

Свен Вермейлен

# **Администрирование системы защиты SELinux**

# SELinux System Administration

Ward off traditional security permissions and effectively secure your Linux systems with SELinux

Sven Vermeulen



BIRMINGHAM - MUMBAI

# Администрирование системы защиты SELinux

Рассмотрение традиционных решений  
по обеспечению безопасности и эффективной защиты  
операционных систем семейства Linux с помощью  
средств SELinux

Свен Вермейлен



Москва, 2020

УДК 004.451  
ББК 32.972.1  
В34

**Вермейлен С.**

В34 Администрирование системы защиты SELinux / пер. с англ. В. Л. Верещагина, О. К. Севостьяновой. – М.: ДМК Пресс, 2020. – 300 с.: ил.

**ISBN 978-5-97060-557-8**

Эта книга показывает, как значительно усилить безопасность операционной системы Linux и устранить имеющиеся уязвимости установленных приложений.

Вы узнаете, как работает SELinux, как можно настроить ее под свои нужды и усилить с ее помощью защиту систем виртуализации, включающих технологию libvirt (sVirt) и контейнеризацию Docker. Также рассказывается об управляющих действиях, позволяющих улучшить безопасность конкретной системы с помощью принудительного контроля доступа – стратегии защиты, определяющей безопасность Linux уже много лет. Большинство возможностей системы защиты рассматривается на реальных примерах.

Книга предназначена для администраторов операционной системы Linux, в задачу которых входит управление ее защищенностью.

УДК 004.451  
ББК 32.972.1

Authorized Russian translation of the English edition of SELinux System Administration ISBN 978-1-78712-695-4 © Packt Publishing.

This translation is published and sold by permission of Packt Publishing, which owns or controls all rights to publish and sell the same.

Все права защищены. Любая часть этой книги не может быть воспроизведена в какой бы то ни было форме и какими бы то ни было средствами без письменного разрешения владельцев авторских прав.

ISBN 978-1-78712-695-4 (анг.)

© Packt Publishing

ISBN 978-5-97060-557-8 (рус.)

© Дополнительный текст, Верещагин В. Л., 2020

© Оформление, издание, перевод, ДМК Пресс, 2020

# Содержание

Об авторе .....	13
О рецензентах.....	15
Предисловие .....	16
<b>Глава 1. Фундаментальные концепции SELINUX.....</b>	<b>21</b>
1.1. Предоставление большей безопасности в Linux.....	21
1.1.1. Использование модулей безопасности Linux .....	24
1.1.2. Расширение возможностей стандартного дискреционного разграничения доступа .....	25
1.1.3. Ограничение привилегий пользователя root.....	27
1.1.4. Сокращение воздействия уязвимостей.....	28
1.1.5. Включение возможностей SELinux в операционной системе .....	29
1.2. Маркировка всех ресурсов и объектов .....	31
1.2.1. Описание параметров безопасности.....	32
1.2.2. Принудительный доступ посредством типов функциональных ограничений .....	35
1.2.3. Распределение по ролям наборов функциональных ограничений ....	36
1.2.4. Разделение пользователей по ролям.....	38
1.2.5. Контроль информационных потоков посредством мандатного механизма .....	39
1.3. Формирование и распределение политик .....	40
1.3.1. Создание политик SELinux .....	41
1.3.2. Распределение политик в виде модулей.....	43
1.3.3. Комплектация модулей в хранилище политик.....	45
1.4. Различия между политиками.....	45
1.4.1. Поддержка многоуровневой защиты (MLS) .....	46
1.4.2. Манера поведения с неизвестными разрешениями .....	46
1.4.3. Поддержка неограниченных доменов.....	47
1.4.4. Ограничение межпользовательского обмена .....	48
1.4.5. Последовательные изменения версий политик .....	49
1.4.6. Качественное изменение версий политик.....	50
1.5. Заключение.....	51
<b>Глава 2. Режимы работы и регистрация событий .....</b>	<b>53</b>
2.1. Включение и выключение защиты SELinux.....	53
2.1.1. Установка глобального состояния защиты.....	54

2.1.2. Переключение в рекомендательный и принудительный режимы .....	55
2.1.3. Использование параметров загрузки ядра .....	57
2.1.4. Отключение защиты SELinux для отдельно взятого сервиса.....	58
2.1.5. Определение приложений, активно взаимодействующих с SELinux.....	61
2.2. Регистрация событий и аудит в SELinux.....	61
2.2.1. Последовательность контроля событий о нарушениях безопасности.....	62
2.2.2. Исключение конкретных отказов в доступе из числа регистрируемых.....	64
2.2.3. Конфигурирование подсистемы контроля событий безопасности Linux.....	65
2.2.4. Настройка локального системного регистратора событий.....	66
2.2.5. Разбор информации об отказах SELinux .....	67
2.2.6. Другие типы событий, связанные с SELinux .....	72
2.2.7. Использование команды ausearch .....	76
2.3. Получение помощи при отказах.....	77
2.3.1. Диагностика неисправности с помощью службы setroubleshoot .....	77
2.3.2. Отправка электронной почты, когда случился отказ SELinux .....	80
2.3.3. Использование утилиты audit2why .....	81
2.3.4. Взаимодействие с журналом system.....	82
2.3.5. Использование здравого смысла .....	83
2.4. Заключение.....	85
<b>Глава 3. Управление учетными записями пользователей.....</b>	<b>86</b>
3.1. Параметры безопасности пользователей.....	86
3.1.1. Сложность допустимого набора функций.....	87
3.1.2. Определение неограниченных доменов .....	89
3.2. Пользователи SELinux и их роли.....	90
3.2.1. Перечень сопоставлений пользователей с пользовательскими типами SELinux.....	90
3.2.2. Сопоставление учетных записей с пользовательскими типами .....	92
3.2.3. Настройка учетных записей относительно служб .....	93
3.2.4. Создание типов пользователей SELinux.....	94
3.2.5. Перечень типов допустимого набора функций у ролей.....	95
3.2.6. Управление категориями .....	96
3.3. Управление ролями SELinux.....	98
3.3.1. Настройки присвоения допустимых ролей пользователю .....	98
3.3.2. Проверка параметров безопасности при помощи утилиты getseuser .....	100
3.3.3. Подключение ролей с помощью команды newrole .....	100
3.3.4. Управление доступом к роли с помощью команды sudo .....	101
3.3.5. Переключение параметров безопасности посредством runcon.....	102

3.3.6. Переключение на системную роль .....	102
3.4. SELinux и PAM (подключаемые модули аутентификации) .....	104
3.4.1. Назначение параметров безопасности с помощью подключаемых модулей аутентификации .....	104
3.4.2. Запрещение доступа в рекомендательном режиме работы защиты .....	105
3.4.3. Многоэкземпляльность каталогов .....	106
3.5. Заключение.....	107

<b>Глава 4. Домены как допустимые наборы функций для процессов и контроль доступа на уровне файлов .....</b>	<b>109</b>
4.1. О параметрах безопасности файлов.....	109
4.1.1. Получение информации о параметрах безопасности.....	110
4.1.2. Интерпретация наименований типов SELinux .....	111
4.2. Закрепление параметров безопасности за объектом и их игнорирование.....	112
4.2.1. Наследование параметров безопасности по умолчанию.....	113
4.2.2. Правила преобразования типов и их вывод .....	113
4.2.3. Копирование и перемещение файлов .....	115
4.2.4. Временное изменение параметров безопасности файла .....	117
4.2.5. Установка категорий для файлов и каталогов .....	118
4.2.6. Использование многоуровневой защиты для файлов .....	118
4.2.7. Резервное копирование и восстановление расширенных атрибутов .....	119
4.2.8. Использование опций монтирования для установки параметров SELinux.....	119
4.3. Формулировка параметров безопасности для файлов .....	121
4.3.1. Использование выражений, описывающих параметры безопасности.....	121
4.3.2. Регистрация изменений параметров безопасности файлов .....	123
4.3.3. Использование заказных типов .....	125
4.3.4. Различные виды файлов file_contexts и их компиляция.....	126
4.3.5. Обмен локальными изменениями.....	127
4.4. Изменение параметров безопасности у файлов.....	127
4.4.1. Использование команд setfiles, rlpkg и fixfiles .....	128
4.4.2. Изменение параметров безопасности на всей файловой системе ...	128
4.4.3. Автоматическое приведение к заданным значениям изменившихся параметров безопасности .....	129
4.5. Параметры безопасности процесса .....	130
4.5.1. Получение параметров безопасности процесса .....	130
4.5.2. Преобразование типа процесса .....	131
4.5.3. Проверка соответствия параметров безопасности .....	133
4.5.4. Другие способы преобразования типов .....	134

4.5.5. Изначально заданные параметры в структуре идентификатора безопасности.....	134
4.6. Определение границ возможных преобразований.....	135
4.6.1. Очистка переменных окружения во время преобразования к другому типу .....	135
4.6.2. Невыполнение преобразований, когда нет ограничивающего родительского типа .....	137
4.6.3. Использование флага, исключающего новые привилегии у процесса.....	138
4.7. Типы, разрешения и ограничения .....	139
4.7.1. Объяснение атрибутов типа .....	139
4.7.2. Запрос разрешений, предоставленных типу процесса.....	140
4.7.3. Рассмотрение наложенных ограничений.....	142
4.8. Заключение.....	143
<b>Глава 5. Контроль сетевого взаимодействия.....</b>	<b>144</b>
5.1. От контроля межпроцессного взаимодействия (IPC) до сокетов базовых протоколов (TCP/UDP) транспортного уровня.....	144
5.1.1. Использование разделяемой памяти .....	145
5.1.2. Локальное взаимодействие, осуществляемое по каналам .....	147
5.1.3. Обращение через сокеты домена UNIX.....	148
5.1.4. Рассмотрение сокетов netlink .....	149
5.1.5. Действия с сокетами протоколов TCP и UDP .....	150
5.1.6. Вывод списка сетевых соединений с параметрами безопасности....	152
5.2. Межсетевой экран и маркировка сетевых пакетов .....	152
5.2.1. Вводные сведения о межсетевом экране netfilter.....	153
5.2.2. Реализация маркировки сетевых пакетов и соединений .....	154
5.2.3. Назначение меток пакетам .....	155
5.3. Промаркированные сети .....	157
5.3.1. Резервная маркировка в NetLabel.....	158
5.3.2. Ограничение потоков данных на уровне сетевого интерфейса.....	160
5.3.3. Ограничение потоков данных на уровне элементов сети .....	160
5.3.4. Проверка однорангового потока.....	161
5.3.5. Применение управления в старом стиле .....	162
5.4. Метки безопасности для IPsec .....	163
5.4.1. Установка стандартного IPsec .....	165
5.4.2. Подключение маркировки IPsec .....	166
5.4.3. Использование Libreswan .....	167
5.5. Технология маркировки сетей NetLabel с параметром CIPSO.....	168
5.5.1. Настройка сопоставлений потоков данных с доменами .....	169
5.5.2. Добавление сопоставлений для типов допустимого набора функций .....	170
5.5.3. Локальное использование параметра CIPSO .....	171



5.5.4. Поддержка опции безопасности для IPv6 .....	172
5.6. Заключение.....	172
<b>Глава 6. Поддержка sVirt и Docker .....</b>	<b>173</b>
6.1. Виртуализация, защищенная SELinux .....	173
6.1.1. Представление о виртуализации .....	173
6.1.2. Обзор рисков виртуализации.....	175
6.1.3. Использование типов для объектов виртуальной инфраструктуры .....	176
6.1.4. Перенастраиваемое применение существующих типов виртуализации.....	177
6.1.5. Рассмотрение защиты различных категорий .....	179
6.2. Поддержка библиотеки libvirt .....	180
6.2.1. Различные случаи маркировки ресурсов .....	181
6.2.2. Оценка архитектуры libvirt .....	181
6.2.3. Настройка libvirt для работы с sVirt.....	182
6.2.4. Использование статических параметров безопасности .....	184
6.2.5. Гибкая настройка параметров безопасности.....	184
6.2.6. Использование разных мест хранения.....	185
6.2.7. Интерпретация информации в поле вывода данных о метке .....	185
6.2.8. Управление доступными категориями.....	186
6.2.9. Поддержка интерпретирующих доменов .....	186
6.2.10. Изменение параметров безопасности, установленных по умолчанию .....	187
6.3. Защищенные контейнеры Docker.....	188
6.3.1. Представление о защите контейнера .....	188
6.3.2. Интеграция системы защиты с контейнерами без sVirt.....	189
6.3.3. Перестраховка безопасности Docker средствами защиты sVirt .....	190
6.3.4. Ограничение привилегий контейнера .....	191
6.3.5. Применение различных параметров безопасности для контейнеров .....	193
6.3.6. Перемаркировка подключенного тома данных.....	194
6.3.7. Понижение контроля со стороны SELinux для специальных контейнеров.....	195
6.3.8. Изменение параметров безопасности, установленных по умолчанию .....	195
6.4. Заключение.....	196
<b>Глава 7. D-Bus и systemd .....</b>	<b>197</b>
7.1. Фоновый процесс системы (systemd).....	197
7.2. Способ поддержки в systemd служб .....	198
7.2.1. Введение понятия модульных файлов.....	198

7.2.2. Установка параметров безопасности SELinux для какой-либо службы.....	199
7.2.3. Использование переходных служб.....	200
7.2.4. Требование включения или отключения SELinux для конкретной службы .....	201
7.2.5. Перемаркировка файлов во время запуска службы.....	202
7.2.6. Использование активизации, основанной на сокетах.....	204
7.2.7. Управление доступом к операциям с модулями .....	205
7.3. Регистрация событий с помощью systemd .....	206
7.3.1. Получение информации, относящейся к SELinux.....	206
7.3.2. Запрос событий, содержащих параметры безопасности SELinux.....	207
7.3.3. Интеграция диагностики неисправностей с журналом .....	207
7.4. Использование контейнеров systemd .....	209
7.4.1. Инициализация контейнеров systemd.....	209
7.4.2. Использование специальных параметров безопасности SELinux.....	209
7.5. Управление файлами устройств .....	210
7.5.1. Использование правил udev .....	210
7.5.2. Назначение метки SELinux на узле устройства.....	212
7.6. Взаимодействие с шиной сообщений D-Bus .....	212
7.6.1. Представление о взаимодействии между процессами D-Bus .....	212
7.6.2. Контроль получения доступа к службам с помощью SELinux .....	215
7.6.3. Управление потоками сообщений .....	216
7.7. Заключение .....	217
<b>Работа с политиками SELinux.....</b>	<b>218</b>
8.1. Логические параметры SELinux.....	218
8.1.1. Вывод списка логических параметров .....	219
8.1.2. Изменение значений логических параметров .....	220
8.1.3. Проверка влияния логического параметра.....	221
8.2. Усиление политик SELinux .....	222
8.2.1. Список модулей политики .....	222
8.2.2. Загрузка и удаление модулей политики.....	223
8.2.3. Создание политик с использованием программы audit2allow.....	224
8.2.4. Использование говорящих за себя наименований для модулей политики .....	226
8.2.5. Использование макрокоманд посреднической политики с программой audit2allow .....	227
8.2.6. Использование скрипта selocal.....	228
8.3. Создание модулей политик по специальным требованиям .....	229
8.3.1. Создание модулей SELinux с помощью исходного языка описания политик .....	230
8.3.2. Создание модулей SELinux с помощью посреднического стиля описания политик .....	231

8.3.3. Создание модулей SELinux с помощью обобщенно-промежуточного языка.....	232
8.3.4. Добавление описаний для параметров безопасности файла .....	232
8.4. Создание ролей и пользовательских типов допустимого набора функций .....	233
8.4.1. Создание файла <code>pgsql_admin.te</code> .....	233
8.4.2. Создание прав пользователя.....	234
8.4.3. Предоставление доступа для взаимодействия с командным интерфейсом.....	235
8.4.4. Формирование структуры файлов пользовательской политики.....	236
8.5. Создание новых типов для приложений.....	237
8.5.1. Создание файлов <code>mojomojo.*</code> .....	238
8.5.2. Создание интерфейсов политик .....	239
8.5.3. Создание структуры файлов политики для приложений.....	240
8.6. Замена существующих политик.....	241
8.6.1. Замена политик Red Hat Enterprise Linux .....	241
8.6.2. Замена политик в Gentoo .....	243
8.7. Другие варианты усиления политики безопасности.....	244
8.7.1. Создание типов SECMARK по специальным требованиям .....	244
8.7.2. Регистрация попыток доступа в журнале событий.....	245
8.7.3. Создание типов, соответствующих специальным требованиям .....	245
8.8. Заключение.....	246
<b>Глава 9. Анализ поведения политики .....</b>	<b>248</b>
9.1. Одноступенчатый анализ.....	248
9.1.1. Использование различных файлов политик SELinux.....	249
9.1.2. Отображение информации об объектах политики .....	249
9.1.3. Применение утилиты <code>sesearch</code> .....	251
9.1.4. Запрос разрешающих правил .....	251
9.1.5. Запрос сведений о правилах преобразования типов .....	251
9.1.6. Запрос правил для других типов.....	252
9.1.7. Запрос правил, связанных с ролями .....	253
9.1.8. Отображение данных с помощью графической программы <code>apol</code> .....	253
9.2. Анализ преобразований типов процессов .....	257
9.2.1. Использование программы <code>apol</code> .....	258
9.2.2. Использование программы <code>sedta</code> .....	259
9.3. Анализ потоков информации .....	261
9.3.1. Использование программы <code>apol</code> для анализа потоков информации .....	262
9.3.2. Использование программы <code>seinfoflow</code> для анализа потоков информации .....	265
9.4. Другие виды анализа политик .....	266
9.4.1. Сравнение политик при помощи <code>sediff</code> .....	266

---

9.4.2. Анализ политик при помощи sepolicy.....	267
9.5. Заключение.....	268
<b>Глава 10. Частные случаи настройки защиты.....</b>	<b>269</b>
10.1. Усиление защиты веб-серверов .....	269
10.1.1. Описание условий работы .....	270
10.1.2. Настройка для установки нескольких экземпляров программ .....	271
10.1.3. Создание категорий SELinux .....	272
10.1.4. Выбор необходимых параметров безопасности .....	273
10.1.5. Включение администраторов в систему защиты .....	275
10.1.6. Управление работой веб-сервера.....	275
10.1.7. Работа с обновлением содержания .....	277
10.1.8. Настройка сети и правил межсетевого экрана.....	279
10.2. Защита командно-строчного интерфейса .....	279
10.2.1. Разделение SSH на несколько экземпляров .....	280
10.2.2. Обновление правил работы сети .....	281
10.2.3. Изменение корневого каталога для отдельной программы .....	282
10.2.4. Предоставление параметров безопасности пользователю в зависимости от способа доступа .....	283
10.2.5. Настройка правил для SSH .....	285
10.2.6. Включение многопользовательского режима использования .....	286
10.3. Общий доступ к файлам через сетевую файловую систему NFS .....	287
10.3.1. Базовая настройка службы NFS .....	287
10.3.2. Включение поддержки NFS на стороне защищенного клиента .....	288
10.3.3. Настройка правил безопасности для NFS на сервере .....	288
10.3.4. Подключение общих сетевых ресурсов с различными параметрами безопасности .....	289
10.3.5. Работа с промаркированной сетевой файловой системой .....	290
10.3.6. Сравнение файлового сервера Samba с сетевой файловой системой NFS .....	291
10.4. Заключение.....	292
<b>Предметный указатель.....</b>	<b>293</b>

# Об авторе

**Свен Вермейлен** (Sven Vermeulen) – постоянный участник различных проектов свободного программного обеспечения и автор многочисленных руководств и ресурсов в интернете. Свой первый опыт в разработке свободного программного обеспечения он получил в 1997 году и с тех пор только развивал и совершенствовал свои навыки в этом направлении. В 2003 году он присоединился к проекту Gentoo Linux как разработчик документации и затем выступал в разных ролях, включая такие, как доверенное лицо фонда Gentoo, член совета, руководитель проекта по различным инициативам в области документирования, а также руководитель проектов по усилению системой защиты SELinux операционной системы Gentoo и системному интегрированию.

В течение этого времени Свен получил экспертные знания как на уровне операционной системы, так и на уровне серверного прикладного программного обеспечения. Он использовал свой интерес к безопасности, чтобы направлять свои проекты, связанные с формированием руководств, в область защиты информации. Для этой цели им стали применяться:

- языки описания политики безопасности, механизмов контроля и результаты оценки SCAP (*Security Content Automation Protocol* – Протокол автоматизации информационного обеспечения безопасности);
- средства контроля разграничения доступа, реализованные в SELinux;
- аутентификация при помощи средства обеспечения защиты PAM (*Pluggable Authentication Modules* – подключаемые модули аутентификации);
- программные межсетевые экраны
- и многое другое.

Для SELinux Свен внес несколько вариантов политик в проект Посреднической политики (Reference Policy project), и он является активным участником проектов по разработке политик и пользовательского пространства.

В своей ежедневной работе Свен – архитектор информационных технологий в одном из европейских финансовых институтов, а также самостоятельно действующий инженер и консультант. Создание безопасных инфраструктур (и сопутствующая архитектурная интеграция) является, конечно, важной частью его работы. Свое образование – степень магистра компьютерной инженерии – Свен Вермейлен получил в Бельгии, в университете города Гент. Вторая степень была получена в магистратуре организации INNOCOM (<https://www.inno.com>) по специальности информационно-коммуникационной архитектуры предприятия. Работал инженером инфраструктуры веб-приложений.

Свен является основным автором книги *Gentoo Handbook* (Справочник по Gentoo), которая охватывает вопросы установки и настройки операционной системы Gentoo на нескольких архитектурах. Он также автор публикации в интернете *Linux Sea* (Mope Linux) – [http://swift.siphos.be/linux\\_sea](http://swift.siphos.be/linux_sea), которая является

базовым введением в операционную систему Linux для начинающих системных администраторов. Кроме того, он является автором таких книг издательства Packt Publishing, как *SELinux System Administration* (Администрирование системы защиты SELinux, 1-е изд.) и *SELinux Cookbook* (Книга готовых рецептов для системы защиты SELinux).

Я хотел бы поблагодарить сообщество разработчиков ПО с открытым исходным кодом и свободного программного обеспечения за его бесконечное стремление создавать отличное программное обеспечение, документацию, настоящие произведения искусства и сервисы. Именно благодаря этому стремлению компании и организации во всем мире пользуются предоставляемыми средствами высокого качества со всей свободой, которую дает это программное обеспечение. В частности, я хотел бы поблагодарить сообщество Gentoo, поскольку оно предоставляет отличные метадистрибутив и операционную систему. Люди, которых я там встречаю, – все они очень высоко мотивированные, опытные и/или эксперты в определенных областях. Присутствие в сообществе заставляет меня стремиться узнать больше.

# О рецензентах

**Дэвид Куигли** (David Quigley) начал свою карьеру исследователем компьютерных систем в Национальной исследовательской лаборатории по обеспечению информационной безопасности при Агентстве национальной безопасности США, где он работал в качестве члена команды SELinux. Дэвид возглавил работы по проектированию и реализации маркировки сетевой файловой системы NFS в SELinux. До этого участвовал в сообществе открытого программного обеспечения, поддерживая кодовую базу проекта вспомогательной файловой системы Unionfs 1.0, и вносил свой вклад в различные другие проекты. Дэвид выступал с докладами на таких конференциях, как Оттавский симпозиум по Linux, семинар по StorageSS, LinuxCon, а также на нескольких локальных собраниях групп пользователей Linux, где темы презентаций включали хранение, файловые системы и безопасность. В настоящее время Дэвид работает инженером по ядру файловой системы ZFS в отделе высокопроизводительных данных в Intel. И ранее рецензировал книгу *SELinux Cookbook*, опубликованную издательством Packt.

Я хотел бы поблагодарить мою замечательную жену Кэти за все, что она делает, чтобы у меня было время заняться такими вещами, как обзор этой книги, и поездками, связанными с презентациями о SELinux. Она – радость моей жизни и помогла мне стать тем, кем я являюсь сегодня. Я также хотел бы поблагодарить моих детей Зою Джейн и Кэрлайн, которые напоминают нам о том, что нужно любить и ценить время, которое мы проводим вместе с семьей.

**Сэм Уилсон** (Sam Wilson) – старший инженер по системам и безопасности, недавно увлекся конструированием радиотехнического оборудования и специализируется на Red Hat Enterprise Linux. Благодаря обширным знаниям в области безопасности, охватывающим микросервисы, инфраструктуру, и в организации работы команды при обеспечении коллективных целей по обеспечению безопасности (*SecOps*) к Сэму регулярно обращаются за наставничеством и советами по SELinux организации, с которыми он сотрудничает и с которыми работает. Сэм активно участвует в сообществах GNU/Linux с начала 2007 года и добровольно посвятил себя работе над проектами NTFreenet, Darwin Community Arts, Ansible и Fedora.

Сэм является автором сайта <https://www.cycloptivity.net>, а также работает с командой интеллектуальной безопасности Atlassian Security Intelligence над визуализацией, эксплуатационной безопасностью и средствами управления для поддержки и защиты клиентов Atlassian в облаке.

# Предисловие

Безопасное состояние операционной системы или какой-либо службы является результатом многоуровневого подхода к обеспечению безопасности. Системы могут быть защищены от внешнего мира при помощи межсетевых экранов, операционные системы должны регулярно получать обновления безопасности, работающие службы должны быть правильно настроены, необходимо разделять обязанности для конечных пользователей и т. д.

Контроль доступа – это еще один уровень обеспечения защиты, который администраторы должны учитывать. Благодаря системе защиты SELinux (Security Enhanced Linux – повышенная безопасность Linux) в экосистеме Linux появилась надежная и, что удобно, встраиваемая система принудительного контроля доступа. Некоторые дистрибутивы включают SELinux по умолчанию, другие позволяют администраторам включать ее самим. Android, одна из самых популярных операционных систем для мобильных устройств, также использует технологию SELinux под названием SEAndroid.

Но, в отличие от Android, где пользователи и приложения находятся под жестким контролем и где недопустимы отклонения в настройке и организации файлов и ресурсов, настольные компьютеры, рабочие станции и серверы, которые реализуют Linux, имеют большее разнообразие в возможностях управления защитой. В результате настройка SELinux в этих системах требует больше знаний о том, что такое SELinux, как он работает и как его можно использовать.

В этой книге мы обсуждаем, что такое SELinux и как он встроен в операционную систему Linux. Мы рассмотрим различные аспекты конфигурации SELinux и разберем несколько вариантов применения, которые используют сильные стороны SELinux для дальнейшего усиления безопасности системы и служб, размещенных на ней.

## О ЧЕМ ЭТА КНИГА

Глава 1 «*Фундаментальные концепции SELinux*» дает администраторам представление о том, что такое система защиты SELinux и как она взаимодействует на уровне ядра операционной системы Linux. Здесь объясняются различия в реализациях SELinux между дистрибутивами и описывается характерная для SELinux терминология, которая дальше будет часто использоваться по мере углубления в технологию SELinux.

Глава 2 «*Режимы работы и регистрация событий*» описывает различные состояния работы SELinux и показывает, где SELinux регистрирует свои события. Эта глава поможет администраторам разобраться с тем, как следует интерпретировать и анализировать эти события.



Глава 3 «*Управление учетными записями пользователей*» рассказывает администраторам, как управлять пользователями Linux и их правами, а также выполнять сопоставление этих пользователей с различными ролями, которые SELinux поддерживает с помощью собственной организации пользовательского пространства и подключаемых модулей аутентификации Linux. Кроме того, глава охватывает такую сущность, как категории защищаемой информации, реализованные в SELinux.

Глава 4 «*Домены как допустимые наборы функций для процессов и контроль доступа на уровне файлов*» знакомит администраторов с метками параметров безопасности SELinux. С тем, как эти метки хранятся в файловой системе или предоставляются другим ресурсам. В этой главе администраторы и конечные пользователи узнают, как устанавливать и обновлять метки параметров безопасности.

Глава 5 «*Контроль сетевого взаимодействия*» рассматривает стандартные службы сетевой безопасности, утилиту iptables и протокол IPSec с точки зрения их совместной работы с функциями защиты SELinux. Администраторы смогут научиться включать поддержку SELinux в этих службах безопасности и даже включать маркировку, выполняемую между распределенными по сети системами, при помощи таких методов, как Labeled IPSec и NetLabel/CIPSO.

Глава 6 «*Поддержка sVirt и Docker*» рассказывает, как компания Red Hat разработала технологию защищенной виртуализации (sVirt) и реализовала ее в двух системах виртуализации: библиотеке libvirt и контейнерах Docker. В этой главе объясняется, как настроить эти службы с помощью поддержки SELinux и контролировать перемещение ресурсов, используемых гостевыми системами виртуальной инфраструктуры или контейнерами.

Глава 7 «*D-Bus и systemd*» рассказывает о сферах влияния упомянутых системных служб уровня ядра и о том, как они используют правила SELinux для дальнейшего усиления безопасности своих собственных функциональных возможностей. Получив эти знания, администраторы смогут настроить защищенную работу службы межпроцессного взаимодействия D-Bus, а также управлять средствами доступа SELinux, применяемыми через подсистему инициализации systemd.

Глава 8 «*Работа с политиками SELinux*» посвящена настройке и управлению политиками SELinux. Она показывает, как можно создавать политики безопасности по заданным требованиям или даже заменять политику, официально предоставляемую в рамках дистрибутива.

Глава 9 «*Анализ поведения политики*» углубляется в инструменты анализа, которые позволяют инженерам и администраторам более детально рассматривать политику безопасности SELinux. С помощью этих инструментов можно получить наиболее полное представление о том, что политика в себя включает и как поведет себя в различных ситуациях.

Глава 10 «*Частные случаи настройки защиты*» описывает ряд распространенных случаев использования серверов, таких как веб-серверы и файловые серверы, и способы использования SELinux для их защиты. В этой главе рас-

сказывается, как можно изолировать пользовательское окружение с помощью SELinux и как администраторы могут построить защищенную многопользовательскую систему.

## Что необходимо для понимания книги

Поскольку SELinux является компонентом для операционной системы Linux, то читателям желательно иметь ее в своем распоряжении вместе с установленной системой защиты SELinux. Процесс установки SELinux не входит в материал данной книги, по этому вопросу стоит обратиться к документации используемого дистрибутива. Кроме того, настройка системы защиты требует наличия привилегий администратора в системе.

## Для кого предназначена эта книга

Эта книга предназначена для системных администраторов Linux, которые имеют достаточный опыт обслуживания систем Linux, хотят разбираться в технологии защиты SELinux и работать с ней. Кроме того, данная книга может быть полезна для архитекторов информационных систем, чтобы понять, как можно применять SELinux для повышения безопасности систем Linux и служб, работающих под управлением этой операционной системы для обеспечения нужд компании.

## Соглашения

В этой книге вы найдете несколько стилей оформления текста, которые позволяют отличать одни виды информации от других. Вот примеры этих стилей и объяснение их значения.

Ключевые слова в тексте, имена таблиц базы данных, каталогов, файлов, расширения файлов, имена путей, адреса сайтов и данные, вводимые пользователем, выделены моноширинным шрифтом: «Мы выполняем это с помощью команды `semanage login`».

Блок текста, связанного с настройками, выводом команд и программами, выглядит в тексте книги так:

```
dbadm_r
  Dominated roles:
    dbadm_r
  Types:
    qmail_inject_t
    dbadm_t
    ...
    user_mail_t
```

Любая фраза командно-строчного интерфейса дополнительно выделяется жирным шрифтом:

```
# seinfo -amcs_constrained_type -x | grep virt_
```

**Новые термины** и **важные слова** выделены жирным шрифтом в тексте описания. Слова, которые пользователь может увидеть на экране (например, в меню или диалоговых окнах), отображаются в тексте книги примерно так: «После загрузки выберите пункт меню **Новый анализ** (New Analysis), для того чтобы начать работу с функциями анализа политики».

**i** Предупреждения и важные замечания отмечаются таким значком.

**✓** Советы и подсказки выглядят так.

**P** Тематические пояснения для русскоязычного издания.

## ОТЗЫВЫ И ПОЖЕЛАНИЯ

Мы всегда рады отзывам наших читателей. Расскажите нам, что вы думаете об этой книге – что понравилось или, может быть, не понравилось. Отзывы важны для нас, чтобы выпускать книги, которые будут для вас максимально полезны.

Вы можете написать отзыв прямо на нашем сайте [www.dmkpress.com](http://www.dmkpress.com), зайдя на страницу книги, и оставить комментарий в разделе «Отзывы и рецензии». Также можно послать письмо главному редактору по адресу [dmkpress@gmail.com](mailto:dmkpress@gmail.com), при этом напишите название книги в теме письма.

Если есть тема, в которой вы квалифицированы, и вы заинтересованы в написании новой книги, заполните форму на нашем сайте по адресу [http://dmkpress.com/authors/publish\\_book/](http://dmkpress.com/authors/publish_book/) или напишите в издательство по адресу [dmkpress@gmail.com](mailto:dmkpress@gmail.com).

## СКАЧИВАНИЕ ИСХОДНОГО КОДА ПРИМЕРОВ

Скачать файлы с дополнительной информацией для книг издательства «ДМК Пресс» можно на сайте [www.dmkpress.com](http://www.dmkpress.com) на странице с описанием соответствующей книги.

## СПИСОК ОПЕЧАТОК

Хотя мы приняли все возможные меры для того, чтобы удостовериться в качестве наших текстов, ошибки все равно случаются. Если вы найдете ошибку в одной из наших книг – возможно, ошибку в тексте или в коде, – мы будем очень благодарны, если вы сообщите нам о ней. Сделав это, вы избавите других читателей от расстройств и поможете нам улучшить последующие версии данной книги.

Если вы найдете какие-либо ошибки в коде, пожалуйста, сообщите о них главному редактору по адресу [dmkpress@gmail.com](mailto:dmkpress@gmail.com), и мы исправим это в следующих тиражах.

## **НАРУШЕНИЕ АВТОРСКИХ ПРАВ**

Пиратство в интернете по-прежнему остается насущной проблемой. Издательства «ДМК Пресс» и Apress очень серьезно относятся к вопросам защиты авторских прав и лицензирования. Если вы столкнетесь в интернете с незаконно выполненной копией любой нашей книги, пожалуйста, сообщите нам адрес копии или веб-сайта, чтобы мы могли применить санкции.

Пожалуйста, свяжитесь с нами по адресу [dmkpress@gmail.com](mailto:dmkpress@gmail.com) со ссылкой на подозрительные материалы.

Мы высоко ценим любую помощь по защите наших авторов, помогающую нам предоставлять вам качественные материалы.

# Глава 1

## Фундаментальные концепции SELINUX

SELinux – система защиты, обеспечивающая повышенную безопасность в Linux. Она включает дополнительные меры защиты в операционную систему для большей целостности, доступности и конфиденциальности ее ресурсов.

В данном разделе будут рассмотрены следующие вопросы:

- 1) почему SELinux использует метки для идентификации ресурсов;
- 2) как качественно SELinux отличается от штатных систем контроля доступа Linux за счет наличия правил обеспечения безопасности;
- 3) каким образом правила контроля доступа повышают уровень защиты, распространяясь через файлы политик безопасности SELinux.

В конце главы будут рассмотрены различия между версиями SELinux, реализованными в дистрибутивах Linux.

### 1.1. ПРЕДОСТАВЛЕНИЕ БОЛЬШЕЙ БЕЗОПАСНОСТИ В LINUX

Опытные администраторы систем, построенных на базе Linux, и проектировщики систем защиты знают, что им необходимо доверять пользователям и процессам в обслуживаемой системе. То есть администраторам требуется заручиться некоторой лояльностью со стороны пользователей и гарантиями для выполняемых процессов в системе, чтобы обеспечивался нужный уровень защищенности. Это частично связано с тем, что пользователи могут пытаться использовать уязвимости, найденные у запущенных программ, с которыми они должны работать «по долгу службы», но еще больший их вклад в снижение уровня угроз связан как раз с тем, что безопасное состояние системы зависит от непосредственного поведения пользователей. Дело в том, что пользователь с доступом к конфиденциальной информации (к такой, которая не должна быть предоставлена в общий доступ), сам же и управляет приложениями, в т. ч. запускает и совершает другие действия, которые оказывают влияние на защиту системы. Одним из механизмов стандартной защиты является дискреционный контроль доступа, который настраивается в соответствии с потребностями пользователя.

Механизм **дискреционного контроля доступа** DAC (*Discretionary Access Control*) в операционной системе Linux основывается на сопоставлении данных процесса, выполняемого пользователем (и/или группой пользователей), с информацией о разрешениях для пользователя (и/или группы), характерных для какого-либо файла, директории или другого управляемого ресурса. Рассмотрим файл `/etc/shadow`, который содержит информацию о паролях и именах пользователей некой локальной системы:

```
$ ls -l /etc/shadow
-rw----- 1 root root 1010 Apr 25 22:05 /etc/shadow
```

Без дополнительных механизмов контроля доступа, имеющихся в наличии, этот файл является разрешенным для чтения и записи любым процессам, которыми владеет пользователь `root`, вне зависимости от целей процесса, преследуемых в системе. Файл `shadow` представляет типичный пример такого конфиденциального файла, который никто из пользователей не хотел бы видеть опубликованным или использованным недобросовестным образом. По сути, кто-то один имеет доступ к данному файлу и может скопировать его в какое-либо другое место, например в домашний каталог, по почте отправить на другой компьютер или попытаться выполнить атаку на защищенное криптографией значение пароля (*hash*) пароля, размещенное в этом файле.

Другой пример того, что стандартный механизм дискреционного контроля доступа требует доверия к пользователям, – это случай, когда в системе расположена база данных. Файлы базы данных сами по себе являются (очень хочется верить) доступными только для сеансовых пользователей, зарегистрированных в *системе управления базой данных* (Database management system – DBMS), и для пользователя операционной системы `root`.

Должным образом защищенные системы будут предоставлять доступ к этим файлам только хорошо проверенным пользователям (например, через команду `sudo`), позволяя им заменить свои эффективные пользовательские идентификаторы на идентификаторы пользователей базы данных или даже на пользователя с правами `root`, и это для строго заданного набора команд. Эти пользователи также могут анализировать файлы базы данных и получать доступ к потенциальной конфиденциальной информации в базе данных без входа через СУБД.

Однако пользователи операционной системы не являются единственной угрозой, из-за которой необходимо защитить систему. Многие фоновые программы (*software daemons*) запускаются с правами пользователя `root` или имеют необходимые для работы привилегии в системе. Ошибки внутри этих программ могут легко привести к утечке информации или даже к удаленному использованию уязвимостей. Программы резервирования, контроля выполнения, администрирования, планирования и им подобные программы: они все часто запускаются с высшими пользовательскими привилегиями, доступными в операционной системе Linux. Даже когда администратор не предоставляет привилегии пользователям, их взаимодействие со службами подразумевает потенциальный риск безопасности. А раз так, то пользователи продолжают

пользоваться своими расширенными правами для корректного взаимодействия с приложениями и обеспечения корректного функционирования системы в целом. Поэтому администратор вынужден строить безопасность системы на порядочности ее пользователей.

Рассмотрим теперь SELinux, который содержит дополнительный уровень контроля доступа, стоящий выше дискреционного механизма. SELinux предоставляет *мандатный контроль доступа* (Mandatory access control – MAC) системы, который нивелирует рассмотренные недостатки дискреционного контроля, предоставляя администратору полный контроль над тем, что является дозволенным в системе, а что нет. Он достигает этого путем применения метода управления политиками, определяющими, являются или не являются процессы разрешенными к выполнению, а также путем приведения этих политик в жизнь через ядро операционной системы Linux.

Мандатные средства, которые контролируют доступ, приводятся в исполнение операционной системой и определяются исключительно правилами политики, задействованными системным администратором (или администратором безопасности). Пользователи и процессы не имеют прав на изменение правил безопасности, таким образом они не могут что-либо делать в части контроля доступа; защита больше не находится во власти свободы их действий.

Здесь слово «мандатный» так же, как и ранее слово «дискреционный», было выбрано для описания возможностей системы контроля доступа не случайно: оба термина являются известными в области информационной безопасности и имеют описания в других публикациях, включая стандарт «Критерии оценки доверенных компьютерных систем»<sup>1</sup> (известный также как «Оранжевая книга»), выпущенный Министерством обороны Соединенных Штатов Америки в 1985 году. Этот документ предшествовал стандарту «Общие критерии»<sup>2</sup>, предназначенному для сертификации компьютерных систем.

**Р** В России данные термины по защите информации определяются и раскрываются в нижеприведенных и других документах:

- 1) Национальный стандарт Российской Федерации. Информационная технология. Методы и средства обеспечения безопасности. Критерии оценки безопасности информационных технологий;
- 2) руководящий документ ФСТЭК<sup>3</sup> «Защита от несанкционированного доступа к информации. Термины и определения». Утверждено решением председателя Гостехкомиссии России от 30 марта 1992 г.;
- 3) руководящий документ ФСТЭК «Средства вычислительной техники. Защита от несанкционированного доступа к информации. Показатели защищенности от несанкционированного доступа к информации».

<sup>1</sup> Trusted Computer System Evaluation Criteria (TCSEC) 1985 [Электронный ресурс] // <http://csrc.nist.gov/publications/history/dod85.pdf>.

<sup>2</sup> Common Criteria (ISO/IEC 15408) [Электронный ресурс] // <http://www.commoncriteria-portal.org/cc/>.

<sup>3</sup> ФСТЭК – Федеральная служба технического и экспортного контроля.

### 1.1.1. Использование модулей безопасности Linux

Снова рассмотрим в качестве примера файл `shadow`. Система мандатного контроля доступа может быть сконфигурирована таким образом, чтобы позволять чтение и запись в файл только ограниченному набору процессов. В системах, настроенных подобным образом, пользователь, зашедший под учетной записью `root`, не имеет прямого доступа к файлу или даже к перемещению его. Он также не может изменить атрибуты файла:

```
# id
uid=0(root) gid=0(root)
# cat /etc/shadow
cat: /etc/shadow: Permission denied
# chmod a+r /etc/shadow
chmod: changing permissions of '/etc/shadow': Permission denied
```

**P** В данном примере автор выполняет проверку текущего пользователя и убеждается, что пользователь зашел под учетной записью `root`. После чего выполняет команду вывода на экран содержимого файла, содержащего защищенную форму представления паролей (hash) паролей, но в результате получает сообщение от системы, что доступ запрещен (Permission denied). Тогда, имитируя мышление взломщика, автор предполагает, что доступ запрещен стандартным механизмом дискреционного доступа, и выполняет команду изменения прав доступа к файлу, пытаясь получить разрешение на чтение для всех пользователей. В результате система снова пишет, что изменение прав доступа для данного файла запрещено.

Команда «`id`» выводит на экран некоторый используемый набор идентификационной информации для текущего пользователя. В данном случае приводится `uid` (сокр. от *user identifier* – идентификатор пользователя) и `gid` (сокр. от *group identifier* – идентификатор группы).

Команда `cat` выводит содержимое текстового файла `/etc/shadow` на экран.

Команда `chmod` выполняет изменение прав доступа к файлам и директориям. Параметр `a` (сокр. от *all* – все) определяет, что изменение прав будет для всех пользователей, а параметр `r` (сокр. от *read* – читать) указывает на то, что файл должен быть разрешен на чтение.

Невозможность изменения атрибутов файла достигается с помощью правил, которые описывают, при каких условиях содержимое файла может быть прочитано. Данные условия определяются в политике SELinux и загружаются, когда система проходит начальную загрузку. Само ядро Linux отвечает за соблюдение этих правил. Мандатная система контроля доступа легко включается в ядро Linux через его поддержку модулей безопасности Linux (Linux Security Modules – LSM):



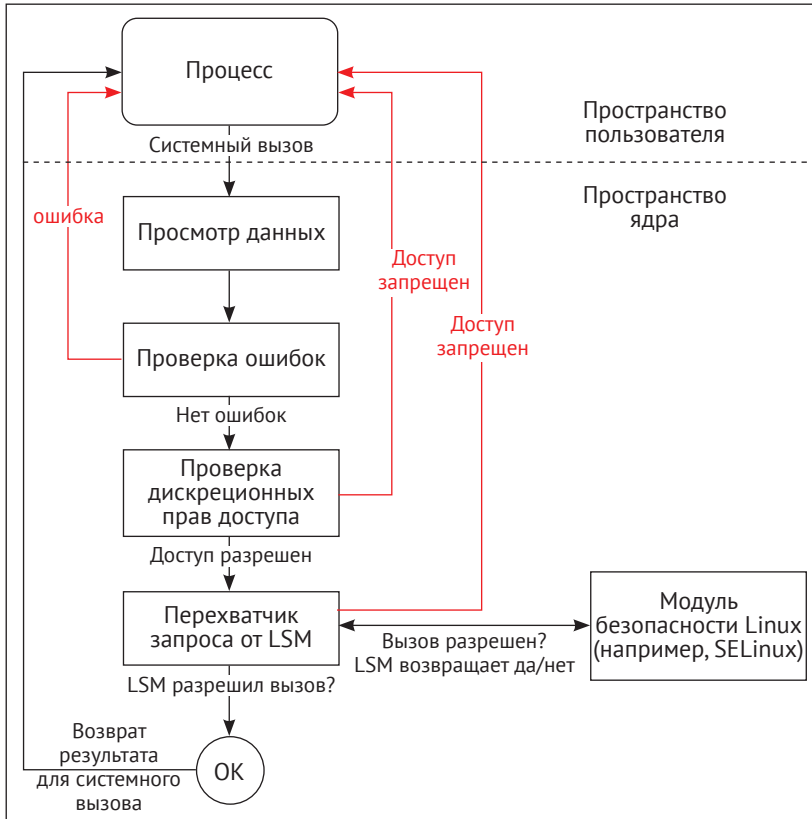


Рис. 1.1 ❖ Высокоуровневое описание процесса интеграции модулей безопасности в ядро Linux

### 1.1.2. Расширение возможностей стандартного дискреционного разграничения доступа

SELinux не изменяет реализацию механизма дискреционного контроля доступа, а также не может отменять правила, установленные механизмом контроля доступа самого Linux. Если стандартная система (без SELinux) исключает кому-то персональный доступ, тогда SELinux не может изменить это решение. Это связано с тем, что модули безопасности системы защиты (LSM<sup>1</sup>) срабатывают *после* включения стандартного дискреционного механизма, что специально спроектировано в рамках реализации проекта LSM.

<sup>1</sup> LSM [Linux Security Module] – модули безопасности операционной системы Linux.

Например, если необходимо предоставить дополнительному пользователю доступ к файлу, то нельзя добавить политику SELinux, которая решит эту задачу. Вместо нее необходимо найти среди других возможностей операционной системы такие, которые используют списки контроля доступа (ACLs<sup>1</sup>) стандарта, обеспечивающего переносимость приложений между различными версиями операционных систем (POSIX<sup>2</sup>). При помощи команд `setfacl` и `getfacl` (распространяются в составе пакета с утилитами для управления доступом ACL) пользователь может установить дополнительные разрешения на файлы или директории, предоставляя выбранный ресурс дополнительным пользователям или группам.

В качестве примера давайте предоставим пользователю `lisa` доступ к некоторому файлу с правом его чтения и записи, используя команду `setfacl`:

```
$ setfacl -m u:lisa:rw /path/to/file
```

**P** Команда `setfacl` (сокращение от фразы «To set file access control list» – «Применить список управления доступом к файлу»), запускаемая из командной строки, предназначена для установки прав доступа к файлу или каталогу. В приведенном примере команда запущена с опцией `-m` (от слова *modify* – модифицировать), указывающей необходимость внесения изменений в некий файл, для которого приведен полный путь размещения в файловой системе `/path/to/file`. С одной стороны, данный путь является условным и образуется совокупностью слов, формирующих фразу «`path to file`» (путь к файлу), а с другой – является вполне допустимой последовательностью вложенных каталогов, где каталог `to` вложен в каталог `path`, а имя самого файла – `file`.

Так как команда распознает разные форматы записи списка контроля доступа (ACL), то в примере приведен тот, который необходим для решения задачи по добавлению пользователя: `[u[ser]:]uid [:perms]`.

В данном формате:

- 1) `u` – сокращение от *user* (пользователь) определяет, что далее будет указан уникальный идентификатор конкретного пользователя – *uid* (*user identifier* – идентификатор пользователя, имеющий числовое значение). В примере идентификатор заменен символьным именем (логином) – `lisa`;
- 2) `perms` – сокращение от *permissions* (полномочия), определяющие права комбинациями обозначений из списка:
  - `r` (*read*) – иметь доступ на чтение файла или каталога;
  - `w` (*write*) – иметь доступ на запись в файл или каталог;
  - `x` (*execute*) – иметь доступ на запуск (исполнение) файла.

В примере указана комбинация параметров «`rw`», позволяющая прочитать содержимое файла и изменить его.

Для успешного выполнения команды необходимо, чтобы была включена поддержка в файловой системе списков контроля доступа (ACL) при подключении (монтировании)

<sup>1</sup> Access Control List [Список контроля доступа] – список контроля доступа (управления доступом).

<sup>2</sup> POSIX [Portable Operating System Interface for computer environments (for Unix)] – интерфейс переносимой операционной системы для компьютерного окружения (сред UNIX).

устройства. Например, в гипотетически возможном содержании файла `/etc/fstab` следует добавить запись `acl` следующим образом:

```
/dev/sda1 / reiserfs noatime,acl 0 2
```

В файловой системе `xfs` поддержка списков контроля доступа включена по умолчанию.

Подобным образом, для того чтобы увидеть примененный к файлу список управления доступом, в формате стандарта (POSIX) следует использовать такую команду:

```
$ getfacl /path/to/file
```

```
1: # file: file
2: # owner: swift
3: # group: swift
4: user::rw
5: user: lisa:rw
6: group::r--
7: mask::r--
8: other::r--
```

**P** Команда `getfacl` (сокращение от фразы «To get file access control list» – «Получить список управления доступом к файлу»), запускаемая из командной строки, предназначена для получения данных об установленных правах доступа для файла или каталога. В приведенном примере команда запущена с указанием полного пути размещения в файловой системе `/path/to/file`.

Из полученных сведений определяется имя объекта (в данном случае `file`), владелец (`swift`), а также владеющая объектом группа (`swift`), каждому из которых предусмотрено соответствующее обозначение: `# file`, `# owner` и `# group`.

В 4-й строке вывода определяются права пользователя-владельца (`swift`), позволяющие читать файл и записывать в него.

В 5-й строке вывода отображаются права доступа пользователя `lisa` – чтение и запись файла – то, что было установлено при помощи утилиты `setfacl`.

В 6-й строке вывода указаны права на чтение для группы `swift`, владеющей файлом.

В 7-й строке вывода, обозначенной словом `mask` (маска), указываются права, которые накладываются на права всех владеющих объектом групп и отдельно указанных пользователей, не считая владельца и так называемых «остальных» пользователей, т.е. которые не указаны явно. Эффективными будут те разрешения, которые совпадают с указанными в маске. Маска рассчитывается автоматически, если не была конкретно задана в команде `setfacl`.

8-я строка определяет права всех остальных пользователей.

В приведенном примере у пользователя `lisa` фактическим (эффективным) правом является только чтение файла, согласно указанной маске.

### 1.1.3. Ограничение привилегий пользователя root

Стандартный механизм дискреционного контроля доступа в Linux доступен для всемогущего пользователя, имеющего имя `root`. В отличие от многих других пользователей системы, пользователь, вошедший под учетной записью `root`, имеет необходимые права для полного управления всей системой, начиная с важнейшего контроля доступа, управления системой регистрации событий и заканчивая изменениями пользовательских идентификаторов, на-

стройкой сети и многим другим. Это все обеспечивается благодаря концепции безопасности, называемой **capabilities**<sup>1</sup> (для общего представления об этих возможностях Linux посмотрите, что написано в руководстве: `man capabilities`). SELinux способен ограничить доступ к этим широким возможностям системы с большой точностью воздействия.

Благодаря этому избирательному разрешающему подходу SELinux даже пользователь `root` может быть ограничен без возникновения конфликтов в системе. Предыдущий пример неудавшегося обращения к файлу `/etc/shadow` является только одним примером деятельности, которую неограниченный в правах пользователь, такой как `root`, все-таки может не быть способным выполнить из-за того, что был применен механизм защиты SELinux.

Когда SELinux был добавлен в ядро операционной системы Linux, некоторые проектные замыслы по защите даже дошли до того, что предоставили общий доступ с правами пользователя `root` к командной оболочке системы, защищенной средствами SELinux, обращаясь с просьбой к хакерам и другим исследователям в области безопасности скомпрометировать такую систему. Умение ограничивать пользователя `root` было доброжелательно встречено системными администраторами, которым иногда приходилось временно передавать пароль от этого пользователя или доступ к командной оболочке, имеющей его права, другим пользователям (к примеру, администраторам базы данных), которым необходимы `root`-привилегии, когда их программное обеспечение выходит из строя. Благодаря SELinux администраторы могут теперь передавать командную оболочку с правами пользователя `root` на время своего отсутствия с уверенностью в том, что пользователь имеет только необходимые ему права и неполные права системного администратора.

#### 1.1.4. Сокращение воздействия уязвимостей

Если существует одно преимущество SELinux, на котором надо сделать акцент из-за частого неверного толкования, то им является именно способность снижения воздействий со стороны уязвимостей.

Хорошо написанная политика безопасности ограничивает прикладные программы таким образом, что разрешенные для них действия сведены к минимально необходимому набору функций. Такая модель наименьших прав (*least-privilege model*) гарантирует, что некорректное поведение приложения не только обнаруживается и регистрируется в журналах, но и предотвращается. Многие уязвимости приложений могут быть использованы для выполнения заданий, которые приложение не намеревалось выполнять. Когда такое случается, SELinux способен это предотвратить.

Однако существует два заблуждения о возможностях этой системы защиты препятствовать реализации таких угроз, и одно из них связано с реализацией политик, а другое – с контролем использования уязвимостей приложений.

<sup>1</sup> В русскоязычном варианте: «возможности», «способности», «мощность».

Если политика не написана с учетом модели наименьших прав, тогда система защиты может расценивать такое нестандартное поведение как нормальное и позволять действиям выполняться. Авторам такой политики следует иметь в виду, что их правила должны быть очень точно детализированы. К сожалению, такой подход к написанию политик очень затратный по времени, т. к. существует более 80 классов и более 200 разрешений, используемых в системе защиты, а в правилах политики необходимо учитывать все эти классы и разрешения для каждого взаимодействия между объектами и субъектами доступа. Вследствие чего тексты политик имеют тенденцию к запутанности и сложности поддержания.

Некоторые авторы создают более либеральные политики, чем это в действительности необходимо, которые могут допустить успешное использование уязвимости, реализованное через действие, являющееся неожиданным поведением даже с точки зрения самой программы. Есть такие политики приложений, которые прямо отмечены как *неограниченные* (unconfined), демонстрируя, что они очень либеральны в своих разрешениях. Разработчики операционной системы Red Hat Enterprise Linux даже исходно создают политики приложений как полностью разрешающие действия субъектов по отношению к объектам и включают усиление контроля доступа для приложений только после выхода нескольких версий (и дополнительного тестирования).

Второе заблуждение – это контроль такой защитой использования злоумышленниками имеющихся уязвимостей.

Если некая уязвимость приложения позволяет не прошедшему аутентификацию пользователю использовать возможности этого приложения, как если бы он был авторизован, тогда SELinux не будет принимать участия в снижении воздействия уязвимостей; система защиты только отмечает поведение самого приложения, но не внутренних действий приложения.

До тех пор, пока приложение ведет себя как ожидается (например, при доступе к своим собственным файлам и не шурует кругом в других файловых системах), система защиты будет успешно позволять действиям совершаться.

И только когда приложение начинает вести себя беспорядочно, система защиты останавливает эксплуатацию программы. Уязвимости, такие как удаленное выполнение команд (remote command execution – RCE) в связке с приложениями, которые не должны выполнять случайные команды (например, системы управления базами данных или веб-сервера, исключая CGI-подобную функциональность), будут предотвращены, несмотря на то что атаки в стиле *sql-инъекций* или *перехвата управления* являются неконтролируемыми через политики SELinux.

### 1.1.5. Включение возможностей SELinux в операционной системе

Включение возможностей дополнительной системы защиты в операционной системе Linux – это не просто вопрос включения модулей безопасности (LSM) в ядро операционной системы.

Реализация системы защиты включает в себя следующее:

- 1) базовую подсистему, реализованную в ядре ОС через модули безопасности (LSM);
- 2) библиотеки, используемые приложениями, которые необходимы для взаимодействия с SELinux;
- 3) утилиты, используемые администраторами для взаимодействия с SELinux;
- 4) политики, которые, собственно, определяют контроль доступа.

Библиотеки и утилиты объединены посредством проекта сообщества SELinux (<https://github.com/SELinuxProject/selinux/wiki>).

**P** Это основной репозиторий (место хранения и поддержания) пользовательских библиотек и инструментов SELinux.

Функции программного обеспечения, предоставляемые данным проектом, дополняются возможностями SELinux, интегрированными в ядро ОС Linux, и используются дистрибутивами операционной системы.

Инструменты предоставляют следующие возможности:

- 1) компиляция политики – низкоуровневые инструменты, которые выполняют преобразование языка политики SELinux, основанного на тексте, в формат, используемый ядром для приведения политики в действие;
- 2) управление политикой – инструменты (такие как `semodule` и `semanage`) и библиотеки (такие как `libsemanage`), используемые для установки, удаления и обновления политик SELinux на работающих системах;
- 3) разработка политики – инструменты, нацеленные на создание и обновление политик (например, `audit2why` и `audit2allow`);
- 4) сервисы SELinux – библиотеки (например, `libselinux`) для приложений, которые должны быть осведомлены о SELinux или исполнять решения в части контроля доступа при помощи SELinux (например, `DBus`);
- 5) утилиты SELinux – низкоуровневые утилиты (такие как `setenforce` и `restorecon`) для управляющей и SELinux-поддерживающей систем.

Программное обеспечение может быть получено как в виде протестированной версии продукта, так и в виде версий из рабочего репозитория.

Вслед за прикладными программами сообщества и библиотеками различные компоненты операционной системы Linux обновляются со специфическим SELinux-кодом, охватывающим подсистему инициализации и некоторые утилиты ядра.

Так как непросто выбрать, что именно надо включить в поставляемый SELinux, дистрибутивы операционных систем, которые поддерживают эту систему защиты, обычно поставляются с предопределенными и загруженными компонентами: операционные системы Fedora и Red Hat Enterprise Linux (с их ответвлениями типа CentOS и Oracle Linux) являются хорошо известными примерами. Некоторые дистрибутивы могут иметь SELinux, не автоматически включенный, но при этом поддерживать легкую установку дополнительных пакетов (как в случае с операционными системами Debian и Ubuntu), а другие могут иметь хорошо документированные способы приведения системы к защищенному варианту средствами SELinux (например, Gentoo и Arch Linux).

В этой книге примеры будут даваться для ОС Gentoo и ОС Red Hat Enterprise Linux версии 7.2. Будут рассматриваться обе операционные системы, потому что они имеют различия в деталях реализации, позволяющие нам продемонстрировать полный потенциал SELinux.

## 1.2. МАРКИРОВКА ВСЕХ РЕСУРСОВ И ОБЪЕКТОВ

Когда SELinux должен решить, следует ли позволить определенное действие или запретить его, он принимает его, основываясь на параметрах безопасности *context* как субъекта (инициирующего действие), так и объекта (являющегося целью действия). Эти параметры (или его части) указаны в правилах политики, которую применяет SELinux.

Параметры безопасности процесса – это то, что идентифицирует процесс в SELinux. SELinux не имеет представления о принадлежности процессов Linux и, будучи однажды запущенной, не заботится о том, как процесс называется, какой идентификационный номер (ID) имеет и с какой учетной записью связан. Все, что ей надо знать, – это каковы параметры безопасности у процесса, что и представлено пользователям и администраторам как **метка**. *Метка* и *параметры безопасности* часто используются как взаимозаменяемые, и хотя здесь есть небольшая разница (поскольку одно понятие является отображением другого), мы не будем долго на этом задерживаться.

Давайте посмотрим на пример метки: параметры безопасности обычного пользователя (вы можете попробовать это сами, если находитесь во включенном режиме SELinux):

```
$ id -Z
unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
```

Команда `id`, которая дает информацию о текущем пользователе, представлена здесь ключом `-Z` (принятый общий ключ для отображения дополнительной информации по безопасности, полученной из подсистем защиты, основанных на LSM). Он показывает нам параметры безопасности текущего пользователя (в действительности параметры самого процесса идентификации в момент его выполнения). Как мы можем видеть, параметры безопасности имеют линейную структуру строки и выглядят так, как если бы у них было пять полей (на самом деле это не так, у них четыре поля, последнее из них просто содержит знак двоеточия).

Разработчики SELinux решили использовать метки вместо метаданных реального процесса и файла (или другого ресурса) для своего механизма контроля доступа. Это отличается от других систем защиты, реализующих мандатное разграничение доступа, например таких, как AppArmor, которые используют путь исполняемого файла (и это имя процесса) и пути ресурсов для управления проверками доступа. Решение сделать SELinux системой, основанной на механизме мандатного контроля доступа, применяющем метки, было принято по следующим нескольким причинам:

- использование путей могло бы быть более легким для понимания администраторами, но оно не позволяет содержать информацию о параметрах безопасности, закрытой внутри ресурса. Если файл или каталог перемещается, или перемонтируется, или процесс имеет другой вид наименования, отличающийся от имени файла, тогда инструменты контроля доступа могли бы повести себя иначе, поскольку обращаются к пути вместо файла. При помощи параметров, основанных на метках, эта информация сохраняется, и система продолжает контролировать ресурс правильно;
- параметры безопасности очень хорошо раскрывают цель процесса. Приложение с одним и тем же бинарным кодом может работать с различными параметрами безопасности, в зависимости от того, как оно было запущено. Значения параметров (которые были выше показаны в результате работы команды `id -Z`) содержат именно то, что необходимо администратору. Вместе с этим он знает, что права есть у каждого из запущенных экземпляров, но он может также проследить, как конкретный процесс мог быть запущен и что является у него целью;
- параметры безопасности также создают обобщение самого объекта. Мы их используем для разговора о процессах и файлах, но параметры безопасности являются еще прикладными для таких небольших сущностей, как каналы межпроцессного взаимодействия или объектов баз данных. В то время как идентификация, основанная на адресе размещения в файловой системе, сможет быть полезна только тогда, когда можно этот адрес (путь) указать для защищаемого объекта.

В качестве примера рассмотрим следующие политики:

- предоставление фоновым процессам `httpd`, обеспечивающим работу с протоколом `http`, связывание с портом 80 протокола `TCP`;
- предоставление процессам, промаркированным как `httpd_t`, взаимодействия с портами протокола `TCP`, помеченными как `http_port_t`.

В первом примере мы не можем легко повторно использовать эту политику, когда процесс веб-сервера не использует исполняемый файл `httpd` (может быть, потому что он был переименован или не является `Apache`, а каким-нибудь другим веб-сервером) или когда мы хотим выполнить `http`-доступ на другой порт. В случае же метода маркировки исполняемые файлы могут носить названия вроде `apache2` или `MyWebServer.py`, в то время как процессы их могут иметь маркировку `httpd_t`, которой будет оперировать политика безопасности. Точно так же обстоит дело и с определениями портов: можно промаркировать порт 8080 с помощью типа `http_port_t` и таким образом позволить веб-серверам подключаться к этому порту.

### 1.2.1. Описание параметров безопасности

Перейдем к параметрам безопасности: в SELinux применяется по крайней мере три, а иногда и четыре значения. В качестве примера посмотрим на па-



параметры безопасности веб-сервера Apache:

```
$ ps -eZ | grep httpd
system_u:system_r:httpd_t:s0 511 ? 00:00:00 httpd
```

Как мы можем видеть, с процессом веб-сервера связан набор параметров безопасности, который содержит следующие поля:

- `system_u`: данное поле определяет *имя пользователя* (SELinux user), зарегистрированного в системе защиты;
- `system_r`: второе поле указывает название одной из разрешающих *ролей* (SELinux role), определенной в системе защиты и предоставленной субъекту/объекту доступа;
- `httpd_t`: поле представляет *тип* (SELinux type) (также поле известно как *домен* (domain) в случае, когда речь идет о процессе);
- `s0`: четвертое поле содержит сведения о *мандатной конфигурации* («sensitivity level»).

**P** Понятие «домен» в данном случае применяется в качестве «*допустимого набора функций*». Далее будет сказано, что понятие домен распространяется не только на процесс, но и в принципе на субъект доступа. Само понятие «домен», как известно, имеет широкое распространение в информационных технологиях и может относиться как к адресному пространству интернета в качестве структурированного глобального адреса узла сети (см. документ РД 45.134–2000), так и к базам данных, серверам, ресурсам и другим объединениям. Учитывая, что слово имеет французские корни, означающие «владение», корректно провести параллель и сказать, что в данном случае домен – это те функции и возможности, которыми «владеет» пользователь, процесс, файл, ресурс с точки зрения системы защиты.

В русскоязычной технической литературе слово «sensitivity» практически не попадает под прямой перевод, подразумевая мандатный принцип разграничения доступа в соответствии с ГОСТ Р 50739–95 «Средства вычислительной техники. Защита от несанкционированного доступа информации. Общие технические требования», где мандатный принцип характеризуется назначением объектам и субъектам так называемых «уровней иерархической классификации» и «иерархических категорий». Далее в этой главе будет сказано, что параметр безопасности «sensitivity level» основывается на классической модели Белла-ЛаПадула, в которой рассматриваются уровни доступа, и на разделении информации по категориям. В данном же разделе стоит уточнить, что уровнями могут быть следующие наименования: «открыто», «конфиденциально», «строго конфиденциально» и др., в зависимости от того, как их определит правообладатель. А категории информации отделяют сведения, например, бухгалтерские от сведений отдела кадров. При этом бухгалтерские и сведения отдела кадров могут быть открытыми, конфиденциальными и т. п. Тогда бухгалтеры будут иметь доступ к своим материалам, а работники отдела кадров – к своим, и смешиваться эти данные не будут. В свою очередь, параметр безопасности «sensitivity level» объединяет в себе «уровни доступа» и «категории», определяя конкретные варианты комбинаций из их возможного множества. Например, на рис. 1.2 показана следующая запись «s0-s0:c0.c1023», которая подразумевает наличие доступа (например, у какого-то пользователя) ко всем 1024 категориям информации, в которых сама информация имеет низший уровень доступа s0. Но можно дать пользователю доступ и ко всем уровням и ограничить его в категориях, оставив, допустим, толь-

ко одну, и таким образом выполнять администратору высокоточную настройку доступа к данным. И от того, как он ее настроит, зависит способность получения (восприятия) тем же пользователем полного объема данных или какой-то большой/сокращенной части целого. Вот эта способность и отражена, по задумке авторов, в термине «sensitivity level», который дословно можно перевести как «уровень восприятия». Говоря о русскоязычном соответствии, следует учесть, что наиболее часто, когда речь идет о мандатном уровне, подразумевают только уровень доступа, но не категорию информации, характерную мандатному *принципу* тоже. В связи с этим наиболее корректно, говоря далее в тексте про 4-й параметр безопасности, использовать понятие «*мандатная конфигурация*», подразумевающее сочетание «уровней доступа» и «категорий информации».

Структура набора из четырех параметров может быть изображена, как показано на рис. 1.2.

<обозначение>_u	<обозначение>_r	<обозначение>_t	s0-s0:c0.c1023
Пользователь SELinux	Роль SELinux	Тип SELinux	Мандатная конфигурация

**Рис. 1.2** ❖ Структура параметров безопасности SELinux, формируемая в результате использования команды `id -Z`, в качестве примера

Когда мы работаем с SELinux, то «параметры безопасности» – это практически все, что нам нужно. В большинстве случаев третий параметр (называемый «доменом», или [функциональным] «типом») является наиболее важным, т. к. многие правила политики системы защиты (порядка 99 %) состоят из правил, определяющих, каким образом осуществляется взаимодействие между двумя такими «типами» (без обращения внимания на «роли», «пользователей защиты» или «мандатные конфигурации»).

Параметры безопасности как и атрибуты модулей безопасности Linux (LSM) предьявляются в пользовательское пространство стандартизированным образом – совместимой с множественными реализациями модульной защитой LSM, позволяющей конечным пользователям и приложениям легко запросить конкретные параметры безопасности. Характерным местом, где эти атрибуты представлены, является псевдофайловая система `/proc`.

Внутри каждого места размещения процесса (`/proc/<pid>`) мы найдем поддиректорию, которая называется `attr`, внутри которой могут быть найдены следующие файлы:

```
$ ls /proc/$$/attr
current  fscreate  prev
exec     keycreate sockcreate
```

Все эти файлы, если их открыть в режиме чтения, отобразят либо отсутствие данных, либо параметры безопасности SELinux. Если какой-то конкретный файл не содержит никаких данных, то это означает, что рассматриваемому

приложению явно не заданы параметры безопасности для этого конкретного аспекта работы и они будут получены либо из определенной политики, либо унаследованы от родительского процесса.

Назначение у этих файлов следующее:

- файл `current` содержит текущие параметры безопасности для данного процесса;
- файл `exec` содержит параметры безопасности, которые будут назначены дочерним процессам от текущего. Обычно данный файл является пустым;
- файл `fscreate` содержит параметры безопасности, которые будут назначены некоему файлу, после внесения в него изменений данным приложением. Обычно этот файл является пустым;
- файл `keycreate` содержит параметры безопасности, которые будут назначены ключам, временно хранимым (*cached*) данным приложением в ядре. Обычно этот файл является пустым;
- файл `prev` содержит *предшествующий* текущему набор параметров безопасности для определенного процесса. Обычно это параметры родительского приложения;
- файл `sockcreate` содержит параметры безопасности, которые будут назначены следующему сокету, созданному посредством данного приложения. Обычно данный файл является пустым.

Если приложение имеет множество подзадач, тогда такая же информация будет доступна в директории каждой подзадаче по следующему пути: `/proc/<pid>/task/<taskid>/attr`.

### 1.2.2. Принудительный доступ посредством типов функциональных ограничений

Тип функциональных ограничений (третья часть параметров безопасности) процесса (называемый **доменом**) является основой тщательного контроля доступа этого процесса по отношению как к нему самому, так и к другим типам (которые могут быть применимы к процессам, файлам, сокетам, сетевым интерфейсам и многому другому). В большинстве литературы о SELinux механизм контроля доступа, основанный на метках, является настолько хорошо настраиваемым, что можно было сказать, что сам SELinux – это система обязательного **соблюдения типов** допустимого набора функций. Когда какие-то действия являются запрещенными, наиболее вероятной причиной этому является тщательный контроль доступа на уровне типа допустимого набора функций (в т. ч. отсутствие его корректной настройки).

С помощью механизма соблюдения типа SELinux является способным контролировать, что позволено делать любому приложению, основываясь на том, как оно получает управление (команду на выполнение): веб-сервер, который запущен в диалоговом режиме пользователем, будет иметь тип, отличный от

веб-сервера, запущенного через систему `init`<sup>1</sup>, даже если исполняемый файл процесса и путь размещения являются теми же самыми. Веб-сервер, запущенный системой `init`, является наиболее вероятно заслуживающим доверия (и, таким образом, веб-серверу позволено выполнять все функции, необходимые для его полноценной работы), в то время как веб-сервер, запущенный вручную, является наименее вероятно подлежащим рассмотрению в качестве образца нормального поведения, и раз так, то он будет иметь отличающиеся привилегии.

**i** Большинство средств SELinux будут сосредоточены на типах допустимого набора функций. Даже когда тип – просто третий компонент параметров безопасности, он является наиболее важной сущностью для многих администраторов. Многие документы будут даже просто рассказывать о типах, таких как `http_t`, охотнее, чем обо всех параметрах безопасности в целом.

Взгляните на следующий процесс фоновой программы обеспечения межпроцессного взаимодействия `dbus-daemon`:

```
# ps -eZ | grep dbus-daemon
system_u:system_r:system_dbusd_t 4531 ?      00:00:00 dbus-daemon
staff_u:staff_r:staff_dbusd_t    5266 ?      00:00:00 dbus-daemon
```

В данном примере первый процесс `dbus-daemon` является фоновой программой системы D-Bus, запущенной с типом, соответственно названным `system_dbusd_t`, тогда как другой процесс является запущенным с назначенным ему типом `staff_dbusd_t`. Будучи полностью одинаковыми, даже на уровне бинарного кода, они оба преследуют различные цели в системе и таким образом имеют разные назначения типов защиты. Затем SELinux использует этот тип для ограничения действий, разрешенных процессу по отношению к другим типам, включая и то, как `system_dbusd_t` взаимодействует с `staff_dbusd_t`.

Типы SELinux обусловлены в обозначении окончанием `_t`, хотя это и необязательно.

### 1.2.3. Распределение по ролям наборов функциональных ограничений

Роли SELinux (второй компонент параметров безопасности) позволяют системе защиты поддерживать контроль доступа. Несмотря на то что тип допустимого набора функций (*type enforcement*) является наиболее используемым (и наиболее известным) компонентом SELinux, контроль доступа, основанный на роли, является важнейшим методом, для того чтобы обеспечивать надежную систему защиты, в особенности от вредоносных пользовательских покушений. Роли SELinux определяют, какие разрешенные наборы функций (домены) процессов могут быть запущены вместе. Эти типы типов допустимого набора функций, назначаемые процессам (домены) в своей области параметров безопасности

<sup>1</sup> `init` – система инициализации, запускающая все остальные процессы после инициализации ядра. Является первым процессом пользовательского режима и отвечает за дальнейшую загрузку системы. – *Прим. перев.*

определяют разрешения. По существу, роли помогают определить, что именно некий пользователь (который имеет доступ к одной или более ролям) может и не может делать.

Роли SELinux обусловлены в обозначении окончанием `_r`. В большинстве систем, включающих SELinux, следующие роли исходно доступны для сопоставления с пользователями:

**Таблица 1.1. Описание предустановленных ролей**

Роли	Описание
<code>user_r</code>	Эта роль предназначена для ограниченных в действиях обычных пользователей: она позволяет использовать приложения, чьи процессы запускаются только со специфическими типами. Привилегированные типы, включающие возможности для выбора другого Linux-пользователя, являются непозволительными для этой роли
<code>staff_r</code>	Эта роль предназначена для некритичных операций: она в основном ограничивает те же приложения, что и в первой роли, но с возможностью поменять роль. Эта роль предназначена для операторов, находящихся на работе (для того чтобы предоставлять этим пользователям роли с наименьшими привилегиями до тех пор, пока это возможно)
<code>sysadm_r</code>	Эта роль предназначена для системных администраторов: она является очень привилегированной, включающей различные системные задачи по администрированию. Однако определенные типы приложений для конечного пользователя могут не быть поддержаны (особенно если эти типы являются предназначенными для потенциально уязвимого или непроверенного программного обеспечения) в целях сохранения системы, защищаемой от зловредных воздействий
<code>secadm_r</code>	Эта роль предназначена для администраторов безопасности: позволяет менять политику SELinux и выполнять действия по управлению данной системой защиты. Роль в основном используется, когда разделение обязанностей является необходимым между системным администратором и службой управления системными политиками
<code>system_r</code>	Данная роль предназначена для фоновых системных процессов: является наиболее полно привилегированной для поддержания различных типов защиты скрытых от пользователя процессов ( <i>daemon</i> ) и процессов операционной системы. При этом типы защиты приложений, предназначенные для программ конечных пользователей, а также типы, обеспечивающие администрирование, являются недоступными из этой роли
<code>unconfined_r</code>	Данная роль (в переводе «неограниченная») предназначена для конечных пользователей: она предоставляет конечное количество типов, но эти типы являются очень привилегированными, так как ориентированы на выполнение любых приложений, запущенных каким-либо пользователем в более или менее свободной манере (без ограничений правилами SELinux). Данная роль как таковая является доступной, исключительно если системный администратор хочет защитить конкретные процессы (часто выполняемые в фоновом режиме), оставляя в покое остальные системные операции почти не затронутыми защитой SELinux

В зависимости от дистрибутива другие роли могут также поддерживаться, например `guest_r` и `xguest_r`. Для того чтобы узнать больше о поддерживаемых ролях, лучше всего обратиться к документации конкретного дистрибутива. Некий обзор доступных ролей может быть предоставлен посредством команды `seinfo`, являющейся частью пакета `setools-console` в RHEL или `app-admin/setools` в Gentoo.

```
# seinfo --role
Roles: 14
  auditadm_r
  dbadm_r
  ...
  unconfined_r
```

### 1.2.4. Разделение пользователей по ролям

Пользователи, зарегистрированные в системе защиты SELinux (определяемые в первой части параметров безопасности), отличаются от штатных пользователей операционной системы Linux. В противовес тому, что информация о пользователе Linux может быть изменена даже в то время, когда пользователь работает в системе (при помощи таких инструментов, как `sudo` или `su`), политика SELinux может обеспечить (и обеспечивает), чтобы пользователь SELinux оставался таким же и тогда, когда пользователь Linux сам изменился. На основе неизменного состояния пользователя SELinux можно реализовать особый контроль доступа, дающий уверенность в том, что пользователи не смогут работать в обход набора установленных им разрешений даже в том случае, когда они имеют привилегированный доступ.

В качестве примера такого контроля доступа является **пользователь-ориентированный контроль доступа** (User-based access control – UBAC), возможности которого некоторые дистрибутивы Linux поддерживают (по желанию) и который оберегает пользователей от доступности файлов различных SELinux-пользователей, даже если те пользователи пытаются применять дискреционный механизм Linux для открытия доступа к файлам друг друга.

Наиболее важной характерной чертой пользователей SELinux, однако, является то, что они формулируют ограничения в части того, какие роли Linux-пользователи принимают, а какие нет. Какой-либо Linux-пользователь сначала назначается какому-то пользователю SELinux – множество Linux-пользователей может быть назначено этому же пользователю SELinux. После назначения этот пользователь не может выбрать какую-либо роль SELinux, т. к. он не является способным в этом участвовать.

Роль-ориентированный контроль доступа, реализованный в SELinux, представлен на рис. 1.3.

**Пользователи SELinux** по договоренности создаются с дописыванием в конце `_ч`, хотя это и не обязательно. Те пользователи SELinux, которые доступны во многих дистрибутивах, являются названными так же, как и пред-

ставляющие их роли, но вместо окончания `_г` они имеют окончание `_у`. Например, для роли `sysadm_г` существует пользователь SELinux `sysadm_у`.

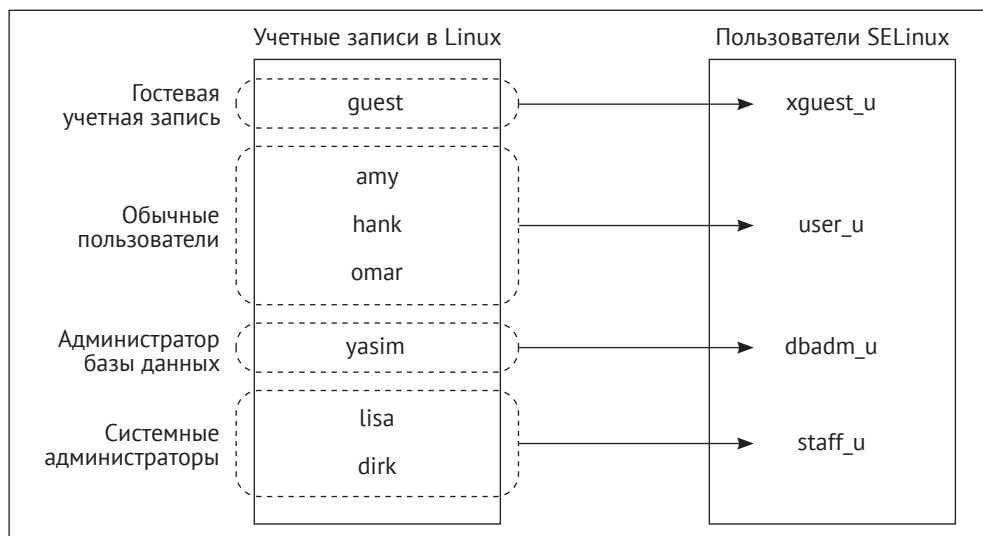


Рис. 1.3 ❖ Соотнесение учетных записей Linux с пользователями SELinux

### 1.2.5. Контроль информационных потоков посредством мандатного механизма

Четвертая часть параметров безопасности SELinux – мандатная конфигурация – не всегда присутствует (некоторые дистрибутивы Linux по умолчанию не включают мандатные метки). Если же она все-таки есть, тогда эта часть маркировки требуется для обеспечения **многоуровневой защиты MLS** (*multilevel security*), поддерживаемой в SELinux. Мандатные метки позволяют разделить защищаемые ресурсы на классы и ограничить доступ к этим ресурсам на основе соответствующих разрешений. Эти метки состоят из двух частей: значения конфиденциальности (обозначаются префиксом `s`) и значения категории (обозначаются префиксом `c`).

Во многих больших организациях и компаниях документы подразделяются на «общедоступные», «конфиденциальные» и «строго конфиденциальные». SELinux может назначить процессам определенный уровень доступа по отношению к этим ресурсам. С помощью механизма многоуровневой защиты система SELinux может быть сконфигурирована в соответствии с моделью Белла–ЛаПадула – моделью защиты, которая может быть охарактеризована как «не читать все, что выше, и не записывать во все, что ниже» (англ. *no read up and no write down*). Данная модель основана на уровне дозволенности процесса: этот процесс не может читать что-либо с более высоким уровнем конфиденци-

альности и не может писать в (или обмениваться информацией как-то иначе с) любой(ым) ресурс(ом) с более низким уровнем конфиденциальности. SELinux не использует метки с названиями вроде «общедоступно», «конфиденциально» и т. п. Вместо этого он применяет числовые значения от 0 (для самого низкого уровня конфиденциальности) до того числа, которое администратор определит как предельное значение (оно конфигурируется и устанавливается после сборки политики SELinux).

Часть метки, связанная с категориями, позволяет ресурсам быть отнесенными к одной категории или нескольким, на которые контроль доступа также распространяется. Одной функциональной возможностью из их общего числа, появляющихся от использования категорий, является поддержка разделения объектов доступа по принадлежности разным владельцам (например, в рамках системы размещения и предоставления доступа к приложениям для нескольких клиентов) в системе Linux, когда процессам и ресурсам, принадлежащим одному владельцу, назначается собственный набор категорий, в то время как процессы и ресурсы другого владельца получают отличающийся набор категорий. Когда процесс не имеет должных категорий, он не может выполнять какие-либо операции с ресурсами (или другими процессами), которые имеют другие присвоенные категории.

**i** Неписаное соглашение в мире SELinux состоит в том, что (по крайней мере) две категории применяются для формирования различий между двумя участниками (владельцами). Имея возможность случайного выбора двух категорий для некоего участника из непредопределенного набора категорий, мы убеждаемся, что каждый участник имеет уникальную комбинацию и таким образом обеспечивается необходимая изоляция. Использование двух категорий не является обязательным, но реализуется такими сервисами, как **sVirt** и **Docker**.

В этом смысле категории можно рассматривать как особый признак, позволяющий предоставить доступ, только когда он совпадает и у субъекта, и у объекта доступа. Поскольку многоуровневая защита является не часто используемой, преимущества исключительного использования категорий представляются в том, что называется **многокатегорийная защита** (Multi-category security – MCS). Это частный случай многоуровневой защиты, где присутствует только один уровень конфиденциальности (s0).

### 1.3. ФОРМИРОВАНИЕ И РАСПРЕДЕЛЕНИЕ ПОЛИТИК

При включении SELinux принудительный режим (*enforcement*) контроля доступа автоматически не запускается. Если SELinux является включенным, но он не может найти политику, тогда запуск будет отменен. Это связано с тем, что политики определяют поведение системы (которое SELinux должен позволять). Политики в основном распространяются в откомпилированном виде (подобно программам) и представляют собой некие модули политик. Эти модули затем объединяются в единое хранилище политик и загружаются в па-



мять, для того чтобы позволить SELinux претворять в жизнь правила политик на данной системе.

- И** Операционная система «Gentoo», будучи метадистрибутивной, полностью основанной на исходных текстах, с тем же успехом распространяет политики SELinux как (исходный) код, который является компилируемым и собираемым во время установки, подобно тому, как это делается с другими программами.
- Р** Под метадистрибутивом, в контексте создателей системы, понимается качественно новая сущность, появившаяся в связи с основным процессом создания Linux-систем (а именно после него), позволяющая наиболее гибко настраивать конечную систему и оптимизировать производительность.

Следующее изображение показывает взаимосвязи между **правилами политики** (*policy rules*), **модулями политики** (*policy modules*) и **пакетом политики** (*policy package*) (который часто является один к одному отражением **хранилища политики** (*policy store*)).

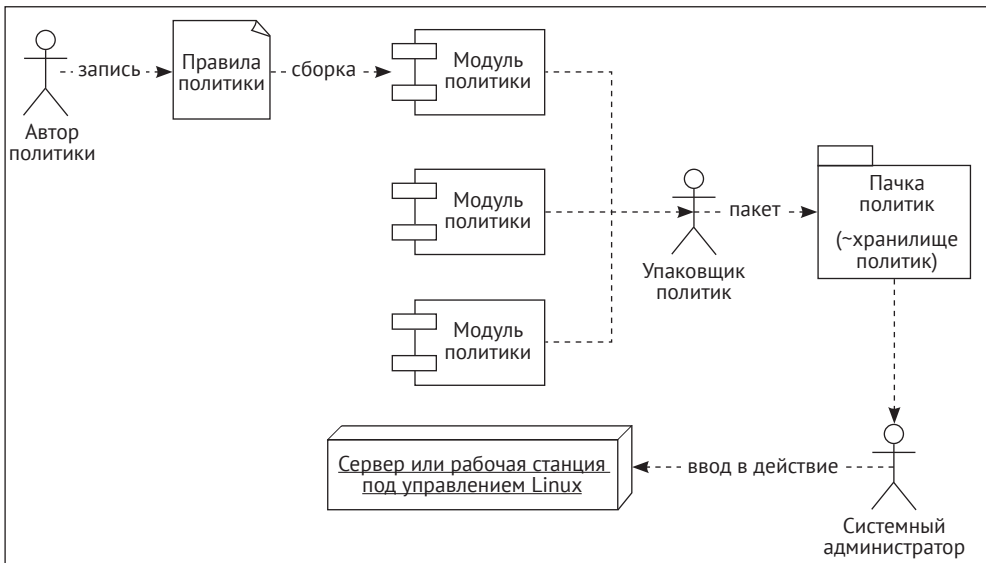


Рис. 1.4 ❖ Взаимосвязь между правилами политики, модулями политик и хранилищем политик

### 1.3.1. Создание политик SELinux

Специалист, формирующий политики, может записывать правила на трех возможных языках:

- 1) в стандартном для SELinux формате, воспринимаемом человеком («format-a human-readable») при помощи хорошо устоявшегося языка для написания политик SELinux;

- 2) в посредническом стиле политики («reference policy style») – он расширяет стандартный формат SELinux макросами «M4» для содействия разработке политик;
- 3) на обобщенно-промежуточном языке SELinux («common intermediate language» – «CIL») – формат для политик, воспринимаемый компьютером (и с большей долей напряжения человеком тоже).

Наибольшее число дистрибутивов, поддерживающих SELinux, основывается на посреднической политике (<https://github.com/TresysTechnology/refpolicy/wiki>). В этом случае полностью функциональный набор политик SELinux управляется как проект свободного программного обеспечения. Такой подход позволяет поставлять дистрибутивы с набором функциональных политик, а не писать их самим. Многие участники проекта являются разработчиками дистрибутивов, пытающимися включить изменения их дистрибутива в сам проект посреднической политики, где изменения рецензируются специалистами, чтобы уверенно избежать правил, привносящих в проект то, что может подвергнуть опасности любую платформу. Без широкого набора макросов M4, предложенного проектом посреднической политики, писать повторно используемые модули политики быстро становится очень беспокойным занятием.

Формат обобщенно-промежуточного языка является довольно новым, несмотря на это, он уже много где используется (новое пользовательское пространство SELinux преобразует все в данный формат (CIL) в фоновом режиме), но при этом он еще не является общепринятым для создателей политик, чтобы применять его безоговорочно.

В качестве примера рассмотрим правило для веб-сервера, которое мы обсуждали ранее, но повторим здесь еще раз для вашего удобства: оно позволяет процессам, промаркированным как `httpd_t`, связываться с TCP-портами, промаркированными как `httpd_port_t`.

В стандартном формате это записывается следующим образом:

```
allow httpd_t http_port_t : tcp_socket { name_bind };
```

Использование посреднического стиля записи политик приводит к тому, что данное правило становится частью следующего вызова макроса:

```
corenet_tcp_bind_http_port(httpd_t)
```

В обобщенно-промежуточном языке это правило должно быть изображено следующим образом:

```
(allow httpd_t http_port_t (tcp_socket (name_bind)))
```

В большинстве представлений мы можем видеть, что данное правило отражает следующее:

- 1) субъект (тот, кто является выполняющим действие); в данном случае это набор процессов, промаркированных как имеющие тип `httpd_t`;

- 2) целевой ресурс или объект (целевой для воздействия); в данном случае он является набором TCP-сокетов (`tcp_socket`), имеющих маркер `httpd_port_t`. В посредническом стиле записи политики данный объект реализован в имени функции;
- 3) конкретное действие или разрешения; в данном случае оно является действием по связыванию с портом (`name_bind`). В посредническом стиле записи политики данный объект реализован в имени функции;
- 4) результат, который данная политика будет претворять в жизнь; в данном случае это то, что действие разрешено (`allow`). В посредническом стиле записи политики данный объект реализован в имени функции.

Любая политика в основном пишется для какого-либо приложения, или набора приложений. Так, предыдущий пример будет частью политики, написанной для веб-сервера.

Создатели политик обычно формируют три файла на приложение или на набор приложений:

- 1) файл с расширением `.te`, который содержит тип претворяемых в жизнь правил;
- 2) файл с расширением `.if`, который содержит интерфейс и шаблон определений, позволяющих авторам политик проще использовать вновь сгенерированные правила политик для улучшения других политик совместным образом. Вы можете сравнить их с заголовочными файлами в других языках программирования;
- 3) файл с расширением `.fc`, который содержит выражения, определяющие параметры безопасности файла. Эти выражения являются правилами, которые назначают маркировку для ресурсов, расположенных в файловой системе.

Завершенная политика упаковывается в модуль политики SELinux.

### 1.3.2. Распределение политик в виде модулей

Исходно SELinux использовал единственный монолитный метод работы с политиками: все правила, контролирующие возможный доступ, хранились в одном файле политик. Вскоре стало ясно, что он не поддается управлению при долгосрочном использовании, и в результате родилась идея разработки модульного подхода к политикам.

С модульным подходом разработчики политик могут писать наборы обособленных политик для отдельных приложений (или групп приложений), ролей и т. п. Эти политики затем создаются и распространяются как модули политик. Платформы, которые нуждаются в контроле доступа для отдельных приложений, загружают модуль политики SELinux, который определяет правила доступа для этого приложения.

Процесс построения модулей политик показан на рис. 1.5. На нем также показывается, где обобщенно-промежуточный язык (CIL) вступает в игру, даже когда правила политики сами не написаны на нем. Для дистрибутивов, кото-

рые еще не поддерживают обобщенно-промежуточный язык, модуль политики (`semodule`) будет напрямую перемещен из состояния `.pp` файла к файлу хранилища политик «`policy.##`».

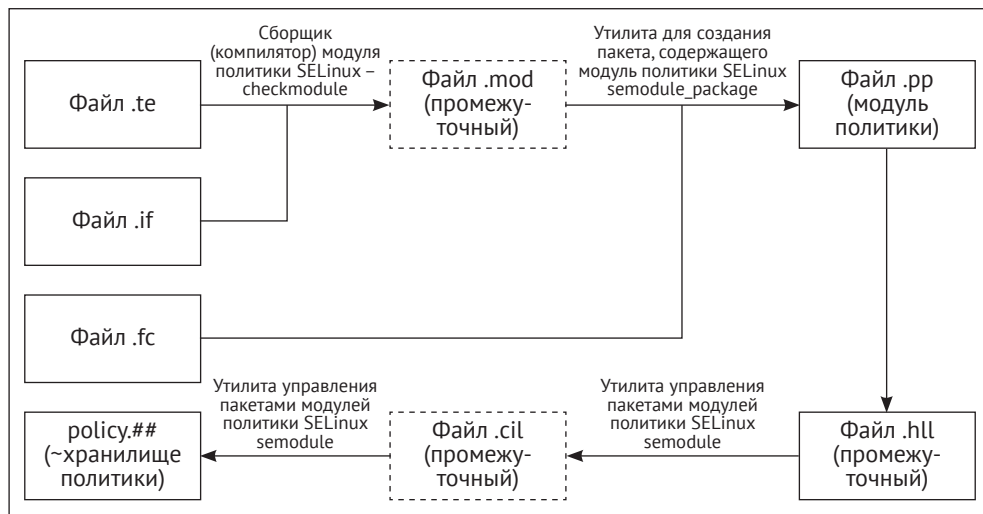


Рис. 1.5 ❖ Процесс преобразования: от правила политики в хранилище готовых политик

Вместе с новыми пользовательскими программами системы SELinux файлы `*.pp` (которые являются модулями политики SELinux) продуманы для того, чтобы быть записанными на языке высокого уровня (*high-level language* – HLL). Не следует ожидать, что это означает большую читаемость их человеком: данные файлы являются двоичными. В подобном случае подразумевается, что разработчики SELinux хотят поддерживать создаваемые политики SELinux в разных форматах, среди которых этот относят к высокоуровневым языкам, раз уж включают в состав средство грамматического разбора (*parser*), которое может конвертировать эти файлы в обобщенно-промежуточный формат (CIL). Маркировка форматов двоичного модуля как высокоуровневых позволяет в проект SELinux вводить различие между языками высокого уровня (HLL) и обобщенно-промежуточным форматом (CIL) в манере обратной совместимости.

При распространении политик многие дистрибутивы Linux помещают модули `*.pp` внутри каталога `/usr/share/selinux`, обычно в подкаталог, называемый позже хранилищем политик (такой как `targeted`). Здесь эти модули являются готовыми для активации администраторами.

Когда активируется модуль, команда `semodule` (часть пакета `policycoreutils`) будет копировать его в предназначенную для этого директорию:

- `/etc/selinux/targeted/modules/active/modules` (для Red Hat Enterprise Linux)
- или
- `/var/lib/selinux/mcs/active/modules` (для Gentoo).

Данное размещение определяется версией пользовательского пространства SELinux – более ранние версии используют каталог `/var/lib`. Когда все модули, наконец, сгруппированы в одном месте, собирается окончательная бинарная политика, размещается в файле `/etc/selinux/targeted/policy/policy.30` (последняя цифра может быть иной) и загружается в память.

В операционной системе Red Hat Enterprise Linux политики предоставляются пакетом `selinux-policy-targeted` (или `-minimum`, или `-mls`). В Gentoo – различными пакетами `sec-policy/selinux-*` (Gentoo использует отдельные пакеты для каждого модуля, сокращая число политик, загруженных на некоторую усредненную систему).

### 1.3.3. Комплектация модулей в хранилище политик

**Хранилище политик** (policy store) содержит единственную, но всеобъемлющую политику, и только одна политика может быть активна в системе в любой момент времени. Администраторы могут выбирать хранилища политик, хотя это часто требует от системы перезагрузки и может даже обязывать выполнять перемаркировку записей в системе (перемаркировка – это действие по перенастройке параметров безопасности на всех файлах и ресурсах, используемых в данной системе).

Активная политика, применяемая в системе, может быть определена при помощи команды `sestatus` (статус SELinux, предоставляемый средствами пакета `policycoreutils`) следующим образом:

```
# sestatus | grep "Loaded policy name"
Loaded policy name:                targeted
```

В этом примере текущая загруженная политика (хранилище) называется `targeted`. Имя политики, которое SELinux будет использовать после следующей перезагрузки, определяется в конфигурационном файле `/etc/selinux/config` параметром `SELINUXTYPE`.

Это система инициализации на уровне ОС (будь это **SysV-совместимая** или `systemd`), которая является в основном ответственной за загрузку политики SELinux, эффективно активизирующей поддержку функций SELinux в системе. Система инициализации считывает конфигурацию, размещенную в хранилище политики, и загружает файл политики в память. Если система инициализации не поддерживает такие возможности (другими словами, не SELinux-ориентированная), тогда политика может быть загружена при помощи команды `load_policy`, являющейся частью пакета `policycoreutils`.

## 1.4. Различия между политиками

Наиболее общие хранилища политик носят названия `strict`, `target`, `mcs` и `mls`. Ни одно из этих имен, связанных с хранилищами политик, не является жестко закрепленным, несмотря на то что это и считается предметом соглашения. Поэтому их рекомендуется уточнять в документации к дистрибутиву, для того

чтобы проверить, какое предпочитаемое имя политики должно быть. Тем не менее имя часто предоставляет некоторую информацию о параметрах SELinux, которые включены в политике.

### 1.4.1. Поддержка многоуровневой защиты (MLS)

Одна из опций, которая может быть включена, – это поддержка многоуровневой защиты (MLS). Если она отключена, тогда параметры безопасности SELinux не будут содержать четвертый компонент с мандатной конфигурацией, определяя параметры безопасности для процессов и файлов, как показано ниже:

```
staff_u:sysadm_r:sysadm_t
```

Для того чтобы проверить, выключена ли поддержка многоуровневой защиты, достаточно посмотреть только на отсутствие четвертого компонента среди параметров безопасности, но это также можно узнать из строки «Policy MLS status»<sup>1</sup>, выводимой командой `sestatus`:

```
# sestatus | grep MLS
Policy MLS Status:      disabled
```

Третий способ – это посмотреть в псевдофайле `/sys/fs/selinux/mls`. Значение `0` говорит, что многоуровневая защита выключена, значение `1` – что включена:

```
# cat /sys/fs/selinux/mls
0
```

Многоуровневая защита включена в основном в такие хранилища политик, как `targeted`, `mcs` и `mls`, тогда как в `strict` обычно выключена.

### 1.4.2. Манера поведения с неизвестными разрешениями

Разрешения (такие как «чтение», «открытие» и «блокирование») являются определенными одновременно в ядре Linux и в самой политике. Однако иногда более новые ядра поддерживают разрешения, которые текущая политика пока что еще не понимает.

Возьмем разрешение `block_suspend` (быть способным блокировать системные зависания) в качестве примера. Если ядро Linux поддерживает (и проверяет) данный доступ, а загруженная политика SELinux не понимает пока эти разрешения, тогда SELinux должен решить, как нужно иметь дело с ними.

SELinux может быть сконфигурирован так, чтобы выполнять при этом одно из следующих действий:

- 1) `allow`: принимать все, что является неизвестным, как дозволенное;
- 2) `deny`: принять, что нет никаких разрешений для выполнения данных действий;
- 3) `reject`: остановить систему и дать команду на прекращение выполнений всех функций процессора.

---

<sup>1</sup> В переводе с англ. «Статус политики многоуровневой защиты».

Данная конфигурация настраивается через значение `deny_unknown`. Для того чтобы определить заданное состояние для неизвестных разрешений, надо обратиться к строчке `sestatus` для `Policy deny_unknown status`:

```
# sestatus | grep deny_unknown
Policy deny_unknown status: denied
```

**Р** В данном примере автор выполняет команду просмотра состояния SELinux `sestatus` и указывает через вертикальную черту, что необходимо выполнить поиск и выборку всех строк, содержащих фразу `deny_unknown`, при помощи утилиты `grep`. Результатом этой команды является вторая строчка, показывающая, что состояние политики запрет\_неизвестного (`Policy deny_unknown status`) находится в статусе отказа (`denied`) для выполнения любых действий, разрешения которых неизвестны в SELinux.

Администраторы могут устанавливать данный параметр для себя в файле `/etc/selinux/semanage.conf` через переменную `handle-unknown`, указав ей одно из трех значений: `allow`, `deny` или `reject`.

Операционная система «Red Hat Enterprise Linux» по умолчанию принимает все неизвестные разрешения как дозволенные, тогда как операционная система «Gentoo» по умолчанию запрещает их.

**Р** Название политики `block_suspend` можно в русскоязычном варианте представить как воспрепятствование\_зависаниям. Названия конфигураций в переводе на русский выглядят в данном контексте следующим образом:

- `allow` – разрешать,
- `deny` – отказывать;
- `reject` – подавлять.

Конфигурационный параметр `deny_unknown` – запрет\_неизвестного. Здесь и далее составное название `sestatus` имеет смысл рассматривать как сокращение от `SELinux status` – состояние SELinux. `Policy deny_unknown status` – состояние политики запрет\_неизвестного. Переменная `handle-unknown`, определяемая в файле контроля конфигурации и действий утилит `semanage` (управление политикой SELinux) и `semodule` (управление пакетами модулей политики SELinux), может переводиться как определитель неизвестного.

### 1.4.3. Поддержка неограниченных доменов

Политика SELinux может быть очень строгой, ограничивающей приложения как можно ближе к их необходимой функциональности, но также может быть и весьма свободной, в которой приложениям разрешено работать. Одна из концепций, доступных во многих политиках, – это идея неограниченных доменов (`unconfined domains`). Когда она включена, то означает, что некоторые SELinux-домены (параметры безопасности, свойственные процессу) позволяют делать едва ли не все, что захочется (конечно, в пределах разрешений прав дискреционного разграничения доступа, который по-прежнему сохраняется), и только избранное число доменов являются в полной мере ограниченными в своих действиях. Неограниченные домены были привнесены для того, чтобы

позволить защите SELinux быть активной на рабочих станциях и серверах, где администраторы не хотят полностью ограничивать всю систему, а только некоторые приложения, запущенные на ней. В основном такие реализации имеют прикладное значение для ограничения сервисов, имеющих сетевой интерфейс (такие как веб-сервер и система управления базами данных), позволяя конечным пользователям и администраторам неограниченно перемещаться повсюду.

В случаях, когда речь идет о других защитах мандатного разграничения доступа, таких как AppArmor, *неограниченность* (unconfinement) является, по существу, частью дизайна системы, поскольку системы защиты контролируют действия только для определенных приложений или пользователей. Однако SELinux был спроектирован как система полного контроля мандатного разграничения доступа, и таким образом необходимо предоставить правила разграничения доступа даже для тех приложений, для которых не нужно никаких правил. Помечая эти приложения как неограничиваемые, мы почти не имеем дополнительных ограничений, налагаемых средствами SELinux.

Мы можем видеть, являются ли включенными неограниченные домены или нет, в системе с помощью команды `seinfo`, которую используем для запроса поддержкой текущей политикой типа защиты с названием `unconfined_t`. В системах, где неограниченные домены поддерживаются, этот тип будет доступен:

```
# seinfo -tunconfined_t
unconfined_t
```

Для систем, где не поддерживаются, этот тип не будет частью политики:

```
# seinfo -tunconfined_t
ERROR: could not find datum for type unconfined_t
```

Многие дистрибутивы, которые включают неограниченные домены, называют свои политики `targeted`, но это только соглашение, которое не всегда выполняется. Поэтому всегда лучше это уточнить, используя команду `seinfo`. RHEL включает неограниченные домены, тогда как в Gentoo это конфигурируемая настройка через флаг `unconfined USE`.

#### 1.4.4. Ограничение межпользовательского обмена

Когда включен пользователь-ориентированный контроль доступа (UBAC), некоторые типы SELinux будут защищены дополнительными ограничениями. Это будет гарантировать, что один пользователь SELinux не получит доступ к файлу (или другому специфическому ресурсу) другого пользователя, даже когда эти пользователи предоставят доступ к своим данным через стандартные средства разрешения Linux. Пользователь-ориентированный контроль доступа предоставляет некоторый дополнительный контроль доступа над информационными потоками между ресурсами, но он весьма далек от совершенства. В сущности, он позволяет изолировать пользователей SELinux одного от другого.



**i** Ограничение в SELinux представлено в виде некоего правила ограничения доступа, которое использует все части параметра безопасности, для того чтобы принять решение. В отличие от правил принудительного исполнения, которые являются основанными исключительно на типах, ограничения могут учитывать пользователя SELinux, роль SELinux или мандатную конфигурацию. Ограничения обычно разрабатываются один раз и остаются неизменными, иначе многие авторы политик не будут касаться ограничений в своих разработках.

Многие дистрибутивы Linux, включая Red Hat Enterprise Linux, исключают пользователь-ориентированный контроль доступа. Gentoo позволяет пользователям выбрать, хотят ли они его использовать или нет, с помощью флага `use USE` (который является включенным по умолчанию).

### 1.4.5. Последовательные изменения версий политик

Изучая выданные сведения командой `sestatus`, мы видим, что имеется следующая запись о версии политики:

```
# sestatus | grep version
Max kernel policy version:      28
```

Приведенная здесь версия не имеет ничего общего с обозначением версии правил политики, но при этом определяет функции SELinux, которые поддерживаются в настоящее время запущенным ядром. В приведенном выводе число 28 определяет наиболее позднюю версию политики, которая поддерживается ядром. Постоянно новые возможности добавляются в SELinux, и номер версии увеличивается. Сам файл политики (содержащий все правила SELinux, загруженные в момент запуска системы) может быть найден в файле `/etc/selinux/targeted/policy` (где `targeted` относится к используемому хранилищу политики, поэтому если система применяет хранилище с именем `strict`, то путь будет `/etc/selinux/strict/policy`).

Если существует множество файлов с политиками, тогда мы можем использовать команду `seinfo`, для того чтобы найти в выведенной ею информации сведения о том, какой файл политики используется:

```
# seinfo
Statistics for policy file: /etc/selinux/targeted/policy/policy.30
Policy Version & Type: v.30 (binary, mls)
...
```

В табл. 1.2 предоставлен текущий список последовательных изменений политик в плане реализованных в них возможностей, соотнесенных с версией ядра Linux, в которых эти политики были реализованы. Многие функции представляют интерес только для разработчиков политик, но приведенная здесь эволюция функциональных возможностей дает нам хорошее представление о развитии самого SELinux.

Таблица 1.2. История развития возможностей SELinux

Версия	Ядро Linux	Описание
12		Старый API для SELinux, сейчас устарела
15	2.6.0	Введены новые API для SELinux
16	2.6.5	Добавлена поддержка условных расширений политики
17	2.6.6	Добавлена поддержка протокола IPv6
18	2.6.8	Добавлена поддержка детализированных разрешений для сокетов взаимодействия приложений пользователя с процессами ядра (netlink-сокеты)
19	2.6.12	Добавлена поддержка многоуровневой защиты (MLS)
20	2.6.14	Сокращен размер таблицы векторов доступа
21	2.6.19	Добавлена поддержка для диапазона переходов многоуровневой защиты
22	2.6.25	Введены возможности политики (policy capabilities)
23	2.6.26	Добавлена возможность для каждого домена рекомендательного режима (permissive mode)
24	2.6.28	Добавлена поддержка четкой иерархии (границы типов)
25	2.6.39	Добавлена поддержка переходов, основанных на именах файлов
26	3.0	Добавлена поддержка для ролевых переходов у классов, не относящихся к процессам. Добавлена поддержка атрибутов ролей
27	3.5	Добавлена поддержка гибкого наследования пользователей и ролей для вновь созданных объектов
28	3.5	Добавлена поддержка гибкого наследования типов для вновь созданных объектов
29	3.14	Добавлена поддержка атрибутов в ограничениях (constraints) SELinux
30	4.3	Добавлена поддержка для расширенных разрешений и реализация их в первую очередь на устройствах контроля ввода/вывода (IOCTL). Улучшение поддержки XEN

По умолчанию, когда собирается политика SELinux, используется новейшая версия, поддерживаемая ядром и библиотекой `linsepol` – ответственной за сборку бинарного вида политики. Администратор может сам изменить версию на более низкую, используя параметр `policy-version` в файле `/etc/selinux/semanage.conf`.

### 1.4.6. Качественное изменение версий политик

По сравнению с возможностями политик, описанных выше, основным различием между политиками (и дистрибутивами) является собственно их содержание. Мы уже описывали, что многие дистрибутивы основывают свою политику на проекте посреднической политики (reference policy). Но хотя этот

проект рассматривается как главный (master) во многих дистрибутивах, при этом каждый дистрибутив имеет свои собственные отклонения от основного набора политики.

Многие дистрибутивы создают широкие дополнения к политике без прямого интегрирования их в исходную посредническую политику. Существует несколько возможных причин, почему это так не делается:

- 1) конкретная политика, модернизированная или расширенная, является еще достаточно незрелой: Red Hat изначально работает с активными политиками, но в рекомендательном режиме (permissive), преднамеренно политики не являются принудительными (enforced). Взамен SELinux регистрирует то, что должно быть предотвращено, и, основываясь на этих журналах, политики затем улучшаются. Это означает, что политики являются готовыми для фактического применения только после нескольких выпусков;
- 2) конкретная политика, модернизированная или расширенная, является также специфичной для конкретного дистрибутива: если набор политик не носит характера многократного использования в других дистрибутивах, то некоторые поставщики предпочтут оставить эти политики при себе, т. к. действия по размещению изменений в отслеживаемом (upstream) проекте требуют довольно много усилий;
- 3) конкретная политика, модернизированная или расширенная, не следовала следующим правилам и рекомендациям для внесения изменений в репозиторий проекта: посредническая политика имеет набор рекомендаций, которым должны следовать политики. Если набор правил политики не удовлетворяет этим требованиям, тогда он не будет принят;
- 4) конкретная политика, модернизированная или расширенная, не реализовала прежнюю модель защиты, как того требует проект посреднической политики: раз SELinux является весьма обширной системой принудительного контроля доступа, то вполне возможно написать полностью отличные от нее политики;
- 5) распространитель не имеет времени или ресурсов для внесения изменений в отслеживаемую ветку проекта в репозитории.

Все это означает, что политики SELinux между дистрибутивами (и даже конкретными версиями одного и того же дистрибутива) могут, являясь корректными по содержанию, быть совершенно различными. Например, Gentoo стремится внимательно отслеживать проекты посреднических политик и вносить изменения, соединяя содержание, в течение нескольких недель.

## 1.5. ЗАКЛЮЧЕНИЕ

В этой главе мы увидели, что SELinux представляет собой некий дополнительный высокоточный механизм контроля доступа верхнего уровня в иерархии систем контроля доступа операционной системы Linux. SELinux реализуется

через модульную систему безопасности Linux (LSM) и использует метки для идентификации защищаемых ресурсов и процессов, основанных на принадлежности (пользователе), роли, типе защиты и даже на мандатном уровне и категории ресурса. Было рассмотрено, как SELinux-политики обрабатываются в системах, поддерживающих SELinux, и мы кратко коснулись того, как авторы политик их структурируют.

Дистрибутивы Linux реализуют политики SELinux, которые могут иметь минимальные различия от других, основанных на поддерживаемых функциях, таких как мандатные метки, поведение при появлении неизвестных разрешений, поддержка ограничивающих уровней или специфические ограничения, приложенные конкретно, такие как пользователь-ориентированный контроль доступа (UBAC). Однако большинство правил политики сами по себе являются подобными и даже основаны на одном и том же проекте посреднической политики.

Переключение между режимами работы SELinux и рассмотрение журналов событий, которые SELinux создает, когда запрещает определенный доступ, являются предметом рассмотрения следующей главы. В ней мы также разберем, как подступить к часто встречающемуся требованию отключения SELinux и почему это решение неудачное для реализации.

# Глава 2

---

## Режимы работы и регистрация событий

Однажды включенная в операционной системе защита SELinux начинает выполнять контроль доступа так, как описано в предыдущей главе. Однако это может иметь некоторые неожиданные эффекты, поэтому в этой главе мы будем:

- 1) выполнять переключение между SELinux, работающим в полнопринудительном режиме (*full-enforcement mode*) (схожим с локальной системой предотвращения вторжений) и работающим в отладочном, рекомендательном режиме (*permissive mode*), который только регистрирует события, идейно близкие к системе обнаружения вторжений;
- 2) использовать различные способы переключений состояний системы (включенный и выключенный, принудительный и рекомендательный режимы);
- 3) отключать принудительный режим контроля для отдельно взятого набора функций (домена), а не для системы в целом;
- 4) изучать для понимания те зарегистрированные в журнале SELinux события, которые описывают, какие действия были предотвращены.

Мы закончим обзором общих методов анализа этих регистрируемых событий в действиях, выполняемых изо дня в день.

### 2.1. ВКЛЮЧЕНИЕ И ВЫКЛЮЧЕНИЕ ЗАЩИТЫ SELINUX

Вероятно, странно начинать с этого раздела, но отключение SELinux – это обычно часто требуемое действие. Некоторые компании не поддерживают работу своих приложений, запущенных на платформе, где работает система защиты SELinux. Системные администраторы в основном не желают использовать средства контроля безопасности, которые они не понимают или находят их слишком сложными для поддержания в рабочем состоянии. К счастью, число таких администраторов все меньше, а SELinux, в свою очередь, способен к выборочному отключению контроля доступа для части системы, вместо того чтобы требовать от нас его полного отключения.

### 2.1.1. Установка глобального состояния защиты

SELinux поддерживает три главных состояния, которые могут быть использованы: **отключенное** (*disabled*), **рекомендательное** (*permissive*) и **принудительное** (*enforcing*). Эти состояния устанавливаются в конфигурационном файле `/etc/SELinux/config`, в переменной состояния SELINUX. Чтобы взглянуть на текущую настройку, достаточно выполнить следующую команду и получить результат ее выполнения:

```
$ grep ^SELINUX= /etc/SELinux/config
SELINUX=enforcing
```

Когда системный процесс `init` загружает политику SELinux, код SELinux проверяет состояние, которое сконфигурировал администратор. Состояния описаны следующим образом:

- 1) если значение переменной состояния `disabled`, тогда код SELinux отключает дальнейшее выполнение, загружая систему дальше без активной защиты;
- 2) если `permissive`, тогда защита активна, но при этом не будет принудительно соблюдаться политика безопасности в системе. Вместо этого любое нарушение политики будет зафиксировано в отчете, но будет разрешено. Это состояние иногда называют **локальным обнаружением вторжений** (*host intrusion detection*), поскольку защита работает только в режиме регистрации событий;
- 3) если значение переменной состояния `enforcing`, тогда защита активна и будет полностью соблюдать политику безопасности в системе. Нарушения регистрируются, и несанкционированные действия запрещаются. Это состояние иногда называют локальной системой предотвращения вторжений (*host intrusion prevention*), поскольку соблюдаются правила безопасности вместе с регистрацией нарушающих действий.

Мы можем использовать команду `getenforce` (принадлежащую пакету `libSELinux-utils` дистрибутива Red Hat Enterprise Linux или `sys-libs/libSELinux` дистрибутива Gentoo) или команду `sestatus` для получения информации о текущем состоянии SELinux, как показано ниже:

```
# sestatus | grep mode
Current mode: enforcing
# getenforce
Enforcing
```

Также можно запросить псевдофайл `/sys/fs/SELinux/enforce` для получения похожей информации. Если запрос к файлу возвращает `1`, тогда SELinux в состоянии `enforcing`. Если он возвращает `0`, тогда он в состоянии `permissive`.

```
# cat /sys/fs/SELinux/enforce
1
```

Если изменился файл конфигурации `/etc/SELinux/config`, тогда систему необходимо перезагрузить для вступления в силу этих изменений. Однако если

система была загружена без поддержки SELinux (в состоянии `disabled`), одного включения поддержки SELinux будет недостаточно: администратору будет необходимо сделать так, чтобы все файлы в системе были заново промаркированы (т. е. необходимо для всех файлов установить параметры безопасности). Операционная система Linux, работающая без включенной защиты SELinux, будет создавать и обновлять файлы без установки новых или обновления ранее созданных меток безопасности, связываемых с этими файлами. Переформирование меток безопасности для файловой системы описано в главе 4 «Домены как допустимые наборы функций для процессов и контроль доступа на уровне файлов».

Во многих ситуациях администраторы часто хотят выключить SELinux, когда он начинает препятствовать определенным задачам. Это весьма легкомысленно, мягко говоря, и вот почему:

- SELinux – это компонент защиты, являющийся частью операционной системы. Его отключение подобно полному прекращению работы межсетевого экрана, потому что он блокирует некое взаимодействие. Отключение может помочь, потому что это наиболее быстрый путь восстановления нормальной работы того, что было заблокировано, но при этом вы устраняете и меры, которые были направлены на защиту вас;
- так же, как в этом вопросе обстоят дела и с межсетевым экраном, SELinux конфигурируется при помощи правил. Если какое-то приложение не работает корректно, то нам необходимо изменить правила для этого приложения, подобно тому как дополнительные правила межсетевого экрана могут быть включены для разрешения конкретных потоков;
- на худой конец, когда мы хотим позволить каждому действию какого-либо приложения выполняться бесконтрольно, мы можем все же оставить включенным SELinux и работать только с данным приложением без включенного контроля доступа.

Распространители вкладывают много усилий в интеграцию SELinux с их продуктом, но они имеют прекрасные каналы поддержки для оказания вам помощи, если все рухнет.

### 2.1.2. Переключение в рекомендательный и принудительный режимы

Большинство распространителей, предоставляющих ядро операционной системы, позволяют переключение между принудительным и рекомендательным режимами при помощи простой команды администрирования. Эта возможность называется **режим разработки SELinux** (`SELinux development mode`) и устанавливается через конфигурационный параметр ядра `CONFIG_SECURITY_SELINUX_DEVELOP`. Несмотря на то что это может рассматриваться как риск (все, что нужно будет сделать злоумышленнику, – это переключить SELinux на рекомендательный (отладочный) режим состояние для блокировки контроля доступа),

переключение состояния требует строгих административных привилегий, которых нет у большинства доменов приложений.

Команда для переключения между рекомендательным и принудительным режимами называется `setenforce` (являющаяся частью пакета `libselinux-utils` в RHEL или `sys-libs/libselinux` в Gentoo). Она получает единственный аргумент 0 (`permissive`) или 1 (`enforcing`). Строчные наименования аргументов `permissive` и `enforcing` тоже, как правило, разрешены.

Изменения приобретают силу сразу же. Например, следующая команда используется для включения в разрешающий режим:

```
# setenforce 0
```

Эффект от команды `setenforce` такой же, как от записи целочисленного значения в псевдофайл `/sys/fs/SELinux/enforce`:

```
# echo 0 > /sys/fs/SELinux/enforce
```

Возможность переключения между разрешающим и принуждающим режимами может быть в интересах разработчиков политик или системных администраторов, которые модифицируют систему для использования SELinux должным образом. Переключение также может быть использовано, для того чтобы быстро проверить, действительно ли предупреждения и ошибки какого-либо приложения возникают по причине работающего контроля доступа SELinux или же защита не имеет отношения к приложению, с которым они связаны. Об этом мы поговорим позже в данной главе.

В производственных системах, вероятно, было бы интересно блокировать возможность включения в разрешающий режим. Блокировка этой возможности обычно требует пересобрать ядро Linux, но разработчики политик SELinux также продумали и другой путь лишения возможности пользователей переключать состояния SELinux. Привилегии, которые пользователям необходимы для переключения в рекомендательный режим, являются *условными*, и системные администраторы могут легко отключить возможность обратного переключения из принудительного режима в рекомендательный. Данное условие выполняется посредством переменной булева типа с названием `secure_mode_policyload`, чье значение по умолчанию `off` (что разрешает переключение состояния SELinux).

**Логические параметры SELinux** (SELinux booleans) – это конфигурируемые параметры, которые принимают единственное значение (`on` или `off`, хотя `true/false` и `1/0` являются обычно допустимыми значениями) и управляют частями активной политики SELinux. Эти значения могут быть постоянными (имеется в виду, что они выдерживают перезагрузки системы) или актуальными только в течение работы одной сессии. Для того чтобы сохранить значение во время перезагрузки, добавьте `-P` к команде `setsebool` (части пакета `policycoreutils`), используемой для ее переключения:

```
# setsebool -P secure_mode_policyload on
```



Для того чтобы получить обзор всех доступных логических параметров SELinux вместе с небольшим описанием того, что они контролируют, используйте команду `semanage boolean`:

```
# semanage boolean -l
SELinux boolean   State Default Description
ftp_home_dir      (off, off)       Determine whether ftpd can read
                  and write files in user home
                  directories
xdm_sysadm_login  (off, off)       Allow the graphical login program
                  to login directly as
                  sysadm_r:sysadm_t
xen_use_nfs       (off, off)       Allow xen to manage nfs files
...
```

**P** Представленный вывод команды имеет следующую русскоязычную интерпретацию:

Булево значение	Состояние	По умолчанию	Описание
<code>ftp_home_dir</code>	(выключено,	выключено)	Определяет, может ли сервис <code>ftpd</code> читать из домашнего каталога пользователя и писать в него
<code>xdm_sysadm_login</code>	(выключено,	выключено)	Разрешает графическую программу входа в систему напрямую в качестве системного администратора
<code>xen_use_nfs</code>	(выключено,	выключено)	Позволяет гипервизору Xen управлять файлами системы NFS
...			

Команда `semanage` является частью пакета `polycoreutils-python` в Red Hat Enterprise Linux или пакета `sys-apps/polycoreutils` в Gentoo. Булевы значения SELinux описаны более подробно в главе 8 «Работа с политиками SELinux».

Использование параметра `secure_mode_policyload` позволяет администраторам ограничить переключение с принудительного режима обратно к рекомендательному. Однако переключение в состояние `disabled` (или из него) не поддерживается: если SELinux активен (либо `permissive`, либо `enforcing`) и его политика загружена, тогда только перезагрузка может эффективно отключить SELinux снова.

### 2.1.3. Использование параметров загрузки ядра

Использование команды `setenforce` имеет смысл, когда мы хотим изменить режим на рекомендательный или принудительный в тот момент времени, когда имеем интерактивный доступ в систему. Но что, если нам нужно это при загрузке операционной системы? Если система отказывается загружаться должным образом из-за подсистемы контроля доступа SELinux, и мы не можем отредактировать файл `/etc/SELinux/config`? К счастью, мы можем так же хорошо изменить состояние SELinux при помощи других средств.

Это решение заключается в том, чтобы использовать параметры загрузки ядра. Мы можем загружать ОС Linux с одним или двумя следующими параметрами, которые дают приоритет над настройками в файле `/etc/SELinux/config`:

- 1) `SELinux=0`: данный параметр информирует систему о необходимости полностью отключить защиту SELinux и имеет такой же эффект, как настройка `SELINUX=disabled` в файле `config`. Когда этот параметр установлен, другие параметры (`enforcing`) не принимаются во внимание. Пожалуйста, запомните, если система загружена с отключенными средствами SELinux, то, для того чтобы их включить снова, файловая система должна быть полностью перемаркирована метками безопасности;
- 2) `enforcing=0`: данный параметр информирует систему о том, что SELinux работает в рекомендательном режиме `permissive` и имеет такой же эффект, как настройка `SELINUX=permissive` в файле конфигурации `config`;
- 3) `enforcing=1`: данный параметр информирует систему о том, что SELinux работает в принудительном режиме `enforcing` и имеет такой же эффект, как настройка `SELINUX=enforcing` в файле конфигурации `config`.

Рассмотрим систему Linux, которая использует загрузчик GRUB2. Мы хотим добавить строку `enforcing=0` в запись загрузки. Это может быть выполнено во время загрузки следующим образом:

- 1) перезагрузите систему до появления экрана загрузки GRUB2;
- 2) выберите при помощи клавиш со стрелками запись загрузки, для которой должно быть изменено состояние SELinux. Как правило, это запись загрузки, используемая по умолчанию, и она уже бывает выделена подсветкой;
- 3) нажмите клавишу `E` для редактирования строки загрузки. Сделайте это до того, как таймер GRUB2 достигнет «0»; иначе система будет продолжать загрузку;
- 4) пользуйтесь клавишами со стрелками, чтобы установить курсор в конце строки, которая начинается со слов `linux`, `linux16` или `linuxefi`;
- 5) добавьте в конец строки фразу `enforcing=0`;
- 6) нажмите сочетание клавиш **Ctrl+X** или **F10** для загрузки данной записи.

Другие загрузчики имеют похожие методы для изменения строки загрузки без их сохранения для каждой перезагрузки. Обратитесь к документации на ваш дистрибутив для получения более подробной информации об этом.

Поддержка параметра загрузки `SELinux=` включается через параметр конфигурации ядра `CONFIG_SECURITY_SELINUX_BOOTPARAM`. Параметр загрузки `enforcing=` поддерживается через параметр конфигурации `CONFIG_SECURITY_SELINUX_DEVELOP`, который мы уже встречали.

Используя SELinux в производственных системах, может быть, было бы мудро либо отключить опции, либо должным образом защитить меню загрузки, например путем установки пароля на доступ к нему и регулярной проверки целостности файлов меню загрузки.

#### 2.1.4. Отключение защиты SELinux для отдельно взятого сервиса

Начиная с 23-й версии политики (которая появилась вместе с Linux 2.6.26) SELinux стал поддерживать более направленный подход к выделению сущностей,

для которых может быть применен рекомендательный или принудительный режим: он стал применяться к наборам функций (доменам). Как было отмечено ранее, домен является термином, который применим в SELinux для типов (часть параметров безопасности) объектов в том случае, когда объектом является процесс. При помощи **рекомендательных доменов** (permissive domains) мы можем отметить определенный домен как работающий в рекомендательном режиме (и, таким образом, не требующий принудительного соблюдения правил политики SELinux), даже если вся остальная система работает в принудительном режиме.

Предположим, мы запускаем сервер для технологии **DLNA** (Digital Living Network Alliance – Объединение цифровых сетевых устройств для жизни), соединяющий домашние компьютеры, мобильные телефоны, бытовую технику в единую цифровую сеть, для того чтобы обрабатывать наши праздничные фотографии на других наших медиаустройствах, или для предоставления последнего внутрикорпоративного видео на каком-то наборе мониторов на всей территории организации. Почему-то у нас не получается показать недавно доступные к просмотру медиафайлы, и мы выясняем, что причина тому – средства SELinux, которые блокируют работу DLNA. Несмотря на то что строго рекомендуется решить данную проблему, возможно даже путем корректирования политики безопасности, мы вполне можем решиться сначала на то, чтобы просто обойти проблему, а уже потом исправить ее как следует. Вместо полного отключения системы контроля SELinux мы можем выделить домен, в котором сервер DLNA был запущен (какой-нибудь `minidlna_t`), и определить его в разряд рекомендательных доменов.

Для того чтобы перевести домен в разряд рекомендательных, мы используем команду `semanage`:

```
# semanage permissive -a minidlna_t
```

При помощи все той же команды `semanage` мы можем вывести список определенных на данный момент таких доменов. В дистрибутиве RHEL какое-то количество доменов будет по умолчанию работать в режиме `permissive`, поскольку это часть их (производителей) подхода к прохождению жизненного цикла разрабатываемой политики безопасности:

```
# semanage permissive -l
Customized Permissive Types
minidlna_t
Builtin Permissive Types
mon_procd_t
mon_statd_t
...
ptp4l_t
```

- P** В представленном выводе команды имеются следующие подлежащие переводу фразы:
- Customized Permissive Types – привнесенные [настроенные в процессе работы] рекомендательные типы;
  - Builtin Permissive Types – встроенные рекомендательные типы.

Другой метод вывода привносимых (настроенных в процессе работы) рекомендательных типов (которые в дистрибутиве не были промаркированы как рекомендательные) заключается в использовании команды `semodule`. В предыдущей главе мы кратко затронули эту команду, когда говорили о модулях политик безопасности SELinux. Мы можем ее использовать для вывода списка модулей политик SELinux, которые имеют в своем названии фразу `permissive_`, потому что команда `semanage permissive` будет в действительности формировать небольшой модуль политики SELinux, для которого домен отмечен как рекомендательный:

```
# semodule -l | grep permissive_
permissive_minidlna_t          1.0
```

**i** Дистрибутивы, которые имеют относительно недавнее пространство пользователя SELinux, такие как Gentoo, могут не отображать версию.

Для того чтобы домен вывести из режима `permissive`, следует использовать аргумент `-d` в команде `semanage`. Это возможно только для доменов, которые были промаркированы администратором как рекомендательные, а те, которые поставлялись вместе с дистрибутивом, нельзя перевести в принудительный режим при помощи этого метода:

```
# semanage permissive -d minidlna_t
```

Когда домен отмечен как рекомендательный, приложение будет вести себя так, будто бы SELinux не был включен в системе, упрощая нам задачу по выяснению, действительно ли SELinux является причиной некорректной работы приложения. Стоит отметить, что, несмотря на это, другие домены, в т. ч. те, которые взаимодействуют с рекомендательным доменом, продолжают управляться в принудительном режиме контроля доступа SELinux.

Другой способ использования рекомендательных доменов предназначен для разработчиков политик. Когда приложение запущено в рекомендательном домене, каждое действие, которое оно совершает, противоречащее политике безопасности, будет зарегистрировано в журнале SELinux. Разработчики политик могут запустить приложение через различные пользовательские сценарии и потом использовать сгенерированные журналы регистрации для формирования предметных правил защиты.

**i** Данный подход к формированию политик имеет значительный недостаток, который заключается в том, что иногда в приложениях будут срабатывать действия (отражаемые в журналах SELinux), которые приложениям в действительности не нужны (например, сканирование всех двоичных файлов для обнаружения собственных вспомогательных скриптов). Разработчикам политик следует быть внимательными, когда они изменяют политики таким методом.

В любом случае, если приложение требует, чтобы SELinux был отключен, имеет гораздо больший смысл создать ложный домен для этого и отметить его как рекомендательный, нежели отключить защиту SELinux для всей системы.

### 2.1.5. Определение приложений, активно взаимодействующих с SELinux

Большинство приложений сами по себе не имеют представления о том, работают ли они со включенным SELinux или с выключенным. И поэтому рекомендательный режим приводит к тому, что приложение ведет себя так, как если бы SELinux не был включен с самого начала. При этом есть такие приложения, которые активно обращаются к функциям SELinux. Подобные приложения могут быть названы SELinux-зависимыми (*SELinux-aware*), потому что их поведение основано на доступной информации, связанной с SELinux.

Такие приложения изменяют свое поведение, когда защита SELinux включена, например запрашивают политику или выполняют проверку параметров безопасности, в которых приложение должно работать. Большинство из таких зависимых от работы SELinux приложений не могут правильно определить, запущены они в рекомендательном режиме или нет. Поэтому запуск подобных приложений в рекомендательном домене (или в системе, где SELinux работает в таком режиме) будет в основном не эффективен, настолько же, как если бы SELinux был бы выключен в принципе.

В качестве примеров таких приложений можно назвать сервис защищенной командной оболочки удаленного доступа SSH, службу аутентификации, инициализацию операционной системы и некоторые программы, выполняющие задачи в заданные моменты времени, а также ряд утилит ядра Linux (вроде `ls` и `id`). Они могут демонстрировать ошибки доступа или различное поведение, основанное на политике SELinux, даже когда SELinux не находится в принудительном режиме работы.

Мы можем определить, является ли приложение зависящим от SELinux, путем проверки динамической связанности приложения с `libselinux`. Это может быть сделано с помощью команды `readelf` или `ldd` следующим образом:

```
# readelf -d /bin/ls | grep libSELinux
0x0000000000000001 (NEEDED) Shared library: [libSELinux.so.1]
# ldd /bin/ls | grep SELinux
libSELinux.so.1 => /lib64/libSELinux.so.1 (0x00007f77702dc000)
```

Понимание того, является ли приложение зависящим от SELinux или нет, может помочь в устранении неисправностей.

## 2.2. РЕГИСТРАЦИЯ СОБЫТИЙ И АУДИТ В SELINUX

Разработчики SELinux хорошо осведомлены, что подсистема, ориентированная на защиту, такая как SELinux, может быть успешна, если она имеет возможность расширенной регистрации событий, а также возможность обнаружения и устранения ошибок. Каждое действие, о котором получает сведения SELinux, будучи составной частью перехвата событий на уровне модулей безопасности (LSM), должно быть проверяемым. Отказы (действия, которые запрещает SELi-

них) всегда следует протоколировать так, чтобы администраторы смогли бы предпринять надлежащие меры. Настройки и изменения в SELinux, такие как загрузка новых политик или смена альтернативных параметров SELinux, следует всегда завершать контрольным сообщением на экране.

### 2.2.1. Последовательность контроля событий о нарушениях безопасности

По умолчанию SELinux будет отправлять эти сообщения в подсистему контроля событий безопасности Linux (*audit subsystem*), если ядро операционной системы было сконфигурировано с поддержкой аудита, при помощи опции конфигурации ядра CONFIG\_AUDIT. Из подсистемы контроля событий безопасности (аудита) сообщения забираются программой *auditd* и записываются ею в файл журнала */var/log/audit/audit.log*. Дополнительные правила обработки могут быть определены при помощи процесса, координирующего контроль событий безопасности, – диспетчерского аудита (*audisp*), который получает контролируемые события и передает их одному или нескольким отдельным процессам для дальнейшей обработки. Этот подход используется, например, программой поиска неисправностей SELinux (*setroubleshootd*), неким дополнительным сервисом по оказанию помощи в части поиска проблемы на основе событий SELinux.

Общая схема контроля событий безопасности (аудита) показана на рис. 2.1.

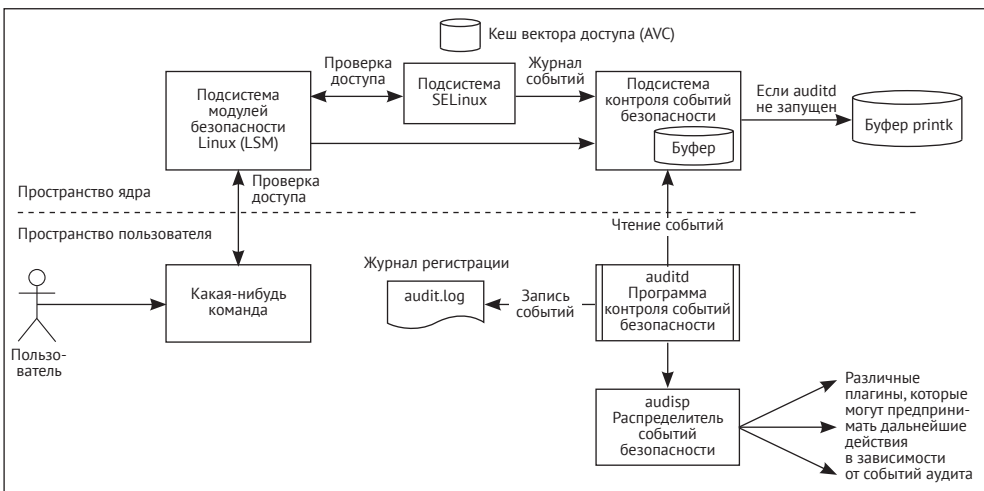


Рис. 2.1 ❖ Обработка событий безопасности в SELinux

Во включенном состоянии SELinux регистрирует в журнале почти каждую проверку доступа, которая была отклонена. Если при этом включен контроль событий безопасности, эти отказы регистрируются программой аудита в фай-

ле `audit.log`. Если нет, события сохраняются в буфере сообщений ядра Linux, к которому можно обратиться при помощи команды `dmesg`, и при этом часто захватываются средствами подсистемы регистрации событий самой операционной системы.

Когда установлена программа поиска неисправностей SELinux, являющаяся частью пакета `setroubleshoot-server` в RHEL, тогда программа контроля событий безопасности (аудита) будет совместно с регистрацией событий также распределять события при помощи программы распределения `audisp` в направлении процесса `sedispatch`. Этот процесс будет в дальнейшем обрабатывать событие и отправлять его через D-Bus (реализацию системного межпроцессного взаимодействия, популярного в системах Linux) к программе поиска неисправностей SELinux. Эта программа будет проводить анализ события и может затем предложить администратору одно или более решений по исправлению проблемы. Мы опишем программу поиска неисправностей SELinux в этой главе чуть позже.

Далеко не каждый раз, когда SELinux проверят права, выполняется обращение к правилам политики безопасности. Вместо этого защита использует такое средство, как **кеш вектора доступа (access vector cache – AVC)**, в котором сохраняются результаты предыдущих попыток получения доступа. Этот кеш гарантирует, что SELinux сможет быстро отреагировать на действия без особых нагрузок на производительность системы. Аббревиатура этого кеша используется как тип сообщения для большинства событий SELinux, как мы можем увидеть из следующего примера:

```
type=AVC msg=audit(1470312632.027:4702304): avc: denied { append }
  for pid=14352 comm="rsyslogd" name="oracle_audit.log" dev="dm-2"
  ino=387512
  scontext=system_u:system_r:syslogd_t:s0
  tcontext=system_u:object_r:usr_t:s0 tclass=file permissive=0
```

Кеш вектора доступа может быть немного перенастроен путем установки объема памяти. Объем задается в псевдофайле `/sys/fs/SELinux/avc/cache_threshold`. Например, для увеличения размера памяти до значения 768 (по умолчанию 512) следует использовать следующую команду:

```
# echo 768 > /sys/fs/SELinux/avc/cache_threshold
```

В псевдофайле `hash_stats` хранятся статистические сведения о хешах [векторах доступа], отражающих сведения о записях кеша вектора доступа (AVC entries), объеме занятой памяти кеша (buckets), максимальной длине цепочек (longest chain):

```
# cat /sys/fs/SELinux/avc/hash_stats
entries: 510
buckets used: 287/512
longest chain: 6
```

Если администраторы предполагают, что ухудшение производительности системы связано с работой SELinux, тогда рекомендуется посмотреть на стро-

ку вывода, определяющую длину цепочки, – `longest chain` в `hash_stats`. Если она больше значения 10, тогда вполне возможно влияние на производительность системы, и в этом случае может помочь обновление размера кеша.

Любое разрешение, которое необходимо проверить, представляется как вектор доступа (*access vector*), и тогда этот кеш позволяет определить, было ли это конкретное разрешение ранее уже проверено. Если было, тогда решение принимается на основе тех сведений, которые хранятся в кеше; в другом случае обращение выполняется к политике, и кеш в результате обновляется. Это внутреннее устройство работы SELinux не так важно для большинства администраторов, но в итоге мы теперь хотя бы знаем, откуда происходит термин AVC.

### 2.2.2. Исключение конкретных отказов в доступе из числа регистрируемых

Существует важная директива политики SELinux, которая тоже влияет на кеш вектора доступа, и эта директива – `dontaudit`. Правило `dontaudit` в политике SELinux сообщает системе защиты, что данный конкретный отказ в доступе *не* должен быть зарегистрирован в журнале. Это исключительный пример того, когда SELinux не будет регистрировать отказ в доступе, т. к. автор политики однозначно отключил контроль событий. Это обычно делается, чтобы сократить нагромождение записей в журнале и скрыть внешние отказы, которые не имеют влияния на безопасность системы.

Утилита `seinfo` может предоставить нам сведения о том, как много в настоящий момент не регистрируется конкретных отказов, а также как много регистрируется, наоборот, разрешенных действий (определяемых родственной директивой `auditallow`):

```
# seinfo --stats | grep -i audit
Auditallow:    152 Dontaudit:    8381
```

К счастью, эти установки `dontaudit` по желанию могут быть отключены. При помощи следующей команды `semodule` эти правила удаляются из активной политики:

```
# semodule --disable_dontaudit -build
```

Аргументы команды могут быть также сокращены до отдельных символов `-D` и `-B` соответственно. Для того чтобы заново включить эти настройки конкретных отказов в доступе, надо просто пересобрать политику следующей командой:

```
# semodule -B
```

Отключение правил, основанных на директиве `dontaudit`, может иногда помочь в работе по обнаружению неисправностей, которая не дает результатов на основе любых применяемых контролируемых событий, регистрируемых в журнале. В основном говорят о том, что контролируемые события, которые автор политики отметил как внешние, не являются причиной сбоя в работе.



### 2.2.3. Конфигурирование подсистемы контроля событий безопасности Linux

SELinux будет пытаться использовать подсистему контроля событий безопасности (аудита), когда это возможно, и будет вынужден возвращаться к обычной системной регистрации, когда подсистема аудита недоступна. Это может быть и по причине того, что подсистема контроля событий безопасности уровня ядра Linux не сконфигурирована, или потому, что программа аудита ОС Linux не работает сама по себе.

Для контроля событий безопасности в Linux нам обычно не требуется что-нибудь настраивать, поскольку отказ в доступе SELinux регистрируются по умолчанию (как сообщения AVC-типа). Конкретные отказы в доступе будут записаны в файле журнала контроля событий безопасности (`/var/log/audit/audit.log`), обычно вместе с системным вызовом, который, собственно, и привел к этому отказу:

```
time->Thu Aug 4 08:28:57 2020
type=SYSCALL msg=audit(1470313737.195:322): arch=c000003e
  syscall=105 success=yes exit=0 a0=0 a1=7f9c3fdde1d0
  a2=8000020 a3=7f9c37ae92e0 items=0 ppid=14542 pid=14544
  auid=1001 uid=1001 gid=1001 euid=0 suid=0 fsuid=0 egid=1001
  sgid=1001 fsgid=1001 tty=pts0 ses=6 comm="su"
  exe="/usr/bin/su" subj=user_u:user_r:user_t:s0 key=(null)
type=AVC msg=audit(1470313737.195:322): avc: denied { setuid }
  for pid=14544 comm="su" capability=7
  context=user_u:user_r:user_t:s0
  tcontext=user_u:user_r:user_t:s0 tclass=capability
```

Файл журнала регистрации для системы контроля событий безопасности может быть настроен через параметр `log_file` в файле `/etc/audit/auditd.conf`.

Для включения регистрации удаленных событий безопасности (т. е. для централизованного сбора событий аудита в одном месте от различных сетевых узлов) у вас есть возможность либо перенаправить системный журнал `syslog`, либо включить плагин `audisp-remote`.

Вместе с перенаправлением системного журнала программа распределения событий конфигурируется для отправки событий безопасности, как правило, локальному системному регистратору. Затем администратору необходимо сконфигурировать этот локальный системный регистратор событий для дальнейшей передачи какой-нибудь удаленной системе.

Отредактируйте файл `/etc/audisp/plugins.d/syslog.conf` и установите параметру `active` значение `yes`:

```
# vi /etc/audisp/plugins.d/syslog.conf
active = yes
direction = out
path = builtin_syslog
type = builtin
args = LOG_INFO
format = string
```

Использование системного регистратора для централизованного контроля событий безопасности может быть нелучшим выбором, поскольку системный регистратор в основном использует незашифрованную, и часто даже негарантированную, передачу данных. При помощи дополнительного программного модуля `audisp-remote` контролируемые события могут быть отправлены в зашифрованном виде и с гарантированной доставкой к удаленному серверу, с запущенной программой `auditd`.

Первое, что надо сделать, – это сконфигурировать программу контроля событий безопасности (аудита) на конечном сервере, чтобы он мог принимать регистрационную информацию с такими событиями от удаленных узлов сети. Для этого надо указать 60-й порт в настройках программы:

```
# vi /etc/audit/auditd.conf
tcp_listen_port = 60
```

Затем настроить дополнительный программный модуль `audisp-remote` для присоединения к серверу централизованного контроля событий безопасности:

```
# vi /etc/audisp/audisp-remote.conf
remote_server = <targethostname>
port = 60
```

И в завершение надо, собственно, включить этот самый программный модуль:

```
# vi /etc/audisp/plugins.d/au-remote.conf
active = yes
```

Распространяется модуль `audisp-remote` в рамках программного пакета `audisp-plugins` операционной системы RHEL и через стандартный пакет `sys-process/audit` дистрибутива ОС Gentoo.

Рекомендуется использовать подсистему контроля событий безопасности (аудита) Linux все время. Она не только хорошо взаимодействует с утилитами поиска неисправностей, но также позволяет администраторам использовать инструменты подсистемы контроля, чтобы запрашивать контрольные записи или даже формировать отчеты, как это делается при помощи программы `augerport`:

```
# augerport --avc --start recent
AVC Report
=====
# date time comm subj syscall class permission obj event
=====
...
12. 08/04/2020 09:00:38 su user_u:user_r:user_t:s0 105 capability
setuid user_u:user_r:user_t:s0 denied 376
```

### 2.2.4. Настройка локального системного регистратора событий

Когда подсистема контроля событий безопасности не включена, или не работает программа аудита Linux, тогда события SELinux будут собираться системным регистратором через компоненты регистрации ядра (`kernel.*`). Большин-

ство системных регистраторов будут сохранять эти журналируемые события ядра в общем файле событий, таком как `/var/log/messages`.

Мы же можем, в свою очередь, сконфигурировать системный регистратор так, чтобы направлять сообщения SELinux типа AVC в его собственный журнал, такой как `/var/log/avc.log`. Например, для системного регистратора `syslog-ng` конфигурация могла бы быть следующей:

```
source kernsrc { file("/proc/kmsg"); };
destination avc { file("/var/log/avc.log"); };
filter f_avc { message(".*avc: .*"); };
log { source(kernsrc); filter(f_avc); destination(avc); };
```

Для такого системного регистратора, как `rsyslog`, правило могло бы выглядеть так:

```
:msg, contains, "avc: " -/var/log/avc.log
```

Когда протоколирование событий SELinux управляется через локальный системный регистратор, тогда простой способ быстро получить последние отказы в доступе (или другие сообщения) – это использовать команду `dmesg`:

```
# dmesg | grep avc | tail
```

Правда, стоит иметь в виду, что, в отличие от журналов аудита, многие системы позволяют результаты команды `dmesg` читать и непривилегированным пользователям. Это может привести к утечке информации к недоверенным пользователям. По данной причине некоторые политики SELinux не дают обычным пользователям доступ к буферу уровня ядра (`kernel ring buffer`) (и как таковому использованию команды `dmesg`) до тех пор, пока параметру SELinux `user_dmesg` не будет явно установлено значение `on`:

```
# setsebool user_dmesg on
```

В дистрибутиве RHEL, однако, значение `user_dmesg` отсутствует. В нем имеется только стандартный неограниченный (*unconfined*) тип пользователя, который, так же как и пользователь с администраторскими правами, имеет доступ к буферу ядра ОС. Для того чтобы ограничить доступ на чтение этой информации каких-либо пользователей, необходимо им исключить администраторские права путем отнесения к любой соответствующей группе пользователей SELinux, например к `user_u` или `(x)guest_u`.

## 2.2.5. Разбор информации об отказах SELinux

Единственное, что каждый из нас должен будет делать неоднократно с системами SELinux, – это читать и интерпретировать информацию об отказах доступа. Когда SELinux запрещает доступ и отсутствует правило `dontaudit`, относящееся к тому действию, для которого был выполнен запрет, то SELinux будет регистрировать о нем сведения в журнале. Если ничего не записано, то вполне возможно, что не SELinux является виновником случившегося отказа. Стоит напомнить, что очередь SELinux наступает после того, как отработают штатные средства

проверки дискреционных прав доступа Linux, поэтому если существует запрещение на их уровне, то защита SELinux никогда не задействуется.

Сообщения SELinux об отказах регистрируются в тот момент, когда SELinux предотвращает некоторую попытку доступа. Когда SELinux работает в принудительном режиме, приложение обычно возвращает ошибку **Permission denied** («Отсутствие прав доступа»), хотя иногда это может выглядеть и более туманно. Например, следующая попытка непривилегированного пользователя применить команду `su` для получения прав доступа пользователя `root` демонстрирует другую ошибку – *incorrect password* («неправильный пароль»):

```
$ su -
Password: (вводится корректный пароль)
su: incorrect password
```

Но в большинстве случаев, тем не менее, возникающая ошибка – это ошибка прав доступа:

```
$ ls /proc/1
ls: cannot open directory /proc/1: Permission denied
# ls -ldZ /proc/1
dr-xr-xr-x. root system_u:system_r:init_t:s0      /proc/1
```

Итак, как выглядит какое-нибудь сообщение об отказе в доступе? Следующий вывод команды показывает отказ от подсистемы контроля событий безопасности, который мы можем запросить при помощи команды `ausearch`:

```
# ausearch -m avc -ts recent
----
time-->Thu Aug 4 09:00:38 2020
type=AVC msg=audit(1470315638.218:376): avc: denied { search }
  for pid=5005 comm="dnsmasq" name="net" dev="proc" ino=5403
  scontext=system_u:system_r:dnsmasq_t
  tcontext=system_u:object_r:sysctl_net_t tclass=dir permissive=0
```

Давайте разберем это сообщение об отказе на части. Следующая таблица дает больше информации о каждой части указанного выше сообщения. Как некий администратор, знающий, насколько важно умение разбирать сообщения об отказах, также уделим этому время и отразим сведения в табл. 2.1.

**Таблица 2.1. Разбор полей сообщения об отказе**

Наименование поля	Описание	Пример
Действие системы защиты SELinux <i>SELinux action</i>	Действие, которое SELinux уже выполнил или собирается совершить, работая в принудительном режиме ( <i>enforcing</i> ). Обычно это <i>denied</i> (отказ), хотя некоторые операции явно отмечены как тоже подлежащие контролю событий безопасности (аудиту) и приведут к тому, что будет указан <i>granted</i> (разрешенный доступ)	denied

Таблица 2.1 (продолжение)

Наименование поля	Описание	Пример
Выполняемое действие субъектом доступа <i>Permissions</i>	Конкретное действие, которое было проведено (операция, выполняемая процессом). Обычно это какое-то единичное действие, хотя иногда может быть набором действий (например, <i>read</i> (чтение), <i>write</i> (запись)). В данном примере это <i>search</i> (поиск)	{ search }
Идентификатор процесса (субъекта доступа) <i>Process ID</i>	Идентификатор ( <i>ID</i> ) процесса, который выполнял действие	для pid=5005
Имя процесса <i>Process name</i>	Символьное имя процесса (команды). При этом запись не отображает каких-либо аргументов, с которыми был запущен процесс. О назначении данного процесса см. в комментарии к русскому изданию ниже	comm="dnsmasq"
Имя цели <i>Target name</i>	Имя цели (ресурса), по отношению к которой процесс выполняет действие. Если целью является файл, тогда приводится его символьное имя или директория. О назначении данного каталога см. в комментарии к русскому изданию ниже	name="net"
Целевое устройство <i>Target device</i>	Устройство, на котором находится целевой ресурс. Вместе со следующим полем (номер индексного дескриптора целевого файла) данное поле позволяет однозначно идентифицировать ресурс в системе. О назначении данного устройства см. в комментарии к русскому изданию ниже	dev="proc"
Номер индексного дескриптора целевого файла <i>Target file inode number</i>	Номер индексного дескриптора ( <i>inode</i> ) целевого файла или директории. Вместе с полем целевого устройства ( <i>target device</i> ) позволяет нам точно идентифицировать ресурс в системе	ino=5403
Параметры безопасности источника <i>Source context</i>	Параметры безопасности, которые назначены для процесса, осуществляющего доступ к ресурсу (или отдельно взятый набор функций – домен, характерный для процесса)	scontext=system_u:system_r:dnsmasq_t
Параметры безопасности цели <i>Target context</i>	Параметры безопасности целевого ресурса	tcontext=system_u:object_r:sysctl_net_t
Класс объекта <i>Object class</i>	Класс целевого объекта, например каталог, файл, сокет, узел, канал, файловый дескриптор, файловая система или право доступа	tclass=dir

Таблица 2.1 (окончание)

Наименование поля	Описание	Пример
Режим работы системы защиты <i>Permissive mode</i>	Режим, в котором находилась система, когда действие было выполнено. Если установлено значение 0, значит, SELinux находился в принудительном режиме <i>enforcing</i> , в другом случае – в рекомендательном <i>permissive</i> . Это поле доступно начиная с версии ядра Linux 3.16	<code>permissive=0</code>

**Р** «dnsmasq» – это открытое программное обеспечение, предоставляющее следующую инфраструктуру для небольших сетей: сервера доменных имен (DNS), протокола динамического конфигурирования узла сети (DHCP), автоконфигурации для узлов, работающих по протоколу IPv6, и сетевой загрузки. Включается во многие дистрибутивы Linux. Официальный сайт – <http://www.thekelleys.org.uk/>.

«proc» – это обозначение файловой системы *procfs*, обрабатывающей сущности, которые предоставляют доступ к образу каждого запущенного процесса в системе. Имя каждой записи в директории `/proc` представляет собой пятизначный номер в десятичной системе счисления, соответствующий какому-то конкретному процессу. Размером файла является вся виртуальная память процесса.

«net» – директория виртуальной файловой системы *procfs*, располагающаяся в `/proc` и предоставляющая всесторонний обзор различных сетевых параметров. Каждая поддиректория описывает аспекты системной конфигурации сети. Более подробно о них можно узнать в документации на соответствующую операционную систему.

Таким образом, вышеуказанное сообщение об отказе в доступе может быть сформулировано следующим образом:

SELinux запретил операцию поиска, инициированную процессом `dnsmasq` (с идентификатором `5005`), по отношению к директории `net` (с индексным дескриптором `5403`) в устройстве `proc`. Процесс `dnsmasq` был запущен с меткой безопасности `system_u:system_r:dnsmasq_t`, а каталог `net` имел в этот момент метку `system_u:object_r:sysctl_net_t`. При этом SELinux работал в принудительном режиме.

Некоторые сообщения об отказах имеют отличающиеся от разобранных поля, например такие, как это:

```
avc: denied { send_msg } for msgtype=method_call
interface=org.gnome.DisplayManager.Settings
member=GetValue dest=org.gnome.DisplayManager
spid=3705 tpid=2864
scontext=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
tcontext=system_u:system_r:xdm_t:s0-s0:c0.c1023
tclass=dbus permissive=0
```

Несмотря на то что некоторые поля отличаются, тем не менее текст является читаемым и может быть интерпретирован следующим образом:

SELinux запретил процессу с идентификатором 3705 выполнить удаленный вызов метода по шине межпроцессного взаимодействия D-Bus (вызов метода получения значения – GetValue, относящийся к интерфейсу org.gnome.DisplayManager.Settings) в рамках реализации org.gnome.DisplayManager, предложенной процессом с идентификатором 2864. При этом вызывающий процесс был запущен со следующими параметрами безопасности: unconfined\_u:unconfined\_r:unconfined\_t:s0-s0.c0.c1023, а удаленный процесс – с такими: system\_u:system\_r:xdm\_t:s0-s0.c0.c1023.

В зависимости от выполняемых действий и класса объекта доступа SELinux использует разные поля, чтобы дать наиболее полную и необходимую нам информацию для обнаружения неисправностей. Рассмотрим следующую запись об отказе:

```
avc: denied { name_bind } for pid=23849
    comm="postgres" src=6030
    scontext=system_u:system_r:postgresql_t
    tcontext=system_u:object_r:unreserved_port_t
    tclass=tcp_socket permissive=0
```

Приведенный выше случай запрета доступа произошел, потому что база данных PostgreSQL была сконфигурирована для получения запросов на порт, не установленный по умолчанию (6030 вместо 5432 стандартного).

Определение какой-либо конкретной проблемы – это, по существу, понимание того, как выполняются действия, а также правильная интерпретация сообщений об отказах. Приведенное выше сообщение о блокировке доступа при использовании шины межпроцессного взаимодействия D-Bus является довольно сложным для обнаружения неисправности, если мы не знаем, как работает система D-Bus (или как она применяет типы сообщений, элементы и интерфейсы в своих базовых протоколах). Для определения неисправности подобные записи журналов об отказах в доступе дают нам достаточно информации, чтобы начать выполнять эту работу. Они дают четкое понимание того, что было заблокировано.

Было бы неправильно сразу же начинать продумывать, как разрешить запрещенное действие, путем добавления соответствующего правила в политику SELinux (об этом будет сказано в главе 8 «Работа с политиками SELinux»), потому что существуют другие варианты, и они часто бывают лучше, например такие:

- установка правильной метки на объекте доступа (обычно это случай, когда им является нестандартный порт, нестандартное локальное размещение и т. д.);
- переключение флагов, которые определяют особенности политики SELinux для разрешения дополнительных прав;
- установка правильной метки у иницирующего действие процесса (часто этот случай возникает тогда, когда приложение, к которому относится данный процесс, установлено без использования штатных средств установки);

- использование приложения так, как предусмотрено, и отсутствие попытки применять его через другие средства (поскольку SELinux разрешает выполнять только ожидаемое поведение). Так, запуск какого-либо фонового процесса (*daemon*) следует выполнять через какой-нибудь предназначенный для этого сервис (специальный скрипт инициализации или подсистему инициализации операционной системы *systemd*), а не через интерфейс командной строки.

## 2.2.6. Другие типы событий, связанные с SELinux

Хотя большая часть событий журнала SELinux относятся к типу AVC, они не являются единственными, с которыми администратору придется иметь дело. Многие события контроля безопасности (аудита) будут показывать информацию SELinux как часть этого события, даже когда SELinux имеет мало общего с самим событием. Но несколько типов событий аудита напрямую относятся к SELinux.

- ✔ **Рассмотрим все возможные типы событий контроля безопасности:** полный список всех таких событий можно найти в заголовочном файле `linux/audit.h`, размещенном в каталоге `/usr/include` (установленном в рамках пакета `kernel-headers` операционной системы RHEL).

### USER\_AVC

Событие `USER_AVC` подобно обычным событиям контроля безопасности, имеющим тип AVC, но в данном случае оно является сформированным каким-либо диспетчером объектов пользовательского пространства. Этот диспетчер относится к тем приложениям, которые применяют правила политики SELinux, но они обеспечивают выполнение этих правил собственными силами, а не посредством ядра операционной системы.

Следующий пример показывает такое событие, сформированное системой межпроцессного взаимодействия D-Bus:

```
type=USER_AVC msg=audit(1467890899.875:266): pid=693 uid=81
  aid=4294967295 ses=4294967295
  subj=system_u:system_r:system_dbusd_t:s0-s0:c0.c1023
  msg='avc: denied { acquire_svc }
    for service=org.freedesktop.resolve1 spid=1434
  scontext=system_u:system_r:systemd_resolved_t:s0
  tcontext=system_u:system_r:system_dbusd_t:s0-s0:c0.c1023
  tclass=dbus exe="/usr/bin/dbus-daemon" sauid=81
  hostname=? addr=? terminal=?'
```

Событие имеет две части. Все, что находится выше строки `msg=`, является информацией о диспетчере объектов пользовательского пространства, который и сформировал это событие. По сути же наиболее содержательная информация размещается во второй части и включает поля, подобные тем, которые мы уже знаем из формата стандартного AVC-события.



## ***SELINUX\_ERR***

Событие `SELINUX_ERR` возникает тогда, когда к SELinux поступает запрос на выполнение чего-либо такого, что не только противоречит разрешенным правам доступа, но и нарушает политику безопасности. Данная задача не может быть решена авторами политик SELinux путем простого *разрешения* этого запроса. Эти события обычно указывают на неправильное использование приложений и служб, которое политика никак не предусматривает:

```
type=SELINUX_ERR msg=audit(1387729595.732:156): security_compute_sid:
  invalid context unconfined_u:system_r:hddtemp_t:s0-s0:c0.c1023 for
  scontext=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
  tcontext=system_u:object_r:hddtemp_exec_t:s0 tclass=process
```

В вышеуказанном примере пользователь (работающий в неограничивающем его наборе функций (домене), который называется `unconfined_t`) выполнял программу, позволяющую определить температуру жесткого диска `hddtemp`, входящего в домен `hddtemp_exec_t`, а политика хотела переключиться в домен `hddtemp_t`. Однако такой вариант в рамках всего набора параметров безопасности (`unconfined_u:system_r:hddtemp_t:s0-s0:c0.c1023`) оказался некорректным. Потому что пользователь, работающий с правами домена `unconfined_u`, не может иметь роль `system_r`, предназначенную для системных процессов, работающих в фоновом режиме.

## ***MAC\_POLICY\_LOAD***

Событие `MAC_POLICY_LOAD` формируется всякий раз, когда система загружает новую политику SELinux в память. Оно возникает, когда администратор загружает новый или обновленный модуль политики SELinux, когда пересобирает политику с отключенными правилами `dontaudit` или изменяет конфигурационные параметры у SELinux, которые должны сохранить новые значения и после перезагрузки:

```
type=MAC_POLICY_LOAD msg=audit(1470381810.215:178): policy loaded
  auid=1001 ses=2
```

Событие `MAC_POLICY_LOAD` может сопровождаться событием `USER_MAC_POLICY_LOAD`. Оно возникает, когда диспетчер объектов пользовательского пространства обнаруживает, что политика была обновлена, и в результате предпринимает действия по регистрации события. Заметим, что не все диспетчеры объектов пользовательского пространства отправляют это событие: некоторые диспетчеры объектов будут запрашивать текущую (*live*) политику, и таким образом им не требуется выполнять каких-либо действий, когда новая политика загружена.

## ***MAC\_CONFIG\_CHANGE***

Когда логика работы SELinux изменена в конфигурационных файлах, но изменения носят временный характер, тогда будет отправлено событие `MAC_CONFIG_CHANGE`. Оно говорит администратору о том, что активная политика проин-

структурирована о необходимости немного изменить свое поведение, но лишь пока текущая политика загружена:

```
type=MAC_CONFIG_CHANGE msg=audit(1470381810.200:177):
  bool=user_ping val=0 old_val=1 auid=1001 ses=2
```

В приведенном примере двухвариантный параметр `user_ping` был изменен со значения 1 (on) на 0 (off).

Комментарий: тем самым отменяя возможность контроля использования команд `ping` и `traceroute` пользователем.

## MAC\_STATUS

Событие `MAC_STATUS` появляется тогда, когда меняется режим работы (статус) SELinux. Например, если администратор использует команду `setenforce 0` и таким образом переводит SELinux в разрешающий (permissive) режим, то появится сообщение следующего вида:

```
type=SYSCALL msg=audit(1470383274.576:74): arch=c000003e syscall=1
  success=yes exit=1 a0=3 a1=7ffe4d5ee270 a2=1 a3=7ffe4d5edff0
  items=0 ppid=8977 pid=9226 auid=0 uid=0 gid=0 euid=0 suid=0
  fsuid=0 egid=0 sgid=0 fsgid=0 tty=tty1 ses=1 comm="setenforce"
  exe="/usr/sbin/setenforce"
  subj=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
  key=(null)
type=MAC_STATUS msg=audit(1470383274.576:74): enforcing=0
  old_enforcing=1 auid=0 ses=1
```

Здесь событие `SYSCALL` приводится вместе с описываемым событием, поскольку оно предоставляет больше детальной информации о случившемся: кто изменил состояние, при помощи какой команды и т. д. По возможности команда `ausearch` будет группировать все связанные события (включая `SYSCALL`), для того чтобы предоставить администратору наиболее полный обзор происшедшего.

## Шесть событий системы NetLabel

NetLabel – это проект уровня ядра Linux, поддерживающий маркированные сетевые пакеты и позволяющий параметрам безопасности, таким как в SELinux, корректно проходить между узлами сети. Один из протоколов, реализацию которого поддерживает NetLabel в операционной системе Linux, – это стандартизированный протокол установки меток **CIPSO** (Common IP Security Option – Общий параметр безопасности интернет-протокола), который мы более подробно рассмотрим в главе 5 «Контроль сетевого взаимодействия».

**Р** Параметр безопасности интернет-протокола (Internet Protocol Security Option – IPSO) относится к одному из трех типов параметров безопасности протокола сетевого уровня (IP), являющихся полями, которые могут быть добавлены к дейтаграмме этого протокола, для переноса информации о безопасности этой дейтаграммы. Один из этих типов называется «Общий параметр безопасности интернет-протокола». Был разработан компанией «Trusted Systems Interoperability Working Group» для переноса иерархических

и неиерархических меток безопасности. Прежде был назван «Коммерческий параметр защиты интернет-протокола» (Commercial IP Security Option). Проект версии 2.3 был опубликован 9 марта 1993 г. как интернет-проект, но в формат RFC переведен не был [см. RFC 4949 «Словарь безопасности интернета», версия 2, стр. 160: <https://tools.ietf.org/html/rfc4949>].

Следующие события контроля безопасности связаны с возможностями системы NetLabel:

- `MAC_UNLBL_STCADD` и `MAC_UNLBL_STCDEL` возникают тогда, когда статическая метка добавляется или удаляется соответственно. Статическая маркировка означает, что если какой-то пакет был принят или передан без какой-либо метки, то статическая метка назначается ему как вид метки, устанавливаемый по умолчанию;
- `MAC_MAP_ADD` и `MAC_MAP_DEL` возникают тогда, когда добавляются и удаляются соответствия между параметрами протокола, поддерживающего маркировку (такого как CIPSO), и функциональными наборами (доменами) модулей безопасности SELinux;
- `MAC_CIPSOV4_ADD` и `MAC_CIPSOV4_DEL` возникают, когда конфигурация CIPSO (IPv4) была соответственно добавлена или удалена.

### **События протокола IPsec, поддерживающего маркировку**

Другим сетевым протоколом, поддерживающим маркировку параметрами безопасности, является IPsec. Благодаря этой возможности SELinux-параметры безопасности у процесса, иницирующего коммуникационное взаимодействие с целевым ресурсом по защищенному каналу средствами IPsec, являются известными для всех программных компонент IPsec, размещенных по обе стороны защищенного канала связи. Более того, SELinux будет содержать правила о том, какие функциональные группы (домены) имеют право выполнять сетевое взаимодействие, используя канал IPsec, и какие домены разрешены для взаимодействия по сети друг с другом.

Следующие события контроля безопасности связаны с работой защиты набора протоколов IPsec:

- `MAC_IPSEC_ADDDSA` и `MAC_IPSEC_DELSA` применяются, когда соответственно добавляется или удаляется безопасное соединение (*security association*) – т. е. когда очередной канал IPsec является созданным или же, наоборот, удаляется из списка имеющихся;
- `MAC_IPSEC_ADDSPD` и `MAC_IPSEC_DELSPD` применяются, когда добавляется или соответственно удаляется база данных политики безопасности (*security policy database*). Политика безопасности обычно описывает, действительно ли необходимо сетевой пакет защищать средствами IPsec, и если да, то через какое именно безопасное соединение;
- `MAC_IPSEC_EVENT` представляет общий тип сообщения для контроля безопасности, связанной с работой протоколов IPsec.

В главе 5 «Контроль сетевого взаимодействия» описана работа IPsec, поддерживающего маркировку параметрами безопасности SELinux.

### 2.2.7. Использование команды `ausearch`

Команда `ausearch`, являясь частью пакета `audit`, представляет собой часто используемую команду для формирования запросов к журналам подсистемы контроля событий безопасности в операционной системе. Мы уже кратко касались этой темы, когда впервые знакомимся с кешем контроля доступа SELinux и регистрацией им событий безопасности, но столь незначительное упоминание в данном случае недостаточно.

При помощи команды `ausearch` мы можем выполнять поиск для событий, которые происходили после определенного момента времени. Когда чуть раньше мы разбирали информацию в журнале событий, то использовали данную команду с параметром `-ts recent`, который подразумевает под «новейшим временем» (*recent*) последние 10 минут и отображает события за этот отрезок времени. Параметр может также указывать и на то, чтобы выдавать события не за отрезок, а произошедшие в четко фиксированное время. Другими условными обозначениями, используемыми в качестве параметров, являются следующие:

- `today`: это обозначение («сегодня») указывает на временной отрезок, начинающийся с 1-й секунды после наступления полуночи текущего дня;
- `yesterday`: это обозначение («вчера») указывает на временной отрезок, равный вчерашнему дню, также отсчитанный с 1-й секунды после полуночи;
- `this-week`, `this-month` и `this-year`: эти обозначения («текущая неделя», «текущий месяц» и «текущий год») указывают на временные отрезки, начинающиеся с 1-й секунды после полуночи первого дня текущей недели, текущего месяца или текущего года соответственно;
- `checkpoint`: это обозначение («контрольная точка») использует фиксированное время, указанное в файле `checkpoint.txt`.

Использование `checkpoint` особенно полезно во время решения задач по обнаружению неисправностей, поскольку может нам показать запреты выполнения, а также и другие события SELinux, начиная с последнего вызова команды `ausearch`:

```
# ausearch --checkpoint /root/ausearch-checkpoint.txt -ts checkpoint
```

Такое средство позволяет администраторам делать небольшие корректировки изменения, затем воспроизводить проблему, которую решают, и смотреть уже только те события, которые произошли с момента, определяемого контрольной точкой, вместо того чтобы пересматривать все события снова и снова.

По умолчанию команда `ausearch` отображает все события, которые произошли и отражены в журнале контроля событий безопасности. В системах с большой функциональной нагрузкой такой журнал может содержать очень много событий, и в результате выполнения команды будут отражены и те события, которые, по сути, не нужны. К счастью, пользователи могут ограничивать тип запрашиваемых событий при помощи параметров команды `ausearch`.

Для диагностики неисправностей SELinux выделяемые среди остальных события с типами `avc`, `user_avc` и `SELinux_err` успешно ограничивают объем обрабатываемой информации достаточным количеством сведений:

```
# ausearch -m avc,user_avc,SELinux_err -ts recent
```

Если количество отображаемых полей, таких как идентификаторы пользователей и временные метки, слишком неудобно для восприятия, то у команды `ausearch` существует возможность найти соответствие и выполнить преобразование идентификаторов в символьные имена пользователей, а временные метки – в отформатированные форматы времени. Для этого надо добавить опцию `-i` к команде `ausearch`, что позволит получить соответствующую интерпретацию этих полей и отобразить более просто воспринимаемые значения взамен исходных.

## 2.3. ПОЛУЧЕНИЕ ПОМОЩИ ПРИ ОТКАЗАХ

В некоторых дистрибутивах доступны дополнительные инструменты поддержки, которые помогают нам определить причину случившегося отказа. Эти инструменты имеют некоторые знания об общих ошибках (например, таких, которые возникают при неправильной настройке параметров безопасности у файлов приложения, ограничивающих веб-сервер в том, чтобы прочитать эти файлы). Другие дистрибутивы требуют от нас использования имеющегося опыта для принятия верных решений, предоставляя поддержку через почтовую рассылку, через сайты, собирающие сведения об ошибках, и через другие сопутствующие средства, например такие, как протокол коммуникации пользователей интернета в режиме реального времени (IRC).

### 2.3.1. Диагностика неисправности с помощью службы `setroubleshoot`

В операционных системах Fedora и RHEL присутствуют дополнительные инструменты, которые помогают нам выполнять диагностику произошедших отказов. Эти инструменты работают совместно для обнаружения какого-либо случившегося запрещения, ищут возможные решения и информируют администратора об отказе и предлагаемых вариантах разрешения ситуации.

Когда используется графический интерфейс на рабочей станции, то блокировки могут даже приводить к возникновению всплывающих окон, которые запрашивают у администратора незамедлительное рассмотрение. Установите программный пакет `setroubleshoot` для получения такой поддержки. На серверах без графического интерфейса администраторы могут видеть информацию в системных журналах или даже конфигурировать систему так, чтобы она отправляла сообщения об отказах SELinux по электронной почте. Для этого вам надо установить пакет `setroubleshoot-server`.

В сущности, именно фоновая программа аудита запускает приложение, распределяющее события безопасности (auditd). Это приложение поддерживает интегрируемые программные модули (плагины), что, собственно, разработчики SELinux и использовали с большим удовольствием. Они написали одну программу, названную sedispatch, которая была призвана действовать как некий плагин для auditd. Приложение sedispatch проверяет, действительно ли контролируемое событие безопасности относится к числу отказов, формируемых системой защиты SELinux, и если это так, то перенаправляет его системе межпроцессной передачи данных D-Bus. Уже от нее событие перенаправляется приложению setroubleshoot (в т. ч. запускает его, если не было ранее запущено), которое выполняет анализ этого отказа и затем формирует ответ для администратора.

Когда запущена рабочая станция с графическим интерфейсом пользователя, небольшая утилита с названием seapplet выведет на экран администратора вот такое всплывающее окно:

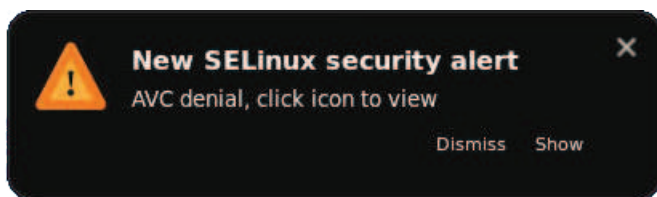
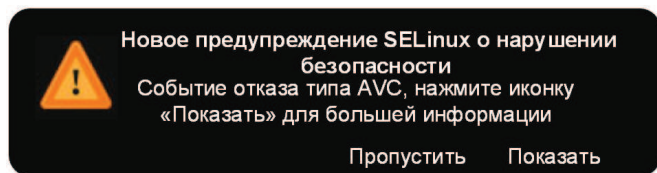


Рис. 2.2 ❖ Пример всплывающего окна SELinux, сообщающего о возникновении угрозы безопасности

**P** Для русскоязычного варианта данное окно могло бы быть представлено следующим видом, если бы программисты придерживались на этот счет мнения переводчика данной книги:



Администратор может выбрать возможность **Показать** (*Show*) для получения больших деталей. Но, кстати, эти детали доступны для чтения даже без графического интерфейса. Результаты анализа сохраняются в файловой системе, а сообщения отражаются в системном журнале:

```
Jun 03 10:41:48 localhost setroubleshoot: SELinux is preventing
/usr/sbin/httpd from 'getattr' accesses on the directory
/var/www/html/infocenter. For complete SELinux messages, run
sealert -l 26d2a1c3-a134-452e-c69b-4ef233e20909
```

**P** Июнь 03 10:41:48 диагностика неисправности на локальном узле: SELinux предотвращает доступ программы `/usr/sbin/httpd` к функции получения атрибутов `'getattr'` директории `/var/www/html/infocenter`. Для получения полного сообщения об этом событии запустите команду

```
sealert -l 26d2a1c3-a134-452e-c69b-4ef233e20909
```

Выполнив команду `sealer`, как написано в журнале, мы можем увидеть более полную информацию о случившемся:

```
# sealert -l 26d2a1c3-a134-452e-c69b-4ef233e20909
```

```
SELinux is preventing /usr/sbin/httpd from getattr access on the directory infocenter
```

```
**** Plugin restorecon (99.5 confidence) suggests ****
```

```
If you want to fix the label.
```

```
/var/www/html/infocenter default label should be httpd_sys_content_t.
```

```
Then you can run restorecon.
```

```
Do
```

```
# /sbin/restorecon -v /var/www/html/infocenter
```

```
Additional Information:
```

```
Source Context          system_u:system_r:httpd_t:s0
Target Context          unconfined_u:object_r:user_home_t:s0
Target Objects          infocenter [ dir ]
Source                  httpd
Source Path              /usr/sbin/httpd
Port                    <Unknown>
Host                    <Unknown>
Source RPM Packages     httpd-2.4.6-40.el7.x86_64
Target RPM Packages
Policy RPM               SELinux-policy-3.13.1-23.el7.noarch
SELinux Enabled         True
Policy Type              targeted
Enforcing Mode          Enforcing
Host Name                localhost.localdomain
Platform                Linux localhost.localdomain
                        3.10.0-327.13.1.el7.x86_64 #1
                        SMP Fri Jun 3 11:36:42 UTC 2020 x86_64

Alert Count              2
First Seen               2020-06-03 10:33:21 EDT
Last Seen                2020-06-03 10:41:48 EDT
```

**P** # sealert -l 26d2a1c3-a134-452e-c69b-4ef233e20909

SELinux предотвращает доступ программы `/usr/sbin/httpd` к функции получения атрибутов `getattr` директории `infocenter`

\*\*\*\* Рекомендуется программа `restorecon` (с точностью 99.5) \*\*\*\*

если вы хотите исправить маркировку.

Каталог `/var/www/html/infocenter` должен иметь по умолчанию тип функциональных ограничений `httpd_sys_content_t`.

Если так, то вы можете запустить команду `restorecon` для восстановления параметров безопасности, установленных по умолчанию.

Выполните следующую команду:

```
# /sbin/restorecon -v /var/www/html/infocenter
Дополнительная информация:
Параметр безопасности субъекта доступа: system_u:system_r:httpd_t:s0
Параметр безопасности объекта доступа: unconfined_u:object_r:user_home_t:s0
Объект доступа: infocenter [директория]
Субъект доступа httpd
Полный путь к субъекту доступа /usr/sbin/httpd
Порт <Неизвестен>
Узел <Неизвестен>
Установочный пакет субъекта httpd-2.4.6-40.el7.x86_64
Установочный пакет объекта
Установочный пакет политики SELinux-policy-3.13.1-23.el7.noarch
SELinux включен Истина
Тип политики целевой (targeted)
Режим защиты Принудительный (Enforcing)
Имя узла localhost.localdomain
Платформа Linux localhost.localdomain
3.10.0-327.13.1.el7.x86_64 #1
SMP Fri Jun 3 11:36:42 UTC 2020 x86_64

Количество предупреждений 2
Первое посещение 2020-06-03 10:33:21 EDT
Последнее посещение 2020-06-03 10:41:48 EDT
```

Приложение `sealert` является программой, запускаемой в командной строке, которая разбирает информацию, сохраненную фоновым процессом `setroubleshoot` (в каталоге `/var/lib/setroubleshoot`).

Оно будет предоставлять нам некий набор возможностей для разрешения проблем, связанных с отказами в доступе. В вышепоказанном случае отказа, связанного с веб-сервером Apache, `sealert` предоставил нам один из возможных вариантов с количественной оценкой правильности предположения. В зависимости от проблемы может быть показано не одно, а множество решений.

Как мы можем видеть в данном примере, приложение `setroubleshoot` само использует подключаемые модули для анализа возникших отказов доступа. Эти модули (распространяемые в рамках пакета `setroubleshoot-plugins`) рассматривают какой-либо отказ на принадлежность к какому-нибудь конкретному общеизвестному случаю использования (например, когда требуется изменить логические значения или параметры безопасности объекта доступа неправильные) и предоставляют ответ приложению `setroubleshoot` о том, с какой *точностью* подключаемый модуль определил, что данная проблема с отказом может быть решена, рекомендованным им методом.

### 2.3.2. Отправка электронной почты, когда случился отказ SELinux

Когда какая-то система является хорошо настроенной и отказы долго не случаются в порядке вещей, тогда администратор может предпочесть, чтобы про-



грамма `setroubleshootd` отправляла сообщения на электронную почту, когда происходит подобная неприятность. Это, несомненно, предоставляет прекрасные возможности для определения/предотвращения вторжения на узел, защищенный SELinux, поскольку администраторам нет необходимости постоянно просматривать свои журналы в поисках данной информации.

Откройте конфигурационный файл `/etc/setroubleshoot/setroubleshoot.conf` в текстовом редакторе вроде `vi` и найдите секцию `[email]`. И обновите в ней информацию в соответствии с используемой почтовой инфраструктурой:

```
# vi /etc/setroubleshoot/setroubleshoot.conf
...
[email]
recipients_filepath = /var/lib/setroubleshoot/email_alert_recipients
smtp_port = 25
smtp_host = localhost
from_address = SELinux@infra.example.com
subject = [infra] SELinux Alert for host infra.example.com
```

Затем отредактируйте файл `email_alert_recipients` (определяемый в переменной `recipients_filepath`) и добавьте адрес электронной почты, который необходимо использовать для уведомлений о возникновении предупреждений SELinux.

В завершение перезагрузите программу межпроцессного взаимодействия D-Bus (потому что `setroubleshootd` управляется посредством этой системы) следующей командой:

```
# systemctl restart dbus
```

Когда же работаете без подсистемы инициализации `systemd`, то используйте вместо этой команды другую:

```
# service dbus restart
```

### 2.3.3. Использование утилиты `audit2why`

Если приложение `setroubleshoot` и команда `sealert` отсутствуют в дистрибутиве Linux, мы все равно можем получить некоторую информацию о случившемся отказе. Хотя это и не так масштабируемо, как посредством подключаемых модулей, предлагаемых в `setroubleshoot`, но утилита `audit2why` (которая является сокращением утилиты `audit2allow`, запущенной с ключом `-w`, и предоставляется в рамках пакета `polyscoreutils-python` операционной системы RHEL) все же дает некоторую ответную реакцию на случившийся отказ. Правда, эта реакция может оказаться не всегда верной в сделанном заключении о происшедшем.

Например, так мы попробовали использовать эту утилиту для того же самого случая отказа, для которого ранее запускали команду `sealert`:

```
# ausearch -m avc -ts today | audit2why
type=AVC msg=audit(1371204434.608:475): avc: denied { getattr }
```

```
for pid=1376 comm="httpd" path="/var/www/html/infocenter"
dev="dm-1" ino=1183070 scontext=system_u:system_r:httpd_t:s0
tcontext=unconfined_u:object_r:user_home_t:s0 tclass=dir
```

Was caused by:

The boolean `httpd_read_user_content` was set incorrectly.

Description:

Determine whether httpd can read generic user home content files.

Allow access by executing:

```
# setsebool -P httpd_read_user_content 1
```

**P** Для русскоязычного варианта данное сообщение могло бы быть представлено следующим видом, если бы программисты придерживались на этот счет мнения переводчика данной книги:

```
# ausearch -m avc -ts today | audit2why
тип=AVC msg=audit(1371204434.608:475): avc: отказ { getattr }
  для процесса с pid=1376, связанного с "httpd", путь="/var/www/html/infocenter"
  устройство="dm-1", номер узла файла в ФС=1183070, параметры безопасности субъекта
  доступа=system_u:system_r:httpd_t:s0
  параметры безопасности объекта=unconfined_u:object_r:user_home_t:s0, класс объекта=dir
```

Причиной сообщения об отказе стало:

Параметр `httpd_read_user_content` был установлен некорректно.

Описание:

Определить, действительно ли процесс `httpd` может читать основные файлы, содержащиеся в домашнем каталоге пользователя.

Предоставьте доступ, выполнив следующую команду:

```
# setsebool -P httpd_read_user_content 1
```

В этом примере утилита `audit2why` не содержит сведений о том, что параметры безопасности объекта доступа были неправильные, и советует нам разрешить для веб-сервера чтение содержимого каталога пользователя.

### 2.3.4. Взаимодействие с журналом `system`

Наряду с системой контроля событий безопасности (аудита) в Linux, которую мы используем для большинства регистрируемой информации, и событий от SELinux информация может быть также собрана при помощи других систем регистрации. Например, журнал подсистемы инициализации `systemd` учитывает информацию о параметрах безопасности SELinux с событиями и позволяет администратору использовать эти сведения, когда он обращается к журналу.

Может быть использована следующая команда, для того чтобы посмотреть события в журнале `systemd-journal`, которые являются сгенерированными каким-либо приложением, связанным с параметрами безопасности `user_u:user_r:user_t:s0`:

```
# journalctl _SELINUX_CONTEXT="user_u:user_r:user_t:s0"
-- Logs begin at Fri 2020-08-05 03:12:39 EDT, end at Fri
2020-08-05 05:46:36 EDT. --
Aug 05 04:31:25 SELinuxtest su[11586]: pam_unix(su-l:auth):
```

```

authentication failure; logname=lisa uid=1001 euid=0 tty=pts/0
ruser=lisa rhost= user=root
Aug 05 04:31:25 SELinuxtest su[11586]: pam_succeed_if(su-l:auth):
requirement "uid >= 1000" not met by user "root"
Aug 05 04:31:27 SELinuxtest su[11586]: FAILED SU (to root)
lisa on pts/0

```

В связи с тем, что `systemd-journal` добавляет параметры безопасности SELinux для порождающего события приложения, вредоносным программам значительно сложнее сгенерировать поддельные события (*fake events*). В то время как обычные системы регистрации просто учитывают строку события, `systemd-journal` извлекает еще и параметр безопасности SELinux из системы.

Создание поддельных событий (например, об аутентификации, порождаемые каким-либо сервисом, который на самом деле не имеет подсистемы аутентификации) тем не менее возможно. Но, используя параметры безопасности SELinux, можно легко группировать события по приложениям и таким образом иметь хорошую гарантию, что конкретные события действительно были сформированы каким-то конкретным приложением.

Когда установлен пакет `bash-completion`, выполняющий автоматическое дополнение команд в терминале по нажатию кнопки **ТАВ**, мы можем даже использовать его для просмотра тех параметров безопасности SELinux, которые уже присутствуют в журнале, что делает формирование запросов к журналу более простым:

```

# journalctl _SELINUX_CONTEXT=<tab><tab>
system_u:system_r:audisp_t:s0 system_u:system_r:rpcd_t:s0
system_u:system_r:auditd_t:s0 system_u:system_r:sshd_t:s0-s0:c0.c1023
...
system_u:system_r:rhnsd_t:s0 user_u:user_r:user_t:s0

```

Но `systemd-journal` пошел дальше, чем просто извлечение параметров безопасности SELinux. Имеется хорошее взаимодействие между подсистемой регистрации `systemd-journal` и службой диагностики неисправностей `setroubleshootd` (о которой мы говорили ранее). Когда отлаживается некая проблема, связанная, скажем, с веб-сервером Apache, мы можем запросить командой `journalctl` показать все события, связанные с исполняемым файлом `httpd`, – и результат вывода будет включать события SELinux, записанные также и службой диагностики неисправностей.

### 2.3.5. Использование здравого смысла

Здравый смысл не просто описывать, но изучение случившегося отказа часто приводит к правильному решению, когда мы имеем некоторый опыт в работе с маркировкой файлов параметрами безопасности. Если мы посмотрим на предыдущий пример с отказом в доступе (где речь о каталоге `/var/www/html/infocenter`), то увидим, что его параметр `user_home_t` должен, как говорят англичане, «звонить в колокол» (т. е. напоминать нам о чем-то). Этот параметр используется для файлов домашнего каталога конечного пользователя, но никак

не для системных файлов внутри каталога `/var`, содержащего файлы, которые изменяются в ходе работы системы.

Один из способов удостовериться в том, что параметр безопасности объекта является корректным, заключается в том, чтобы проверить его с помощью команды `matchpathcon` (предоставляемой в рамках пакета `libSELinux-utils` для операционной системы RHEL или с пакетом `sys-libs/libSELinux`, если для ОС Gentoo). Данная утилита возвращает параметры безопасности такие, какими они должны быть в соответствии с политикой:

```
# matchpathcon /var/www/html/infocenter
/var/www/html/infocenter
system_u:object_r:httpd_sys_content_t:s0
```

Выполнение этой команды для событий отказов, связанных с файлами или директориями, может помочь в обнаружении правильного решения проблемы более быстро.

К тому же многие допустимые в SELinux наборы функций (домены) для субъектов доступа имеют определенные страницы описаний в руководствах, которые информируют читателя о допустимых для объектов типах SELinux, обычно использующихся для каждого домена, а также о том, как иметь дело с доменом в деталях (к примеру, доступные элементарные настройки, характерные ошибки и т. п.). Эти страницы руководства вызываются с указанием наименования субъекта и добавления к нему окончания `_SELinux`:

```
$ man ftpd_SELinux
```

Во многих случаях метод управления случившимися отказами может быть описан следующей последовательностью анализа:

- 1) имеет ли объект доступа метку безопасности SELinux (такую как у файла), единственно правильную? Проверьте это при помощи команды `matchpathcon` или сравните с метками подобных объектов, которые не блокируются системой защиты;
- 2) имеет ли субъект доступа маркировку (принадлежит ли домену), которая должна быть? Какая-нибудь программа, обрабатывающая в фоне SSH-соединения, должна быть запущена в домене `sshd_t`, а не в домене `init_t`. Если это не так, то сделайте, чтобы маркировка соответствующего приложения (его исполняемого файла) являлась корректной (снова можно использовать команду `matchpathcon` для этого);
- 3) не является ли случившийся отказ чем-то, что может быть результатом выбранных установок в системе? Могут быть установлены какие-нибудь логические настройки защиты (*SELinux boolean*) для разрешения определенного правила. Это будет отражено в таком случае программой `setroubleshootd`, и обычно страницы описания домена (такого как `httpd_SELinux`) будут также включать сведения о возможностях логических настроек SELinux. Сведения о таких настройках приводятся в главе 8 «Работа с политиками SELinux».

Изменения маркировки файла будут обсуждаться более детально в главе 4 «Домены процессов и контроль доступа на уровне файлов».

## 2.4. ЗАКЛЮЧЕНИЕ

В этой главе мы рассмотрели, как включать и выключать защиту SELinux и на полностью системном уровне, и на уровне сервисов, используя различные методы: настройки загрузки ядра, конфигурационные файлы SELinux, а также консольные команды. Одной из таких команд является `semanage permissive`, которая может отключать защиту SELinux и для отдельных сервисов.

Затем мы разобрали, где SELinux регистрирует свои события и как интерпретировать их, учитывая, что они являются одной из наиболее важных возможностей администратора по взаимодействию с SELinux. В помощь нам для этой интерпретации существуют такие инструменты, как `setroubleshoot`, `sealert` и `audit2why`. Также мы погрузились в некоторые утилиты, связанные с контролем событий безопасности Linux, для того чтобы помочь нам обработать необходимые события из их общего множества.

В следующем разделе мы будем рассматривать первостепенную задачу администрирования в SELinux-системах: управление учетными записями пользователей, их связывание с ролями SELinux и проведение расчетов защиты для ресурсов системы.

# Глава 3

## Управление учетными записями пользователей

Когда мы входим в систему с включенным SELinux, нам для дальнейшей работы назначаются некие стандартные параметры безопасности. Эти параметры включают в себя сведения о соответствующем пользователе, зарегистрированном в системе защиты SELinux, о разрешающей роли, наименовании типа, определяющего функциональные ограничения и мандатную конфигурацию.

В этой главе мы будем:

- определять пользователей, имеющих достаточные права для выполнения своей работы, разделяющихся на обычных пользователей со строгими мерами защиты и на полностью привилегированных, административных пользователей с небольшим набором ограничений SELinux;
- создавать и назначать категории и мандатные конфигурации;
- назначать пользователям роли и использовать разные инструменты для смены ролей.

Мы закончим главу изучением того, как SELinux интегрируется с процессом аутентификации Linux.

### 3.1. ПАРАМЕТРЫ БЕЗОПАСНОСТИ ПОЛЬЗОВАТЕЛЕЙ

Некогда зарегистрированный в системе пользователь будет работать внутри системы с определенными параметрами безопасности. Эти параметры определяют права и привилегии, которые пользователь имеет в системе. Команда для вывода информации о текущем пользователе – `id` – также может предоставить сведения о параметрах безопасности пользователя в SELinux.

```
$ id -z
unconfined_u:unconfined_r:unconfined_t
```

В системах SELinux с предустановленными политиками (по умолчанию размещаемыми в каталоге `targeted`) велики шансы, что все пользователи окажутся связанными с типом учетной записи SELinux, имеющей минимальные ограничения, – `unconfined_u` (см. первую часть параметров). В более строгих случаях

пользователь может быть связан с типом записи `user_u` (определяющей обычных пользователей с ограниченными правами), с типом `staff_u` (предназначенной для операторов), с типом `sysadm_u` (для системных администраторов) или с каким-либо еще из числа созданных пользовательских типов SELinux. По умолчанию в системе доступно фиксированное число типов пользователей SELinux, но администраторы могут создавать дополнительные пользовательские типы. В задачи администратора также входит выполнение соотношения имени пользователя, заведенного в Linux с пользовательским типом в системе защиты.

Кроме этого, конкретному пользователю SELinux могут быть назначены различные роли, которые пользователь может менять. Эти роли обозначают некоторые разрешенные действия, которые доступны пользователю. При этом надо учитывать, что роли SELinux не могут быть созданы посредством команд администратора. Для того чтобы их можно было создать, политика SELinux должна содержать дополнительные правила, которые и создают роли. Для того чтобы вывести список заданных текущих ролей, используется команда `seinfo`:

```
# seinfo --role
Roles: 14
  auditadm_r
  dbadm_r
  ...
  unconfined_r
```

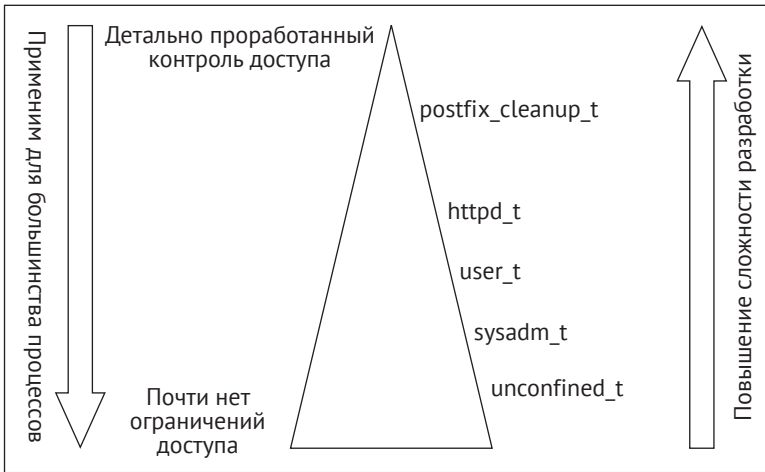
Перед тем как рассматривать пользователей SELinux и их роли, давайте обратим внимание на различные уровни сложности, которые политики могут предоставлять с того момента, когда они будут направлять наш выбор прав пользователя SELinux и затем ролей. Например, они будут помогать нам проводить различие между пользователем с широкими правами (таким как `sysadm_u`) и пользователем с узкими функциональными возможностями (таким как `dbadm_u`).

### 3.1.1. Сложность допустимого набора функций

Система защиты SELinux способна предоставить полную функционально-замкнутую среду системы: любое и каждое приложение запускается в собственном ограниченном окружении, которое оно не может преодолеть и выйти за его пределы. Но это требует детально проработанных политик, которые нужно разрабатывать настолько быстро, насколько быстро выходят новые версии приложений, которые они призваны ограничивать.

Следующая диаграмма показывает отношение в рамках политики применимости допустимого набора функций к большому числу процессов и сложности разработки таких наборов. В качестве примера набор функций с именем `postfix_cleanup_t` показан на диаграмме как детально проработанный в рамках политики безопасности и используется для процесса очистки для инфраструктуры почтовой системы *Postfix*, в то время как набор с именем `unconfined_t` рас-

смотрен в примере как очень широкая, почти неограниченная область функциональных возможностей.



**Рис. 3.1** ❖ Взаимосвязь между сложностью разработки наборов функциональных возможностей и тщательностью проработки в них контроля доступа

Сложность политики можно примерно разделить на несколько категорий следующим образом:

- хорошо детализированная политика, с отдельными допустимыми наборами функций для отдельных приложений и команд;
- политика на уровне приложения;
- политика широкого применения, используемая для множества различных приложений, реализующих схожую функциональность;
- укрупненные политики, в том числе включающие неограниченный доступ к системе.

Давайте рассмотрим данные категории политик более подробно.

1. **Хорошо детализированные политики** (*fine-grained policies*) – имеют явное преимущество, так как они действительно нацелены на ограничение приложений настолько, насколько это возможно сделать в принципе. При помощи них роли, разработанные для пользователей и администраторов, тоже становятся хорошо продуманными и детализированными. Недосток таких политик в том, что они сложны для поддержания в связи с тем, что требуют немедленного обновления вместе с развитием самого приложения. Политикам также необходимо учитывать влияние различных опций конфигурации, поддерживаемых приложением, которое требуется ограничить:

Такие детально проработанные политики нечасто встречаются. Примером может являться набор политик, предусмотренный для инф-



раструктуры почтовой системы *Postfix*. Каждая подсистема инфраструктуры *Postfix* имеет свой собственный набор функциональных ограничений.

2. **Политики на уровне приложений** (*application-level policies*). Эти политики используют один допустимый набор функций для некоего приложения, независимо от числа его подсистем. Они обеспечивают баланс между требованием замкнутости работы приложения и возможностью поддержки развития приложения. Такие политики, работающие на уровне приложений, наиболее часто встречаются в большинстве распространяемых политик.
3. **Политики широкого применения** (*category-wide policies*). Данные политики используют один допустимый набор функций для целого множества приложений, которые реализуют схожую функциональность. Они наиболее часто используются для сервисов, которые действуют подобно друг другу и для которых роли пользователей могут не учитывать различия, имеющиеся между этими сервисами:  
 Популярным примером политики широкого применения является политика для веб-серверов. Изначально написанная для сервиса Apache HTTP, политика стала повторно использоваться для других веб-серверов, предоставляемых в рамках проектов Cherokee, Hiawatha, Nginx и Lighttpd.
4. **Укрупненные политики** (*coarse-grained policies*). Такие политики используются для приложений или служб, чье поведение очень сложно формализовать. Функциональные возможности конечных пользователей – хороший тому пример как неограниченного набора допустимых возможностей без ярко выраженных запретов.

### 3.1.2. Определение неограниченных доменов

Широкие возможности в изменении сложности политик приводят к появлению различных моделей политик безопасности, созданных для разных дистрибутивов Linux и поддерживаемых тоже только для них. Red Hat Enterprise Linux, например, фокусирует свое внимание на политиках, где службы с сетевым взаимодействием ограничиваются, а пользовательская активность – в основном нет (при этом данные политики размещаются в основном каталоге `targeted`, используемом по умолчанию). Неограниченная активность достигается при помощи назначения следующих параметров безопасности конечному пользователю: значение `unconfined_u` для типа пользователя в SELinux, `unconfined_r` для назначаемой ему роли и `unconfined_t` для типа допустимых функций.

Кроме неограниченных типов для пользователя, мы также имеем неограниченные домены (*domains*) (те же допустимые наборы функций, только применительно к процессам), для фоновых процессов (*daemons*) и других приложений. Как правило, ряд из них запускается с допустимым набором функций без

ограничений – `unconfined_t`, но большинство из них работают в своем собственном домене, даже если они и не ограничены в действиях.

Чтобы выяснить, обладает ли домен неограниченными возможностями, мы можем сделать запрос к политике SELinux на получение данных о тех доменах, которые имеют какой-либо SELinux-атрибут, связанный с неограниченным набором допустимых функций. Эти SELinux-атрибуты предоставляют нам возможность группировать типы SELinux и присваивать этим группам права доступа. Общий тип атрибута, определяющего неограниченность функциональных возможностей, называется `unconfined_domain_type`. И мы можем сформировать запрос на предоставление сведений о том, какие типы допустимого набора функций в параметрах безопасности SELinux имеют этот атрибут, используя команду `seinfo`:

```
# seinfo -aunconfined_domain_type -x
unconfined_domain_type
  sosreport_t
  bootloader_t
...
```



Администраторы не могут переводить неограниченные домены в число тех, которые имеют какие-либо ограничения.

## 3.2. Пользователи SELinux и их роли

Когда в системах с установленной защитой SELinux выполняется пользователем вход, подсистема идентификации и аутентификации соотносит имя пользователя с типом пользователя в SELinux. Затем, когда тип пользователя в SELinux найден, система определяет роль и тип допустимого набора функций для пользователя и устанавливает их в качестве параметров безопасности пользователя.

### 3.2.1. Перечень сопоставлений пользователей с пользовательскими типами SELinux

После того как вход в операционную систему выполнен, мы можем использовать команду `id -Z` для получения текущих параметров безопасности пользователя. Для многих пользователей эти параметры будут такими: `unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023` независимо от их имени пользователя. Если это не так, тогда параметры будут начинаться с одного из следующих типов пользователей: `sysadm_u`, `staff_u` или `user_u`. Это потому, что большинство дистрибутивов Linux по умолчанию предоставляют только ограниченное число типов пользователей SELinux, привязанных к конкретным ролям, которые эти дистрибутивы поддерживают.

Когда процесс входа в систему уже запущен, то для проверки соответствия учетной записи Linux типу пользователя SELinux будет применен некий локально размещенный файл. Давайте посмотрим, используя команду `semanage login -l`, как сопоставляются существующие имена пользователей. Представ-

ленный ниже экранный вывод данных является результатом работы данной команды для настроек по умолчанию в системе RHEL.

```
# semanage login -l
```

Login Name	SELinux User	MLS/MCS Range	Service
__default__	unconfined_u	s0-s0:c0.c1023	*
root	unconfined_u	s0-s0:c0.c1023	*
system_u	system_u	s0-s0:c0.c1023	*

В результате работы команды сопоставление имен и типов выводится построчно. Каждая строка содержит следующие элементы, разделенные по колонкам:

- Login Name – имя пользователя Linux, для которого выполняется сопоставление;
- SELinux User – тип пользователя SELinux, с которым сопоставляется имя пользователя;
- MLS/MCS Range – диапазоны уровней и категорий, с которыми сопоставляется имя пользователя;
- Service – служба, для которой применяется сопоставление (этот параметр используется для локальных настроек, которые мы рассмотрим позже).

Имя пользователя может содержать некоторые особые значения, которые не соотносятся напрямую с отдельными учетными записями Linux:

- 1) \_\_default\_\_ – это универсальное правило. Если никакое другое правило не подходит, тогда пользователи соотносятся с пользователями SELinux в соответствии с тем, как указано в данной строке. В приведенном выше примере все пользователи соотносятся в SELinux с пользовательским типом unconfined\_, означающим, что обычные пользователи Linux являются едва ли ограниченными в своих действиях. Когда это не приветствуется, администраторы обычно привязывают имена обычных пользователей к ограниченным пользовательским типам SELinux, в то время как административные учетные записи – к таким типам, как staff\_u или sysadm\_u;
- 2) запись в колонке имен пользователей, предваряемая символом процента %, предлагает связь с группой. Это позволяет администраторам напрямую сопоставлять какую-либо группу людей с некоторым типом SELinux-пользователя, что удобнее, чем делать то же самое, но для каждого индивидуально;
- 3) значение system\_u в строке параметров связано с системными процессами (не для учетных записей системы Linux, интерактивно входящих в систему). Оно никогда не назначается конечным пользователям.

**i** В случае когда сопоставление с различными типами SELinux-пользователей выполнено для конкретного пользователя и для группы, в которую он входит, преимущество имеет то сопоставление, которое относится непосредственно к пользователю. Когда существует несколько определенных групп, куда входит пользователь, тогда использоваться будет сопоставление для первой группы (в том порядке, в котором показывает команда `semanage login`).

Если включена многоуровневая (*MLS*) или многокатегорийная защита (*MCS*), сопоставление включает информацию о значениях мандатной конфигурации доступа, назначенной пользователю и в которой он может работать. Таким образом, два пользователя могут быть оба сопоставлены с одним и тем же ограничивающим в правах типом пользователя SELinux, но одному из них при этом будет позволен доступ на низком уровне мандатного доступа (*s0*), в то время как другой может иметь доступ к данным более высокого уровня защиты (например, *s1*). При этом пользователи могут еще различаться и по категориям доступа.

### 3.2.2. Сопоставление учетных записей с пользовательскими типами

Давайте используем несколько примеров, для того чтобы показать, как работают эти сопоставления. Предположим, у нас есть пользователь в системе Linux с именем *lisa*, и мы хотим, чтобы ее учетная запись была сопоставлена с типом SELinux-пользователя *staff\_u*, тогда как остальные пользователи из группы *users* должны быть соотнесены с типом *user\_u*.

Мы можем добиться этого при помощи команды `semanage login`, используя опцию `-a` (что является сокращением от `add` – добавить):

```
# semanage login -a -s staff_u lisa
# semanage login -a -s user_u %users
```

Параметр `-s` используется для назначения типа SELinux-пользователя, а параметр `-r` управляет уровнями и категориями безопасности. Давайте изменим (используя параметр `-m` вместо `-a`) созданное в последнем примере сопоставление для группы *%users* новым сопоставлением с пользовательским типом *staff\_u* и ограничим этих пользователей уровнями безопасности в диапазоне *s0-s0* и категориями от *c0* до *c4*:

```
# semanage login -m -s staff_u -r "s0-s0:c0.c4" %users
```

**i** Диапазоны уровней и категорий сопоставляемых учетных записей не могут превышать диапазоны, которые назначены пользовательскому типу SELinux. Например, если пользовательскому типу *staff\_u* был ранее назначен доступ только к таким уровням и категориям, как *s0-s0:c0.c3*, тогда предыдущая команда закончится неудачей, поскольку она пытается назначить более широкий диапазон доступа. Мы обсудим, как создавать типы пользователей SELinux и диапазоны мандатной конфигурации, позже в этой главе.

Изменения вступают в силу при новом входе в систему, поэтому нам следует инициализировать выход этих пользователей. Следующая команда снимает все работающие процессы пользователя, форсируя его выход из системы:

```
# pkill -KILL -u lisa
```

Также, когда пользователь изменен, нам следует еще переустановить параметры безопасности домашнего каталога этого пользователя (пока он еще не вошел в систему). Для того чтобы это сделать, применяется команда `restorecon`

с опциями `-F` (`force reset` – принудительная перенастройка) и `-R` (`recursively` – рекурсивно), как показано ниже:

```
# restorecon -RF /home/lisa
```

**i** Выполнение этой команды также перенастраивает параметры безопасности файлов, для которых пользователь выполнил настройки вручную, используя такие инструменты, как `chcon`. Поэтому правильнее было бы выполнять сопоставление с типом пользователя SELinux до настроек параметров безопасности файлов или же рекурсивно изменить только пользовательский тип SELinux, используя команду `chcon -R -u`. Более подробный разбор этой команды и параметров безопасности для файлов описан в следующей главе.

Для того чтобы удалить сопоставление учетной записи с пользовательским типом, используйте опцию `-d` (`delete` – удалить), как показано ниже. И не забудьте после этого запустить команду `restorecon`:

```
# semanage login -d lisa
# restorecon -RF /home/lisa
```

### 3.2.3. Настройка учетных записей относительно служб

Когда имена пользователей сопоставляются с типами SELinux-пользователей при помощи добавления командой `semanage login`, то они применяются ко всем службам. В команде управления `semanage` отсутствует возможность, позволяющая настроить сопоставления с учетом какой-нибудь конкретной службы. Тем не менее это не означает, что это невозможно.

Инструменты и библиотеки пользовательского пространства SELinux будут обращаться к двум файлам конфигурации для определения актуальных параметров сопоставлений:

- файл `/etc/SELinux/targeted/seusers` содержит стандартные, не имеющие отношения к службам сопоставления. Этот файл управляется командой `semanage login` и не предназначен для обновления любыми другими средствами;
- каталог `/etc/SELinux/targeted/logins` содержит специализированные сопоставления, размещая в одном файле параметры для одной учетной записи Linux. Таким образом, специализированное сопоставление для пользователя `root` будет размещено по следующему пути: `/etc/SELinux/targeted/logins/root`.

Внутри файлов, предназначенных для специализированного сопоставления, администраторы могут устанавливать для какой-либо службы другой пользовательский тип SELinux, нежели тот, с которым пользователь уже сопоставлен.

Службы, посредством которых пользователь может зарегистрироваться (войти в систему), – это службы PAM (`pluggable authentication module` – подключаемые модули аутентификации).

Например, чтобы пользователя `root` при запуске защищенной шифрованием консоли SSH снабдить ограниченными правами, какие предоставляются через

тип `user_u` (взамен его по умолчанию установленным правам с неограниченными возможностями у пользовательского типа `unconfined_u`), файл со специализированными настройками сопоставления будет содержать следующее:

```
sshd:user_u:s0
```

Запрос параметров сопоставления, выполненный при помощи команды `semanage login`, будет показывать данное специализированное сопоставление следующим образом:

```
# semanage login -l
Login Name      SELinux User      MLS/MCS Range      Service
%users         staff_u           s0-s0:c0.c1023    *
__default__    unconfined_u     s0-s0:c0.c1023    *
root           unconfined_u     s0-s0:c0.c1023    *
system_u       system_u         s0-s0:c0.c1023    *
Local customization in /etc/SELinux/targeted/logins
root           user_u           s0                  sshd
```

Конечно, этому сопоставлению нет необходимости придавать столь радикальный характер. И оно может быть использовано для ограничения диапазона уровней безопасности и категорий (MLS/MCS), с которым пользователь входит в систему. Например, для ограничения категорий до диапазона `s0.c8` (вместо установленного по умолчанию `s0.c1023`) следует использовать следующие указания:

```
sshd:unconfined_u:s0-s0:c0.c8
```

### 3.2.4. Создание типов пользователей SELinux

По умолчанию существует лишь небольшое количество типов пользователей SELinux, с которым могут быть сопоставлены регистрационные имена пользователей Linux. Если мы хотим более гибко контролировать систему через имена пользователей и им сопоставленные типы, то нужно создавать дополнительные типы пользователей SELinux.

Для начала выведем список уже известных на текущий момент типов пользователей SELinux при помощи команды `semanage user -l`:

```
# semanage user -l
SELinux
User      Labeling Prefix  MLS/  MLS/  SELinux Roles
          Prefix  MCS   MCS
          Level Range
guest_u   user    s0     s0     guest_r
root      user    s0     s0-s0:c0.c1023  staff_r sysadm_r system_r unconfined_r
staff_u   user    s0     s0-s0:c0.c1023  staff_r sysadm_r system_r unconfined_r
sysadm_u  user    s0     s0-s0:c0.c1023  sysadm_r
system_u  user    s0     s0-s0:c0.c1023  system_r unconfined_r
unconfined_u user    s0     s0-s0:c0.c1023  system_r unconfined_r
user_u    user    s0     s0     user_r
xguest_u  user    s0     s0     xguest_r
```

На следующем шаге создадим нового пользователя SELinux с помощью команды `semanage user`, используя опцию `-a` (*add* – добавить). И еще нам необходимо предоставить дополнительную информацию системе защиты SELinux о создаваемом типе пользователя, следующего характера:

- мандатный уровень, устанавливаемый по умолчанию (при помощи опции `-L`) для пользователя SELinux. Это уровень безопасности, с которым пользователь начинает работу;
- допустимые ограничения для уровней и категорий (используя опцию `-r`), применяемые к типу пользователя. Это диапазон значений, которые может использовать пользователь;
- роль или роли (опция `-R`), которые могут применяться к пользователю.

В следующем примере мы создаем тип пользователя SELinux `finance_u` с заданными параметрами безопасности:

```
# semanage user -a -L s0 -r "s0-s0:c0.c127" -R user_r finance_u
```

**i** Роли SELinux включаются через тип пользователя SELinux, с которым соотносится учетная запись Linux. Когда администратор хочет предусмотреть поддержку для дополнительных ролей, он или обновляет существующие типы пользователей SELinux, для того чтобы они включали новую роль (новые роли), или создает новый тип пользователя SELinux, который имеет доступ к новой роли (ролям).

Когда тип пользователя SELinux создан, информация о нем становится частью политики SELinux. С этого момента и далее учетные записи Linux могут сопоставляться с этим типом пользователя.

Так же, как в случае с командой сопоставления имен пользователей типам, команда `semanage user` поддерживает опцию `-m` для изменения существующей записи и опцию `-d` для ее удаления. Например, следующая команда удаляет тип пользователя SELinux с именем `finance_u`:

```
# semanage user -d finance_u
```

Отдельные пользовательские типы SELinux улучшают контроль событий безопасности, возникающих в системе в процессе работы, т. к. пользовательские типы не меняются в течение пользовательской сессии, тогда как эффективный идентификатор – *ID* пользователя Linux может меняться. Если пользователь создает файлы или другие объекты, то они также наследуют от него параметры безопасности в части, касающейся пользовательского типа SELinux.

### 3.2.5. Перечень типов допустимого набора функций у ролей

Создавая пользовательские типы SELinux, одним из параметров, которые следует создать, является роль (или роли) для данного типа. Большинство ролей говорят сами за себя: роль `dbadm_r` предназначена для администраторов баз данных (*DBA*), в то время как роль `webadm_r` – для администраторов инфраструктуры веб-приложений. Если роль не понятна из названия или какой-нибудь администратор не уточнил, какие возможности характерны некой роли,

то можно выполнить запрос к политике SELinux для получения большей информации.

Как было написано ранее, роли определяют, какие типы допустимых наборов функций, связанные с этой ролью, доступны пользователям. Мы видели, что команда `seinfo` может показать нам доступные роли, но она может сделать и больше. Она может вывести список типов допустимых наборов функций, доступных для данной роли, используя опцию `-x`:

```
# seinfo -rdbadm_r -x
dbadm_r
  Dominated Roles:
    dbadm_r
  Types:
    qmail_inject_t
    dbadm_t
    ...
    user_mail_t
```

В этом примере пользователи, работающие с ролью `dbadm_r`, являющейся частью их параметров безопасности, будут иметь возможность использовать, например, допустимый набор функций `qmail_inject_t` (тип, применяемый для чтения сообщений электронной почты и передачи их в очередь почтовой системы `qmail`) и `user_mail_t` (тип, используемый для формирования отправляемых по электронной почте сообщений при помощи командно-строчного интерфейса).

Информация, предоставляемая в рамках сведений о «преобладающих ролях» (*dominated roles*), обычно не имеет существенного значения для администраторов. Такое понятие, как преобладающие роли, хотя и поддерживается в ядре SELinux, но не используется в распространяемых политиках Linux. Теоретически они извещают, какие роли от каких унаследованы, но при этом всегда будут показывать только лишь запрошенную роль.

### 3.2.6. Управление категориями

Мандатная конфигурация системы защиты и относящиеся к ней категории доступа определяются числовыми значениями, что прекрасно для компьютера, но не так понятно пользователям. К счастью, утилиты SELinux поддерживают перевод уровней и категорий в читаемые для человека значения, даже когда они продолжают храниться в виде чисел. В результате почти все инструменты, способные выводить параметры безопасности, будут показывать переведенные, а не числовые значения.

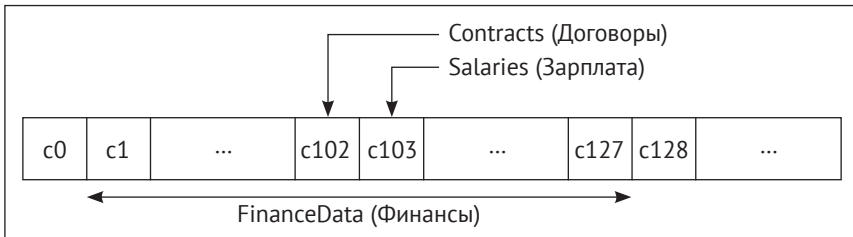
Преобразования числовых значений в читаемые управляются при помощи файла `setrans.conf`, размещаемого в каталоге `/etc/SELinux/targeted`. Внутри этого файла мы можем давать названия определенным значениям (например, `s0:c102`) или диапазонам (вроде таких, как `s0-s0:c1.c127`), указав эти данные в строках файла, что весьма просто для администраторов. Однако, для того чтобы выполненные в файле изменения вступили в силу, необходимо запус-



тить фоновую службу трансляции для компонента многокатегорийной защиты (MCS) – mcstransd. Тем не менее не все дистрибутивы Linux имеют ее установленной по умолчанию. Для операционной системы Red Hat Enterprise Linux требуется первоначально установить пакет mcstrans. Не забудьте запустить его автоматически после установки:

```
# yum install mcstrans
# systemctl enable mcstransd
# systemctl start mcstransd
```

Рассмотрим наш пример с типом SELinux-пользователя `finance_u`, которому был предоставлен доступ в диапазоне категорий `s0.c127`. Двум категориям в рамках этого диапазона мы дадим читаемые наименования: для `c102` – `Contracts` («Договоры»), а для `c103` – `Salaries` («Зарплата»). Диапазон `s0.c127` будет иметь наименование `FinanceData` («Финансы»). Следующая диаграмма показывает отношение между этими разными категориями:



**Рис. 3.2** ❖ Диаграмма «Взаимосвязь между примером отдельных категорий и целым диапазоном»

Для того чтобы это реализовать, надо в файле конфигурации `setrans.conf` поместить следующие строки:

```
s0:c102=Contracts
s0:c103=Salaries
s0-s0:c1.c127=FinanceData
```

**i** После редактирования файла `setrans.conf` служба `mcstransd` должна быть перезапущена.

Преобразования чисел в символьные значения регулируются утилитами SELinux, которые подключаются к фоновой службе `mcstransd` через сокет `.setrans-unix`, расположенный в каталоге `/var/run/setrans`, и выполняют запрос через него к файлу `setrans.conf`. Если служба по каким-то причинам не работает или выполнить взаимодействие с ней не удастся, то информация об уровнях и категориях будет выводиться в числовом виде без преобразований.

Например, при работающей службе вывод по команде `id -Z` сейчас такой:

```
# id -Z
unconfined_u:unconfined_r:unconfined_t:SystemLow-SystemHigh
```

Мы можем увидеть имеющиеся соответствия уровней и категорий их символьным эквивалентам с помощью инструмента `chcat` (являющегося частью пакета `polyscoreutils`, написанного на языке Python для Red Hat Enterprise Linux, или пакета `sysapps/polyscoreutils` в случае с операционной системой Gentoo). Следующий пример показывает перечень соответствующих преобразований после добавления относящихся сведений, которые мы выше задали для финансов:

```
$ chcat -L
s0                SystemLow
s0-s0:c0.c1023   SystemLow-SystemHigh
s0:c0.c1023      SystemHigh
s0:c102          Contracts
s0:c103          Salaries
s0:c1.c127       FinanceData
```

Та же утилита `chcat` может использоваться для назначения категорий пользователям. Например, для назначения категории `Salaries` (предполагаем, что она определена в `setrans.conf`) для пользователя Linux `lisa` нам следует применять следующую команду:

```
# chcat -l -- +Salaries lisa
```



Если для данного пользователя Linux не определен никакой тип пользователя SELinux, то тогда тип назначается автоматически.

В результате выполнения предыдущей команды категория `Salaries` (`c103`) назначается Linux-пользователю `lisa`. Существующие сопоставления пользователя немедленно обновляются с учетом этой информации. Но для того, чтобы изменения вступили в силу, необходимо пользователю `lisa` выйти из системы и войти вновь.

## 3.3. УПРАВЛЕНИЕ РОЛЯМИ SELINUX

Мы уже увидели, что для типов пользователей SELinux определяют конкретные роли, в рамках которых какой-нибудь пользователь может существовать. Но как SELinux выбирает, какую роль надо предоставить пользователю, когда он входит в систему? И как после входа пользователь может переключить одну роль на другую?

### 3.3.1. Настройки присвоения допустимых ролей пользователю

Для выбора параметров безопасности, которыми будет наделен успешно аутентифицированный пользователь, SELinux вводит понятие параметров, устанавливаемых по умолчанию. Основываясь на параметре безопасности той программы, с помощью которой пользователь вошел в систему (или через которую он выполняет команды), выбираются корректные параметры пользователя.

Внутри каталога `/etc/SELinux/targeted/contexts` есть файл `default_contexts`. Каждая строка этого файла начинается с информации о параметрах безопасности SELinux для какого-либо родительского процесса и потом продолжается упорядоченным списком всех параметров безопасности, которые могут быть выбраны, исходя из роли(ей), которая(ые) разрешена(ы) для конкретного пользователя.

Рассмотрим такую строку кода для параметров безопасности, связанных с защищенным шифрованием командной оболочкой SSH:

```
system_r:sshd_t:s0 user_r:user_t:s0 staff_r:staff_t:s0 \
    sysadm_r:sysadm_t:s0 \
    unconfined_r:unconfined_t:s0
```

Эта строка конфигурационного файла указывает, что когда пользователь входит в систему через процесс, работающий в границах функциональных возможностей домена `sshd_t`, перечисленные роли сверяются с ролями этого пользователя. И назначается та роль, которая первая совпадет из имеющегося перечня с фактической ролью пользователя. Вместе с ролью, согласно этому списку, назначается и домен.

К примеру, предположим, мы сопоставили тип пользователя SELinux с ролями `staff_r` и `sysadm_r`. В этом случае мы будем входить в систему с параметрами `staff_r:staff_t`, т. к. это первое найденное соответствие.

Однако, подобно файлу `seusers` с сопоставлениями для имен, зарегистрированных в операционной системе Linux, файл `default_contexts` – это файл, используемый по умолчанию, но который может не учитываться, при наличии особых настроек. Такие настройки хранятся в подкаталоге `/etc/SELinux/targeted/contexts/users/`, и файлы в нем именуются в согласии с идентификаторами типов SELinux-пользователей, для которых они предназначены. Это позволяет нам назначать разные параметры безопасности для определенных типов SELinux-пользователей, даже если они разделяют одни и те же роли с другими типами SELinux-пользователей. А вследствие того, что SELinux проверяет записи настроек построчно, нам не требуется копировать полностью содержание файла `default_contexts`; только конкретные строки для тех служб, которые мы хотим использовать, отличающиеся по конфигурации от заданных по умолчанию.

Давайте изменим параметры безопасности, используемые по умолчанию, таким образом, чтобы учетные записи, сопоставленные с пользовательским типом `dbadm_u`, осуществляли вход в систему с ролью `dbadm_r` (и с типом допустимого набора функций `dbadm_t`) при заходе в систему через SSH. Для этого используем строку для `sshd_t`, но устанавливаем параметры `dbadm_r:dbadm_t:s0` как единственно возможные и сохраняем результат в файле `/etc/SELinux/targeted/contexts/users/dbadm_u`:

```
system_r:sshd_t:s0 dbadm_r:dbadm_t:s0
```

### 3.3.2. Проверка параметров безопасности при помощи утилиты `getseuser`

Чтобы проверить, успешно ли внесены изменения, мы можем запросить SELinux о том, какие параметры безопасности будут установлены в интересующей нас ситуации, без проведения разбора файлов с настройками вручную. Это можно сделать с помощью команды `getseuser`, которая берет (выбирает) два аргумента: имя пользователя Linux и параметры безопасности процесса, с которым собирается взаимодействовать пользователь.

**i** Команда `getseuser` – это вспомогательная утилита, предлагаемая в рамках проекта пользовательского пространства SELinux, но недоступная во всех дистрибутивах. Пользователи Red Hat Enterprise Linux, например, будут напрасно искать команду `getseuser`.

Приведем здесь пример, который показывает, как можно выполнить проверку того, какие будут параметры безопасности для пользователя `lisa`, когда она регистрируется, входит в систему, используя запущенный процесс, с функциональными возможностями типа `sshd_t`:

```
# getseuser lisa system_u:system_r:sshd_t
seuser: dbadm_u, level_(null)
Context 0      dbadm_u:dbadm_r:dbadm_t
```

Одним из преимуществ команды `getseuser` является ее возможность запрашивать настройки SELinux, которые присутствуют не только в файле `default_contexts` и файлах с настройками для типов пользователей SELinux, а также то, что она проверяет, могут ли предустановленные политикой параметры безопасности пользователя быть согласованы с разрешенными параметрами из списка и нет ли других препятствий, запрещающих изменение на необходимые для работы параметры безопасности.

### 3.3.3. Подключение ролей с помощью команды `newrole`

После того как пользователи успешно прошли аутентификацию и вошли в систему, им должны быть назначены параметры безопасности согласно конфигурации, описанной выше. Если же пользовательский тип SELinux при этом имеет доступ ко множеству ролей, тогда пользователь Linux может использовать программу `newrole` для переключения между этими ролями.

Рассмотрим систему SELinux без неограниченных (*unconfined*) типов допустимых наборов функций и такую, где мы по умолчанию регистрируемся с ролью `staff_r`. Допустим, что для выполнения каких-то администраторских задач нам надо переключиться на роль `sysadm_r`, что мы и можем сделать с помощью команды `newrole` (являющейся частью пакета `policycoreutils-newrole` в Red Hat Enterprise Linux или пакета `sys-apps/policycoreutils` в операционной системе Gentoo). Эта команда выполняется, только когда работает через какой-нибудь защищенный терминал из числа размещенных в каталоге `/etc/securetty`:

```
$ id -Z
staff_u:staff_r:staff_t
$ newrole -r sysadm_r
Password:
$ id -Z
staff_u:sysadm_r:sysadm_t
```

Заметьте, что тип SELinux-пользователя при этом остался неизменным, а роль и тип допустимого набора функций (домен) изменились.

Команда `newrole` может также быть использована для перехода к специфическим уровню и категории доступа, подобным этим:

```
$ newrole -l s0-s0:c0.c100
```

Когда мы переключаемся на другую роль или мандатную конфигурацию, задействуется новая сессия (с новой командной оболочкой). При этом не изменяются параметры безопасности текущей сессии пользователя, и выход из нее тоже не выполняется. И мы можем вернуться из назначенной новой роли или перейти обратно к первоначальной сессии, задействуя функции выхода (например, с помощью команд `exit`, `logout` или комбинации клавиш **Ctrl+D**).

### 3.3.4. Управление доступом к роли с помощью команды `sudo`

Большинство администраторов используют команду `sudo` для передачи привилегий: разрешая пользователям выполнять определенные команды при более привилегированных параметрах, чем позволено пользователю. Приложение `sudo` также способно управлять ролями и типами SELinux.

Мы можем получить необходимые роль и тип с помощью `sudo` напрямую. Например, мы можем сказать `sudo`, что надо переключиться на роль администратора баз данных, когда решили отредактировать файл конфигурации системы управления базами данных PostgreSQL:

```
$ sudo -r dbadm_r -t dbadm_t vim /etc/postgresql/pg_hba.conf
```

Вместе с тем мы можем настроить `sudo` через файл `/etc/sudoers`, чтобы разрешить пользователям выполнять отдельные команды в рамках определенной роли или типа. А также получить командную оболочку с определенными параметрами безопасности. Рассмотрим некоего пользователя, который имеет доступ к двум ролям `user_r` и `dbadm_r` (из них роль `dbadm_r` предназначена для администратора базы данных). Включенная в текст файла `sudoers` нижеприведенная строка позволяет пользователю `myuser` выполнять любую команду через `sudo`, которая при запуске будет выполняться с ролью `dbadm_r` и типом допустимого набора функций `dbadm_t`:

```
myuser ALL=(ALL) TYPE=dbadm_t ROLE=dbadm_r ALL
```

Зачастую `sudo` предпочтительнее `newrole`, поскольку большинство операций, для которых нам нужна другая роль, требуют переключения эффективных идентификаторов `-IDs` (связанных с учетной записью `root` или с ориентиро-

ванным к конкретной службе специализированным пользователем) – в любом случае. Программа `sudo` также имеет великолепные возможности по регистрации событий, и мы даже можем использовать команды, переключающие роли без необходимости конечному пользователю четко указывать целевые роль и тип допустимого набора функций. К сожалению, данная команда не поддерживает изменения мандатной конфигурации.

### 3.3.5. Переключение параметров безопасности посредством `runcon`

Другим приложением, позволяющим переключать роли и мандатную конфигурацию, является программа `runcon`. Она доступна всем пользователям и предназначена для запуска какой-либо заданной команды, но с другой ролью, типом допустимого набора функций и/или мандатной конфигурацией. Она даже поддерживает возможность изменения типа SELinux-пользователя – предполагая, что политика SELinux вам это позволит.

Команда `runcon` не имеет своего собственного типа допустимых функций – а запускается с параметрами безопасности пользователя, который запустил ее. Поэтому любые изменения в роли, типе, мандатной конфигурации или даже пользовательском типе SELinux управляются привилегиями пользователя.

В большинстве своем `runcon` используется для запуска приложений с какой-то определенной категорией доступа. Эта программа позволяет пользователям иметь преимущество многокатегорийного (MCS) подхода в SELinux без требования, чтобы в их приложениях была включена поддержка MCS.

Например, для того чтобы запустить командную оболочку с доступом к данным, категоризованным как `Salaries`, сделайте следующее:

```
$ runcon -l Salaries bash
$ id -Z
unconfined_u:unconfined_r:unconfined_t:Salaries
```

### 3.3.6. Переключение на системную роль

Иногда администраторам необходимо задействовать приложения, которые не должны работать с параметрами безопасности текущего типа SELinux-пользователя, а вместо этого запускаться с типом `system_u` и ролью `system_r`.

Эта необходимость признается администраторами политик SELinux, которые позволяют какому-то очень ограниченному числу типов допустимых наборов функций (доменов) переключать один тип SELinux-пользователя на другой тип – возможно, даже вопреки упомянутой ранее цели сохранять неизменность сопоставления типов SELinux-пользователей по отношению к зарегистрированным пользователям Linux.

Все-таки надо признать, что есть такие случаи, когда это необходимо, и системе SELinux придется учесть такую потребность. Одним из приложений, позволяющих переключение типов SELinux-пользователей, является `run_init`.

Приложение `run_init` используется в основном (почти всегда) для запуска фоновых системных служб в Linux. Применяя это приложение, фоновые процессы работают не с параметрами безопасности SELinux-пользователя, а с системными параметрами, как того требуют политики SELinux.

Поскольку это необходимо только на системах, где запуск дополнительных сервисов выполняется посредством сервисных скриптов, то дистрибутивы, использующие для инициализации системных процессов `systemd`, не требуют применения `run_init`. То есть `systemd` уже работает с ролью `system_r` и отвечает за запуск дополнительных служб. Поэтому изменение роли не требуется. Другие системы инициализации, наподобие Gentoo's OpenRC, используют `run_init` таким образом, что администраторам в основном не нужно обращаться к `run_init` вручную.

Еще может возникнуть ситуация, когда это потребуется, так что давайте запустим служебный скрипт с помощью `run_init` и убедимся, что он и в самом деле работает с правами типа SELinux-пользователя `system_u`:

```
# run_init /etc/rc.d/init.d/mcstrans start
# ps -Z $(pidof mcstransd)
system_u:system_r:setrans_t 7972 ? Ss 0:00 mcstransd
```

Большинство политик SELinux содержат поддержку выборочного управления службами, согласуя ее с ролями (не для `systemd`-дистрибутивов). Это позволяет пользователям, не имеющим полных системных административных прав, все-таки манипулировать конкретными службами в Linux, если это разрешено политикой SELinux.

Этим пользователям должна быть предоставлена роль `system_r`, но как только это будет выполнено, им больше не понадобится вызывать команду `run_init` для управления конкретными службами. Преобразования происходят автоматически и только для сервисов, которые являются назначенными для конкретного пользователя, другие службы не могут быть запущены этими пользователями.

Для связи роли `system_r` с типом SELinux-пользователя `finance_u` сначала определим, какая роль назначена в настоящий момент, а потом изменим имеющуюся роль на `system_r`:

```
# semanage user -l
...
finance_u user s0 s0 user_r
# semanage user -m -R user_r -R system_r finance_u
```

Сопоставление роли `system_r` с типом SELinux-пользователя не означает, что этот пользователь теперь способен всегда переходить к этой роли – она будет предоставлена только в рамках ограниченного, четко определенного количества типов допустимого набора функций (доменов), управляемых политикой SELinux.

Когда роль с системными правами `system_r` будет сопоставлена с типом SELinux-пользователя и этому типу пользователя будут даны права на управление

службой СУБД PostgreSQL, тогда конечный пользователь сможет напрямую вызывать службу `postgresql` (желательно через `sudo`) следующим образом:

```
$ sudo /etc/rc.d/init.d/postgresql stop
```

Если пользователи имеют доступ к `run_init` (точнее – имеют в параметрах безопасности тип допустимого набора функций `run_init_t`), тогда они могут запускать любую службу, какую хотят. По этой причине предпочтительнее предоставлять необходимому числу пользователей право применять роль `system_r` и прохождение через специфику доступов, чем предоставлять им привилегию использования инструмента `run_init`.

## 3.4. SELinux и PAM (подключаемые модули аутентификации)

Имея дело со всей информацией о пользователях и ролях SELinux, мы не затронули вопрос о том, каким непосредственно образом приложения способны создавать и назначать параметры безопасности SELinux пользователю.

### 3.4.1. Назначение параметров безопасности с помощью подключаемых модулей аутентификации

Конечные пользователи входят в систему Linux через программу доступа (срабатывающую в рамках работы процесса `getty`), через сетевые службы (например, `OpenSSH`) или через графический менеджер (`xdm`, `kdm`, `gdm`, `slim` и др.).

Эти службы отвечают за переключение нашего эффективного пользовательского идентификационного номера – ID (после успешной аутентификации, конечно), поэтому мы не заходим в систему как пользователь `root`. В случае когда включена система защиты SELinux, эти процессы также требуют переключения соответствующего типа SELinux-пользователя (и роли), а иначе параметры безопасности будут унаследованы от службы входа в систему, что безусловно неверно для любой интерактивной сессии.

Теоретически можно было бы сделать все эти приложения полностью учитываемыми SELinux, связываемыми с библиотеками пользовательского пространства SELinux для получения информации о сопоставлении учетных записей Linux с типами SELinux-пользователей. Но, вместо того чтобы исправлять работу всех этих программ, разработчики системы защиты решили задействовать следующий уровень аутентификации, используя службы механизма обеспечения безопасности PAM (подключаемые модули аутентификации), которые предоставляют системы Linux.

Механизм подключаемых модулей аутентификации предлагает очень гибкий интерфейс для управления разными методами аутентификации в системе Linux (и Unix). Все приложения, упомянутые ранее, используют подключаемые модули аутентификации для выполнения своих задач в этой части. Для



того чтобы включить поддержку SELinux в этих приложениях, нам необходимо обновить их файлы конфигурации PAM – подключить динамическую библиотеку `pam_selinux.so`.

Следующий список строк – это выдержка из файла `/etc/pam.d/system-login` операционной системы Gentoo в части, касающейся служебных директив для сессий, взаимодействующих с подключаемыми модулями аутентификации. В файле запускаются коды библиотеки `pam_SELinux.so` как составная часть процесса аутентификации следующим образом:

```
session    required    pam_selinux.so close
session    optional    pam_loginuid.so
session    required    pam_env.so
session    optional    pam_lastlog.so
session    include     system-auth
session    optional    pam_ck_connector.so nox11
# Примечание: модули, которые запускаются с параметрами безопасности пользователя,
#              должны идти после этой строчки
session    required    pam_selinux.so multiple open
session    optional    pam_motd.so motd=/etc/motd
session    optional    pam_mail.so
```

Аргументы, поддерживаемые кодом библиотеки `pam_selinux`, даны на страницах руководства (*manual page*), описывающего `pam_selinux`. В приведенной выдержке опция `close` очищает текущие параметры безопасности (если они есть), а опция `open` устанавливает параметры безопасности конкретного пользователя. Модуль `pam_selinux` отвечает за запрос конфигурации SELinux, а также должен найти правильные сопоставления и параметры безопасности в соответствии с именем службы, которая используется.

### 3.4.2. Запрещение доступа в рекомендательном режиме работы защиты

Наличие включенной защиты SELinux, работающей в принудительном режиме в системе, улучшает ее устройчивость перед успешными случаями эксплуатации уязвимостей и другими злонамеренными действиями, особенно когда система используется в качестве сервера, взаимодействующего с пользователями через командно-строчный интерфейс (или обеспечивает другие службы для взаимодействия) и пользователи ограничены тем, что их учетная запись сопоставлена с типом SELinux-пользователя `user_u` или какими-нибудь другими, также имеющими ограничения.

Но кому-то из администраторов может потребоваться временное переключение в рекомендательный режим защиты. Это может быть нужно для диагностики проблем или отладки некоторых изменений в системе. При использовании рекомендательного режима было бы неплохо убедиться в том, что службы, предназначенные для взаимодействия, недоступны для использования обычными пользователями.

Это можно обеспечить в системе модулем `pam_sesermit`. Он будет отказывать в доступе к системе множеству зарегистрированных пользователей, если защита находится в рекомендательном режиме. По умолчанию эти пользователи указаны в файле `/etc/security/sesermit.conf`, но может быть настроен для хранения этих настроек и другой файл, если его указать в опции `conf=` внутри самой конфигурации PAM.

В файле `sesermit.conf` есть три способа для описания того, каким пользователям будет отказано в доступе, когда система работает в рекомендательном режиме:

- 1) обычные имена пользователей;
- 2) имена групп, начинающиеся со знака `@`;
- 3) наименования типов SELinux-пользователей, начинающиеся со знака `%`.

Каждое такое описание приводится в отдельной строке и может быть уточнено с помощью одной или нескольких опций. Эти опции описаны на страницах руководства (*manual*), описывающего файл `sesermit.conf`.

Для активации модуля `pam_sesermit` достаточно включить его службу `auth`, относящуюся к PAM, следующим образом:

```
auth required pam_sesermit.so
```

Кстати, не забудьте удалить все активные сессии пользователей, когда переключаетесь на рекомендательный режим, поскольку иначе любая оставшаяся работать сессия останется незатронутой.

### 3.4.3. Многоэкземплярность каталогов

Последний модуль PAM, который мы рассмотрим, – это `pam_namespace.so`. Прежде чем погрузиться в то, как конфигурировать этот модуль, давайте сначала посмотрим, что из себя представляет многоэкземплярность каталогов.

Многоэкземплярность – это некий метод, при котором когда пользователь входит в систему, он получает отображение ресурсов файловой системы, характерное для его конкретной сессии, при этом ресурсы других пользователей от него оказываются скрытыми. Это отличается от обычного контроля доступа, при котором другие ресурсы остаются все равно видны, но могут просто быть недоступны.

Этот характерный для сессии вид представления файловой системы применяется не только обычным монтированиям. Рассматриваемый модуль использует технологию пространства ядра Linux, для того чтобы обеспечить конкретное отображение на файловой системе, которое является изолированным и специфичным для конкретной пользовательской сессии. Другие пользователи получают иной вид файловой системы.

Давайте рассмотрим обычный пример. Представим, что все пользователи, кроме пользователя `root`, не должны иметь доступ ни к домашним каталогам других пользователей, ни к временным файлам, созданным другими пользователями. При стандартном контроле доступа эти ресурсы будут видны

(возможно, нечитаемы, но их существование будет отражено). Вместо этого с многоэкземплярностью каталогов любой пользователь будет видеть только свой собственный домашний каталог `/home`, а также собственные каталоги `/tmp` и `/var/tmp`.

Следующая настройка в файле `/etc/security/namespace.conf` переопределит размещения этих трех каталогов:

```
/tmp      /tmp-inst/          level  root
/var/tmp  /var/tmp/tmp-inst/ level  root
$HOME     $HOME/$USER.inst/  level  root
```

В реально действующей файловой системе они будут размещены внутри подподкаталогов `/tmp-inst`, `/var/tmp/tmp-inst` и `/home/<user>/<user>.inst`. Конечные пользователи не знают или не видят перенесенных размещений – для них каталоги `/tmp`, `/var/tmp` и домашний каталог `HOME` выглядят так, как они и ожидают.

В предыдущем примере только пользователя `root` не касаются такие изменения пространства имен. Пользователи, которым нужно включить многоэкземплярность каталогов, могут быть перечислены (через запятую), или может быть представлен конкретный список пользователей (с указанием перед ним знака `~`). Для того чтобы выполнялось изменение пространства имен, каталоги, в которых будут размещаться экземпляры, должны быть доступны в системе с дискреционными правами `000`:

```
# mkdir /tmp-inst && chmod 000 /tmp-inst
```

Затем следует подключить библиотеку `ram_namespace.so` в конфигурационных файлах `RAM` среди настроек сессии:

```
session    required    ram_namespace.so
```

В заключение убедитесь, что в SELinux разрешено создание множества экземпляров для каталогов. В дистрибутиве Red Hat Enterprise Linux это разрешение дается через параметр `polyinstantiation_enabled`, принимающий либо включенное состояние, либо выключенное. Другие дистрибутивы используют параметр `allow_polyinstantiation`:

```
# setsebool polyinstantiation_enabled on
```

## 3.5. ЗАКЛЮЧЕНИЕ

SELinux сопоставляет учетные записи Linux с типами SELinux-пользователей. Типы, в свою очередь, сопоставлены с ролями SELinux. Мы узнали, как управлять этими сопоставлениями при помощи приложения `semanage`, и смогли предоставлять правильные роли нужным пользователям.

Мы также увидели команды, которые используются для предоставления надлежащего уровня доступа и категории пользователю, и как мы можем описать эту мандатную конфигурацию в файле `setrans.conf`. Мы использовали ин-

струмент `chcat` для совершения большинства управляющих действий, связанных с категориями доступа.

После назначения ролей пользователям мы посмотрели, как переходить от одной роли к другой, используя команды `newrole`, `sudo`, `gungon` и `gun_init`. Мы закончили эту главу рассмотрением важного момента – как SELinux интегрируется в процесс аутентификации Linux и как настроить в дальнейшем систему Linux, используя несколько подключаемых модулей аутентификации, поддерживающих работу с SELinux.

# Глава 4

---

## Домены как допустимые наборы функций для процессов и контроль доступа на уровне файлов

Когда мы работаем во включенном режиме системы SELinux, сбор информации о параметрах безопасности, связанных с файлами и процессами, является основной необходимостью. Мы должны понимать, как эти параметры используются в политиках и какие подходящие правила защиты с механизмами контроля доступа характерны для какого-либо интересующего нас процесса.

В этой главе мы будем:

- работать с параметрами безопасности файлов и изучим, где они хранятся;
- выяснять, как назначаются параметры безопасности;
- изучать и получать информацию о том, как и когда процессы получают свои параметры безопасности;
- формировать первое представление о политике SELinux и о том, как ее запрашивать.

Закончим эту главу рассмотрением другой возможности SELinux, которая вызывает ограничения, и изучим, как они используются, для того чтобы обеспечить контроль доступа на уровне пользователя.

### 4.1. О ПАРАМЕТРАХ БЕЗОПАСНОСТИ ФАЙЛОВ

В ходе этой главы мы будем рассматривать процесс разворачивания веб-приложения DokuWiki. Это популярная программа, написанная на языке PHP для реализации интерфейса пользователя, позволяющего самостоятельно изменять содержимое веб-сайта с помощью инструментов, предоставляемых са-

мим сайтом. Для организации работы серверного уровня представления веб-системы (*backend system*) в DokuWiki не используется база данных, что тоже влияет на простоту ее установки и управления.

### 4.1.1. Получение информации о параметрах безопасности

Предположим, что приложение DokuWiki размещается в каталоге `/srv/web/localhost/htdocs/DokuWiki` и хранит свои wiki-страницы (где содержится публикуемая пользователями информация) в подкаталогах каталога `data/`. Во всяком случае, так будет, если загрузить последнюю версию архива DokuWiki с сайта разработчика и извлечь его в этот каталог. Некоторые дистрибутивы могут иметь другое место для размещения приложения DokuWiki в файловой системе (такое как `/var/lib/DokuWiki`), которое уже имеет корректную маркировку с точки зрения параметров безопасности. Рассматриваемый здесь пример в основном придерживается подобной маркировки, вне зависимости от размещения, позволяющей нам продемонстрировать различные действия, связанные с параметрами безопасности.

Сами параметры безопасности файлов могут быть легко получены с помощью аргумента `-z` команды `ls`. Большинство утилит, способных предоставлять информацию о параметрах безопасности, будут также использовать аргумент `-z`, как мы видели с утилитой `id`.

Посмотрим на текущие параметры безопасности собственно каталога `dokuwiki`:

```
# ls -dZ /srv/web/localhost/htdocs/DokuWiki
drwxr-xr-x. root root system_u:object_r:var_t:s0 dokuwiki
```

В параметрах безопасности, представленных здесь, есть обозначение `var_t`. Позже мы изменим его на правильный параметр (поскольку `var_t` является слишком общим и не предназначенным для размещения веб-содержания).

Параметры безопасности файлов и каталогов хранятся в файловой системе как расширенные атрибуты, если файловая система их поддерживает. **Расширенный атрибут** (*extended attribute*) (часто сокращенный до аббревиатуры **xattr**) – это комбинация «ключ/значение», соотнесенная с **inode** ресурса (неким информационным блоком, который представляет файл, каталог или символическую ссылку – объекты файловой системы). Каждый ресурс может иметь множество расширенных атрибутов, но при этом только одно значение приходится на один уникальный ключ. Кроме того, по соглашению, расширенные атрибуты в системе Linux используют следующий синтаксис:

```
<namespace>.<attribute>=<value>
```

Пространство имен (*namespace*) расширенного атрибута предоставляет дополнительный контроль доступа или возможности. Из пространств имен, поддерживаемых в настоящее время (*security*, *system*, *trusted* и *user*), пространство *security* налагает особые ограничения на действия с атрибутом: если не загружен модуль защиты (например, SELinux не включен), тогда только процессы

с Linux-возможностью `CAP_SYS_ADMIN` (в основном `root` или привилегированные похожим образом процессы) способны изменять этот параметр.

Мы можем сделать запрос на вывод существующих расширенных атрибутов, используя команду `getfattr` (распространяется в рамках пакета `attr` дистрибутива Red Hat Enterprise Linux или `sys-apps/attr` в Gentoo), как показывает следующий пример:

```
$ getfattr -m . -d DokuWiki
# file: DokuWiki
security.SELinux="system_u:object_r:var_t:s0"
```

Как мы можем видеть, параметры безопасности SELinux определяются через расширенный атрибут `security.SELinux`. Это гарантирует, что параметры SELinux не могут быть изменены пользователями, не имеющими административных прав, когда SELinux выключен, и что манипулирование параметрами контролируется при помощи политики SELinux, когда SELinux включен.

Команда просмотра состояния файла `stat` также может быть использована для отображения параметров безопасности SELinux:

```
$ stat dokuwiki
File: 'dokuwiki'
Size: 4096          Blocks: 8          IO Block: 4096 directory
Device: fd01h/64769d Inode: 8570035    Links: 8
Access: (0755/drwxr-xr-x)  Uid: (  0/   root) Gid: (  0/   root)
Context: system_u:object_r:var_t:s0
Access: 2020-08-16 13:46:44.573764039 -0400
Modify: 2020-08-16 13:36:59.698275931 -0400
Change: 2020-08-16 13:36: 59.698275931 -0400
Birth: -
```

Получение информации о параметрах безопасности файла или каталога должно быть так же привычно для администратора, как получение обычной информации о контроле доступа (на уровне отметок `read` (`r`), `write` (`w`) и `execute` (`x`)).

### 4.1.2. Интерпретация наименований типов SELinux

После работы с SELinux в течение какого-то продолжительного времени становится довольно ясно причина использования меток безопасности у файлов. Параметры безопасности файлов именованы в зависимости от их цели, позволяя администраторам более легко видеть, правильно ли присвоен тот или иной параметр.

Рассмотрим типы, применяемые для следующих объектов:

- для пользовательского файла в его домашнем каталоге (`user_home_t`);
- временный каталог в `/tmp` для приложения (`java_tmp_t`);
- сокет `rpcbind` (`rpcbind_var_run_t`).

Все эти файлы или каталоги имеют разные цели в файловой системе, и это отражено в присвоенных им параметрах.

Разработчики политик безопасности будут всегда стараться давать осмысленные имена параметрам, предоставляя нам возможность более легко распознавать цель файла и при этом делая политику не требующей пояснений.

Для стандартной файловой системы, к примеру, файлы имеют такие наименования типов допустимого набора функций, которые напоминают об их размещении относительно корня. Например, мы находим:

- исполняемые файлы в каталоге /bin (и /usr/bin), имеющие тип bin\_t;
- загрузочные файлы в каталоге /boot, отмеченные как boot\_t;
- общие системные ресурсы в каталоге /usr, отмеченные типом usr\_t,
- и т. д.

Наиболее интересны обозначения типов для отдельных приложений. К примеру, для сервера базы данных MySQL (или сопоставимых баз данных вроде MariaDB) мы имеем:

- 1) параметр `mysqld_t`, предназначенный собственно для приложения (тип допустимого набора функций для процесса или домен);
- 2) параметр `mysqld_port_t`, предназначенный для TCP-порта, на котором фоновый процесс MySQL обрабатывает соединения;
- 3) параметры `mysqld_server_packet_t` и `mysqld_client_packet_t`, являются типами, соотносимыми с сетевыми пакетами, которые получены (сервером) или отправлены (клиентом) через MySQL-порт;
- 4) тип `mysql_exec_t`, назначенный исполняемому файлу `mysqld`;
- 5) различные типы `mysql_*` для специфических размещений в файловой системе, относящихся к фоновому сервису приложения, такие как:
  - `mysqld_var_run_t` (для объектов в каталоге `/var/run`);
  - `mysqld_etc_t` (для объектов в каталоге `/etc`);
  - `mysqld_log_t` (для объектов в каталоге `/var/log`);
  - и `mysqld_tmp_t` (для объектов в каталоге `/tmp`);
- 6) тип `mysqld_home_t` предназначен для файлов конечного пользователя (администратора), характерных для управления MySQL (таких как файл конфигурации `~/my.cnf`).

Основываясь на параметрах безопасности файла или ресурса, администраторы могут легко выявить отклонения в установке системы. Примером отклонения может служить то, когда файл перемещен из домашнего каталога пользователя `home` в каталог размещения веб-сервера. Когда это происходит, он сохраняет параметр `user_home_t`, поскольку расширенные атрибуты перемещаются с ним. Когда серверному процессу не разрешен доступ к `user_home_t` по умолчанию, он не сможет предоставить этот файл своим пользователям.

## 4.2. ЗАКРЕПЛЕНИЕ ПАРАМЕТРОВ БЕЗОПАСНОСТИ ЗА ОБЪЕКТОМ И ИХ ИГНОРИРОВАНИЕ

Теперь, когда нам известно, что параметры безопасности файлов сохраняются в виде расширенных атрибутов, как нам убедиться в том, что файлы полу-



чают верный набор параметров безопасности, когда они будут записаны или изменены? Для этого существует несколько рекомендаций, варьирующихся от правил наследования до конкретных команд, для того чтобы установить параметры SELinux на ресурсе файловой системы.

### 4.2.1. Наследование параметров безопасности по умолчанию

По умолчанию система защиты SELinux использует наследственность параметров для того, чтобы идентифицировать, какие параметры безопасности должны быть назначены файлу (или каталогу, сокету и т. д.) при их создании. Файлу, созданному в каталоге с параметром `var_t`, будет также присвоен тип допустимого набора функций `var_t`. Это означает, что наследственность основана на родительском каталоге, а не на параметрах выполняемого процесса.

Несмотря на это, есть несколько исключений:

- в случае с приложением, поддерживающим функции SELinux, оно может само выполнить изменение параметров безопасности файла (если политика безопасности SELinux это, конечно, позволит). Поскольку эти действия полностью относятся к области контроля самого приложения, то настроить его поведение в целом не получится;
- приложение с названием `restorecond` может быть использовано так, чтобы усилить параметры безопасности на каком-то количестве путей/файлов, в соответствии с правилами SELinux. Мы рассмотрим эти правила и программу `restorecond` позже в данной главе;
- политика SELinux позволяет [изменять параметры] по правилам преобразования типов, которые учитывают параметры безопасности процесса, создающего новые файлы или каталоги.

Эти правила перехода мы раскроем далее.

### 4.2.2. Правила преобразования типов и их вывод

Правила преобразования типов (*type transition rules*) – это правила политики, вынуждающие использовать другой тип допустимого набора функций вместо предустановленного. В том случае, когда мы имеем дело с параметрами безопасности файлов, тогда правило преобразования типа может быть проиллюстрировано следующим образом: если некий процесс, работающий с типом допустимого набора функций `httpd_t`, создает файл в каталоге, имеющем тип `var_log_t`, тогда тип файла становится `httpd_log_t` вместо ожидаемого `var_log_t`.

В основном правило политики, позволяющее так сделать, гарантирует, что любому файлу, размещенному в каталоге журнала регистрации веб-серверами, присваивается параметр безопасности, предназначенный для журнала регистрации веб-сервера – `httpd_log_t`, а не `var_log_t`, предполагаемый по умолчанию, что должно было бы произойти в случае стандартного наследования.

Мы можем выполнить запрос на предоставление этих правил, используя команду `sesearch`, являющуюся частью пакета `setools-console` в Red Hat Enterprise Linux или `app-admin/setools` в Gentoo. Команда `sesearch` – одна из самых важных

среди инструментов, способных делать запросы к текущей политике SELinux. Для демонстрации ее работы на предыдущем примере нам понадобится исходный (*source*) тип допустимого набора функций процесса и тип цели (*target*) доступа – каталога, в котором создается файл: `httpd_t` и `var_log_t` соответственно. Затем мы используем, как показано ниже, команду `sesearch` для поиска декларации преобразования типа `httpd_t`, связанного с субъектом доступа и типом `var_log_t` для объекта доступа:

```
$ sestatus -T -s httpd_t -t var_log_t
Found 1 semantic te rules:
  type_transition httpd_t var_log_t : file httpd_log_t;
```

**Р** Для русскоязычного варианта данное сообщение могло бы быть представлено следующим видом:

```
$ sestatus -T -s httpd_t -t var_log_t
Найдено 1 семантическое правило te-формата:
  type_transition httpd_t var_log_t : file httpd_log_t;
```

Строка, начинающаяся с фразы `type_transition` (*преобразование типа*), – это правило политики SELinux, которое в точности соответствует заданному автором политик. Давайте рассмотрим другие правила преобразования типа, которые предназначены для объектов с типом `tmp_t` (назначаемым каталогам верхнего уровня, содержащим временные файлы, такие как `/tmp` и `/var/tmp`):

```
$ sestatus -T -s httpd_t -t tmp_t
Found 4 semantic te rules:
  type_transition httpd_t tmp_t : file httpd_tmp_t;
  type_transition httpd_t tmp_t : dir httpd_tmp_t;
  type_transition httpd_t tmp_t : lnk_file httpd_tmp_t;
  type_transition httpd_t tmp_t : sock_file httpd_tmp_t;

Found 2 named file transition rules:
  type_transition httpd_t tmp_t : file krb5_host_rcache_t "HTTP_23";
  type_transition httpd_t tmp_t : file krb5_host_rcache_t "HTTP_48";
```

**Р** Для русскоязычного варианта данное сообщение могло бы быть представлено следующим видом:

```
$ sestatus -T -s httpd_t -t tmp_t
Найдено 4 семантических правила te-формата:
  type_transition httpd_t tmp_t : file httpd_tmp_t;
  type_transition httpd_t tmp_t : dir httpd_tmp_t;
  type_transition httpd_t tmp_t : lnk_file httpd_tmp_t;
  type_transition httpd_t tmp_t : sock_file httpd_tmp_t;

Найдено 2 именованных правила преобразования файлов:
  type_transition httpd_t tmp_t : file krb5_host_rcache_t "HTTP_23";
  type_transition httpd_t tmp_t : file krb5_host_rcache_t "HTTP_48";
```

Политика говорит нам о том, что если файл, каталог, символическая ссылка или сокет создается (процессом `httpd_t` – типа) в каталоге, имеющем тип `tmp_t`, тогда этому ресурсу присваивается параметр `httpd_tmp_t` (а не унаследованный

tmp\_t, назначенный по умолчанию). Но она (политика) также содержит два более гибких правила преобразования для файлов. Этот пример снова показывает нам степень детализации в SELinux, демонстрируя правила преобразования типов для различных классов объектов: обычные файлы, каталоги, а также файлы, представляющие собой символические ссылки или сокеты. Существуют и другие классы объектов, относящиеся к файловой системе, которые поддерживает SELinux. Это блочные устройства (blk\_file), символьные устройства (chr\_file) и каналы (fifo\_file).

С помощью способа **именованных файловых преобразований** (*named file transitions*) политика может учитывать имя файла (или каталога), для того чтобы распределять параметры безопасности. В предыдущем примере, если файл с именем HTTP\_23 или HTTP\_48 создан в каталоге типа tmp\_t, он получит не параметр httpd\_tmp\_t (как подразумевалось бы обычными правилами передачи типов), а тип krb5\_host\_gcache\_t (используемый для реализации сетевого протокола аутентификации **Kerberos**).

Преобразование типов дает нам не только представление о том, какие параметры безопасности будут назначаться, но также ключ к пониманию того, какие типы целевых объектов связаны с какими типами процессов. В случае с веб-сервером мы выяснили при запросе политики, что его файлы журналов в большинстве своем имеют тип httpd\_log\_t, а его временные файлы – httpd\_tmp\_t.

### 4.2.3. Копирование и перемещение файлов

Параметры безопасности файлов также могут быть переданы вместе с самим файлом во время операций копирования или перемещения. По умолчанию система Linux будет:

- сохранять параметры безопасности файлов в случае операции перемещения (mv) в пределах одной и той же файловой системы (т. к. эта операция не затрагивает расширенные атрибуты, а всего лишь корректирует метаданные файла);
- игнорировать текущие параметры безопасности в случае операции перемещения (mv) за пределы границ одной файловой системы, поскольку это действие приводит к созданию нового файла, включающего его содержание и расширенные атрибуты. Взамен старых система защиты использует для определения конечных параметров безопасности механизм наследования (или преобразования типов файлов);
- игнорировать текущие параметры безопасности при выполнении операции копирования (cp), вместо них также будет использован механизм наследования или преобразования типов файлов для определения целевых параметров.

К счастью, это только поведение по умолчанию (основанное на поддержке расширенных атрибутов данными командами), которое может гибко регулироваться.

Мы можем использовать опцию `-Z` для того, чтобы сообщить команде `mv`, что параметры безопасности для файла должны быть установлены согласно типу [допустимого набора функций], который установлен «по умолчанию» для целевого каталога. Так, в следующем примере два файла перемещаются из домашнего каталога пользователя `home` в каталог `/tmp`. В результате первый будет сохранять свой тип файла (`user_home_t`), в то время как другой получит тип, связанный с местом размещения (каталогом `/tmp`)— `user_tmp_t`:

```
$ mv test1.txt /tmp
$ mv -Z test2.txt /tmp
$ ls -ldZ /tmp/test*
-rw-r--r--. david users user_u:object_r:user_home_t:s0 test1.txt
-rw-r--r--. david users user_u:object_r:user_tmp_t:s0 test2.txt
```

Похожим образом мы можем сообщить команде `cp` через опцию `--preserve=context` необходимость сохранять (*preserve*) параметры безопасности (*context*) SELinux при копировании файлов. Используя тот же пример, получим следующее:

```
$ cp test1.txt /tmp
$ cp --preserve=context test2.txt /tmp
$ ls -ldZ /tmp/test*
-rw-r--r--. david users user_u:object_r:user_tmp_t:s0 test1.txt
-rw-r--r--. david users user_u:object_r:user_home_t:s0 test2.txt
```

Большинство утилит, которые предоставляются через пакет `coreutils`, поддерживают опцию `-Z`: `mkdir` (создание каталога), `mknod` (создание файла устройства), `mkfifo` (создание именованного канала) и т. д. И даже более того, большинство этих утилит позволяют пользователю точно задать параметры безопасности посредством опции `--context`.

Например, для создания каталога `/tmp/foo` с параметрами `user_home_t` использование `mkdir` без дополнительных указаний в параметрах не даст нужного эффекта:

```
$ sesearch -s user_t -t tmp_t -T -c dir
type_transition user_t tmp_t : dir user_tmp_t
```

С помощью опции `--context` мы можем указать утилите установить конкретные параметры:

```
$ mkdir --context=user_u:object_r:user_home_t:s0 /tmp/foo
$ ls -ldZ /tmp/foo
drwxr-xr-x. lisa lisa user_u:object_r:user_home_t:s0 foo/
```

Для того чтобы узнать, как обстоят дела у других утилит, лучше всего обратиться к справочному руководству (*manual page*) и посмотреть, как конкретная утилита работает с расширенными атрибутами. Например, для того чтобы команда `rsync` сохраняла расширенные атрибуты, применяется опция `-X` или `--xattrs`:

```
$ rsync -av -X <source> <destination>
```

#### 4.2.4. Временное изменение параметров безопасности файла

Мы можем использовать инструмент `chcon` (сокр. от *change context* – изменение параметров безопасности) для обновления параметров безопасности файла (файлов) напрямую. В нашем примере, приведенном ранее, мы отметили, что файлы `DokuWiki` имеют тип [допустимого набора функций] – `var_t`. Это основной предусмотренный тип для изменяющихся данных, и он не является удачным типом для веб-содержимого. Мы можем использовать команду `chcon`, чтобы присвоить тип `httpd_sys_content_t` этим файлам, который позволит веб-серверам иметь к ним доступ на чтение:

```
$ chcon -R -t httpd_sys_content_t /srv/www
```

Другая возможность, которую предлагает `chcon`, – это сказать ему, что нужно присвоить файлу тот же параметр, что и у какого-то другого файла. В следующем примере мы используем команду `chcon` для назначения файлу `/srv/www/index.html` тех же параметров безопасности, которые используются для файла `/var/www/index.html`:

```
$ chcon --reference /var/www/index.html /srv/www/index.html
```

Если мы меняем параметры безопасности файла командой `chcon` и устанавливаем им значения, отличающиеся от какого-либо из имеющихся в списке параметров, тогда есть вероятность, что параметры безопасности будут позже возвращены: менеджеры пакетов могут вернуть параметры файла к их предполагаемому значению, или же системный администратор может запустить операцию переназначения (*relabeling operation*) параметров безопасности для всей файловой системы.

До сих пор мы фокусировали внимание только на такой части параметров безопасности, как «тип допустимого набора функций». Тем не менее эти параметры также включают такие части, как «роль» и «тип SELinux-пользователя». Если механизм контроля на уровне пользователя (UBAC) не включен, тогда у пользователя SELinux нет возможности влиять на любые решения, и его возврат к первоначальной настройке будет иметь небольшое значение. Если же UBAC включен, тогда может возникнуть необходимость сбросить в параметрах безопасности файлов значения, определяющие SELinux-пользователя. Утилиты, такие как `chcon`, способны устанавливать в параметрах значения SELinux-пользователя подобно тому, как показано здесь:

```
# chcon -u system_u -R /srv/www
```

Роль для файла обычно назначается как `object_r`, поскольку в настоящий момент роли имеют значение только для пользователей (процессов).



Для того чтобы можно было изменять параметры безопасности, нам необходимы правильные привилегии SELinux, которые называются `relabelfrom` и `relabelto`. Эти права предоставляются для типов допустимого набора функций (доменов) и обозначают, позволено ли какому-то конкретному домену изменять параметры безопасности с (`rela-`

belfrom) конкретного типа (такого как `user_home_t`) на (`relabelto`) другой тип (такой как `httpd_sys_content_t`). Если мы находим отказы в журнале событий безопасности, связанные с этими разрешениями, то тогда это означает, что домену запрещено изменение параметров безопасности.

### 4.2.5. Установка категорий для файлов и каталогов

Мы останавливали наше внимание преимущественно на изменении типов допустимого набора функций и коротко затронули типы SELinux-пользователей, но еще одной важной частью параметров безопасности является мандатная конфигурация, поддерживающая категории (и мандатные уровни). При помощи команды `chcon` мы можем добавлять мандатные уровни и категории следующим образом:

```
$ chcon -l s0:c0,c2 index.html
```

Другим инструментом, который может использоваться для присвоения категорий, является команда `chcat` (сокр. от *change category* – изменить категорию). При помощи `chcat` мы можем назначать дополнительные категории, вместо того чтобы каждый раз заново подтверждать те, которые уже установлены, как в случае с `chcon`, и даже пользоваться удобочитаемыми обозначениями категорий, предоставляемых в файле `setrans.conf`:

```
$ chcat -- +Customer2 index.html
```

Для удаления категории используйте просто знак минус:

```
$ chcat -- -Customer2 index.html
```

Для удаления всех категорий используйте опцию `-d`:

```
$ chcat -d index.html
```

Пользователям и администраторам следует учитывать, что приложения в основном не устанавливают категории сами, так что они должны быть добавлены специально предусмотренными для этого способами.

### 4.2.6. Использование многоуровневой защиты для файлов

Когда система использует политику многоуровневой защиты (*MLS*), то должен применяться инструмент `chcon`. Синтаксис команды подобен тому, какой применялся для работы с категориями. Например, при установке мандатного уровня `s1` для набора категорий, состоящего из `c2`, а также из диапазона категорий, начиная с `c4` и заканчивая `c10`, на всех файлах домашнего каталога конкретного пользователя следует выполнить следующую команду:

```
$ chcon -R -l s1:c2,c4.c10 /home/lisa
```

Имейте в виду, что параметры безопасности пользователя при выполнении `chcon` и параметры пользователя, который намерен применять данные, должны допускать возможность работы с указанным мандатным уровнем.

### 4.2.7. Резервное копирование и восстановление расширенных атрибутов

Так же, как и обычным средствам для выполнения операций над файлами (таким как `mv` и `cp`), программным средствам для резервного копирования тоже необходимо учитывать параметры безопасности SELinux. В действительности существуют два важных требования для инструмента резервного копирования при работе с активной защитой SELinux в системе:

- инструмент резервного копирования должен быть запущен с такими параметрами SELinux, которые позволят читать все файлы, подлежащие резервному копированию и восстановлению. Если нет специальной политики SELinux для инструмента резервного копирования, тогда будет необходимо запустить его с неограниченным (*unconfined*) или с высокопривилегированным типом допустимого набора функций (доменом), для того чтобы он мог выполнить свои задачи успешно;
- инструмент резервного копирования должен быть способен сохранять и восстанавливать расширенные атрибуты.

Одним из популярных инструментов для создания резервных копий (или архивов) является программа `tar`. При создании `tar`-архива добавьте параметр `--selinux` для учета программой соответствующих параметров безопасности (причем как при создании архива, так и при извлечении из него файлов):

```
# tar cvjf home-20200815.tar.bz2 /home --SELinux
```

### 4.2.8. Использование опций монтирования для установки параметров SELinux

Не все файловые системы поддерживают расширенные атрибуты. Когда используется файловая система без поддержки расширенных атрибутов, тогда параметры безопасности файла являются либо основанными на типе самой файловой системы (каждая файловая система имеет собственный сопоставленный с ней набор параметров безопасности), либо задаются в системе через опции команды [подключения файловых систем для работы в операционной системе] – `mount`.

Наиболее широко используемой опцией команды `mount` в таких ситуациях является опция `context=`. При установке она будет использовать указанные после знака «равно» параметры безопасности в качестве таковых параметров для всех ресурсов в подключаемой файловой системе. Например, для подключения внешнего USB-накопителя, который содержит форматирование согласно с файловой системой FAT и предполагает возможность записи на него конечными пользователями информации, следовало бы использовать параметр `user_home_t`, как показано ниже:

```
# mount -o context="user_u:object_r:user_home_t:s0" /dev/sdc1 /media/usb
```

Если файловая система поддерживает расширенные атрибуты, но не имеет пока маркировки всех файлов параметрами безопасности, тогда мы можем использовать опцию `defcontext=`, для того чтобы сообщить системе Linux, что если нет установленного на объекте параметра безопасности SELinux, тогда для него требуется применить параметр по умолчанию:

```
# mount -o defcontext="system_u:object_r:var_t:s0" /dev/sdc1 /srv/backups
```

Другой опцией команды `mount` является `fscontext=`. Она связывает параметр безопасности с типом файловой системы, а не конкретным файлом в ней. Например, файловая система у CD/DVD-дисков может быть ISO 9660, Joliet или UDF. Для SELinux характерно сочетание с первым типом файловой системы с помощью типа допустимого набора функций `iso9660_t`. Этот тип определяется всей файловой системы целиком и применяется системой защиты SELinux для сопоставления разрешений при выполнении таких операций, как подключение (*mount*) и создание файлов. Администраторы могут не желать позволять какой-нибудь файловой системе быть подключенной куда-либо еще, кроме как внутри каталога `/media`. С помощью опции `fscontext=` тип файловой системы может быть установлен иначе, чем был бы при установке типа файловой системы по умолчанию.

Опция `fscontext=` имеет небольшое влияние на параметры безопасности файлов, расположенных внутри этой файловой системы. Например, смонтированная файловая система ISO 9660 будет, вероятно, использовать тип `iso9660_t` для файловой системы как таковой, в то время как файлы будут доступны через тип `removable_t`:

```
# mount -o fscontext="system_u:object_r:iso9660_t:s0" /dev/sdc1 /mnt
```

Последняя опция, которая может быть использована при монтировании файловых систем, – это опция `rootcontext=`. Она применяется для того, чтобы корневой индексный дескриптор (*root inode*) имел заданный (после знака равенства) параметр безопасности даже до того момента, как файловая система станет видна в пользовательском пространстве. Параметры безопасности корня файловой системы могут значительно различаться в зависимости от того, где он расположен, поэтому принудительное принятие параметров через опцию команды `mount` позволяет администраторам использовать необходимые параметры, невзирая на фактическое размещение:

```
# mount -o rootcontext="system_u:object_r:tmp_t:s0" -t tmpfs none /var/cache/eix
```

На этом все опции команды `mount`, которые связаны параметрами безопасности, заканчиваются. Еще следует отметить: опция `context=` является взаимоисключающей для опций `defcontext=` и `fscontext=`. То есть в то время как опции `defcontext=` и `fscontext=` могут использоваться вместе, они не могут быть использованы совместно с опцией `context=`.



## 4.3. ФОРМУЛИРОВКА ПАРАМЕТРОВ БЕЗОПАСНОСТИ ДЛЯ ФАЙЛОВ

Когда мы считаем, что параметр файла не верен, то нам требуется его исправить. SELinux предлагает несколько методов для этого, а некоторые дистрибутивы даже добавляют еще больше. Мы можем использовать такие инструменты, как `chcon`, `restorecon` (вместе с `semanage`), `setfiles`, `rlpkg` (в Gentoo) и `fixfiles` (в Red Hat Enterprise Linux). Конечно, мы также могли бы применить команду `setfattr`, но это был бы наименее удобный для пользователя вариант установки параметров безопасности.

### 4.3.1. Использование выражений, описывающих параметры безопасности

В политике SELinux существует перечень регулярных выражений, которые информируют утилиты и библиотеки SELinux о том, какие параметры следует присвоить файлу (или другому ресурсу файловой системы). Хотя этот перечень не влияет непосредственно на систему, он необходим администраторам для того, чтобы увидеть, являются ли правильными параметры безопасности, а также для программ, которым необходимо сбросить текущие параметры безопасности на те значения, которые у них должны быть. Сам по себе перечень хранится в файлах `file_contexts.*`, размещенных в каталоге `/etc/SELinux/targeted/contexts/files`.

Будучи администратором, мы можем запросить содержание этого перечня при помощи команды `semanage fcontext` следующим образом:

```
# semanage fcontext -l
SELinux fcontext      type          Context
./.*                  all files    system_u:object_r:default_t:s0
/[^/]+               regular file system_u:object_r:etc_runtime_t:s0
/a?quota\.(user|group) regular file system_u:object_r:quota_db_t:s0
...
```

**Р** Для русскоязычного варианта данное сообщение могло бы быть представлено следующим видом:

```
# semanage fcontext -l
Путь                  Тип           Параметры безопасности
./.*                  all files    system_u:object_r:default_t:s0
/[^/]+               regular file system_u:object_r:etc_runtime_t:s0
/a?quota\.(user|group) regular file system_u:object_r:quota_db_t:s0
...
```

В качестве примера программы, которая запрашивает такую информацию, можно привести `matchpathcon`, который мы уже использовали в этой книге ранее:

```
# matchpathcon /etc/SELinux/targeted
/etc/SELinux/targeted system_u:object_r:SELinux_config_t:s0
```

Тем не менее не все записи видны с помощью приложения `semanage`. Записи, относящиеся к характерным домашним каталогам пользователя (таким как `/home/mashenka/.ssh`), не показываются, поскольку эти записи зависят от пользователя Linux (и, что более важно, от сопоставленного с ним типа SELinux-пользователя).

Но для всех других записей вывод команды содержит:

- регулярное выражение, которое соответствует одному или нескольким путям;
- типы объектов, к которым применяется правило, но переведенные в удобочитаемый формат;
- параметры безопасности, назначенные ресурсам, которые попадают в число соответствующих заданным в регулярном выражении путям и типам объектов.

Список типов позволяет нам разделять параметры безопасности по различным типам ресурсов. Вывод команды `semanage fcontext` содержит удобочитаемые идентификаторы, но для наибольшей пользы мы рассмотрим также относящиеся к ним опции: типы ресурса могут быть обычным файлом (*regular file*) (`--`), каталогом (*directory*) (`-d`), сокетом (*socket*) (`-s`), именованным каналом (*named pipe*) (`-p`), блочным устройством (*block device*) (`-b`), устройством посимвольного ввода/вывода (*character device*) (`-c`) или же символической ссылкой (*symbolic link*) (`-l`). Когда этот идентификатор записан как *all files* («все файлы»), то параметры безопасности, указанные в этой строке, применяются независимо от типа объекта.

К этому моменту мы пока еще не определяли сами такие правила, но после следующего раздела даже определение пользовательских выражений параметров безопасности SELinux не будет содержать для нас какой-либо тайны. Важной особенностью списка параметров безопасности является способ определения приоритета среди выражений, т. к. мы можем легко иметь два разных выражения, которые оба подходят. В SELinux побеждает то правило, которое наиболее конкретно. Используемая логика (в порядке приоритета) такова:

- если в строке А есть конструкции регулярных выражений<sup>1</sup>, а строка Б их не содержит, тогда строка Б более конкретна;
- если число символов перед первой частью регулярного выражения в строке А меньше, чем число символов перед первой частью регулярного выражения в строке Б, тогда строка Б более конкретна;
- если число символов в строке А меньше, чем в строке Б, тогда строка Б более конкретна;
- если строка А не сопоставляется с каким-либо особым типом SELinux (редактор политики явно указал SELinux не назначать тип), а строка Б сопоставляется, тогда строка Б считается более конкретной.

<sup>1</sup> То есть содержит некий шаблон, задающий правило поиска. – Прим. перев.

Рассмотрим все правила, которые соответствуют каталогу `/usr/lib/pgsql/test/regress/pg_regress` (отобразив их посредством программы `findcon`, предоставляемой пакетом `setools-console` в дистрибутиве Red Hat Enterprise Linux):

```
$ findcon /etc/SELinux/strict/contexts/files/file_contexts -p \
/usr/lib/pgsql/test/regress/pg_regress
/. * system_u:object_r:default_t
/usr/. * system_u:object_r:usr_t
/usr/(.*)?lib(/.*)? system_u:object_r:lib_t
/usr/lib/pgsql/test/regress(/.*)? system_u:object_r:postgresql_db_t
/usr/lib/pgsql/test/regress/pg_regress -- \ system_u:object_r:postgresql_exec_t
```

Хотя другие правила тоже соответствуют, последнее из них наиболее конкретно, потому что не содержит каких-либо регулярных выражений. Если бы этой строки не было, тогда строка перед ней была бы наиболее конкретной, поскольку число символов в первой части выражения значительно больше, чем в других, и т. д.

Тем не менее существует одна оговорка в этом порядке правил. Когда дополнительные правила добавлены с помощью команды `semanage` (которая описывается далее), тогда играет роль порядок добавления правил, а не их конкретность. То есть будет использоваться для заданного пути то правило, которое было добавлено последним.

### 4.3.2. Регистрация изменений параметров безопасности файлов

Поскольку изменение параметров SELinux с помощью команды `chcon` зачастую носит только временный характер, очень рекомендуется использовать `chcon`, только когда мы проверяем влияние изменения контекста.

Приняв решение сделать какие-либо изменения, нам необходимо зарегистрировать их с помощью команды `semanage`. Например, для того чтобы присвоить каталогу `/srv/www` (и всем его подкаталогам) параметр безопасности `httpd_sys_content_t` на постоянной основе, мы должны выполнить следующие действия:

```
# semanage fcontext -a -t httpd_sys_content_t "/srv/www(/.*)?"
# restorecon -Rv /srv/www
restorecon reset /srv/www context system_u:object_r:var_t:s0
-> system_u:object_r:httpd_sys_content_t:s0
...
```

Все, что мы сейчас сделали, – это зарегистрировали каталог `/srv/www` и его подкаталоги как объект с типом допустимого набора функций `httpd_sys_content_t`, используя команду `semanage`. Затем мы использовали команду `restorecon` для (рекурсивной) перенастройки параметров каталога `/srv/www` к значению, которое зарегистрировано в списке параметров безопасности. Рекомендуется именно таким способом устанавливать параметры безопасности для большинства объектов.

Эти регистрации являются локальными (пользовательскими) регистрациями выражений для параметров безопасности и сохраняются в отдельном файле конфигурации (`file_contexts.local`). В следующем примере по правилам приоритета, описанным выше, первое правило должно было бы быть определяющим, но в действительности оно не будет определять ожидаемое поведение, т. к. как самое последнее правило, которое мы добавили, имеет все равно больший приоритет:

```
# semanage fcontext -a -t httpd_sys_content_t "/srv/www(/.*)?"
# semanage fcontext -a -t var_t "/srv(/.*)?"
```

В этом примере каталогу `/srv/www` будет присвоен тип `var_t` вместо `httpd_sys_content_t`, потому что правило `var_t` было добавлено позже.

Команда `semanage fcontext` (сокр. от *selinux manage file context* – управление в SELinux параметрами безопасности файла) также может использоваться для информирования SELinux о том, что часть дерева файловой системы следует соотносить с параметрами безопасности так, как если бы эта часть располагалась в другом месте. Это дает нам возможность использовать разные пути для установки приложений или места размещения для файлов и сообщать команде `semanage` применять те же параметры безопасности, как если бы новое место размещения было таким, которое предполагалось для размещения по умолчанию.

Давайте увидим это на примере, где все, что размещено внутри каталога `/srv/www`, будет иметь такие же параметры безопасности, как если бы оно было размещено в каталоге `/var/www`. То есть подкаталог `/srv/www/icons` получит те же параметры, что имеются у каталога `/var/www/icons`. Для этого мы используем опцию `-e` команды `semanage fcontext`, что позволит достичь необходимой эквивалентности:

```
# semanage fcontext -a -e /var/www /srv/www
```

В результате будет сформирована такая замещающая запись, которая приведет к тому, что все объекты, расположенные внутри каталога `/srv/www`, получат параметры безопасности такие же, как если бы они размещались в каталоге `/var/www`.

В большинстве дистрибутивов уже настроено некоторое число таких мест, для которых выполняется замещение параметров. Команда `semanage fcontext -l` позволяет увидеть эти места размещения объектов в результате своего вывода, но вы можете также напрямую прочитать эту информацию в конфигурационном файле `file_contexts.subs_dist`, размещающемся в каталоге `/etc/SELinux/targeted/contexts/files`:

```
# cat /etc/SELinux/targeted/contexts/files/file_contexts.subs_dist
/run /var/run
/run/lock /var/lock
/run/systemd/system /usr/lib/systemd/system
...
```

### 4.3.3. Использование заказных типов

Некоторые типы [допустимого набора функций] SELinux предназначены для файлов, чьи пути размещения не могут быть строго определены администраторами, или тех, у которых администраторы не хотят сбрасывать параметры безопасности, когда будет проводиться процедура их переназначения. Для этих целей SELinux поддерживает так называемые заказные типы (*customizable types*). Когда инструменты управления параметрами безопасности файлов, такие как `restorecon`, сталкиваются с файлом, имеющим подобный специально заказанный тип, они не будут приводить его параметры безопасности к тому виду, который зарегистрирован в списке предопределенных параметров.

Заказные типы [допустимого набора функций] объявляются в файле `customizable_types`, расположенном внутри каталога `/etc/selinux/strict/contexts`. Чтобы изменить параметры безопасности у таких файлов, используя команду `restorecon`, администраторам необходимо передать опцию принудительного сброса (`-F`), выполняемого перед тем, как данный инструмент смог выполнить переустановку параметров.

Давайте посмотрим на содержимое этого файла `customizable_types`:

```
$ cat /etc/SELinux/strict/contexts/customizable_types
sandbox_file_t
svirt_image_t
home_bin_t
...
user_tty_device_t
```

В качестве примера мы можем присвоить файлу в домашнем каталоге (в данном случае этот файл называется `convert.sh`) значение типа `home_bin_t`, который относится к заказным типам и поэтому не будет изменен обратно к типу `user_home_t`, когда будет выполняться операция переустановки параметров безопасности:

```
$ chcon -t home_bin_t ~/convert.sh
```

А пока, для того чтобы расширить число заказных типов, которые нужно присвоить каким-либо объектам, требуется внести их в файл `customizable_types`. Поскольку данный файл может быть перезаписан, когда новая политика (во время установки) будет применяться в системе, то его требуется тщательно контролировать.

Несмотря на это, использование заказных типов имеет свои преимущества. Как администратор мы можем захотеть создавать и поддерживать особые типы, полезные для конечных пользователей, которые вправе использовать команду `chcon` для установки конкретных параметров безопасности отдельным файлам в своем домашнем каталоге. Имея файлы с заказными типами, пользователь может рассчитывать, что операция перенастройки параметров безопасности, примененная к каталогу `/home`, не заденет их и параметры безопасности останутся прежними.

В любом случае, предпочтительнее использовать команду `semanage fcontext` для добавления выражения в число регистрируемых и команду `restorecon` для того, чтобы зафиксировать зарегистрированные параметры безопасности у файлов. Если мы опять возьмем в качестве примера файл `convert.sh`, то получим необходимый результат, выполнив следующие действия:

```
# semanage fcontext -a -t home_bin_t /home/myuser/convert\.sh
# restorecon -F /home/myuser/convert.sh
```

Большинство администраторов предпочитают использовать присвоение параметров безопасности на уровне каталогов. Для нижеприведенного примера команды сами файлы располагаются в каталоге `~/bin`:

```
# semanage fcontext -a -t home_bin_t "/home/[^/]*bin(/.*)?"
```

#### 4.3.4. Различные виды файлов `file_contexts` и их компиляция

Внутри каталога `/etc/SELinux/targeted/contexts/files` могут быть найдены пять различных видов файлов, обозначенных как `file_contexts` и содержащих выражения, регулирующие параметры безопасности для файлов:

- собственно файл `file_contexts` (без какого-либо дополнительного окончания) – это основной файл с выражениями, предоставленными политикой SELinux, которая распространяется в рамках дистрибутива системы Linux;
- файл `file_contexts.local` – содержит локально добавленные правила (посредством команды `semanage fcontext`, которую мы рассмотрели ранее в этом разделе);
- файл `file_contexts.homedirs` – содержит выражения для домашних каталогов пользователей. Когда новые сопоставления создаются и управляются через `semanage login` и `semanage user`, этот файл корректируется с учетом новой ситуации;
- файл `file_contexts.subs_dist` – содержит правила эквивалентного замещения параметров безопасности объектов, размещенных по определенному пути, на параметры безопасности, свойственные другому пути размещения. Правила предоставляются политикой SELinux, распространяемой в рамках дистрибутива Linux, которая предписывает системе защиты SELinux рассматривать определенную часть файловой системы соответствующей какой-то другой обозначенной части файловой системы с точки зрения параметров безопасности;
- файл `file_contexts.subs` – тоже содержит правила эквивалентного замещения параметров безопасности, но те, которые устанавливаются локально (посредством команды `semanage fcontext`, которую мы рассмотрели ранее в этой главе).

Рядом с этими файлами вы найдете соответствующие им исполняемые файлы `*.bin` (так, файл `file_contexts.bin` будет исполняемым для файла `file_con-`

texts, а файл `file_contexts.local.bin` – для файла `file_contexts.local` и т. п.). Эти файлы содержат те же правила, что и главный файл, но предварительно скомпилированы для того, чтобы можно было выполнить более быстрый поиск. Эти \*.bin-файлы создаются автоматически, но в случае появления расхождений между исходным текстовым вариантом и исполняемым администраторы могут перестроить с тем же успехом файлы самостоятельно, используя команду `sefcontext_compile`:

```
# cd /etc/SELinux/targeted/contexts/files
# sefcontext_compile file_contexts.local
```

### 4.3.5. Обмен локальными изменениями

Когда изменения параметров безопасности, свойственные одной какой-то системе, регистрируются при помощи команды `semanage fcontext`, они применяются локально – только к ней одной. Если локальные определения параметров безопасности должны быть применены также и на других различных системах, администраторы могут извлечь эти локальные изменения и импортировать их на другую систему.

Для того чтобы извлечь локальные изменения, используйте команду `semanage export`:

```
# semanage export -f local-mods.conf
```

Файл, где теперь хранятся локальные изменения параметров безопасности (в приведенной выше команде это файл `local-mods.conf`), может быть скорректирован по усмотрению. Например, администраторы могут удалить все строки, кроме тех, которые они хотят применить на других системах.

Сохранив в файле локальные изменения, можно переместить файл на другие системы и импортировать их следующим образом:

```
# semanage import -f ./local-mods.conf
```

Импортируемые установки немедленно регистрируются. Конечно, в случае изменений параметров безопасности для объектов файловой системы (`semanage fcontext`) не забывайте запускать команду `restorecon` для каталогов, которые содержат эти объекты.

## 4.4. ИЗМЕНЕНИЕ ПАРАМЕТРОВ БЕЗОПАСНОСТИ У ФАЙЛОВ

Теперь мы знаем, как устанавливать параметры безопасности SELinux и управлять ими с помощью таких инструментов, как `chcon` и `restorecon` – приложение, которое запрашивает перечень параметров безопасности SELinux для понимания того, какие именно должны быть у какого-нибудь файла. Но не только `restorecon` является приложением, которое использует в работе этот список параметров.

### 4.4.1. Использование команд `setfiles`, `rlpkg` и `fixfiles`

Использование команд `semanage fcontext` и `restorecon` является предпочтительным методом для изменения параметров безопасности файлов, но существуют и другие инструменты, которые воздействуют на них в системе.

Приложение `setfiles` является довольно старым и требует указания пути непосредственно к файлу, содержащему перечень начальных параметров безопасности при сбросе текущих. Несмотря на то что оно часто используется в составе других программ, большинству администраторов не нужно больше вызывать `setfiles` напрямую, как это показано ниже:

```
# setfiles /etc/SELinux/targeted/contexts/files/file_contexts /srv/www
```

Другой набор инструментов – это `rlpkg` (в Gentoo) и `fixfiles` (в Red Hat Enterprise Linux). Они оба имеют замечательную особенность: могут использоваться для переустановки параметров безопасности файлов на начальные значения для конкретного приложения, вместо того чтобы выполнять итерации над файлами этого приложения вручную и запускать затем для них команду `restorecon`. В следующем примере мы используем эти инструменты для восстановления параметров безопасности у файлов, входящих в состав пакета `openssh`:

```
# rlpkg openssh
# fixfiles -R openssh restore
```

Другой особенностью обоих инструментов является то, что они могут использоваться для изменения параметров безопасности на всей файловой системе без необходимости производить перезагрузку системы. И делается это следующим образом:

```
# rlpkg -a -r
# fixfiles -f -F relabel
```

### 4.4.2. Изменение параметров безопасности на всей файловой системе

Использование команд `rlpkg` и `fixfiles` – не единственные предусмотренные варианты для изменения параметров безопасности на всей файловой системе, когда мы работаем с дистрибутивом Red Hat Enterprise Linux (или его производными). Есть два других метода, позволяющих сказать системе, что надо выполнить операцию приведения всех параметров безопасности к исходному состоянию (*relabeling*) для объектов файловой системы во время (пере)загрузки: выполнить команду `touch` для создания специального файла или указать в параметрах загрузки ядра.

Файл, который формируется командой `touch` и определяет необходимость приведения параметров к исходному состоянию, называется `.autorelabel`, и его следует размещать в корне файловой системы. После выполнения команды системе требуется перезагрузить:



```
# touch /.autorelabel
# reboot
```

Такое же поведение инициируется, если в перечень параметров загрузки ядра добавлен параметр `autorelabel` (подобно тому, как параметры `selinux=` и `enforcting=` могут быть установлены, что мы обсуждали в главе 2 «Режимы работы и регистрация изменений»).

Приведение параметров к исходному состоянию во всей файловой системе займет какое-то время. По окончании система перезагрузится снова. Если была использована команда `touch` для запуска операции приведения в соответствие параметров безопасности, то файл `.autorelabel` будет удален автоматически.


### 4.4.3. Автоматическое приведение к заданным значениям изменившихся параметров безопасности

Параметры безопасности также могут быть применены фоновым процессом `restorecond`. В его задачи входит использование правил из перечня [регулярных] выражений и задействование их по отношению к объектам из списка, заданным в файле `/etc/SELinux/restorecond.conf`.

Следующий набор файлов и каталогов представляет собой примерный список объектов, сформированных в файле `restorecond.conf` таким образом, что фоновый процесс `restorecond` автоматически присваивает параметры безопасности SELinux этим файлам и каталогам каждый раз, когда обнаруживает изменения в параметрах безопасности:

```
/etc/resolv.conf
/etc/mntab
/var/run/utmp
/root/*
~/public_html
~/mozilla/plugins/libflashplayer.so
```

В случае когда были произведены такие изменения в каком-нибудь файле, который соответствует любому из данного списка объектов, процесс `restorecond` будет оповещен об этом (через Linux-подсистему уведомлений об изменениях в файловой системе **inotify**) и затем выполнит приведение параметров безопасности в соответствии с правилами [регулярных] выражений.

 Процесс `restorecond` в основном используется по историческим причинам, потому что было время, когда SELinux не поддерживал именованные файловые преобразования (*named file transitions*). Запись в файл `resolv.conf` могла быть неотличима от записи в файл `passwd`, находящийся в одном каталоге `/etc`. Привнесение в SELinux такой функциональной возможности, как именованные файловые преобразования, значительно снизило необходимость в использовании процесса `restorecond`.

## 4.5. ПАРАМЕТРЫ БЕЗОПАСНОСТИ ПРОЦЕССА

Поскольку все в системе SELinux маркируется параметрами безопасности, то даже процессы маркируются типом допустимого для них набора функций (также известного как *домен*).

### 4.5.1. Получение параметров безопасности процесса

Мы посмотрели, что веб-сервер Apache выполняется с набором функций, имеющим название `httpd_t`. Об этом можно узнать, если выполнить команду `ps -eZ`, следующим образом:

```
# ps -eZ | grep httpd
system_u:system_r:httpd_t:s0 2270 ?          00:00:00 httpd
```

Наряду с этим способом существует несколько других путей, для того чтобы получить параметры безопасности процесса. И хотя метод с помощью команды `ps` наиболее очевиден, другие методы могут доказать свою пользу при написании скриптов или в службах постоянного наблюдения.

Первый метод заключается в чтении псевдофайла `/proc/<pid>/attr/current`, с которым мы уже сталкивались ранее в этой книге. Он отображает текущие параметры безопасности процесса:

```
# pidof httpd
1952 1951 1950 1949 1948 1947
# cat /proc/1952/attr/current
system_u:system_r:httpd_t:s0
```

Для получения информации в более удобочитаемом виде лучше использовать команду `secon` для того же **идентификатора процесса** (process ID – PID):

```
# secon --pid 1952
user: system_u
role: system_r
type: httpd_t
sensitivity: s0
clearance: s0
mls-range: s0
```

**P** Для русскоязычного варианта данное сообщение могло бы быть представлено следующим видом:

```
# secon --pid 1952
тип SELinux-пользователя: system_u
роль: system_r
тип допустимого набора функций: httpd_t
текущий мандатный уровень: s0
максимальный мандатный уровень допуска: s0
диапазон многоуровневой защиты: s0
```

Наконец, в рамках проекта пользовательского окружения SELinux имеется вспомогательная утилита, которая называется `getpidcon` и предоставляется

в рамках библиотеки `libselinux`. Хотя эта утилита недоступна в дистрибутиве Red Hat Enterprise Linux, другие дистрибутивы, такие как Gentoo, ее имеют. Утилите передается какой-нибудь один идентификатор процесса, и затем она возвращает параметры безопасности для него:

```
# getpidcon 1950
system_u:system_r:httpd_t:s0
```

В настоящее время процессы Apache сами по себе не сообщают SELinux, что им нужно выполняться с типом допустимого набора функций `httpd_t`. Для этого в политике SELinux существуют правила преобразования типов (*transition rules*), которые определяют, когда и как процессы выполняются с конкретным набором функций.

## 4.5.2. Преобразование типа процесса

Подобно тому, как уже видели на примере файлов, если процесс разветвляется и создает новый процесс, этот новый процесс по умолчанию наследует параметры безопасности породившего его процесса. В случае с веб-сервером основной процесс работает в границах возможностей, которые ему предоставляет тип допустимого набора функций `httpd_t`, так что все рабочие процессы веб-сервера, которые запускаются, наследуют этот тип (домен) от основного.

Для того чтобы отличался один процесс от другого, можно преобразовывать наименования из типов допустимого набора функций (доменов). Любое **преобразование домена** (*domain transition*) (также известное как **преобразование типа процесса** (*process transition*)) – это некое правило, которое сообщает системе защиты SELinux, что для ветвления процесса должен быть использован другой тип набора функций (в действительности это происходит на том этапе, когда родительский процесс вызывает уже функцию непосредственного выполнения другой программы `execve()`, производимую обычно после действий команды `fork()`).

Подобно тому, как это происходило с файлами в п. 4.2.2, правила преобразования типов процессов могут быть получены при помощи команды `sesearch`. Давайте посмотрим на те типы допустимого набора функций, которые разрешены для преобразования к типу `httpd_t`:

```
$ sesearch -T | grep "process httpd_t"
type_transition piranha_pulse_t httpd_exec_t : process httpd_t;
type_transition kdumpctl_t httpd_exec_t : process httpd_t;
type_transition initrc_t httpd_exec_t : process httpd_t;
...
type_transition init_t httpd_exec_t : process httpd_t;
```

В приведенном примере SELinux будет менять параметры безопасности веб-сервера на тип `httpd_t`, если его родительский процесс работает с одним из указанных типов (в частности, с типом `initrc_t`), а исполняемый файл `httpd` веб-сервера, который этот процесс запускает, имеет тип `httpd_exec_t`.

Но для того, чтобы это действительно произошло, должны существовать и некоторые другие разрешения. Следующий список описывает эти различные разрешения:

- исходному (*source*) родительскому процессу (такому как `initrc_t`) должно быть разрешено (*Allow*) преобразование к целевому (*target*) типу `httpd_t`, что регулируется правом (*privilege*) *transition* (*преобразование*) для класса (*class*) таких объектов, как *process* (*процесс*):

```
$ sestatus -s initrc_t -t httpd_t -c process -p transition -A
```

- исходному родительскому процессу (`initrc_t`) необходимо иметь право *execute* (*исполнения*) для запускаемого им объекта класса *file* (*файл*) с типом `httpd_exec_t`:

```
$ sestatus -s initrc_t -t httpd_exec_t -c file -p execute -A
```

- тип `httpd_exec_t` должен быть идентифицирован как некая отправная точка (*entry point*) для преобразования к типу `httpd_t`. Параметр `entrypoint` используется системой SELinux, для того чтобы гарантировать, что какое-либо преобразование типа процесса произойдет только тогда, когда этот конкретный параметр безопасности файла применяется к исполняемому файлу (в т. ч. скрипту):

```
$ sestatus -s httpd_t -t httpd_exec_t -c file -p entrypoint -A
```

- целевой тип допустимого набора функций должен быть разрешен для роли, которая характерна для исходного родительского процесса. Для системных фоновых процессов роль называется `system_r`:

```
$ seinfo -rsystem_r -x | grep httpd_t
```

Графическое изображение описанных разрешений представлено на рис. 4.1.

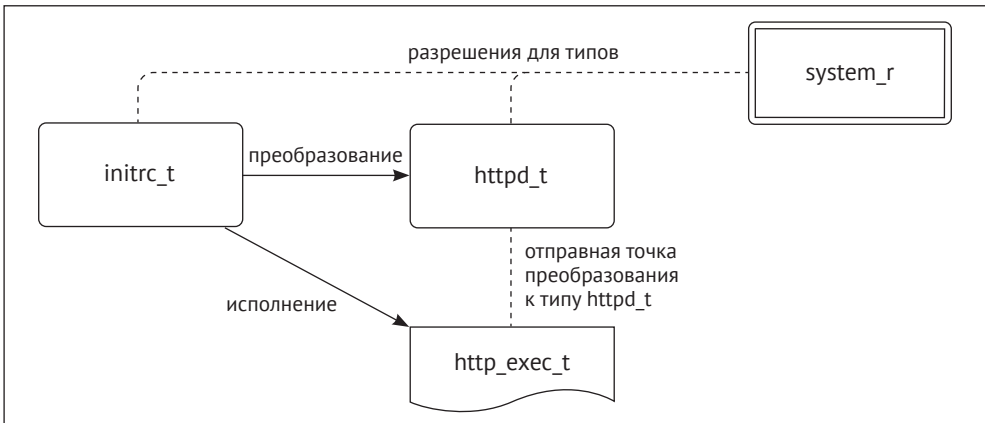


Рис. 4.1 ❖ Графическое представление разрешений, участвующих в успешном преобразовании из одного типа в другой

Только когда все эти разрешения соблюдены, будет выполнено преобразование типа процесса. Если нет, тогда не удастся запустить приложение (если у типа родительского процесса нет прав на выполнение (execute) целевого файла или же есть право на выполнение, но без права преобразования типа – `execute_no_trans`). Или приложение все-таки будет запущено, но с тем типом допустимого набора функций, который характерен для родительского процесса.

Преобразования типов процессов – это важная концепция, поскольку они информируют администратора о том, как приложение оказывается с теми параметрами безопасности, которые обеспечивают ему конкретные привилегии. Для того чтобы разобраться с этим, многие администраторы защиты смотрят на то, как один параметр безопасности может быть преобразован к другому. Об этом более подробно будет рассказано в главе 9 «Анализ поведения политики».

Для тех, кто разрабатывает политику, принять решение о том, когда надо выполнить преобразование типа процесса, а когда сохранять тип процесса в соответствии с типом родительского процесса, – это вопросы проектирования системы в целом. В большинстве случаев разработчики политик стараются сохранять параметры безопасности родительского процесса такими, чтобы у него был набор необходимых и достаточных возможностей и при этом каждая дополнительная привилегия была бы поводом для рассмотрения необходимости преобразования к другому типу [допустимого набора функций], которое имеет эту конкретную привилегию. Другими словами, преобразование типа является необходимым тогда, когда запускаемому приложению требуется больше разрешений или иных разрешений по сравнению с разрешениями, которые есть у родительского процесса.

Вот, кстати, почему тип допустимого набора функций `unconfined_t`, с которым выполняются пользовательские приложения, имеет немного преобразований типа процесса, по сравнению с более ограниченными типами, как `user_t` или `guest_t`: тип `unconfined_t` уже содержит много привилегий, поэтому преобразование к какому-то другому типу имеет небольшое значение. Стоит отметить, что это решение о преобразовании типа процесса принимается либо авторами конкретных политик, либо на уровне политик дистрибутивов Linux, а никак не самой технологией SELinux. Все, что делает система SELinux, – это соблюдает правила политики.

### 4.5.3. Проверка соответствия параметров безопасности

В рамках уже выполняющегося приложения (например, командной оболочки) политика SELinux может заставить запускаться команду с другим типом допустимого набора функций, чем тип самого приложения. И хотя, чтобы узнать, с каким именно мы могли бы начать узнавать обо всех правилах с помощью `sesearch`, есть более простая утилита, которая сообщает нам, какие будут параметры безопасности у конкретного объекта, после того как мы выполним интересующую нас команду или скрипт.

Утилита эта называется `selinuxexeccon` и предоставляется вместе с пакетом `libselinux-utils` в дистрибутиве Red Hat Enterprise Linux или `sys-libs/libselinux` в Gentoo. Для выполнения ей требуется как минимум один аргумент (путь исполняемого файла или скрипта, который должен быть выполнен), и дополнительно можно указать второй (параметры безопасности родительского процесса).

Например, чтобы выяснить, к какому типу допустимого набора функций будет относиться команда `passwd`, после того как будет запущена из приложения с текущими параметрами безопасности, нужно использовать эту утилиту следующим образом:

```
# selinuxexeccon /usr/bin/passwd
unconfined_u:unconfined_r:passwd_t:s0
```

Приведенная ниже команда позволит выяснить, какой тип допустимого набора функций будет присвоен веб-серверу при его запуске из приложения, имеющего тип `init_t`:

```
# selinuxexeccon /usr/sbin/httpd system_u:system_r:init_t:s0
system_u:system_r:httpd_t:s0
```

#### 4.5.4. Другие способы преобразования типов

Преобразования типа процесса на основе регулярных выражений являются наиболее часто встречающимися преобразованиями в SELinux, но существуют также и другие.

Например, некоторые приложения (такие как `cron` или `login`) поддерживают функции SELinux и будут сами определять, какой тип [допустимого набора функций] процесса в какой преобразовывать. Эти приложения вызывают метод `setexeccon()`, для того чтобы задать тип у какого-либо процесса, и при этом они не пользуются правилами преобразования типа. Другие же требования привилегий, как бы то ни было, остаются в силе.

Некоторые приложения, поддерживающие функции SELinux, даже способны изменять свои текущие параметры безопасности (а не только параметры того приложения, которые исполняют). Для того чтобы так делать, тип [допустимого набора функций], с которым запущено приложение, должен иметь разрешение `dyntransition` (сокр. от *dynamic transition* – динамическое преобразование). Это одно из разрешений, имеющих для действий на уровне процессов). В качестве примера можно привести приложение OpenSSH, которое по умолчанию запускается с типом `sshd_t`, но может быть преобразовано к типу `sftpd_t`.

#### 4.5.5. Изначально заданные параметры в структуре идентификатора безопасности

Если маркировка параметром безопасности отсутствует (или недействительна), система SELinux будет рассматривать такой процесс как имеющий тип `unlabeled_t`. Это потому, что тип `unlabeled_t` определен для такого вида объек-

тов, как «файл», в качестве изначального (*initial*) значения, заданного в структуре **идентификатора безопасности** (*security identifier – SID*).

Изначальные параметры для различных идентификаторов безопасности могут быть получены при помощи команды `seinfo`:

```
# seinfo --initialsid -x
Initial SID: 27
  devnull: system_u:object_r:null_device_t:s0
  scmp_packet: system_u:object_r:unlabeled_t:s0
  ...
  file: system_u:object_r:unlabeled_t:s0
  kernel: system_u:system_r:kernel_t:s0
```

## 4.6. ОПРЕДЕЛЕНИЕ ГРАНИЦ ВОЗМОЖНЫХ ПРЕОБРАЗОВАНИЙ

Из соображений безопасности системы Linux могут в определенных ситуациях ограничивать возможности процессов в получении повышенных привилегий или вводить дополнительные ограничения, для того чтобы снизить использование в будущем потенциальных уязвимостей. Разработчики SELinux тоже придерживаются таких взглядов.

### 4.6.1. Очистка переменных окружения во время преобразования к другому типу

Когда выполняется команда с более высоким уровнем привилегий (например, это может быть приложение с установленным атрибутом `setuid` (позволяющим обычному пользователю запустить его с правами владельца этого приложения) или какое-то такое приложение, у которого дополнительные возможности добавляются во время сессии), стандартная библиотека языка C – **glibc (GNU C Library)** – будет очищать среду окружения такой команды. Это означает, что если переменные окружения способны негативно повлиять на работу системы в случае утечки или модификации, то они будут удалены, для того чтобы можно было бы уверенно защититься от воздействия через них атакующих или вредящих злоумышленников либо программ.

Такое безопасное исполнение файла контролируется параметром `AT_SECURE` из набора параметров вспомогательного вектора (*auxiliary vector*), который используется при обработке файлов, созданных в формате **ELF (Executable and Linking Format – формат сборки и выполнения)**. Когда этот параметр установлен, такие переменные окружения, как `LD_PRELOAD`, `LD_AUDIT`, `LD_DEBUG`, `TMPDIR` и `NLSPATH`, удаляются из рабочей сессии файла.

**P** Когда программа выполняется, она получает информацию от операционной системы о среде, в которой работает. Форма этой информации представляет собой таблицу, состоящую из пар ключ/значение, где ключи взяты из набора значений «AT\_», находящихся в файле `elf.h`. Некоторые данные предоставляются ядром для использования в `libc` и могут быть получены обычными интерфейсами, такими как `sysconf` (функцией, которая считывает информацию о настройках во время работы системы). Тем не менее в за-

висимости от платформы может существовать информация, которая недоступна другим способом [см. описание для вспомогательных векторов в документации проекта GNU C библиотеки [https://www.gnu.org/software/libc/manual/html\\_node/Auxiliary-Vector.html](https://www.gnu.org/software/libc/manual/html_node/Auxiliary-Vector.html)]. Код загрузчика считывает информацию из заголовка исполняемого файла, созданного в формате ELF, и на ее основании определяет, какие библиотеки требуются данной программе для работы. После чего динамический линковщик приводит в соответствие указатели адресов приложения с указателями адресов загруженных библиотек, в результате этих действий приложение может начать работу.

Вспомогательный вектор – это средство, которое загрузчик исполняемых файлов формата ELF (работающий на уровне ядра ОС) использует для передачи определенной информации в пространство пользователя при выполнении какой-либо программы. Каждая запись во вспомогательном векторе состоит из пары значений: тип того идентификатора, который данная запись представляет, и значение для этого типа.

Один из типов – это `AT_SECURE`: он имеет ненулевое значение, если выполняемая программа должна быть обработана безопасно. Когда это значение не равно нулю, динамический компоновщик отключает использование основных переменных окружения [более подробно см. описание на страницах руководства Linux <http://man7.org/linux/man-pages/man3/getauxval.3.html>].

`LD_PRELOAD` – переменная окружения, определяющая список дополнительных, определенных пользователем, общедоступных ELF-объектов, которые должны быть загружены раньше других.

`LD_AUDIT` – переменная окружения, определяющая список определенных пользователем общедоступных ELF-объектов, которые должны быть загружены раньше других в отдельном пространстве компоновщика. Такие объекты могут быть использованы для того, чтобы выполнять регистрацию событий, связанных с действиями компоновщика.

`LD_DEBUG` – переменная окружения, позволяющая вывести подробную информацию о работе динамического компоновщика [более подробно см. описание на страницах руководства Linux <http://man7.org/linux/man-pages/man8/ld.so.8.html>].

`TMPDIR` – переменная окружения, предоставляющая путь к каталогу, доступному для программ, которым требуется место для создания временных файлов.

`NLSPATH` – переменная окружения, которая должна содержать последовательность шаблонов, которую функция `catopen()` использует при попытке найти каталоги сообщений. Сама же функция `catopen()` открывает заданный в шаблоне каталог сообщений и возвращает дескриптор каталога [более подробно см. описание на страницах руководства Linux [https://pubs.opengroup.org/onlinepubs/9699919799/basedefs/V1\\_chap08.html](https://pubs.opengroup.org/onlinepubs/9699919799/basedefs/V1_chap08.html)].

Система SELinux будет также проводить принудительную очистку при преобразовании одного типа [допустимого набора функций] процесса к другому типу, для того чтобы гарантировать, что при работе с новым набором функций будет отсутствовать доступ к этим уязвимым переменным окружения. Правда, иногда вновь присвоенный тип требует, чтобы эти переменные все же присутствовали с исходными значениями (не все же типы допустимого набора функций могут быть уязвимы через них, так что безусловный сброс значений переменных окружения может привести к тому, что какие-то типы процессов будут бесполезны и неприменимы для дальнейшей работы).

Для того чтобы выполнить преобразование типа процесса без очистки переменных окружения, следует явно использовать параметр `noatsecure`, приме-



няемый по отношению к целевому процессу. Например, когда выполняется какой-нибудь подключаемый модуль браузера Firefox (который в результате преобразования получает тип `mozilla_plugin_t`), то переменные окружения должны сохраняться для его корректной работы. Поэтому у его преобразования имеется установленный параметр `noatsecure`:

```
# sestatus -t mozilla_plugin_t -p noatsecure -A
Found 4 semantic av rules:
  allow xguest_t mozilla_plugin_t : process { ... noatsecure };
  allow staff_t mozilla_plugin_t : process { ... noatsecure };
  allow user_t mozilla_plugin_t : process { ... noatsecure };
  allow unconfined_t mozilla_plugin_t : process { ... noatsecure };
```

#### 4.6.2. Невыполнение преобразований, когда нет ограничивающего родительского типа

Второй особенностью, влияющей на безопасность и поддерживаемой в ОС Linux, является подключение файловой системы с опцией `nosuid` (которая запрещает операции с битами `suid` и `sgid`, позволяющими пользователю выполнять программу с правами ее владельца, а не с правами пользователя, запустившего ее). Когда эта опция установлена, никакие атрибуты файлов `setuid` и `setgid` на этой файловой системе не будут влиять на то, с каким идентификатором пользователя и группы будет выполняться сессия. Другими словами, на файловой системе, подключенной с опцией `nosuid`, программа, имеющая установленный атрибут `setuid`, будет работать точно так же, как когда работала бы без него.

Для SELinux любой файл, содержащий исполняемый код с параметрами безопасности, которые должны привести к преобразованию типа процесса, действительно приведет к этому преобразованию только тогда, когда тип, к которому выполняется преобразование, сам ограничен родительским типом допустимого набора функций. Если же он не имеет никаких границ, тогда преобразование типа процесса не произойдет и сессия останется с текущими параметрами безопасности (или же команда не будет исполнена, когда приложению не позволено работать с текущими параметрами безопасности).

Ограничивающий тип не вычисляется тут же, исходя из имеющихся прав доступа. У SELinux есть четкое правило, которое ограничивает целевой тип неким родительским типом допустимого набора функций. Даже если разрешения будут добавлены позже в ограниченный тип [допустимого набора функций] процесса, то они будут отклонены подсистемой защиты SELinux, поскольку они не являются частью родительского типа процесса.

Чтобы просмотреть имеющиеся ограниченные типы, можно использовать команду `seinfo`. Эта возможность представлена в наборе инструментов `setools` начиная с 4-й версии, и поэтому в дистрибутивах, вышедших раньше, ее не должно быть. Для ОС Gentoo эта команда доступна в таком формате:

```
# seinfo --typebounds
Typebounds: 1
  typebounds mozilla_t mozilla_plugin_t;
```

### 4.6.3. Использование флага, исключающего новые привилегии у процесса

Использование файловых систем, подключенных с опцией `nosuid`, – это особый случай поддержки концепции «**Никаких новых привилегий**» (No New Privilege – NNP) в операционной системе Linux. Данная концепция реализуется через свойственный процессам атрибут, который сообщает ядру Linux, что процесс не должен обладать более никакими дополнительными правами. Начиная с этого момента, способы ограничения, упомянутые ранее, актуализируются, и SELinux станет позволять преобразования типов процессов, только если они происходят в отношении ограниченного типа [допустимого набора функций] процесса.

Этот атрибут может быть установлен самими приложениями при помощи функции управления процессами `prctl()`, но пользователи могут тоже оказывать воздействие на него. Команда `setpriv` может быть использована для запуска приложений с установленным параметром `PR_SET_NO_NEW_PRIVS` (который приложения могут передавать через функцию `prctl()` в качестве ее аргумента).

Для примера, в домашнем каталоге текущего пользователя создадим на языке программирования Python простую программу, взаимодействующую с веб-сервером через общий шлюзовый интерфейс – CGI (*Common Gateway Interface*):

```
$ mkdir ~/cgi-bin
$ cat > ~/cgi-bin/test.py << EOF
#!/usr/bin/env python
import sys, time
import subprocess
import cgi, cgitb
cgitb.enable()
print 'Content-Type: text/html;charset=utf-8\n'
PIPE = subprocess.PIPE
STDOUT = subprocess.STDOUT
pd = subprocess.Popen(['ping', '-c', '1', 'localhost'], stdout=PIPE,
stderr=STDOUT)
while True:
    output = pd.stdout.read(1)
    if output == '' and pd.poll() != None:
        break
    if output != '':
        sys.stdout.write(output)
        sys.stdout.flush()
EOF
```

Теперь, когда этот CGI-скрипт нам доступен, запустим простой веб-сервер, поддерживающий CGI (мы укажем ему порт 6020, поскольку не имеющие привилегий пользователи должны иметь возможность связываться с ним процессы), и присоединимся к нему:

```
$ python -m CGIHTTPServer 6020
```

Присоединимся к веб-серверу из другой сессии и вызовем (созданный выше) CGI-скрипт `test.py`:

```
$ curl http://localhost:6020/cgi-bin/test.py
PING localhost (127.0.0.1) 56(84) bytes of data
64 bytes from localhost (127.0.0.1): icmp_seq=1 ttl=64 time=0.002 ms

-- localhost ping statistics --
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.002/0.002/0.002/0.000 ms
```

Теперь запустим такой же веб-сервер (поддерживающий общий шлюзовый интерфейс), но с включенным атрибутом `NNP`, исключающим получение новых привилегий:

```
$ setpriv --no-new-privs python -m CGIHTTPServer 6020
```

Снова присоединимся к веб-серверу и вызовем CGI-скрипт `test.py`:

```
$ curl http://localhost:6020/cgi-bin/test.py
ping: icmp open socket: Permission denied
```

Поскольку атрибут `NNP` системы Linux был включен, команда `ping` не смогла получить большие привилегии, необходимые для открытия сетевого сокета.

Иногда политика SELinux даже не позволяет приложению выполняться без преобразования типа [допустимого набора функций] (домена). В этом случае появится отказ в доступе `execute_no_trans`:

```
type=AVC msg=audit(1150125191.592:740): avc: denied
{ execute_no_trans } for pid=2793 comm="pipe"
name="PostFix.mail.SpamAssassin.spamfilter.sh" dev=md9 ino=56842
scontext=system_u:system_r:postfix_pipe_t:s0
tcontext=system_u:object_r:ql_spamassassin_client_exec_t:s0
tclass=file permissive=0
```

## 4.7. Типы, РАЗРЕШЕНИЯ И ОГРАНИЧЕНИЯ

Теперь, когда мы знаем больше о типах допустимых наборов функций (как для процессов, так и для файлов и других ресурсов), давайте посмотрим более внимательно на то, как они используются в политике SELinux.

### 4.7.1. Объяснение атрибутов типа

Мы уже обсуждали приложение `sesearch` и то, как оно может быть использовано для запроса данных о текущей политике SELinux. Давайте еще раз посмотрим на преобразование типа у процесса:

```
$ sesearch -s initrc_t -t httpd_t -c process -p transition -A
Found 1 semantic av rules:
allow initrc_domain daemon : process transition ;
```

Даже когда мы запросили правила, позволяющие (allow) родительским процессам с типом `initrc_t` выполнять преобразование типа для объекта [файла], имеющего изначальный тип [допустимого набора функций] `httpd_t`, то получили в результате правило для родительского процесса с типом `initrc_domain` и целевого объекта с типом `daemon`. В итоге команда `sesearch` показала нам, как интересующее разрешение предоставляется системой защиты SELinux, но показала, используя атрибуты, назначенные типам `initrc_t` и `httpd_t`.

Атрибуты типа (*type attributes*) в SELinux используются того, чтобы группировать различные типы и назначать привилегии этим группам, вместо того чтобы присваивать их каждому типу отдельно. При помощи команды `seinfo` можно увидеть, какие типы снабжены меткой со значением `initrc_domain` в качестве атрибута:

```
$ seinfo -ainitrc_domain -x
initrc_domain
  piranha_pulse_t
  initrc_t
  openshift_initrc_t
  kdumplib_t
  init_t
  glusterd_t
  cluster_t
  condor_startd_t
```

Как мы можем видеть, тип `initrc_t` – действительно один из числа тех, которые объединены атрибутом `initrc_domain`. Аналогично атрибут `daemon` назначается нескольким типам (даже нескольким сотням). Таким образом, единственное разрешающее правило, указанное выше, объединяет в себе более чем тысячу правил (по несколько сотен разрешающих правил для каждого из указанных ранее восьми типов, относящихся к атрибуту `initrc_domain`).

Атрибуты используются в политике все больше и больше как способ упрощения разработки политик за счет укрупнения правил. С помощью команды `seinfo -a` вы можете получить общие сведения обо всех атрибутах, поддерживаемых текущей политикой.

#### 4.7.2. Запрос разрешений, предоставленных типу процесса

Большинство стандартных правил системы SELinux являются правилами разрешения (*allow rules*), информирующими подсистему SELinux о том, какие права предоставлены типу процесса (домену). Правила разрешения имеют следующий синтаксис:

```
allow <source> <destination> : <class> <permissions>;
```

Поле `<source>` [«источник»] – это почти всегда домен (тип допустимого набора функций процесса), тогда как поле `<destination>` [«целевой объект»] может отражать тип как процесса, так и других объектов доступа.

Поле `<class>` [«класс»] позволяет нам различать привилегии по виду ресурса, будь то обычный файл, каталог, TCP-сокеты или что-нибудь другое. Полные све-

дения о поддерживаемых классах могут быть получены при помощи команды `seinfo -c`. Каждый класс имеет свой собственный набор разрешений, которыми SELinux управляет. Например, класс `sem` (используемый для семафорного доступа (*semaphore access*)) имеет следующие разрешения, связанные с ним:

```
$ seinfo -csem -x
sem
  associate
  create
  write
  unix_read
  destroy
  getattr
  setattr
  read
  unix_write
```

В поле `<permissions>` [`<разрешения>`] большинство правил будут объединять допустимые действия в наборы при помощи фигурных скобок (`{}`):

```
allow user_t etc_t : file { ioctl read getattr lock execute
                          execute_no_trans open };
```

Такой синтаксис позволяет авторам политик создавать детальные и конкретизированные правила управления доступом. Мы можем использовать команду `sesearch` для получения списка этих правил. Чем больше уточняющих параметров мы укажем в команде для вывода правил `sesearch`, тем более детальными получатся наши результаты поиска. Например, `sesearch -A` предоставит нам все правила разрешения, действующие в настоящий момент.

Добавление наименования источника после параметра `-s` отфильтрует выходные данные по правилам разрешения, относящимся к указанному таким образом домену. Если добавить сведения о целевом объекте после параметра `-t`, то выборка будет еще более конкретная. Другие параметры, которые могут быть использованы для фильтрации вывода у правил разрешения (при использовании команды `sesearch`), – это класс (`-c`) и разрешения (`-p`).

Данный синтаксис также прекрасно позволяет учитывать информацию, предоставляемую в сообщениях об отказе в доступе, для внесения каких-либо изменений:

```
type=AVC msg=audit(1371993742.009:15990): avc: denied
{ getattr } for pid=31069 comm="aide"
path="/usr/lib64/postgresql-9.2/bin/postgres"
dev="dm-3" ino=803161 scontext=root:sysadm_r:aide_t
tcontext=system_u:object_r:postgresql_exec_t tclass=file
```

Если мы захотим исправить эту проблему с помощью добавления правила разрешения в политику SELinux, тогда оно будет выглядеть следующим образом:

```
allow aide_t postgresql_exec_t : file { getattr };
```

### 4.7.3. Рассмотрение наложенных ограничений

Однако конструкция правил разрешения учитывает только типы допустимого набора функций. Тем не менее иногда возникает потребность распределить определенные действия, основываясь на конкретных пользователях или ролях. В SELinux это возможно сделать при помощи наложенных ограничений (*constraints*).

Наложённые ограничения в SELinux – это правила, которые применяются к какому-либо классу [объектов] и к набору допустимых действий, которые должны быть расположены в определенном порядке, для того чтобы SELinux мог в дальнейшем обработать запрос. Рассмотрим следующее наложенное ограничение на преобразование типа процесса:

```
constrain process
{ transition dyntransition noatsecure siginh rlimitinh }
(
  u1 == u2 or
  (
    t1 == can_change_process_identity and
    t2 == process_user_target
  ) or (
    t1 == cron_source_domain and
    (
      t2 == cron_job_domain or
      u2 == system_u
    )
  ) or (
    t1 == can_system_change and
    u2 == system_u
  ) or (
    t1 == process_uncond_exempt
  )
);
```

Это наложенное ограничение говорит нам о том, что хотя бы одно из следующих правил должно быть верным, если разрешение на выполнение преобразования типа *transition*, разрешение на выполнение динамического преобразования типа *dyntransition* или любое другое из трех упомянутых выше разрешений потребуется процессу для выполнения:

- SELinux-пользователь исходного родительского процесса (*u1*) и SELinux-пользователь целевого объекта (*u2*) являются одним и тем же субъектом;
- тип [допустимого набора функций] у исходного родительского процесса (*t1*) должен иметь установленный атрибут `can_change_process_identity` [*<возможно изменение идентификатора процесса>*], а тип целевого объекта (*t2*) должен иметь установленный атрибут `process_user_target` [*<целевой объект пользовательского процесса>*];
- тип у исходного родительского процесса (*t1*) должен иметь установленный атрибут `cron_source_domain` [*<задающий планировщику задание родительский домен>*], а также тип целевого объекта (*t2*) должен иметь

в качестве атрибута `con_job_domain` [*<домен\_запускаемого\_планировщиком\_целевого\_объекта>*]; или же пользователем целевого объекта (`u2`) должна быть указана системная учетная запись SELinux – `system_u`;

- тип у исходного родительского процесса (`t1`) должен иметь установленный атрибут `can_system_change` [*<возможно\_изменение\_системы>*], а пользователем (`u2`) должна быть системная учетная запись SELinux – `system_u`;
- тип у исходного родительского процесса (`t1`) должен иметь установленный атрибут `process_uncond_exempt` [*<процесс\_свободный\_от\_наложенных\_ограничений>*].

Именно через наложенные ограничения реализован механизм контроля доступа, ориентированный на пользователя (UBAC):

```
u1 == u2
or u1 == system_u
or u2 == system_u
or t1 != ubac_constrained_type
or t2 != ubac_constrained_type
```

Мы можем вывести список включенных наложенных ограничений в данный момент, используя команду `seinfo -constrain`, но в выводе немедленно будут раскрыты атрибуты и будет использована постфиксная нотация записи, которая приводит к более неудобному пониманию при прочтении.

## 4.8. ЗАКЛЮЧЕНИЕ

В этой главе мы изучили, как параметры безопасности файлов сохраняются в виде расширенных атрибутов файловой системы и как мы можем управлять параметрами безопасности файлов и других ресурсов файловой системы. Затем мы выяснили, где система защиты SELinux содержит свои описания того, какие параметры безопасности должны быть присвоены каким файлам.

Мы также научились работать с инструментом `semanage` для манипулирования этой информацией и поработали с несколькими инструментами, которые используют эту информацию для включения принудительной защиты на ресурсах.

В первый раз мы посмотрели на политики SELinux, работающие на уровне процесса и определяющие, как какой-либо процесс выполняется в границах некоторого типа допустимого набора функций. Здесь же с помощью приложений `sesearch` и `seinfo` мы научились запрашивать политику SELinux. В конце изучили некоторые средства защиты механизмы ОС Linux, которые ограничивают возможности преобразования типов процессов.

В следующей главе мы расширим наши знания о защите операционной системы с помощью возможностей SELinux, связанных с сетью.

# Глава 5

## Контроль сетевого взаимодействия

Подсистема мандатного разграничения выходит далеко за пределы контроля доступа, ограниченного лишь защитой файлов и процессов. Одна из возможностей, предоставляемая системой защиты SELinux, – это контроль сетевых взаимодействий. Исходно механизм контроля доступа на уровне сокетов является основным механизмом, но и другие способы контроля также реализованы.

В этой главе мы будем:

- изучать, как система защиты управляет сетевым контролем доступа;
- рассматривать, что администраторы могут делать для того, чтобы способствовать усилению безопасности сетевых соединений, используя утилиту управления работой межсетевого экрана – «iptables»;
- описывать, как политики SELinux могут быть использованы для обеспечения межсистемной защиты при помощи маркированного протокола сетевого обмена IPsec.

Закончим мы эту главу вводными сведениями об использовании специальных параметров безопасности, применяемых для маркировки IP-пакетов и разработанных в рамках протокола CIPSO<sup>1</sup>, а также интеграции данного механизма с системой защиты SELinux.

### 5.1. От контроля межпроцессного взаимодействия (IPC) до сокетов базовых протоколов (TCP/UDP) ТРАНСПОРТНОГО УРОВНЯ

Linux-приложения взаимодействуют друг с другом или напрямую, или через сеть. Но, с точки зрения прикладного программиста, разница между прямым

---

<sup>1</sup> Commercial Internet Protocol Security Option – параметры безопасности коммерческого протокола интернет // <https://tools.ietf.org/pdf/draft-ietf-cipso-ipsecurity-01.pdf>. – Прим. перев.



и сетевым взаимодействиями не так уж и велика. Давайте рассмотрим различные методы взаимодействия, поддерживаемые в Linux, и то, как система защиты SELinux согласуется с ними.

### 5.1.1. Использование разделяемой памяти

Использование разделяемой памяти (*shared memory*) – метод, который меньше всего похож на сетевое взаимодействие. Приложения могут предоставлять для совместного использования конкретные области памяти друг для друга и использовать эти разделяемые сегменты для взаимодействия между двумя (и более) процессами. Для управления доступом к разделяемой памяти программисты прикладного программного обеспечения могут использовать семафорные механизмы **взаимного исключения** – **мьютексы** (*mutexes*) или **семафоры** (*semaphores*). Семафор – это нечто целое, способное на минимальное значение увеличиваться или уменьшаться (с целью обеспечить работу двух приложений таким образом, чтобы они не перезаписали доступное каждому из них некое значение без ведома другого), в то время как мьютекс может быть интерпретирован как специального вида семафор, который только принимает значения 0 или 1.

В операционной системе Linux предусмотрено два вида реализации для контроля и доступа разделяемой памяти: модели POSIX<sup>1</sup> и SysV<sup>2</sup>. Мы не будем задерживаться на описании достоинств и недостатков каждой, а просто посмотрим, как система защиты SELinux управляет доступом в рамках этих моделей.

SELinux контролирует обмен данными, выполняемый в рамках модели SysV, посредством специально предусмотренных классов: 'sem' для семафоров и 'shm' для разделяемой памяти. Семафоры, мьютексы и сегменты разделяемой памяти получают такие же параметры безопасности, какие имеются у первого процесса, который создает их.

В качестве примера отказа в доступе, связанного с семафором модели SysV, и в рамках неудачного взаимодействия с совместно используемой памятью можно привести следующее сообщение системы защиты SELinux:

```
type=AVC msg=audit(1443147735.370:444): avc: denied
{ unix_read unix_write } for pid=1454 comm="gnome-shell"
key=17908779 scontext=system_u:system_r:xdm_t:s0-s0:c0.c1023
tcontext=system_u:system_r:xserver_t:s0-s0:c0.c1023
tclass=sem permissive=0
```

Администраторы, которые хотят управлять элементами, предназначенными в модели SysV для обмена данными, могут использовать различные вариации команды `ipcs*`:

- 1) с помощью команды `ipcs` все элементы (такие как сегменты разделяемой памяти и семафоры) могут быть выведены списком;

<sup>1</sup> POSIX – от англ. *portable operating system interface* – переносимый интерфейс операционной системы.

<sup>2</sup> SysV – System V – одна из версий ОС Unix. – *Прим. перев.*

- 2) с помощью команды `ipcrm` они могут быть удалены;
- 3) с помощью команды `ipcrm` могут создаваться новые сегменты разделяемой памяти и семафоры.

Например, давайте сначала выведем перечень элементов, а затем удалим один семафор:

```
# ipcs
----- Message Queues -----
key      msqid  owner  perms  used-bytes  messages

----- Shared Memory Segments -----
key      shmid  owner  perms  bytes      nattch   status
0x01124612  0      root   600    1000       6        dest

----- Semaphore Arrays -----
key      semid   owner  perms  nsems
0x00000000  65536  apache  600    1
0x00000000  98305  apache  600    1
0x00000000  131074 apache  600    1
0x00000000  163843 apache  600    1
0x00000000  196612 apache  600    1
```

```
# ipcrm -s 98305
```

**P** Для русскоязычного варианта данный вывод мог бы быть представлен следующим видом:

```
# ipcs
----- Очереди сообщений -----
ключ      идентификатор очереди сообщений  владелец  разрешения  используемые байты  сообщения

----- Сегменты разделяемой памяти -----
ключ      идентификатор разделяемой памяти  владелец  разрешения  байты      количество текущих подключений  статус
0x01124612  0      root      600          1000       6          dest

----- Массивы семафоров -----
ключ      идентификатор семафора  владелец  разрешения  количество семафоров в наборе
0x00000000  65536  apache    600          1
0x00000000  98305  apache    600          1
0x00000000  131074 apache    600          1
0x00000000  163843 apache    600          1
0x00000000  196612 apache    600          1

# ipcrm -s 98305
```

Когда применяется POSIX-модель, то SELinux контролирует операции, связанные с семафорами, мьютексами и сегментами разделяемой памяти, посредством контроля доступа к файлам. В данном случае контролируются файлы, размещенные в директории `/dev/shm`, что является обычным делом для администратора в рамках контроля и управления.

### 5.1.2. Локальное взаимодействие, осуществляемое по каналам

Вторым большим семейством методов взаимодействия в операционных системах является использование программных каналов (*pipe*). Оправдывая свое название, каналы в основном являются однонаправленными тоннелями, с информацией, перемещающейся от одного (или более) отправителя к одному получателю (существуют исключения этого, например такое, как в операционной системе Solaris, где реализованы двунаправленные каналы, но они не поддерживаются в Linux. Другое название, которое часто используется для программного канала, – FIFO («first-in, first-out» – «первым вошел – первым вышел»).

Мы имеем два типа каналов в операционной системе Linux: **анонимные каналы** (*anonymous pipe*), также известные как **неименованные каналы** (*unnamed pipes*), и **именованные каналы** (*named pipes*). Разница между ними в том, что именованный канал использует файл специального типа в имеющейся файловой системе и может быть по нему идентифицирован, в то время как анонимные каналы создаются между приложениями без представления в файловой системе.

В обоих случаях система SELinux будет рассматривать каналы как файлы класса `fifo_file`. Именованные каналы имеют свой путь в файловой системе и создаются командами `mknod` или `mkfifo` (или при помощи функции `mkfifo()`, когда канал создается в рамках работы программного обеспечения). Анонимные же каналы будут видны как часть файловой системы `pipefs`. Это псевдофайловая система, недоступная пользователям, но тем не менее позиционируемая как файловая система за счет абстрагирования на уровне ядра компонентом «**Виртуальная файловая система**»<sup>1</sup> (*virtual file system – VFS*).

Указанные ниже события являются двумя отказами в доступе, связанными с каналами: первый является именованным каналом с путем `/run/systemd/initctl/fifo`, а второй – анонимный. Стоит заметить, что второй представлен как часть устройства `pipefs`:

```

avc: denied { getattr } for pid=16755 comm="su"
  path="/run/systemd/initctl/fifo" dev="tmpfs" ino=822
  scontext=staff_u:sysadm_:sysadm_su_t:s0-s15:c0.c1023
  tcontext=system_u:object_r:initctl_t:s0
  tclass=fifo_file permissive=0
avc: denied { write } for pid=4445 comm="entrypoint.sh"
  path="pipe:[596879]" dev="pipefs" ino=596879
  scontext=system_u:system_r:svirt_lxc_net_t:s0:c845,c982
  tcontext=system_u:system_r:kernel_t:s0
  tclass=fifo_file permissive=0

```

С точки зрения политики SELinux управление доступом осуществляется на файловом уровне представления каналов (FIFO). Два процесса смогут взаимо-

<sup>1</sup> См. доп. статью М. Джонса «Анатомия файловой системы Linux». [Электронный ресурс] // <https://www.ibm.com/developerworks/ru/library/l-linux-filesystem/>, яз. рус.; <https://developer.ibm.com/tutorials/l-linux-filesystem/>, яз. англ. – *Прим. перев.*

действовать друг с другом при помощи канала (*FIFO file*), если их типы допустимого набора функций (домены) будет иметь корректный набор привилегий по отношению к файлу, представляющему этот канал.

Администраторы могут определить, какой процесс взаимодействует через каналы с другими процессами при помощи таких инструментов, как *lsof*, или путем формирования запроса к виртуальной файловой системе */proc* (являющейся частью от общего перечня файловых дескрипторов, содержащихся в каталоге */proc/<pid>/fd*). К примеру, следующая выдержка из перечня результатов команды *lsof* (*list of opened files – список открытых файлов*) показывает параметры взаимодействия, выполняемого через канал, между двумя связанными (в рамках работы почтовой программы *Postfix*) процессами (колонка *NODE* (номер индексного дескриптора) содержит идентификатор канала, являющегося одинаковым для каждого из них):

```
COMMAND PID USER   FD TYPE DEVICE SIZE/OFF  NODE  NAME
master 1320 root   5r FIFO 0,8   0t0    17553 pipe
qmgr   1345 postfix 92w FIFO 0,8   0t0    17553 pipe
```

**Р** Для русскоязычного варианта данный вывод мог бы быть представлен следующим видом:

ИМЯ ПРОЦЕССА	ИДЕНТИФИКАТОР ПРОЦЕССА	ПОЛЬЗОВАТЕЛЬ	ФАЙЛОВЫЙ ДЕСКРИПТОР	ТИП ФАЙЛА	УСТРОЙСТВО РАЗМЕЩЕНИЯ ФАЙЛА	РАЗМЕР ФАЙЛА	НОМЕР ИНДЕКСНОГО ДЕСКРИПТОРА	ИМЯ ФАЙЛА
master	1320	root	5r	FIFO	0,8	0t0	17553	pipe
qmgr	1345	postfix	92w	FIFO	0,8	0t0	17553	pipe

### 5.1.3. Обращение через сокеты домена UNIX

В каналах реализуется только однонаправленное взаимодействие, а тип взаимообратного взаимодействия между двумя процессами будет требовать двух каналов. Подобие же клиент-серверного взаимодействия посредством каналов реализовать проблемно. Для реализации более совершенных связующих потоков между процессами были введены **сокеты** (*sockets*).

Редкий администратор не знает, что такие сетевые протоколы транспортного уровня, как *TCP* и *UDP*, осуществляют взаимодействие через сокеты. Приложения могут связываться с каким-либо сокетом и ожидать входящих через него соединений либо использовать конкретный сокет для подключения к другим, удаленным сервисам. Но даже внутри одной операционной системы *Linux* могут использоваться специальные сокеты для содействия в организации потоков взаимодействия. Они называются **сокетами домена UNIX** (*UNIX domain sockets*).

Мы можем найти различие между двумя определениями сокетов, по аналогии с каналами: неименованные сокеты и именованные сокеты. И так же, как с каналами, различие заключается в их размещении – в пути, который используется для идентификации сокета. Именованные сокеты создаются в обычной файловой системе, в то время как неименованные являются частью псевдофайловой системы *sockfs*. Так же, как и в примерах, которые мы видели

ранее, сведения о сокетах могут быть запрошены через такие утилиты, как `lsof`, или в виде списка объектов, размещенных в каталоге `/proc/<pid>/fd`.

Кроме того, существует еще одно различие, касающееся сокетов домена UNIX, и им является вид взаимодействия, который позволен сокету домена UNIX. Они могут быть созданы как **дейтаграммные сокет** (*datagram sockets*) (когда данные, передаваемые в сокет, считываются в одного размера блоки заданного формата) или как **поток** (когда данные, передаваемые в сокет, могут быть считаны в блоки различного размера). Это различие влияет на правила политики SELinux.

В системе защиты SELinux, при взаимодействии через сокет домена UNIX двух процессов, требуется, чтобы типы допустимого набора функций обоих процессов имели необходимые (в зависимости от задач коммуникации) права – `open` (открыть), `read` (читать) и `write` (писать) – по отношению к конкретному типу файла сокета.

Дополнительно типу [допустимого набора функций] отправляющего данные процесса (клиента), как правило, требуются следующие привилегии по отношению к типу принимающей стороны («сервера»), которые зависят от вида взаимодействия через сокет:

- привилегия `connectto` (<подключаться к>) по отношению к классу `unix_stream_socket` принимающей данные стороны («сервера») (в случае взаимодействия через потоковый сокет);
- привилегия `sendto` (<передать в>) по отношению к классу `unix_dgram_socket` принимающей данные стороны («сервера») (в случае взаимодействия через дейтаграммный сокет).

Так же, как и в случае с каналами, регистрируемые сообщения с отказами в доступе будут показывать тип используемого сокета домена UNIX:

```

avc: denied { connectto } for pid=2597 comm="nginx"
  path="/home/git/gitlab/tmp/sockets/gitlab.socket"
  scontext=system_u:system_r:httpd_t:s0
  tcontext=system_u:system_r:initrc_t:s0
  tclass=unix_stream_socket permissive=0
avc: denied { read write } for pid=31230 comm="iptables"
  path="socket:[224507]" dev=sockfs ino=224507
  scontext=unconfined_u:system_r:iptables_t:s0
  tcontext=unconfined_u:system_r:fail2ban_t:s0
  tclass=unix_stream_socket permissive=0

```

### 5.1.4. Рассмотрение сокетов netlink

Особым случаем среди сокетов домена UNIX являются netlink-сокеты. Это сокеты, которые позволяют приложениям из пространства пользователя связываться и взаимодействовать с процессами ядра. В отличие от стандартных сокетов домена UNIX, у которых целевые параметры безопасности согласуются с параметрами владельца сокета, netlink-сокеты имеют всегда *локальный* характер параметров безопасности.

Другими словами, когда субъект доступа с типом [допустимого набора функций] `sysadm_t` хочет управлять информацией о маршрутизации на уровне ядра, он будет открывать взаимодействие с ядром через `netlink`-сокет маршрута (`netlink_route_socket`), который идентифицируется с помощью класса `netlink_route_socket`:

```
# sestatus -s sysadm_t -t sysadm_t -c netlink_route_socket -A
allow sysadm_t sysadm_t : netlink_route_socket { ioctl... nlmsg_read };
```

Поскольку приложения имеют довольно много функциональных возможностей, может оказаться так, что некоторые из этих функций больше не разрешены текущей политикой SELinux. Администраторам в этом случае необходимо обновить политику SELinux для разрешения какого-то конкретного взаимодействия с пространством ядра через предусмотренные для этого сокет.

Перечень таких поддерживаемых сокетов может быть получен из информации о `netlink` из страниц руководства:

```
# man netlink
```

Например, сокет `NETLINK_XFRM`, обеспечивающий интерфейс для управления защищенным соединением через протоколы IPsec, поддерживается SELinux-классом `netlink_xfrm_socket`.

### 5.1.5. Действия с сокетами протоколов TCP и UDP

Проходя далее по звеньям в цепи описываемых механизмов взаимодействия, мы доходим до коммуникации через сокеты TCP и UDP. В данном случае, в отличие от тех взаимодействий, которые напрямую устанавливаются между процессами (и, соответственно, между SELinux-типами [допустимого набора функций] процессов), потоки данных получают из TCP- или UDP-сокетов и записывают соответственно в TCP- или UDP-сокеты.

Система защиты будет присваивать также типы и TCP/UDP-портам, и эти типы потом должны будут применяться при взаимодействии через сокеты. При включенном SELinux приложение клиента, подключающееся к порту сервера доменных имен (порт TCP 53, которому присвоен тип `dns_port_t` в большинстве политик SELinux), использует разрешение `name_connect` для класса `tcp_socket`, нацеленное на типизированный порт. В случае когда используется протокол UDP (и, соответственно, класс `udp_socket`), разрешение `name_connect` не применяется. С другой стороны, фоновый процесс приложения использует привилегию `name_bind` для собственной связи с портом.

Администраторы могут четко настроить, какую метку какому TCP- или UDP-порту присвоить. Для этого может применяться команда `semanage port`. Например, для вывода списка сведений о текущих портах следует использовать эту команду:

```
# semanage port -l
SELinux Port Type      Proto  Port Number
adb_port_t             tcp    5037
```

```
afs3_callback_port_t tcp 7001
...
http_cache_port_t tcp 3128, 8080, 8118, 10001-10010
http_port_t tcp 80,443,488,8008,8009,8443
```

**P** Для русскоязычного варианта данный вывод мог бы быть представлен следующим образом:

```
# semanage port -l
SELinux-тип порта    Протокол  Номер порта
adb_port_t           tcp       5037
afs3_callback_port_t tcp       7001
...
http_cache_port_t    tcp       3128, 8080, 8118, 10001-10010
http_port_t          tcp       80,443,488,8008,8009,8443
```

В этом примере мы видим, что тип `http_port_t` присвоен набору портов TCP. Типам [допустимых наборов функций] процессов, относящихся к веб-серверу и которым разрешено соединяться с портом, имеющим тип `http_port_t`, таким образом позволено устанавливать связь с любым из перечисленных портов.

Чтобы позволить фоновому процессу (такому как сервер SSH) взаимодействовать с другими (или дополнительными) портами, нам необходимо сообщить системе защиты о маркировке этих портов соответствующей меткой. Чтобы позволить серверу SSH присоединиться к порту 10122, мы сначала должны проверить, назначена этому порту уже специализированная метка или нет. Это можно сделать путем использования команды `sepolicy` (являющейся составной частью пакета `policycoreutils-devel` в операционной системе Red Hat Enterprise Linux или `sys-apps/policycoreutils` в Gentoo):

```
# sepolity network -p 10122
10122: tcp unreserved_port_t 1024-32767
10122: udp unreserved_port_t 1024-32767
```

Метка `unreserved_port_t` не является специализированной, поэтому мы можем присвоить этому порту метку `ssh_port_t`:

```
# semanage port -a -t ssh_port_t -p tcp 10122
```

Удаление назначения порта работает аналогично:

```
# semanage port -d -t ssh_port_t -p tcp 10122
```

Когда специализированная метка порту уже присвоена, утилита будет выдавать следующую ошибку:

```
# semanage port -a -t ssh_port_t -p tcp 80
ValueError: Port tcp/80 already defined
```

**P** Для русскоязычного варианта данный вывод мог бы быть представлен следующим образом:

```
# semanage port -a -t ssh_port_t -p tcp 80
ОшибкаНазначения: Порт tcp/80 уже определен
```

Если это так и другой порт не может быть использован, тогда не остается ничего другого, кроме как изменить саму политику SELinux.

### 5.1.6. Вывод списка сетевых соединений с параметрами безопасности

Многие инструменты в арсенале администратора имеют возможность отображать информацию о параметрах безопасности. Так же как и с утилитами ядра, большинство данных инструментов используют опцию `-Z` для этого. Например, для вывода списка работающих сетевых служб может применяться команда `netstat`:

```
# netstat -nptZ | grep ':80'
tcp      0      0 0.0.0.0:80      0.0.0.0:*      LISTEN   2267/httpd \
        system_u:system_r:httpd_t:s0
```

Даже команда `lsof` при запросе отображает параметры безопасности:

```
# lsof -i :80 -Z | grep httpd
httpd    2267 system_u:system_r:httpd_t:s0 root    3u IPv4  15962 \
        0t0      TCP *:http (LISTEN)
```

Другой и более мощной для запроса данных о соединениях является команда `ss`. Просто вызов команды `ss` будет отображать все соединения текущей системы. При добавлении опции `-Z` она добавляет также и информацию о параметрах безопасности.

Например, следующая команда запрашивает TCP-службы, находящиеся в состоянии ожидания запроса (*listen*):

```
# ss -ltnZ
State Recv-Q Send-Q Local Address:Port Peer Address:Port
LISTEN 0      128      *:22          *: *          \
users:({"sshd",pid=747, \
proc_ctx=system_u:system_r:sshd_t:s0-s0:c0.c1023,fd=3))
```

Также могут быть выполнены более сложные запросы. Для получения большего количества информации об этом следует обратиться к странице руководства по команде `ss`.

## 5.2. МЕЖСЕТЕВОЙ ЭКРАН И МАРКИРОВКА СЕТЕВЫХ ПАКЕТОВ

Подход с организацией защиты на уровне TCP- и UDP-портов имеет несколько недостатков. Одним из них является отсутствие возможности контролировать подключения приложений, исходя из того, к какому удаленному сетевому узлу оно направлено. Кроме того, нет возможности ограничить фоновые процессы от соединения с любым сетевым интерфейсом: в множественно-ориентированной (*multi-homed*) ситуации, т. е. когда защищаемый узел подключен к нескольким физическим линиям связи, которые, в свою очередь, могут относиться как к одной, так и к различным сетям, мы можем быть уверенными в том,



что фоновый процесс взаимодействует только с интерфейсом, обращенным к внутренней сети, а не в интернет, или наоборот.

Раньше система защиты SELinux позволяла решать эту проблему привязки с помощью меток **interface** (<интерфейс>) и **node** (<узел>): какой-либо тип [допустимого набора функций] процесса может быть в этом случае привязан только к одному интерфейсу и больше ни к какому другому, даже к конкретному адресу (называемому узлом). Такое решение имело свои недостатки и в значительной степени устарело. Вместо него предпочтение отдается в пользу фильтрации и маркировки сетевых пакетов при помощи такого средства, как SECMARK.

### 5.2.1. Вводные сведения о межсетевом экране netfilter

Перед тем как объяснить, что такое SECMARK и как администраторы могут его контролировать, давайте сначала кратко коснемся Linux-подсистемы netfilter, которая является де-факто стандартом для локального выполнения функций межсетевого экранирования.

Так же, как и в случае с модулями безопасности в Linux (LSM), подсистема netfilter выполняет перехваты [сетевого потока данных] (*hooks*) на различных стадиях той структуры обработки сетевого стека, которая может затем быть реализована в одном или более модулях. Например, модуль `ip_tables` (который использует команду `iptables` как свою программу управления) является одним из них, а модули `ip6_tables` и `ebtables` являются примерами других модулей подсистемы netfilter. Когда реализуется логика обработки какого-либо перехвата [сетевого потока данных], тогда каждый модуль также сообщает системе netfilter о приоритете обработки для конкретного захвата данных. Это позволяет обеспечивать порядок при выполнении (т. к. многочисленные вызовы одного и того же захвата и могут, и будут использоваться вместе).

Стоит отметить, что `ip_tables` – это тот компонент инфраструктуры, который мы рассмотрим более детально, потому что он поддерживает метод SECMARK. Поскольку он обычно наиболее известен по имени своей программы управления `iptables`, то здесь и далее мы будем использовать наименование `iptables` в качестве основного наименования этого механизма.

Компонент `iptables` предлагает несколько таблиц, которые функционально ориентированы в рамках классификации и обработки сетевых взаимодействий:

- таблица `filter` (<фильтр>) включает стандартные возможности фильтрации сетевого потока данных;
- таблица `nat` (<преобразование сетевых адресов>) предназначена для изменения информации из пакетов, связанной с маршрутизацией, такой как адрес источника и/или назначения;
- таблица `mangle` (<изменения>) используется для изменения большинства полей пакетов;
- таблица `raw` (<необработываемое>) включается, когда администраторы хотят отказаться от некоторых пакетов/потоков из числа отслеживаемых соединений netfilter;

- таблица `security` (<таблица безопасности>) предлагает администраторам маркировать пакеты после обычной обработки.

В пределах каждой таблицы `iptables` предлагает начальный набор цепочек (*chains*). Эти цепочки, работающие по умолчанию, конкретизируют, где в обрабатываемом потоке (и, следовательно, в каком компоненте структуры `netfilter`, обеспечивающем перехват данных) должна быть выполнена обработка по заданным правилам. Каждая цепочка имеет политику, действующую по умолчанию, которая также по умолчанию является возвращаемым значением, если ни одно из правил в цепочке не подошло.

Внутри цепочки администраторы могут добавлять правила. Правила обрабатываются последовательно. Когда правило срабатывает, то выполняется конкретное действие. Это действие может либо разрешить пакету беспрепятственно проходить через конкретный захват [сетевого потока данных] в системе `netfilter`, либо запретить, либо передать на дополнительную обработку.

Обычно предоставляемыми цепочками являются (не все цепочки предлагаются для всех таблиц):

- цепочка `PREROUTING`, которая является первым этапом обработки пакета после его приема;
- цепочка `INPUT` для обработки пакетов, предназначенных для локальной системы;
- цепочка `FORWARD` для обработки пакетов, предназначенных для пересылки в другую удаленную систему;
- цепочка `OUTPUT` для обработки пакетов, исходящих из локальной системы;
- цепочка `POSTROUTING` обеспечивает последний шаг в обработке пакета перед его отправкой.

При сильном упрощении можно сказать, что реализация этих таблиц и их цепочек примерно соответствует приоритету вызовов в инфраструктуре подсистемы `netfilter`. Цепочки являются легко сопоставляемыми с перехватами потоков сетевых данных, которые выполняются в инфраструктуре `netfilter`. А таблица сообщает в `netfilter`, какие цепочки действий должны быть выполнены в первую очередь.

### 5.2.2. Реализация маркировки сетевых пакетов и соединений

Чтобы промаркировать сетевые пакеты, мы можем использовать возможности фильтрации компонента `iptables` (и `ip6tables`) для присвоения меток пакетам и соединениям. Идея заключается в следующем: межсетевой экран используется для маркировки (*to tag*) пакетов и соединений, а средства SELinux – для предоставления типам приложений разрешающих или запрещающих прав. Прав на использование этих промаркированных пакетов и соединений.

Метод маркировки пакетов известен как **SECurity MARKings (SECMARK)** («Маркировка безопасности»). И хотя мы используем термин `SECMARK`, мы должны понимать, что в действительности существует два вида маркировки: одна для сетевых пакетов (`SECMARK`) и одна для соединений (`CONNSECMARK`). Возможно-

сти SECMARK реализуются посредством двух таблиц: `mangle` и `security`. Только эти две таблицы в настоящее время обладают функциональностью в области маркировки пакетов и соединений, заложенной в их наборах правил.

Таблица `mangle` имеет наибольший приоритет исполнения, чем большинство других таблиц. Реализация правил SECMARK на этом уровне главным образом выполняется, когда предполагается, что все пакеты подлежат маркировке, даже когда многие из этих пакетов в конечном итоге будут отброшены.

Таблица `security` имеет следующий по значимости приоритет после таблицы `filter`. Это позволяет обычным правилам межсетевого экрана быть выполненными в первую очередь и маркировать только те пакеты, которые разрешены межсетевым экраном для продолжения. Использование таблицы `security` позволяет таблице `filter` выполнять правила дискреционного контроля доступа в первую очередь, а мандатное разграничение – применять системе защиты SELinux только после их успешного выполнения.

Когда запущен механизм защиты SECMARK, сетевому пакету или соединению будет сопоставляться конкретный тип, зарегистрированный в системе защиты. После этого правила политики SELinux проверят, разрешено ли типу процесса получать (`recv`) или отправлять (`send`) пакеты заданного типа. Например, процессу Firefox (выполняющемуся с типом допустимого набора функций `mozilla_t`) будут разрешены отправка (`send`) и получение (`recv`) клиентских HTTP-пакетов:

```
allow mozilla_t http_client_packet_t : packet { send recv };
```

**i** Другие поддерживаемые наборы разрешений для пакетов, связанных с SECMARK, называются `forward_in` (<переслать\_в>) и `forward_out` (<переслать\_из>). Эти разрешения проверяются, когда используется дальнейшая передача сетевых пакетов за пределы локальной системы, контролируемые в `netfilter`.

Важно знать о том, что раз введенный в действие механизм маркировки SECMARK приведет к тому, что потом все пакеты, достигающие приложений операционной системы, будут иметь метку, связанную с ними, – даже если для этого конкретного пакета или соединения нет ни одного выполняющегося правила SECMARK. Если правила нет, то по умолчанию будет назначена метка `unlabeled_t` (<немаркированный\_тип>). Политика SELinux по умолчанию, реализованная в операционной системе Red Hat Enterprise Linux, позволяет всем типам процессов отправлять и получать пакеты с типом `unlabeled_t`, но так происходит не во всех дистрибутивах Linux.

### 5.2.3. Назначение меток пакетам

Когда в подсистему `netfilter` не загружены никакие правила, относящиеся к механизму SECMARK, тогда SECMARK не подключается и ни одно из правил SELinux, связанное с разрешениями SECMARK, не проверяется. Сетевые пакеты не маркируются, поэтому никакие методы воздействия к ним применены быть не могут. Конечно, обычные механизмы контроля доступа, относящиеся к соке-

там, в это время применяются, а SECMARK – это только дополнительная мера контроля.

Когда правила SECMARK включены, то маркировка методом SECMARK становится активной и система защиты SELinux запускает механизм принудительного контроля маркированных пакетов. Это означает, что все сетевые пакеты теперь должны иметь метку безопасности (поскольку SELinux может иметь дело только с маркированными ресурсами). Для пакетов метка по умолчанию (первоначальный параметр безопасности) определяется как `unlabeled_t`, которая означает, что никакое правило маркировки не относится к этому конкретному сетевому пакету.

Когда включен принудительный контроль маркированных пакетов, то все типы [допустимых наборов функций], которые связаны с процессами и взаимодействуют с сетевыми пакетами, проверяются на предмет того, разрешено ли им отправлять и получать эти пакеты. Для упрощения управления [системой защиты] некоторые дистрибутивы исходно предоставляют права передачи и получения пакетов с меткой `unlabeled_t` для всех типов процессов. Без этих разрешений все сетевые службы прекращали бы свое правильное функционирование с того момента, как подключалось бы хотя бы одно правило SECMARK.

Чтобы присвоить метку пакету, нам требуется сначала определить набор правил, соответствующих какому-то конкретному сетевому потоку, и потом задать логику маркировки методом SECMARK (позначить конкретный пакет или соединение какой-либо меткой), и, возможно, вдобавок сразу же указать в правилах параметр `ACCEPT (<допускать>)`, позволяющий этому конкретному соединению достичь (не быть заблокированным межсетевым экраном) целевой системы.

Давайте реализуем два правила:

- 1) одно для того, чтобы позволить взаимодействие с веб-сайтами (через 80-й порт) и маркировку связанных с этим взаимодействием сетевых пакетов типом `http_client_packet_t1` (таким, чтобы веб-браузерам было разрешено отправлять и получать эти пакеты);
- 2) другое правило для того, чтобы позволить взаимодействие с локально работающим веб-сервером (тоже через 80-й порт) и маркировку относящихся к нему сетевых пакетов типом `http_server_packet_t2` (таким, чтобы веб-серверы могли отправлять и получать подобные пакеты).

Для каждого набора правил мы также подключим наблюдение за соединением таким образом, чтобы связанные с ним пакеты автоматически маркировались необходимым образом и отправлялись дальше.

Для сетевого потока данных, обрабатываемого веб-сервером, следует применить эти команды:

---

<sup>1</sup> Тип сетевого пакета, используемого клиентским ПО для взаимодействующих по сети через протокол http.

<sup>2</sup> Тип сетевого пакета, используемого веб-сервером для взаимодействия с другими компонентами сетевого обмена.

```
# iptables -t filter -A INPUT -m conntrack \
--ctstate ESTABLISHED,RELATED -j ACCEPT
# iptables -t filter -A INPUT -p tcp -d 192.168.100.15 \
--dport 80 -j ACCEPT
# iptables -t security -A INPUT -p tcp --dport 80 -j SECMARK \
--selctx "system_u:object_r:http_server_packet_t:s0"
# iptables -t security -A INPUT -p tcp --dport 80 \
-j CONNSECMARK --save
```

Для сетевого потока данных, обрабатываемого веб-браузером, следует изменить ЭТИ команды:

```
# iptables -t filter -A OUTPUT -m conntrack \
--ctstate ESTABLISHED -j ACCEPT
# iptables -t filter -A OUTPUT -p tcp --dport 80 -j ACCEPT
# iptables -t security -A OUTPUT -p tcp --dport 80 -j SECMARK \
--selctx "system_u:object_r:http_client_packet_t:s0"
# iptables -t security -A OUTPUT -p tcp --dport 80 \
-j CONNSECMARK --save
```

И в конце, для того чтобы можно было скопировать метки соединения в организующие его и связанные с ним пакеты, используйте следующие команды:

```
# iptables -t security -A INPUT -m state \
--state ESTABLISHED,RELATED -j CONNSECMARK --restore
# iptables -t security -A OUTPUT -m state \
--state ESTABLISHED,RELATED -j CONNSECMARK --restore
```

Даже этот простой пример показывает, что описание правил межсетевого экрана – это уже само по себе искусство, а маркировка методом SECMARK – это только малая его часть. Тем не менее применение правил SECMARK делает возможным разрешить конкретный сетевой поток данных и при этом гарантировать, что только строго определенным типам [сопоставляемым с процессами] будет позволено взаимодействовать с этим сетевым потоком. Например, это может быть реализовано в системах, обеспечивающих работу [интернет-] киоска (*kiosk systems*), которому необходимо разрешать одному лишь браузеру взаимодействовать с сетью интернет, запрещая при этом доступ для других браузеров и команд: конкретной меткой маркируется весь поток данных, а разрешение на передачу (*send*) и получение (*recv*) данных, имеющих эту конкретную метку, предоставляется только определенному типу, сопоставленному с нужным браузером.

## 5.3. ПРОМАРКИРОВАННЫЕ СЕТИ

Другим методом для дальнейшей уточняющей настройки контроля доступа на сетевом уровне является построение маркированных сетей. С помощью промаркированных сетей защищаемая информация передается между узлами сети (в отличие от подхода, реализованного с помощью SECMARK, при котором защита начинает работать, только когда пакет уже получен подсистемой

netfilter). Этот метод также известен под названием **одноранговая маркировка** (*peer labeling*), поскольку защищаемая информация передается между самостоятельными узлами сети (ее равноправных членов).

Преимущество промаркированных сетей заключается в том, что параметры, указывающие на защиту информации, сохраняются при ее перемещении в сети, позволяя настройкам мандатного разграничения доступа оставаться актуальными для различных конечных узлов сети, так же как и уровням доступа – для потоков взаимодействия между системами. Тем не менее основной недостаток заключается в том, что этот метод требует дополнительной сетевой технологии (протокола), способной управлять метками на сетевых пакетах или потоках.

Система защиты SELinux в настоящее время поддерживает две реализации подобной сетевой технологии, поддерживающей маркировку сетей: NetLabel и Labeled IPsec. NetLabel использует параметр CIPSO. Дополнительно NetLabel может использовать резервную маркировку (*fallback labeling*), если не применяется Labeled IPsec или CIPSO (отсюда и «резервная маркировка»). В обоих случаях реализации NetLabel сохраняются при передаче по сети только уровни доступа, характерные типу [допустимого набора функций] родительского субъекта. В то же время Labeled IPsec поддерживает передачу всех параметров безопасности, определяемых в SELinux.



На самом деле есть некоторое исключение: NetLabel поддерживает полный набор параметров безопасности при использовании соединения с обратной петлей (*loopback*). В этом случае передается метка полностью (а не только уровни доступа и категории информации). Однако это работает лишь для соединений, которые проходят через петлевой сетевой интерфейс (*loopback interface*).

Относительно недавно поддержка для NetLabel/CIPSO и Labeled IPsec была включена в общую структуру, которая вводит три дополнительные проверки привилегий в SELinux: контроль на уровне сетевого интерфейса (*interface*), узла сети (*node*) и оконечных точек подключения одноранговой сети (*peer*).

Эти проверки привилегий активны *только* тогда, когда используется промаркированный сетевой поток данных; без него данные проверки просто игнорируются.

### 5.3.1. Резервная маркировка в NetLabel

Проект NetLabel поддерживает резервную маркировку, где администраторы могут назначать метки сетевому потоку данных, исходящему из сегментов (или входящему в сегменты) сети, которые на самом деле не используют сетевую маркировку. С помощью резервной маркировки способы контроля оконечных точек подключения одноранговой сети, упомянутые в следующих двух разделах, могут быть применены даже без наличия таких средств, как Labeled IPsec или NetLabel/CIPSO.

Установите пакет `netlabel_tools`, для того чтобы иметь возможность выполнить команду `netlabelctl`, которая нужна для управления конфигурацией NetLabel. Когда это будет сделано, то мы сможем начать добавлять правила. Давайте сделаем так, чтобы присвоение резервных меток производилось для всех потоков, формируемых на оконечном узле с адресом 192.168.100.1:

```
# netlabelctl unlbl add interface:eth0 address:192.168.100.1 \
  label:system_u:object_r:netlabel_peer_t:s0
```

Для вывода перечня актуальных настроек резервной маркировки применяется следующая команда:

```
# netlabelctl -p unlbl list
Accept unlabeled packets : on
Configured NetLabel address mappings (1)
  interface: eth0
  address: 192.168.100.1/32
  label: "system_u:object_r:netlabel_peer_t:s0"
```

**P** Для русскоязычного варианта данный вывод мог бы быть представлен следующим образом:

```
# netlabelctl -p unlbl list
Разрешение немаркированных пакетов : включено
Настроенные сопоставления сетевых адресов в NetLabel (1)
  сетевой интерфейс: eth0
  сетевой адрес: 192.168.100.1/32
  метка: "system_u:object_r:netlabel_peer_t:s0"
```

С установкой этого правила маркировка в сети становится активной. И любой сетевой поток данных, сформированный на узле с адресом 192.168.100.1, будет помечен меткой `netlabel_peer_t:s0`, в то время как другие потоки будут помечены (по умолчанию) меткой `unlabeled_t:s0`.

Конечно, политика SELinux должна позволять всем типам [сопоставляемым с процессами] предоставлять право `recv` (<получать>) от оконечных одноранговых узлов данные, имеющие либо метку `unlabeled_t`, либо метку `netlabel_peer_t`. Если таких прав нет, то будет получен отказ в доступе с записью сообщения AVC-типа, предупреждающего об отсутствии разрешения у типов [процессов] получать сетевой поток данных:

```
avc: denied { recv } for saddr=1.2.3.4 src=500 daddr=4.3.2.1
  dest=500 netif=eth0
  context=system_u:system_r:racoont:s0-s15:c0.c1023
  tcontext=system_u:object_r:unlabeled_t:s15:c0.c1023
  tclass=peer permissive=0
```

Резервная маркировка полезна для применения в смешанных сетях, где присутствуют как выполняющие маркировку сегменты сети, так и те, которые не поддерживают маркировку.

### 5.3.2. Ограничение потоков данных на уровне сетевого интерфейса

Идея проверки интерфейса заключается в том, что каждый пакет, который приходит в систему, проходит входную (ingress) проверку на интерфейсе, в то время как покидающий систему пакет проходит выходную (egress) проверку. Разрешения ingress и egress назначаются системой защиты SELinux, а сетевые интерфейсы получают необходимые параметры безопасности.

Метки для интерфейса могут быть назначены при помощи инструмента `semanage`, который также полезен для присвоения уровней доступа и категорий информации, что мы и сделаем в следующем примере, где установим категории для интерфейса `tap0`:

```
# semanage interface -a -t netif_t -r s0-s0:c0.c128 tap0
```

По аналогии с другими применениями команды `semanage` мы можем увидеть текущие сопоставления параметров безопасности с интерфейсами:

```
# semanage interface -l
SELinux Interface      Context
tap0                   system_u:object_r:netif_t:s0-s0:c0.c128
```

Следует помнить, что тип входящих соединений будет являться типом конкретного однорангового узла сети. В случае когда мы имеем дело с Labeled IPsec, это будет тип, сопоставляемый с клиентом, который инициировал соединение, тогда как в NetLabel/CIPSO это метка, ассоциированная с оконечной одноранговой точкой подключения (такая как `netlabel_peer_t`).

Это показано в следующем отказе:

```
avc: denied { ingress } for saddr=147.32.127.222 src=21
daddr=10.0.2.15 dest=53060 netif=eth0
scontext=system_u:object_r:netlabel_peer_t:s0
tcontext=system_u:object_r:netif_t:s0-s15:c0.c1023
tclass=netif permissive=0
```

По умолчанию сетевой интерфейс промаркирован как `netif_t` и при этом не имеет ограничений по категориям обрабатываемой информации. Однако это не будет показано в выводе результатов от выполнения команды `semanage interface -l`, поскольку его вывод по умолчанию является пустым.

### 5.3.3. Ограничение потоков данных на уровне элементов сети

Узлы сети (*nodes*) представляют собой особые компоненты сети (или некие группы сетевых компонент), от которых данные либо передаются (`sendto`), либо принимаются (`recvfrom`). При этом эти элементы сети управляются через класс сетевых элементов SELinux. Так же, как и в случае с интерфейсами, можно вывести перечень учтенных в системе защиты элементов сети с помощью инструмента `semanage`. В следующем примере мы промаркируем сеть `192.168.100.1`, на-



значив ей тип `node_t`, и укажем для него набор категорий информации, которые можно обрабатывать:

```
# semanage node -a -t node_t -p ipv4 -M 255.255.255.255 -r s0-s0:c0.c128 192.168.100.1
```

А затем выведем на экран список учтенных в системе защиты элементов сети:

```
# semanage node -l
IP Address      Netmask          Protocol Context
192.168.100.1   255.255.255.255  ipv4           system_u:object_r:node_t:s0-s0:c0.c128
```

Аналогично тому, как было в случае с контролем потоков сетевых интерфейсов, фактическим типом для входящих соединений является тип конкретной оконечной точки одноранговой сети [инициировавшей соединение]. Это можно увидеть из текста сообщения AVC-типа, зарегистрированного при отказе в доступе:

```
avc: denied { recvfrom } for pid=7204 comm="client.pl"
      saddr=10.4.2.1 src=56403 daddr=10.4.2.112 dest=7081
      netif=eth0 scontext=system_u:object_r:netlabel_peer_t:s0
      tcontext=system_u:object_r:node_t:s0 tclass=node permissive=0
```

По умолчанию узлы сети маркируются как `node_t` и без ограничений на обработку различных категорий информации. Однако это не будет показано в выводе результатов от выполнения команды `semanage node -l`, поскольку его вывод по умолчанию является пустым.

### 5.3.4. Проверка однорангового потока

Последняя проверка – это проверка класса `peer`. В случае с Labeled IPsec метка [для указания типа] назначается каждому конкретному сетевому сокету, через который данные посылаются во внешнюю сеть (такую как `mozilla_t`). Когда же используется метод маркировки NetLabel/CIPSO, то для оконечных точек подключения одноранговой сети будет назначаться только один тип, в соответствии с источником данных, так как NetLabel (а на самом деле CIPSO) может передавать лишь уровни доступа (*sensitivity level*). Обычно можно видеть такую метку, подходящую для NetLabel, – `netlabel_peer_t`.

Ниже приведен пример сообщения типа AVC, регистрируемого при отказе в доступе для класса `peer`:

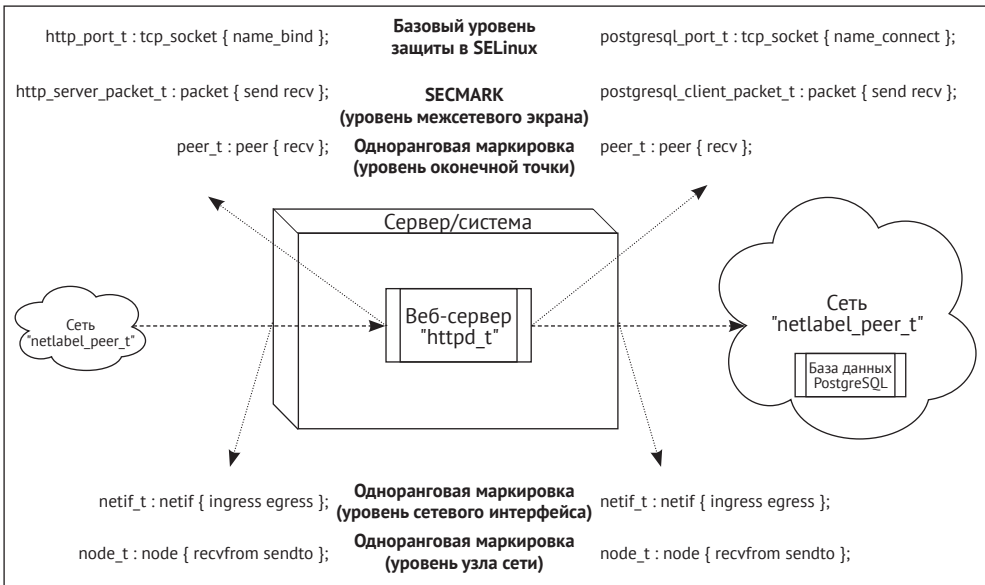
```
avc: denied { recv } for pid=9 comm="rcu_preempt"
      saddr=192.168.100.1 src=40870 daddr=192.168.100.152 dest=80
      netif=eth0 scontext=system_u:system_r:httpd_t:s0
      tcontext=staff_u:staff_r:mozilla_t:s0 tclass=peer permissive=0
```

Как мы можем видеть из этого примера, в отличие от проверок интерфейсов и элементов сети, контроль потока между оконечными точками взаимодействия одноранговой сети предполагает, что тип [допустимого набора функций] относится к целевой (*target*) точке, с которой выполняется соединение, а не

к той, которая его инициализирует (*source*). Исходя из текста сообщения, мы видим, что инициатор соединения с (локальным) типом `httpd_t` не имеет прав на получение сетевого потока данных от оконечной точки с типом `mozilla_t`.

Во всех предыдущих примерах процесс, указанный в поле `comm=` текста сообщения об отказе, не имеет никакого отношения к фактическому отказу. Это связано с тем, что отказ инициируется в подсистеме ядра, а не посредством вызова, выполняемого пользовательским процессом. В результате здесь приведен несвязанный процесс, который был прерван во время подготовки отказом в доступе.

В заключение рассмотрим рис. 5.1, где представлена диаграмма, которая дает представление об этих вариантах контроля и уровнях, к которому они применяются:



**Рис. 5.1** ❖ Схематический обзор различных методов контроля SELinux, связанных с сетью

Высокоуровневый контроль осуществляется на уровне процесса (такого как `httpd_t`), а низкоуровневый уровень проверок выполняется на уровне типов оконечных точек взаимодействия одноранговой сети (таких как `netlabel_peer_t`).

### 5.3.5. Применение управления в старом стиле

Большинство дистрибутивов Linux содержат возможность, называемую `network_peer_control`. Это некоторое улучшение в рамках подсистемы SELinux, ко-

торое использует ранее упомянутый класс `peer` для контроля сетевого потока данных между оконечными точками взаимодействия одноранговой сети (*peer-to-peer flow*).

Однако политики SELinux могут предпочесть использовать более старый метод, в котором такой поток данных не контролируется классом `peer`, а используется класс `tcp_socket` для обеспечения сетевого взаимодействия. В таком случае класс `tcp_socket` будет использоваться по отношению к типу [допустимого набора функций процесса] `peer` с разрешением получения данных `recvfrom` (в качестве приоритетного из числа других разрешений, имеющихся у `tcp_socket`).

Текущее значение возможности `network_peer_control` можно выяснить, используя файловую систему:

```
# cat /sys/fs/SELinux/policy_capabilities/network_peer_controls
1
```

Если значение равно нулю – 0, тогда ранее упомянутый контроль сетевого потока данных между оконечными точками взаимодействия одноранговой сети будет осуществляться с применением класса `tcp_socket` вместо класса `peer`.

Возможности политик (*policy capabilities*) не могут контролироваться администратором: они жестко запрограммированы в самой политике SELinux.

## 5.4. МЕТКИ БЕЗОПАСНОСТИ ДЛЯ IPSEC

Несмотря на то что настройка и техническое описание установки группы протоколов защиты данных, носящих название IPsec, выходят далеко за рамки этой книги, давайте рассмотрим простой пример, для того чтобы показать, как промаркированный метками безопасности IPsec (Labeled IPsec) работает в такой системе. Имейте в виду, что контроль промаркированных сетей на уровнях интерфейсов, узлов и оконечных точек подключения, как было сказано выше, включается автоматически и одновременно с использованием средств организации защищенного канала «IPsec».

При установке «IPsec» следует осознавать два следующих момента:

- 1) **база данных политики безопасности (SPD – security policy database)** содержит правила и информацию для ядра, определяющие, когда взаимодействие должно управляться частной политикой, применяемой к протоколу IP (и как результат управление через безопасное соединение (*security association*));
- 2) **база данных безопасных соединений (SAD – security association database)** содержит отдельные безопасные соединения. **Безопасное соединение (SA – security association)** представляет собой однонаправленный канал между двумя устройствами, подключенными в сеть, и содержит всю информацию о защите канала. В случае когда «IPsec» является промаркированным, то также содержит информацию о параметрах клиента, который организовал это безопасное соединение.

Безопасные соединения с настройкой Labeled IPsec больше не индексируются исключительно по адресу источника и адресу приемника, но также и по параметрам безопасности источника. Как таковая Linux-система, которая принимает участие в процессе настройки Labeled IPsec, будет легко иметь несколько дюжин безопасных соединений для одного-единственного сетевого потока данных, поскольку каждое безопасное соединение теперь также представляет конкретный тип, определяющий допустимый набор функций клиента.

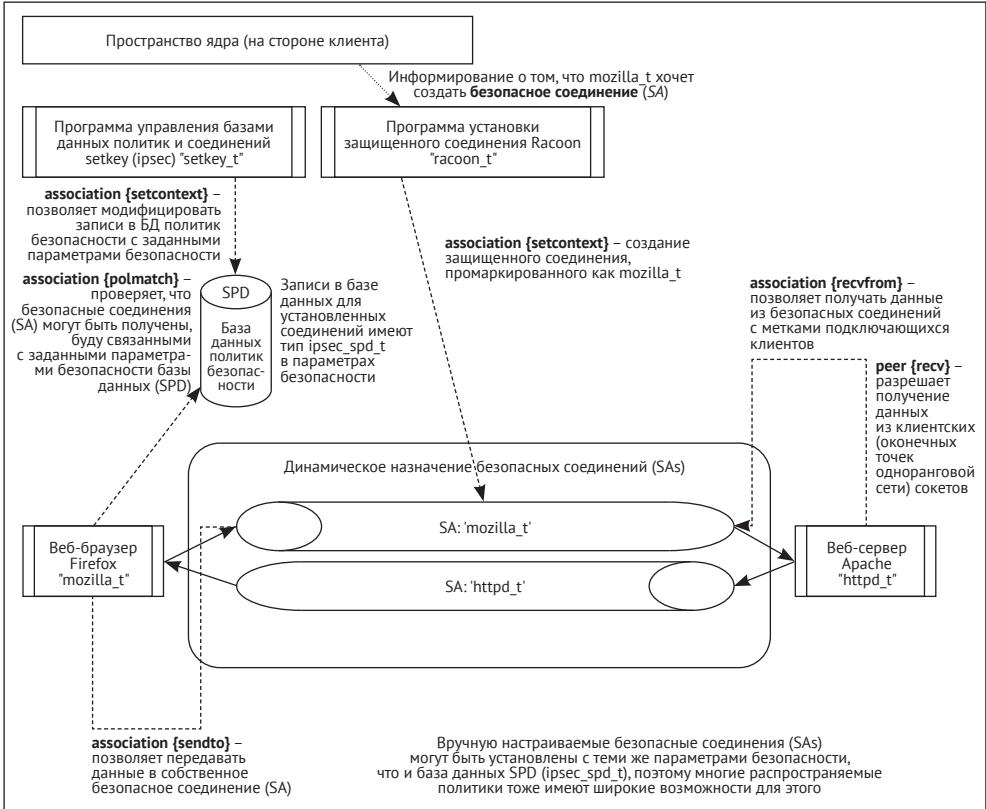
Labeled IPsec включает несколько дополнительных средств контроля доступа на основе возможностей SELinux:

- отдельные записи в базе данных политики безопасности (*SPD*) получают параметры безопасности. Типы [допустимого набора функций], назначенные процессам, которым требуется получить безопасное соединение (*SA*), должны иметь привилегию *polmatch* (часть класса *association*) для конкретного параметра безопасности. Кроме того, они (типы) должны иметь привилегию *setcontext* (также являющуюся частью класса *association*), чтобы позволить иметь возможность инициировать какое-либо безопасное соединение (*SA*);
- только авторизованным (разрешенным) типам [допустимого набора функций] процессов позволено модифицировать базу данных политик безопасности (*SPD*), что также управляется при помощи привилегии *setcontext*, но в данном случае определяется для записей параметров безопасности в базе *SPD*. Данная привилегия предоставляется в основном инструментам IPsec, таким как *setkey* (*setkey\_t*) и *Racoon* (*racoon\_t*);
- типы [допустимых наборов функций] процессов, участвующие в сетевом взаимодействии с применением протоколов IPsec, должны иметь право *sendto* (<переслать в>) для инициализации своего соединения и право *recvfrom* (<получить из>) для установки соединения с типом оконечной точки взаимодействия одноранговой сети (*peer*). Типу принимающей стороны требуется также иметь привилегию *recv* (<получать>) от класса *peer*, сопоставленного с типом [допустимого набора функций] *peer*.

Таким образом, хотя средство защиты Labeled IPsec и не влияет на то, сможет ли субъект доступа с типом *mozilla\_t* взаимодействовать с объектом типа *httpd\_t* (поскольку *mozilla\_t* требует право на передачу данных (*send to*) для инициализации своего собственного соединения), но оно может влиять на право разрешения (или запрещения) входящих соединений к объекту типа *httpd\_t* от типа *mozilla\_t* (так как для этого требуется привилегия *recvfrom*).

Эта, вероятно, сложная игра привилегий отображена на рис. 5.2.

В следующем примере настроен простой тоннель IPsec между двумя узлами сети при помощи приложения *ipsec-tools*. Инструкции для одной из свободных программных реализаций IPsec, названной *Liberswan*, приводятся после этих разделов.



**Рис. 5.2** ❖ Управление SELinux для средства защиты Labeled IPsec на примере сетевого потока обмена данными между веб-браузером Firefox и веб-сервером Apache

### 5.4.1. Установка стандартного IPsec

Для начала фоновый процесс гассоон настраивается сведениями:

- 1) о предварительном общем ключе (*pre-shared key*) (для использования во время протокола рукопожатия (*handshake*) с удаленным абонентом);
- 2) о деталях протокола рукопожатия для удаленного абонента;
- 3) о связи, касающейся присоединяемых сетей.

Следующий текст – выдержка из файла конфигурации для программы гассоон:

```
# File contains remote address with a shared key, like:
# 192.178.100.153 ThisIsABigSecret
path pre_shared_key "/etc/racoon/psk.txt";
remote 192.168.100.153 { ... };
sa!info address 10.1.2.0/24 any address 10.1.3.0/24 any { ... };
```

**P** Для русскоязычного варианта данная выдержка может содержать следующие комментарии:

```
# Файл содержит удаленный адрес с общим ключом, как показано ниже:
# 192.178.100.153 ThisIsABigSecret # Это является Большим Секретом
<...>
```

Большинство дистрибутивов по умолчанию предлагают разумную конфигурацию для работы программы установки защищенного соединения `gasoon`. В предыдущем примере IP-адрес `192.168.100.153` – это адрес удаленного абонента, тогда как параметр `saInfo` определяет диапазоны адресов, которые используются в работе виртуальной частной сети (VPN – Virtual Private Network): `10.1.2.0/24` являются локальными, а `10.1.3.0/24` – удаленными).

Информация для утилиты управления базами данных IPsec SA/SP – `setkey` – выглядит так:

```
#!/usr/sbin/setkey -f
flush; spdflush;
spdadd 10.1.2.0/24 10.1.3.0/24 any
  -P out ipsec esp/tunnel/192.168.1.5-192.168.100.153/require;
spdadd 10.1.3.0/24 10.1.2.0/24 any
  -P in ipsec esp/tunnel/192.168.100.153-192.168.1.5/require;
```

В конце мы корректируем сведения системы маршрутизации таким образом, чтобы любое присоединение к сети `10.1.3.0/24` происходило через VPN-шлюз `10.1.2.1`:

```
# ip addr add 10.1.2.1/24 dev eth0
# ip route add to 10.1.3.0/24 via 10.1.2.1 src 10.1.2.1
```

## 5.4.2. Подключение маркировки IPsec

Для подключения маркировки – Labeled IPsec – нам нужно проинформировать IPsec о добавлении параметров безопасности для записей базы данных политики безопасности (SPD). Когда они будут добавлены, `gasoon` будет автоматически согласовывать свои действия с защитой Labeled IPsec. Добавление параметров безопасности к базе данных SPD означает добавление опции `-ctx` к командам `spdadd` в конфигурации `setkey`. Например, мы можем добавить параметр `ipsec_spd_t` в политику безопасности IPsec таким образом:

```
spdadd ... -ctx 1 1 "system_u:object_r:ipsec_spd_t:s0" -P out ipsec ...
```

В результате подобных изменений мы можем видеть параметры безопасности при выводе команды `setkey -DP`, которая показывает текущую базу данных SPD:

```
# setkey -DP
...
10.1.2.0/24[any] 10.1.3.0/24[any] 255
  out prio def ipsec
  esp/tunnel/192.168.100.152-192.168.100.153/require
  created: Jul 4 21:45:44 2013 lastused:
```

```
lifetime: 0(s) validtime: 0(s)
security context doi: 1
security context algorithm: 1
security context length: 33
security context: system_u:object_r:ipsec_spd_t:s0
spid=1 seq=0 pid=3237
refcnt=1
```

Когда какое-нибудь приложение пытается взаимодействовать по сети через IPsec с удаленными объектами, которым соответствует некий тип SELinux, тогда программа Rasoon (или любой другой клиент IKEv2, поддерживающий маркированный IPsec, такой как pluto из состава продукта Libreswan) будет обмениваться необходимой информацией (включая параметры безопасности) с другой стороной. Обе стороны будут обновлять базу данных SPD и необходимые безопасные соединения (SAs), а также связывать с ними соответствующую **информацию о политике безопасности** (SPI – security policy information). С этого момента отправляющая сторона будет добавлять согласованную SPI-информацию к IPsec-пакетам так, что удаленная сторона сможет без промедления сопоставлять с ней правильные параметры безопасности.

Огромным преимуществом здесь является то, что параметры безопасности клиента и сервера, включая уровни доступа и категории информации, синхронизируются (в действительности они не отправляются по каналу с каждым пакетом, а обмениваются изначально при установке безопасных соединений).

### 5.4.3. Использование Libreswan

Предыдущие инструкции были связаны с программами из набора ipsec-tools. Тем не менее существует много других поддерживающих IPsec утилит. Дистрибутив Red Hat Enterprise Linux использует Libreswan в качестве реализации протоколов IPsec.

Настройка Libreswan определяется в основном конфигурационном файле этой программы – ipsec.conf. Многие дистрибутивы будут использовать include (<подключаемую>) директорию (такую как /etc/ipsec.d), в которой могут находиться настройки, характерные для соединений. В основном include директория применяется для действующих конфигураций IPsec, в то время как основной файл ipsec.conf – для описания поведения самого продукта Libreswan.

Чтобы использовать защиту Labeled IPsec в Libreswan, применяйте управляющие директивы labeled\_ipsec и policy\_label в числе определений IPsec. Например, для установки соединения на базе IPsec между двумя сетевыми узлами (10.1.2.1 и 10.1.3.1) может быть использована следующая конфигурация:

```
# cat /etc/ipsec.d/SELinuxtest.conf
conn SELinuxtest
  auto=start
  rekey=no
  authby=secret
  type=transport
```

```

left=10.1.2.1
right=10.1.3.1
ike=3des-sha1
phase2=esp
phase2alg=aes-sha1
  labeled-ipsec=yes
  policy-label=system_u:object_r:ipsec_spd_t:s0
leftprotoport=tcp
rightprotoport=tcp

```

Секретные данные, используемые для подтверждения подлинности двух абонентов, устанавливающих между собой соединение, хранятся в файле `/etc/ipsec.secrets`. В операционной системе Red Hat Enterprise Linux этот файл включает секреты из файлов `/etc/ipsec.d/*.secrets`, поэтому в ней параметры IPsec могут легко настраиваться через отдельные файлы.

Например, для предыдущей конфигурации файл типа `secrets` мог бы выглядеть так:

```

# cat /etc/ipsec.d/SELinuxtest.secrets
10.1.2.1 10.1.3.1: PSK "some preshared key"

```

**P** В этом примере: PSK – это принятое обозначение для предварительного общего ключа (Pre-Shared Key), а в кавычках его значение, в данном случае указанное как текст «некоторый предварительный общий ключ».

**i** При совместном использовании Libreswan и `ipsec-tools` в одной настройке IPsec может возникнуть необходимость сконфигурировать Libreswan, чтобы использовать одно и то же значение атрибута (*attribute value*) для параметров безопасности, что и для `ipsec-tools`. В `ipsec-tools` жестко запрограммировано это значение до 10, поэтому мы должны адаптировать Libreswan к применению данного значения. Это делается в разделе `config setup`, который определен в файле `ipsec.conf`, где и должно быть прописано значение `secctx-attr-value = 10`.

## 5.5. ТЕХНОЛОГИЯ МАРКИРОВКИ СЕТЕЙ NETLABEL С ПАРАМЕТРОМ CIPSO

С поддержкой метода маркировки NetLabel/CIPSO сетевой поток данных маркируется сведениями об уровнях доступа и категорий информации, которые могут быть использованы внутри сети. В отличие от Labeled IPsec, никакая другая информация о параметрах безопасности не отправляется и не синхронизируется. Так, когда мы посмотрим на такие сетевые потоки данных, то увидим, что они будут формироваться на основе одного параметра безопасности, но будут иметь метки, основанные на метке конфиденциальности удаленной стороны.

В рамках технологии NetLabel сопоставления определяются таким образом, чтобы информировать систему о том, какие сетевые потоки данных (из конкретных сетевых интерфейсов или даже из конкретных IP-адресов) к ка-



кому именно **интерпретирующему домену (DOI – Domain of Interpretation)** относятся.

Стандарт CIPSO определяет интерпретирующий домен как некую совокупность систем, которые одинаково распознают метку CIPSO или же, как в нашем случае, используют одну и ту же политику SELinux и одинаковую мандатную конфигурацию.

При наличии имеющихся сопоставлений NetLabel/CIPSO будет пропускать данные об уровнях доступа (и о категориях информации) между сетевыми ресурсами. Параметр безопасности, который мы будем видеть в сетевых потоках данных, будет иметь наименование `netlabel_peer_t` – это параметр безопасности, который присваивается по умолчанию всему потоку данных, формируемому при участии системы NetLabel/CIPSO.

Рассмотрим следующее зарегистрированное событие AVC-типа с отказом в доступе:

```
type=AVC msg=audit(1368735963.286:1998): avc: denied { recv }
for pid=4773 comm="python-thinlinc" saddr=192.168.100.15
src=46092 daddr=192.168.100.11 dest=9000 netif=eth0
scontext=system_u:system_r:httpd_t:s0-s2:c0.c32
tcontext=system_u:object_r:netlabel_peer_t:s0:c102
tclass=peer permissive=0
```

Данное сообщение показывает, что сетевой поток данных был получен из оконечной точки одноранговой сети, которой разрешена работа с информацией категории `c102`, что непозволительно для типа `httpd_t`, имеющего параметры безопасности `s0-s2:c0.c32`.

При помощи этого метода мы можем запускать фоновые процессы с индивидуальными диапазонами уровней доступа и категорий информации и, следовательно, принимать соединения только от тех пользователей или клиентов, которые имеют корректные диапазоны разрешений, даже на удаленных системах с включенным средством защиты NetLabel/CIPSO.

### 5.5.1. Настройка сопоставлений потоков данных с доменами

Прежде чем говорить о том, что в сети корректно применен параметр CIPSO, следует разобраться с тем, какой интерпретирующий домен (DOI) будет использоваться и к каким результатам это приведет. Маркированные сети могут использовать разные интерпретирующие домены для особых целей.

Далее, имея дело с конкретным интерпретирующим доменом, нам также нужно определиться с тем, какие категории информации и уровни доступа передаются по сети, поддерживающей CIPSO. Это контролируется меткой CIPSO, которая обозначается `tag`. Существует три поддерживаемых значения для такой метки:

- `tag:1` – категории предоставляются в CIPSO-пакете в виде массива битов. Это наиболее стандартный способ, поддерживающий до 240 категорий (от 0 до 239);

- tag:2 – категории перечисляются отдельно. Этот подход предоставляет более широкий диапазон категорий (до 65534), но поддерживает не более 15 перечисленных категорий. Попробуйте использовать tag:2, когда у вас есть много категорий, но для каждой *области применения (scope)* необходимо использовать только несколько категорий;
- tag:5 – категории могут быть указаны в виде диапазона (от номера наименьшего значения до номера наибольшего), при этом максимальное число таких пар диапазонов – «наименьшее/наибольшее» значение – равно семи.

Заметим, что результаты установки CIPSO-меток (tag) обрабатываются «под капотом», т. е. внутри системы: системным администраторам надо только настроить сопоставления NetLabel для использования конкретного значения метки.

Давайте предположим, что у нас есть две сети с включенным параметром CIPSO. Одна из сетей имеет адрес 10.1.0.0/16 и входит в интерпретирующий домен doi:1, а вторая имеет адрес 10.2.0.0/16 и входит в doi:2. Обе используют значение метки CIPSO, равной 1. Перво-наперво мы подключаем параметр CIPSO и разрешаем ему передавать маркированные CIPSO-пакеты, относящиеся к интерпретирующим доменам (DOI) со значениями 1 или 2. Мы не выполняем никаких преобразований, так что категории информации и уровни доступа, установленные в CIPSO-пакете, являются единственными используемыми в системе защиты SELinux:

```
# netlabelctl cipsov4 add pass doi:1 tags:1
# netlabelctl cipsov4 add pass doi:2 tags:1
```

Если нам потребовалось бы выполнить преобразования (скажем, мы используем уровни доступа s0–s3, в то время как сеть CIPSO использует уровни доступа 100–103), то команда могла бы выглядеть так:

```
# netlabelctl cipsov4 add std doi:1 tags:1 levels:0=100,1=101,2=102
```

Затем мы выполняем правила сопоставления (*mapping rules*), сообщая конфигурации NetLabel, какой сетевой поток данных будет связываться с doi:1 или doi:2:

```
# netlabelctl map del default
# netlabelctl map add default address:10.1.0.0/16 protocol:cipsov4,1
# netlabelctl map add default address:10.2.0.0/16 protocol:cipsov4,2
```

Вот и все. Мы удалили сопоставление, установленное по умолчанию (поскольку оно препятствует добавлению новых сопоставлений, устанавливаемых по умолчанию), и потом настроили NetLabel так, чтобы привязать сетевой поток данных к заданным сетям с настроенной конфигурацией CIPSO.

## 5.5.2. Добавление сопоставлений для типов допустимого набора функций

Также NetLabel может быть настроен для обеспечения того, чтобы конкретные SELinux-домены (типы допустимых наборов функций, сопоставленные с про-

цессами) использовали четко определенную совокупность систем, которые одинаково распознают метку CIPSO (DOI), а не настроенную ранее по умолчанию. Например, чтобы фоновый процесс SSH (запущенный с типом `sshd_t`) имел свой сетевой поток данных, промаркированный как `doi:3` с использованием параметра CIPSO, следует использовать такую команду:

```
# netlabelctl cipsov4 add pass doi:3 tags:1
# netlabelctl map add domain:sshd_t protocol:cipsov4,3
```

Правила сопоставления могут быть даже более избирательными, чем эти. Мы можем указать NetLabel использовать значение `doi:2` для сетевого потока, формируемого в рамках SSH-соединений и порождаемого в одной сети, а значение `doi:3` использовать для потока данных SSH, порождаемого в другой сети. И даже применять немаркированный поток данных, когда он исходит из любой другой сети.

```
# netlabelctl map del domain:sshd_t protocol:cipsov4,3
# netlabelctl map add domain:sshd_t address:10.1.0.0/16 protocol:cipsov4,1
# netlabelctl map add domain:sshd_t address:10.4.0.0/16 protocol:cipsov4,3
# netlabelctl map add domain:sshd_t address:0.0.0.0/0 protocol:unlbl
```

Инфраструктура системы NetLabel будет сначала пытаться сопоставлять наиболее конкретизированное правило, поэтому для адреса сети `0.0.0.0/0` будет проверяться соответствие только тогда, когда уже не будет никаких других правил.

### 5.5.3. Локальное использование параметра CIPSO

Как было сказано раньше, NetLabel по умолчанию передает только уровни доступа и категории информации. Тем не менее когда мы используем CIPSO локально (то есть через обратный петлевой интерфейс), есть возможность контролировать все параметры безопасности. Когда включена [система маркировки локально], то контроль окончечных точек взаимодействия одноранговой сети (*peer*) будет относиться не к типу `netlabel_peer_t`, назначенному по умолчанию, а к типу конкретного клиентского или серверного приложения.

Чтобы использовать определения CIPSO локально, сначала надо указать интерпретируемый домен (DOI) для локального использования:

```
# netlabelctl cipsov4 add local doi:5
```

Потом сделаем так, чтобы локальное сетевое взаимодействие использовало заданный домен DOI (5 в нашем примере):

```
# netlabelctl map add default address:127.0.0.1 protocol:cipsov4,5
```

При включении этого правила локальное сетевое взаимодействие будет связано со значением `doi:5` и использовать локальное сопоставление, передавая параметры безопасности полностью при работающем мандатном контроле доступа системы защиты (SELinux).

### 5.5.4. Поддержка опции безопасности для IPv6

Продолжает вестись работа по внедрению **опции безопасности для общей архитектурной метки в интернет-протоколе версии 6 (CALIPSO – Common Architecture Label IPv6 Security Option)** в рамках проекта NetLabel. Когда необходимо включить поддержку этой опции безопасности, то указывается в правилах наименование протокола как `calipso`, а не `cipsov4`.

Существует несколько небольших различий при использовании CALIPSO вместо CIPSO в NetLabel:

- поддерживается только один тип метки, обозначаемой `tag` (в отличие от трех типов в CIPSO). А поэтому нет нужды указывать в правилах `tag:#`;
- CALIPSO использует только сквозной режим передачи параметров безопасности без поддержки их преобразования;
- реализация NetLabel CALIPSO в настоящее время не поддерживает локальный режим, в котором передаются параметры безопасности целиком.

Учитывая эти различия, можно сказать, что использование опции безопасности CALIPSO осуществляется сходным образом с описанным выше подходом:

```
# netlabelctl calipso add pass doi:5
# netlabelctl map add domain:httpd_t protocol:calipso,5
```

## 5.6. ЗАКЛЮЧЕНИЕ

SELinux по умолчанию использует контроль доступа, основанный на файловом представлении элементов взаимодействия или на использовании сокетов. В случае с TCP- и UDP-портами администраторы имеют некоторую свободу в управлении с помощью команды `semanage`, не прибегая к обновлениям политики SELinux.

Когда мы работаем в сфере сетевого взаимодействия, более широкий контроль этого взаимодействия может быть достигнут при поддержке межсетевого экрана `netfilter` системы Linux, при использовании метода маркировки `SECMARK` и при помощи маркировки оконечных точек взаимодействия одноранговой сети (*peer labeling*).

В случае маркировки методом `SECMARK` локальные правила брандмауэра используются для сопоставления параметров безопасности сетевым пакетам, которые потом управляются политикой SELinux.

В случае маркировки оконечных точек взаимодействия одноранговой сети используются или параметры безопасности самого приложения (в случае с `Labeled IPsec`), или его мандатная конфигурация (в случае поддержки средства NetLabel/CIPSO). Это позволяет контролировать почти все сетевые потоки вида «приложение–приложение» через политики SELinux.

В следующей главе мы рассмотрим две платформы, использующие SELinux для дополнительного управления безопасностью: виртуализацию Linux и контейнеризацию при помощи `sVirt` и `Docker`.

# Глава 6

## Поддержка sVirt и Docker

Все больше и больше системных инструментов имеют встроенную поддержку для SELinux или используют возможности SELinux для дальнейшего укрепления своих собственных сервисных предложений. Когда мы смотрим на виртуализацию, приходят на ум два проекта с открытыми исходными текстами: **libvirt** и **Docker**. В то время как первый проект поддерживает полную виртуализацию, второй сосредоточивается на управлении контейнерами. В этой главе администраторы будут:

- узнавать, как SELinux может помочь снизить риски виртуализации;
- разбирать, как политика SELinux настраивается для поддержки этих служб виртуализации;
- иметь дело с опцией безопасной виртуализации, поддерживаемой через программный интерфейс (API) проекта libvirt.

Закончим главу рассмотрением раздела «Защита контейнеров Docker» с помощью SELinux.

### 6.1. Виртуализация, защищенная SELinux

Виртуализация – это составляющая часть многих служб инфраструктуры. Возможности виртуализации выросли с момента возникновения в начале 70-х средств в качестве средства изоляции рабочих задач и абстрагирования от аппаратных средств. Когда мы рассматриваем предложения различных услуг сегодня, то понимаем, что многие поставщики облачных технологий не смогли бы предоставлять свои услуги при отсутствии у них виртуализации.

Изоляция является одной возможностью из общего числа функций, которые предлагает виртуализация и которую SELinux может поддерживать и очень хорошо дополнять.

#### 6.1.1. Представление о виртуализации

Когда мы рассматриваем виртуализацию, то видим абстрактные уровни, которые она предоставляет для того, чтобы скрыть конкретные ресурсы (такие как аппаратные компоненты или вычислительная мощность). Виртуализация способствует:

- 1) развитию более эффективного использования аппаратных средств (что приводит к оптимизации затрат);
- 2) централизованному представлению о ресурсах и системах;
- 3) большей гибкости в численности операционных систем, с которыми может работать компания;
- 4) стандартизации при распределении ресурсов;
- 5) усовершенствованию (улучшению качества) системы защиты.

Существует несколько типов виртуализации:

- **полная эмуляция системы** (*Full system emulation*), при которой аппаратная часть полностью имитируется посредством программного обеспечения. QEMU является примером программного эмулятора, который способен выполнять полную эмуляцию системы;
- **аппаратная виртуализация** (*Native virtualization*), где основные части аппаратного обеспечения являются предоставленными в общее пользование через экземпляры и гостевые сервисы могут работать на них без изменений. В Linux примером этого вида виртуализации служит средство KVM (*Kernel-based Virtual Machine* – виртуальная машина на базе ядра), которое также поддерживается через QEMU;
- **паравиртуализация** (*Paravirtualization*) – это способ виртуализации, при котором гостевая операционная система использует особые программные интерфейсы (API), предлагаемые уровнем виртуализации (на котором немодифицированные операционные системы не могут быть размещены). Изначальные реализации гипервизора Xen поддерживали только паравиртуализацию. Другой пример реализации этого способа виртуализации – это использование KVM с драйверами VirtIO;
- **виртуализация на уровне операционной системы, или контейнеризация** (*OS-level virtualization; containerization*), в этом случае гостевые сервисы используют основную операционную систему (ядро), но не видят никаких других процессов и ресурсов, запущенных на ней. Контейнеры Docker или контейнеры LXC – это примеры виртуализации уровня операционной системы;
- **виртуализация приложений** (*Application virtualization*), при которой приложение работает под управлением специальных программ, обеспечивающих выполнение этого приложения. Популярным примером здесь является поддержка для приложений, написанных на языке Java и выполняющихся под управлением JVM (*Java Virtual Machine* – виртуальной машины Java).

Многие платформы поддерживают несколько типов виртуализации. QEMU, например, работает и в режиме полной эмуляции, и в режиме паравиртуализации, в зависимости от своей конфигурации.

Когда мы работаем с уровнями виртуализации, то сразу появляется несколько терминов, которые часто употребляются:

- 1) **основная система** (*host*) – это (базовая) операционная система или сервер, где работает программное обеспечение виртуализации;

- 2) **гостевая система** (*guest*) – это виртуализированный сервис (в основном какая-нибудь операционная система или контейнер), работающий на основной системе;
- 3) **гипервизор** (*hypervisor*) – это специализированное программное обеспечение виртуализации, которое управляет абстрагированием оборудования и совместным использованием ресурсов на платформе виртуализации;
- 4) **образ** (*image*) – это файл или набор файлов, которые представляют файловую систему гостевой системы;
- 5) **виртуальная машина** (*virtual machine*) – это абстрагированное оборудование или набор ресурсов, где работает гостевая система.

### 6.1.2. Обзор рисков виртуализации

Тем не менее виртуализация привносит и определенные риски. Если мы спросим о виртуализации архитекторов или других осведомленных о рисках лиц, то они будут говорить о беспорядочном росте виртуальных машин, о проблемах, относящихся к надежности и ненадежности программных интерфейсов (API), о высокой сложности виртуальных служб и о том, что отсутствует.

Решение проблем самой виртуализации выходит за рамки этой главы, но существуют некоторые риски, которые находятся в пределах досягаемости SELinux. Если мы можем интегрировать SELinux с виртуализацией, тогда мы можем более легко снизить эти риски.

Первый риск – это **чувствительность** [к модификации, несанкционированному доступу и утечке] **данных** (*data sensitivity*), размещенных на виртуальной машине. Всякий раз, когда несколько виртуальных машин размещаются совместно, может так случиться, что одна гостевая система сможет получать доступ к защищаемым данным, размещенным на другой виртуальной машине. Случиться это может и из-за какого-нибудь изъяна в программных средствах, обеспечивающих виртуализацию, и из-за сетевых возможностей этих программных средств, и из-за атак по каким-либо побочным каналам доступа.

В SELinux чувствительность данных может контролироваться посредством использования средств, предоставляемых в рамках многоуровневой защиты (MLS). Гостевые системы могут работать с различными метками (уровней доступа и категорий информации) таким образом, что защищенность данных гарантирована даже на уровне виртуализации.

Другой риск – **безопасность образов** [дисков] **гостевых систем, находящихся в выключенном состоянии** (*security of offline guest images*). Тогда и администраторы, и неверно настроенные виртуальные машины могут получить доступ к образу другой гостевой системы. SELinux может предотвратить это через правильно промаркированные гостевые образы и гарантировать, что эти образы [дисков], находящиеся в выключенном состоянии, принципиально отличаются от образов включенных виртуальных машин.

Виртуальные машины могут также **израсходовать ресурсы** (*exhaust the resources*) системы. В операционных системах Linux ресурсами можно управлять через **подсистему контрольных групп** (**cggroups** – *control groups*). Поскольку эта подсистема управляется через обычные файловые программные интерфейсы (APIs), SELinux может быть использована для дальнейшего контроля доступа к этому объекту, гарантируя, что контрольные группы, относящиеся, например, к средству контейнеризации Docker, остаются исключительно под контролем Docker.

**Атаки прерывания** [передачи управления] (*Break-out attacks*), когда уязвимости в гипервизоре используются для попыток получить доступ к основной системе; они могут быть сдержаны за счет принудительного применения типов [допустимого набора функций] SELinux к объектам и субъектам доступа, так как даже гипервизор не требует полного административного доступа ко всему, что есть на основной системе (*host*).

SELinux также может использоваться для **авторизации доступа к гипервизору** (*authorize access to the hypervisor*), гарантируя, что только группы, которым разрешено (при помощи средства контроля доступа, основанного на ролях), могут управлять работой гипервизора и его настройками.

Наконец, SELinux предлагает еще и усовершенствование **изоляция гостевой системы** (*guest isolation*), которое выходит за рамки только доступа к образам гостевых систем. Благодаря реализации защиты по категориям информации (MCS) в SELinux гостевые системы могут быть отделены друг от друга при помощи мандатного механизма разграничения доступа. И с помощью принудительного контроля типов (допустимых наборов функций, присваиваемых объектам) разрешенное поведение гостевых систем может быть явно задано и затем проконтролировано. Эта ключевая возможность используется поставщиками услуг по размещению у себя ресурсов клиента (*hosting providers*), поскольку она (возможность) позволяет запускать ненадежные (для поставщиков услуг) гостевые виртуальные машины.

### 6.1.3. Использование типов для объектов виртуальной инфраструктуры

Тем не менее SELinux не является полным решением по защите виртуализации. Один главный недостаток SELinux – это его нединамичность. Когда мы присваиваем некий тип [допустимого набора функций] виртуальной машине, то он является закрепленным и, образно говоря, «высеченным в камне». Однако виртуальные машины при этом будут иметь различное поведение, зависящее от программного обеспечения, работающего под их управлением.

Виртуальная машина, запускающая какой-нибудь веб-сервер, будет вести себя иначе, чем та, в которой работает некая база данных или шлюз электронной почты. Хотя администраторы политики безопасности SELinux могли бы создавать новые типы для каждой виртуальной машины, это является неэффективным решением. В результате большинство политик SELinux предлагают



только несколько типов, которые могут использоваться виртуальной машиной, обладающей широким спектром характеристик.

В дистрибутиве Red Hat Enterprise Linux такие типы являются частью реализации технологии sVirt.

### 6.1.4. Перенастраиваемое применение существующих типов виртуализации

В технологии sVirt компания Red Hat предложила перенастраиваемый метод (*reusable approach*) для поддержки защиты SELinux в виртуализации и контейнеризации. Метод предполагает применение SELinux-типов, которые могут использоваться независимо от базовой платформы виртуализации.

К таким типам (*virtualization domains*) относятся следующие:

- `virtd_t` используется программным обеспечением гипервизора;
- `svirt_t` применяется гостевыми системами (виртуальными машинами), которые не требуют превалирующего использования ресурсов основной системы;
- `svirt_qemu_net_t`, `svirt_kvm_net_t` и `svirt_lxc_net_t` используются гостевыми системами, которые требуют большего взаимодействия с основной системой (допустим, из-за паравиртуализации или из-за полувиртуального метода, такого как контейнеризация);
- `svirt_tcg_t` применяется гостевыми системами, которые требуют более гибкого доступа к памяти (исполнения сегментов записываемой памяти). Этот тип используется для гостевых систем, чья эмуляция/виртуализация требует использования TCG (*Tiny Code Generator* – небольшой генератор кода);
- `svirt_image_t` присваивается файлам образа, который содержит данные гостевой системы;
- `virt_image_t` присваивается файлам образа, которые не используются в конкретный момент;
- `virt_content_t` присваивается файлам образа, когда они используются в режиме «только чтение».

Чтобы придать перечисленным выше типам некоторую гибкость в части разрешений, введены в действие дополнительные логические параметры (*booleans*) SELinux. Для того чтобы получить перечень этих логических параметров и их текущие значения, применяется команда `semanage boolean`, к примеру с такими аргументами:

```
# semanage boolean -l | grep virt_
virt_rw_qemu_ga_data      (off , off)
  Allow qemu-ga to manage qemu-ga date
virt_use_nfs              (on , on)
  Allow confined virtual guests to manage nfs files
virt_use_comm             (off , off)
  Allow confined virtual guests to use serial/parallel communication ports
virt_sandbox_use_fusefs  (off , off)
```

```

Allow virt to sandbox use fusefs
...
virt_use_samba                (off , off)
  Allow confined virtual guests to manage cifs files

```

**P** В русскоязычном варианте вывод может выглядеть следующим образом:

```

# semanage boolean -l | grep virt_
virt_rw_qemu_ga_data          (off , off)
  Позволяет агенту гостевой системы qemu-ga управлять датой qemu-ga
virt_use_nfs                   (on , on)
  Позволяет ограниченному в правах виртуальным гостевым системам управлять файлами NFS
virt_use_compl                 (off , off)
  Позволяет ограниченному в правах виртуальным гостевым системам использовать
  параллельные и последовательные порты для взаимодействия
virt_sandbox_use_fusefs       (off , off)
  Разрешить виртуальным машинам в изолированной среде использовать файловую систему
  в пользовательском пространстве, созданном на основе драйвера fuse
...
virt_use_samba                (off , off)
  Позволяет ограниченному в правах виртуальным гостевым системам управлять
  файлами протокола cifs

```

Управление логическими параметрами выполняется через команды. Например, для того чтобы разрешить виртуальным машинам управлять объектами сетевой файловой системы NFS (*Network File System*), следует выполнить следующую команду:

```
# setsebool -P virt_use_nfs on
```

Использование логических параметров SELinux для управления ограничениями у типов, назначаемых объектам виртуальной инфраструктуры, следует тщательно проработать. Логические параметры влияют на политику безопасности SELinux на уровне основной системы (*host level*) и не могут быть использованы для изменения контроля доступа у отдельных гостевых систем. Поэтому предыдущий пример, позволяющий виртуальным машинам управлять объектами сетевой файловой системы (NFS), применим ко всем виртуальным машинам, запущенным на основной системе.

Если должны быть разрешены операции, способные повлиять на защищаемые параметры гостевых систем, то целесообразно запускать эти гостевые системы на отдельных основных системах (*isolated host*), которым эти операции тоже позволены, в то время как другие гостевые системы следует разместить на иных основных системах, для которых политика не позволяет эти конкретные действия.

Администраторы могут также использовать разные SELinux-типы для конкретных гостевых систем и настраивать в результате контроль доступа для каждой виртуальной машины отдельно. Как именно это можно сделать, зависит, конечно, от конкретной базовой технологии. Чуть позже в этой главе мы рассмотрим данный вопрос на примере виртуализации, основанной на libvirt, и для контейнеризации на базе Docker.

### 6.1.5. Рассмотрение защиты различных категорий

Следует заметить, что применение указанных ранее типов SELinux является необходимой, но недостаточной мерой для построения изолированных друг от друга гостевых систем. Технология sVirt добавляет другой уровень безопасности, обеспечивая **защиту для различных категорий** – **MCS** (*Multi-Category Security*), на которые разделяются защищаемые объекты.

В SELinux есть типы, которые обозначаются как имеющие ограничения по категориям. Для объектов таких типов будет отсутствовать доступ к ресурсам, не имеющим соответствующих наборов категорий в текущих параметрах безопасности.

Внедрение технологии sVirt гарантирует, что объектам виртуальной инфраструктуры, которым назначены ранее упомянутые типы, будут присвоены и ограничивающие. Это можно проверить, выполнив следующий запрос к системе, который выдаст типы, сопоставленные с атрибутом `mcs_constrained_type`:

```
# seinfo -amcs_constrained_type -x | grep virt_
svirt_kvm_net_t
svirt_lxc_net_t
svirt_tcg_t
svirt_t
svirt_qemu_net_t
```



Если создатели политик хотят сделать собственный тип [допустимого набора функций] для работы с ним программных средств, обеспечивающих виртуализацию, им будет необходимо также указать его как тип, работающий с ограничениями в области категорий (используя макрос `mcs_constrained()`), и как тип, относящийся к виртуальной инфраструктуре, используя макрос `virt_domain_template()`.

Разработка политик кратко затронута в главе 8 «Работа с политиками SELinux». Более обстоятельным материалом о разработке политик является книга издательства Packt's *SELinux Cookbook* (<https://www.packtpub.com/networking-and-servers/selinux-cookbook>).

Именно с помощью разграничения по категориям технология sVirt выполняет необходимую изолированность гостевых систем. Каждой работающей виртуальной машине (в основном запущенной с типом `svirt_t`) будут назначены две (произвольные) категории SELinux. Образы [дисков], которые используются при работе такой виртуальной машины, также получают категории, соответствующие категориям этих машин.

И тогда если виртуальная машина захочет получить доступ к чужому образу, то за счет разницы в категориях ею будет получен отказ в доступе от системы защиты. Аналогично этому, если одна виртуальная машина пытается присоединиться к другой виртуальной машине (в т. ч. атаковать), то защитные механизмы MCS снова предотвратят эти действия.

sVirt выбирает для гостевой системы две произвольные категории, чтобы увеличить количество их уникальных комбинаций, даже в тех условиях, когда диапазон поддерживаемых вариантов для категорий не является большим. Предположим, например, что гипервизор работает с диапазоном категорий от

c10 до c99. Это означает, что гипервизор может оперировать только девятью десятками категорий. Если каждая гостевая система получает только одну категорию, тогда гипервизор может поддерживать работу для 90 изолированных гостевых систем, а потом наступит ситуация, когда новые гостевые системы будут находиться в одной категории с уже имеющимися, что может позволить им взаимодействовать (если будет присутствовать какая-нибудь уязвимость, которая позволит это; конечно, программное обеспечение гипервизора в основном будет препятствовать такому непредусмотренному доступу). Тем не менее если каждой гостевой системе будет назначаться по две категории, которые вместе будут формировать уникальную комбинацию, то число поддерживаемых параллельно работающих гостевых систем станет равным 4005 (это число уникальных пар в наборе из 90 элементов, и получено оно по формуле  $n*(n - 1)/2$ ).

## 6.2. ПОДДЕРЖКА БИБЛИОТЕКИ LIBVIRT

Проект libvirt предлагает некоторый уровень абстракции для виртуализации, посредством которого администраторы могут управлять виртуальными машинами без конкретного понимания о них или выяснения, на какой базовой платформе виртуализации они работают. Поэтому администраторы могут использовать инструменты, предлагаемые libvirt, для управления виртуальными машинами, работающими на QEMU, QEMU/KVM, Xen, и т. д.

Для использования технологии sVirt библиотека libvirt может быть собрана (будучи открытым программным обеспечением) с поддержкой функций SELinux. В этом случае, при условии что гостевые системы имеют маркировку SELinux, библиотека libvirt будет учитывать в обязательном порядке типы sVirt. Код libvirt также будет выполняться с учетом категорий, позволяющих обеспечить изоляцию гостевых систем, а также будет гарантировать, что файлам образов присвоены соответствующие (политике) параметры безопасности (образы, которые используются, имеют параметры безопасности, отличающиеся от параметров не используемых в текущий момент образов).

Благодаря внедрению технологии защиты sVirt компания Red Hat смогла получить сертификат **Common Criteria at Evaluation Assurance Level 4+ (CC EAL 4+)** (уровень безопасности платформы) для виртуализации, основанной на Red Hat Enterprise Linux и KVM. Конечно, это заслуга не только технологии sVirt, но она повлияла на получение данного сертификата.

**Р** В Российской системе сертификации, регулируемой Федеральной службой по техническому и экспортному контролю (ФСТЭК), данный уровень безопасности соответствует четвертому усиленному оценочному уровню доверия – «ОУД 4 (усиленный)», если руководствоваться документом «Безопасность информационных технологий. Критерии оценки безопасности информационных технологий».

### 6.2.1. Различные случаи маркировки ресурсов

Различие среди меток для образов обусловливается необходимостью поддержки различных случаев использования виртуальной инфраструктуры. Файл с образом диска, на котором размещается ОС гостевой системы, маркируется в основной системе типом `svirt_image_t`. Пара категорий (о которых говорилось выше) присваивается точно такая же, какая назначена и гостевой системе в работающем состоянии (запускаются виртуальные машины с типом `svirt_t`). Этот файл с образом доступен для записи в него гостевой системой.

В том случае когда есть какой-нибудь файл с образом [диска], на который планируется записывать информацию множеству виртуальных машин, тогда библиотека libvirt может сделать так, что никакие категории назначаться данному файлу с образом не будут. При отсутствии категорий ограничения системы MCS применяться не будут (точнее, ограничения также действуют, но, учитывая, что отсутствие категорий означает их пустой набор, любой другой набор категорий будет иметь преимущество над ним, и поэтому действия по отношению к файлам с такой маркировкой будут разрешены).

Образы, которые нужно подключить в гостевую систему с правами «только для чтения» (такие как загрузочные диски), маркируют типом `virt_content_t`. Если образы узкоспециализированные (например, для выполнения одной какой-нибудь задачи или функции), тогда категории также могут быть назначены. Для общего доступа на чтение никакие категории назначать не нужно.

Стоит отметить, что эти различия в метках применяют главным образом к технологиям виртуализации, а не к технологиям контейнеризации.

### 6.2.2. Оценка архитектуры libvirt

Проект libvirt имеет несколько клиентских приложений, которые взаимодействуют с фоновым процессом `libvirtd`. Этот фоновый процесс отвечает за управление программным обеспечением локального гипервизора (это может быть QEMU/KVM, Xen или любое другое средство виртуализации) и даже способен управлять удаленными гипервизорами.

Эта последняя функциональная возможность часто используется для запатентованных (*proprietary*) [коммерческих] гипервизоров, которые предлагают необходимые программные интерфейсы (API) для управления виртуальными ресурсами на основной системе (*host*).

Эта высокоуровневая архитектура показана на рис. 6.1.

sVirt хорошо соответствует характерной для libvirt природе кросс-платформенности и способности взаимодействовать с различными гипервизорами. Вместо специфичных для конкретного гипервизора типов в SELinux используются обобщенные (но все равно ограниченные) типы, гарантирующие безопасность среды окружения.

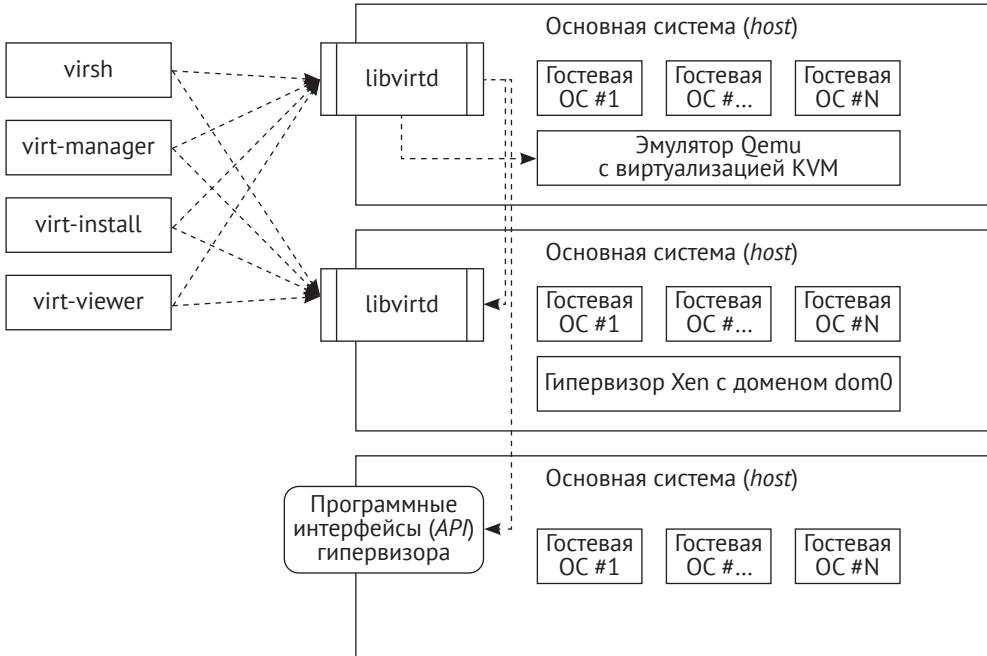


Рис. 6.1 ❖ Высокоуровневая архитектура программного обеспечения libvirt

### 6.2.3. Настройка libvirt для работы с sVirt

Большинство систем, поддерживающих libvirt на операционных системах с защитой SELinux (таких как Red Hat Enterprise Linux), автоматически поддерживают и работу библиотеки libvirt с включенными в нее функциями SELinux. Если это не такой случай, но поддержка SELinux в принципе возможна, тогда все, что нужно сделать, – это настроить libvirt так, чтобы ею поддерживалась модель защиты SELinux.

Например, для драйвера виртуализации на основе QEMU это можно сделать так:

```
# vim /etc/libvirt/qemu.conf
security_driver = "SELinux"
```

Когда гостевая система запущена на платформе виртуализации, то мы можем видеть доказательство использования в ней sVirt через присвоенные метки:

```
# ps -wwC qemu-kvm -o label,command
LABEL                                COMMAND
system_u:system_r:svirt_t:s0:c23,c89 /usr/bin/qemu-kvm ...
system_u:system_r:svirt_t:s0:c48,c52 /usr/bin/qemu-kvm ...
```

В левой колонке (*LABEL* – метка) указаны параметры безопасности, а в правой (*COMMAND*) – наименование исполняемого файла/команды.

Для того чтобы вывести список определенных в текущий момент гостевых систем (называемых доменами в libvirt), следует использовать такую команду:

```
# virsh list
Id   Name           State
-----
10   wastomcat1     running
12   wastomcat2     running
```

**P** В русскоязычном варианте вывод команды может выглядеть следующим образом:

```
# virsh list
Идентификатор  Имя гостевой системы  Состояние
-----
10              wastomcat1             Работа
12              wastomcat2             Работа
```

Текущая конфигурация домена (типа гостевой системы) может быть получена, если использовать параметр `dumpxml` с командой пользовательского интерфейса `virsh`:

```
# virsh dumpxml wastomcat1
<domain type='kvm' id='10'>
  <name>wastomcat1</name>
  ...
  <seclabel type='dynamic' model='SELinux' relabel='yes'>
    <label>system_u:system_r:svirt_t:s0:c23,c89</label>
    <imagelabel>
      system_u:object_r:svirt_image_t:s0:c23,c89
    </imagelabel>
  </seclabel>
</domain>
```

Приведенные здесь настройки параметров безопасности (`seclabel`), будучи частью определяющей структуры для домена, в которую входит конкретная гостевая система, сообщают libvirt, как именно надо применять политику безопасности для данного домена. В предыдущем примере защита гостевых систем использует функции SELinux (что определено параметром `model='SELinux'`), а библиотеке libvirt разрешено динамически распределять параметры безопасности, включая категории. Разрешения для libvirt в примере определяются следующими параметрами: `type='dynamic'` и `relabel='yes'`.

Создание доменов для гостевых систем может быть выполнено с использованием команды `virt-install`. В результате создается некое описание на языке XML, которое в дальнейшем может редактироваться. Также возможно определять новые домены через XML-файл и потом его использовать:

```
# virsh define SomeDomainDefinition.xml
```

Для редактирования настроек уже имеющегося описания домена, поддерживающего настройки, касающиеся технологии sVirt, следует использовать команду `virsh edit`:

```
# virsh edit wastomcat1
```

Следующие несколько подразделов покажут, как обновлять такие домены сведениями, касающимися защиты в SELinux.

### 6.2.4. Использование статических параметров безопасности

Когда администраторы управляют описаниями доменов, используемых для работы в библиотеке libvirt, то они могут указать, что параметры безопасности не подлежат изменению (являются статическими). Такой подход будет гарантировать, что libvirt не станет динамически изменять параметры безопасности. Вместо этого будет использоваться тот набор параметров, который определен администратором.

Когда используются статические параметры безопасности, администратор может выбрать: включить или отключить возможность изменять параметры безопасности (*relabeling*). Если их смена разрешена, то libvirt будет пытаться обеспечить, чтобы указанная метка ресурса (файла образа) была применена к образу, даже несмотря на то, что параметры безопасности у домена являются зафиксированными. Если администратор запретит смену, тогда libvirt будет считать, что имеющиеся параметры безопасности на ресурсе (файле образа) являются верными.

Для установки статических параметров безопасности отредактируйте описание настроек для домена, указав в конструкции `seclabel`, что `type='static'`:

```
# virsh edit wastomcat1
<domain>
  <name>wastomcat1</name>
  ...
  <seclabel type='static' model='SELinux' relabel='no'>
    <label>system_u:system_r:svirt_t:c1,c2</label>
  </seclabel>
  ...
</domain>
```

### 6.2.5. Гибкая настройка параметров безопасности

Администраторы могут выбирать разные параметры безопасности, при этом сохраняя динамическую природу работы libvirt. Например, администраторы могут указать libvirt, что надо использовать конкретный домен, но при этом все же управлять категориями MCS, выбрав базовые параметры безопасности (*base label*):

```
# virsh edit wastomcat1
...
<seclabel type='dynamic' model='SELinux'>
  <baselabel>system_u:system_r:custom_svirt_t:s0</baselabel>
</seclabel>
```

Динамическая модель автоматически подразумевает смену параметров безопасности на ресурсах.



## 6.2.6. Использование разных мест хранения

Когда место для хранения образов libvirt отличается от размещения `/var/lib/libvirt/images/`, то администраторам необходимо убедиться, что новое место размещения имеет тип `virt_image_t`.

Это надо сделать обязательно, т. к. фоновый процесс программного обеспечения libvirt может менять параметры безопасности только для файлов, чьи SELinux-типы относятся к числу назначаемых объектам виртуальной инфраструктуры (кроме Red Hat Enterprise Linux, где libvirtd работает с менее ограниченным типом [допустимого набора функций] и поэтому имеет больше привилегий).

Например, если используется каталог `/srv/virt/images`, то нам стоит сделать следующее:

```
# semanage fcontext -a -t virt_var_lib_t "/srv/virt(/.*)?"
# semanage fcontext -a -t virt_image_t "/srv/virt/images(/.*)?"
# restorecon -R /srv/virt
```

Если место для хранения [файлов] размещается на подключаемом ресурсе с сетевой файловой системой NFS, тогда надо подключить логический параметр SELinux `virt_use_nfs`:

```
# setsebool -P virt_use_nfs on
```

## 6.2.7. Интерпретация информации в поле вывода данных о метке

Когда выводится информация из XML-файла, содержащего описание домена для работы libvirt, то добавляется некоторая информация о параметрах безопасности (*security label*), которую нельзя изменить. Эта исключительно выводимая информация непосредственно относится к параметрам безопасности файла образа.

Давайте посмотрим на следующий пример:

```
<seclabel type='dynamic' model='SELinux' relabel='yes'>
  <label>system_u:system_r:svirt_t:s0:c192,c392</label>
  <imagelabel>
    system_u:object_r:svirt_image_t:s0:c192,c392
  </imagelabel>
</seclabel>
```

В этом примере показано, что тип маркировки является динамическим (*dynamic*). В результате в процессе вывода информации о настройках безопасности libvirt-домена были сгенерированы поля `<label>` и `<imagelabel>`, предназначенные только для вывода данных. Они показывают администратору, что из себя представляют текущие метки параметров безопасности.

Если администратор станет вручную редактировать XML-файл и включит в него информацию для поля `<imagelabel>`, то она будет проигнорирована системой защиты, т. к. эти сведения должны быть динамически созданы. В сущ-

ности, информация поля <label> интерпретируется в зависимости от атрибута типа, который определяется в записи <seclabel>.

### 6.2.8. Управление доступными категориями

Когда система libvirt выбирает произвольные категории [для назначения их гостевым системам], то делает это, исходя из собственного диапазона доступных категорий. Изначально в системе защиты, основанной на разделении по категориям (MCS), диапазон задан от c0 до c1023. Для того чтобы изменить этот диапазон, фоновый процесс libvirtd должен быть запущен с соответствующими параметрами безопасности [которые будут включать необходимый диапазон категорий].

В операционных системах Red Hat Enterprise Linux это может сделать, например, так: обновить соответствующий модуль (libvirtd.service) службы systemd, ответственный за работу фонового процесса libvirtd (путем перемещения этого файла из каталога /usr/lib/systemd/system в каталог /etc/systemd/system) и добавлением в него атрибута с параметрами безопасности SELinuxContext=:

```
# vim /etc/systemd/system/libvirtd.service
...
[Service]
...
SELinuxContext=system_u:system_r:virtd_t:s0:c10.c99
```

В системах, где инициализация выполняется с помощью SysV, в основном требуется обновить иницилирующий скрипт и включить выражение runcon:

```
# vim /etc/rc.d/init.d/virtd
...
runcon -l s0:c10.c99 /usr/sbin/libvirtd --config /etc/libvirt/libvirtd.conf --listen
```

Каждый раз, когда запускается новая гостевая система, код libvirt будет произвольно выбирать две категории и проверять, являются ли эти категории частью его собственного диапазона, а также то, присвоены ли уже эти две категории какой-либо другой гостевой системе. Если сформированная пара категорий уже занята другой системой, то категории будут выбираться до тех пор, пока не будет найдена свободная пара.

### 6.2.9. Поддержка интерпретирующих доменов

Проект libvirt поддерживает управление окружением, состоящим из множества основных систем (т. е. кластера), и управление перемещением гостевых систем с одной основной системы на другую. Когда libvirt использует систему защиты SELinux, то можно подумать, что каждая основная система (*host*) должна быть запущена с одной и той же политикой SELinux (или хотя бы с политикой, использующей одинаковую интерпретацию [правил]), потому что если это не так, то нельзя гарантировать выполнение модели защиты, применяемой на уровне libvirt-доменов.

Хотя этот метод и возможен, он не является обязательным. С помощью libvirt администраторы могут пометать (*tag*) те основные системы, которые используют одну и ту же политику SELinux, и пока это так, гарантировать, что libvirt-домены перемещаются исключительно между основными системами с одинаковой политикой SELinux (или с одинаковой интерпретацией правил политики).

Для того чтобы это обеспечить, libvirt использует концепцию **интерпретирующих доменов** DOI (*domain of interpretation*). Мы уже имели дело с интерпретирующими доменами, когда говорили о технологии маркировки сетей NetLabel/CIPSO. Интерпретирующий домен для libvirt не имеет никакого отношения к такому же домену для NetLabel/CIPSO, но служит похожей цели. И так же, как для протокола CIPSO, libvirt использует целые числа для разграничения систем с различными реализациями политик.

Например, мы можем взять основные системы, где политика многоуровневой защиты (MLS) включена, и отделить их от других основных систем, где включена политика MCS – обеспечивающая разграничение по категориям. Системам с политикой многоуровневой защиты может быть присвоено значение DOI, равное 1 (единице), а для систем с политикой MCS – значение DOI, равное 0 (нулю).

Настройки для интерпретирующих доменов можно сделать в соответствующем файле конфигурации. В каком именно файле, зависит от применяемой технологии виртуализации. Каталог размещения для таких файлов конфигурации называется `/etc/libvirt`, а для того чтобы просмотреть его, надо выполнить основной системе команду `virsh capabilities`:

```
# virsh capabilities
Connecting to uri: qemu:///system
<capabilities>
  <host>
    ...
    <secmodel>
      <model>SELinux</model>
      <doi>0</doi>
    </secmodel>
  </host>
  ...
</capabilities>
```

## 6.2.10. Изменение параметров безопасности, установленных по умолчанию

Когда libvirt включается в маркировку доменов, то использует по умолчанию следующие значения для SELinux-типов: `svirt_t` или `svirt_tcg_t`. Это не жестко запрограммированное решение в проекте libvirt и может настраиваться в файле с параметрами безопасности, используемом для этих целей по умолчанию. Два файла, размещенных внутри каталога `/etc/selinux/targeted/contexts`, ис-

пользуются в libvirt для определения параметров безопасности, которые должны применяться по умолчанию:

- это файл `virtual_domain_context`, который содержит параметры безопасности для запущенной гостевой системы (виртуальной машины),
- и файл `virtual_image_context`, содержащий используемые по умолчанию SELinux-типы для файлов образов гостевых систем.

Администраторы могут обновлять содержание этих файлов, чтобы использовать разные значения по умолчанию, но должны учитывать, что эти файлы в основном перезаписываются, когда устанавливается пакет новой политики. Поэтому рекомендуется придерживаться базовой маркировки самого libvirt.

## 6.3. ЗАЩИЩЕННЫЕ КОНТЕЙНЕРЫ DOCKER

До сих пор мы рассматривали набор средств для управления виртуализацией libvirt и полную виртуализацию. Но есть и другой тип виртуализации, который предполагает контейнеризацию. Частным случаем являются контейнеры Docker.

Во время работы с контейнерами администраторы должны хорошо понимать, что контейнеры не виртуализируют все: само ядро Linux общее, и все программное обеспечение, работающее внутри контейнера, взаимодействует с ядром Linux так же, как и программные средства, работающие вне его. Однако это не означает, что контейнеры являются изолированными системами. Они построены, основываясь на таких функциональных возможностях Linux, как пространства имен (*namespaces*) и контрольные группы (*control groups*).

### 6.3.1. Представление о защите контейнера

Поскольку ядро Linux является общим, то вредоносные программы, способные эксплуатировать уязвимость на уровне ядра, влияют на всю основную систему (*host*), и можно скомпрометировать не только контейнер, в котором выполняется вредоносная программа, но также и другие контейнеры и программные средства, работающие на этой основной системе.

Для предотвращения эксплуатации уязвимостей можно использовать SELinux. Политики могут применяться для обеспечения того, чтобы программные средства, работающие в контейнере, не могли бы производить никаких действий, которые не являются разрешенным поведением для этих программных средств.

При этом, поскольку ядро Linux является общим, контейнеры имеют самые различные способы для поддержки функций безопасности, имеющихся в SELinux. Правда, в то время как полная виртуализация позволяет гостевой операционной системе использовать внутри себя SELinux тоже, то контейнеры не имеют такой роскоши. В настоящее время даже если внутри контейнера работает программное обеспечение, которое проверяет [для дальнейшего ис-

пользования], подключены ли средства системы защиты SELinux, оно получает в качестве ответа статус, говорящий об отключенном состоянии системы, даже если это на самом деле не так.

Причина подобного поведения в том, что самой системе защиты SELinux неизвестно о пространстве имен, т. к. изоляция своего пространства имен – это ключевая особенность контейнеров в Linux. SELinux же является частью ядра операционной системы Linux, и политика применяется ко всей основной системе. Поэтому придется работать с SELinux только на уровне основной системы, во всяком случае до тех пор, пока SELinux не поддерживает различные слои политик (не разделяет политику, относящуюся к основной системе, от политики, относящейся к контейнерам) и не поддерживает контроль доступа в конкретном пространстве имен (гарантируя, что политики, определяющие безопасную работу контейнеров, не могут выйти за пределы пространства имен этих контейнеров).

### 6.3.2. Интеграция системы защиты с контейнерами без sVirt

Когда в процессе установки программного обеспечения Docker не будет использоваться поддержка технологии sVirt, тогда в дальнейшем эта технология и не будет применяться при работе контейнера. Но это не означает, что и система защиты SELinux не будет задействована. Контейнеры будут работать, используя функциональные возможности типов `docker_t` или `container_runtime_t` (которые являются почти неограниченными), которые могут быть настроены при помощи логических параметров SELinux.

**i** Между типами `docker_t` или `container_runtime_t` нет различия. Наименование типа было изменено, чтобы показать поддержку различных средств контейнеризации, а не только Docker. Далее в тексте мы будем использовать тип `docker_t`.

Важными логическими параметрами SELinux, которые влияют на функции управления доступом для объектов с типом `docker_t`, являются следующие:

- логический параметр `docker_connect_any`, разрешающий объектам, имеющим тип `docker_t`, подключаться к любому TCP-порту (а не только к специальным портам управления Docker);
- логические параметры `selinuxuser_execheap` и `selinuxuser_execstack` (в операционной системе Red Hat Enterprise Linux) или параметр `allow_execheap` и `allow_execstem` (в Gentoo), которые, как правило (не только для типа `docker_t`), позволяют выполнять сегменты памяти, доступные для записи.

Из всех средств управления защитой, которые упомянуты в следующих пунктах, средства, указанные в пункте «*Применение различных параметров безопасности для контейнеров*», могут быть использованы, даже когда локальный экземпляр Docker не поддерживает sVirt. Но при этом требуется, чтобы Docker поддерживал бы работу с SELinux, а политики SELinux поддерживали необходимые преобразования типов [допустимых наборов функций].

### 6.3.3. Перестраховка безопасности Docker средствами защиты sVirt

Docker имеет похожий подход к виртуализации, что и другие гипервизорподобные (*hypervisor-like*) решения. Он имеет компонент управления (фоновый процесс Docker) и запускает гостевые системы, в которых работает контейнерное программное обеспечение. Гостевыми системами в этом случае являются процессы, которые подключают новые пространства имен (*namespaces*) и запускают программные средства контейнера в качестве подпроцессов. Новые же пространства имен позволяют скрывать от гостевой системы представления файлов, процессов, пользователей и многое другое, что работает на уровне основной системы.

Исходно Docker не имеет встроенной технологии sVirt (или хотя бы ее подходов к защите). Это означает, что когда Docker запущен на основной системе, которая поддерживает уже включенную в работу SELinux, то процессы, действующие как контейнеры, будут выполняться с типом `docker_t`, с тем типом, который присвоен управляющему фоновому процессу Docker. И часто это работает так, как и ожидается, но не изолирует гостевые системы, как следует.

Компания Red Hat адаптировала Docker для поддержки подходов к защите sVirt. Будучи запущенными на базе программного обеспечения Docker компании Red Hat, новые гостевые системы работают в функциональных границах изолированных типов (`svirt_lxc_net_t`) и даже получают набор MCS-категорий для обеспечения изоляции между контейнерами. Мы вполне можем ожидать, что Docker продолжит развиваться в направлении охвата большего числа средств защиты SELinux, а такой подход компании Red Hat может стать стандартом в ближайшем будущем.

Например, процесс веб-сервера Nginx, запущенный внутри некоего контейнера Docker, является промаркированным типом `svirt_lxc_net_t` и имеет две необходимые для работы категории в параметрах безопасности:

```
~# ps -wwC nginx -o label,command
LABEL                                COMMAND
system_u:system_r:svirt_lxc_net_t:s0:c232,c590 nginx: master ...
system_u:system_r:svirt_lxc_net_t:s0:c232,c590 nginx: worker ...
```

Поскольку Docker не использует такую же концепцию работы с образами, как и технологии, выполняющие виртуализацию более высокого уровня (через систему libvirt), то типы, присвоенные образам (такие как `svirt_image_t` и `virt_content_t`), не применяют к типу `svirt_lxc_net_t` даже принудительно. Вместо этого SELinux-типы файлов, относящихся к основной системе (например, такие типы, как `usr_t`, подразумевающие исключительный доступ на чтение (*read-only*)), могут использоваться только с типом `svirt_sandbox_file_t`, явно применяемым к ресурсам, которые могут быть полностью управляемыми контейнером.

В отличие от системы `libvirt`, параметры контейнеров Docker не определяются через XML-файлы. Внесение изменений в управление защитой SELinux, применяемой к контейнерам, осуществляется через командную строку.

### 6.3.4. Ограничение привилегий контейнера

Сильной стороной Docker является возможность изменять привилегии (*capabilities*) контейнера. Привилегии определяют крупномасштабные разрешения в Linux и используются системой контейнеризации Docker для ограничения того программного обеспечения, которое выполняется внутри контейнера.

Например, выключение основной системы (*host*) (а это является командой, выполняемой на уровне ядра) не разрешается по умолчанию выполнять из контейнера Docker.

Исходно Docker ограничивает привилегии, которые являются действующими для контейнера. Администраторы же могут в дальнейшем провести точную настройку этих привилегий или даже напрямую включить только конкретный набор привилегий. Ограничениями привилегий контейнера может быть подорвана деструктивная деятельность программ, эксплуатирующих уязвимости ядра. Также они могут привести к ограничению сбоев в работе контейнеров.

Для того чтобы удалить привилегии, надо использовать команду `docker run` с параметром `--cap-drop` (в следующем примере `nimmis/alpine-nginx` – это наименование общедоступного микроконтейнера, который размещает веб-сервер Nginx):

```
~# docker run --cap-drop mknod --name nginx -d nimmis/alpine-nginx
```

Мы можем использовать команду `ps tree` (часть пакета `psmisc`), чтобы увидеть процессы, которые являются запущенными. При помощи параметра `-S` можно даже показать, какие процессы переключили пространства имен (делая немало более очевидными те процессы, которые являются контейнерами):

```
~# docker ps
CONTAINER ID   IMAGE                COMMAND             CREATED         ...
bb59d5e2ef4f   260f538c3be0        "/boot.sh"         46 minutes ago ...
```

```
~# ps -ef | grep "boot.sh"
root  2955 2554  0 05:20 ?    00:00:00:00 /bin/sh /boot.sh
```

```
~# pstree -apS 2554
docker-current,2554,mnt daemon --exec-opt \
  native.cgroupdriver=systemd --SELinux-enabled \
  --log-driver=journald
  |-boot.sh,2955,ipc,mnt,net,pid,uts /boot.sh
  |   `--runsvdir,3001 -P /etc/service...
  |       |--runsv,3003 crond
  |       |   |--crond,3006 -f
  |       |   |--runsv,3004 rsyslogd
  |       |   `--rsyslogd,3008 -n
```

```

|         |         |-{rsyslogd},3010
|         |         |-{rsyslogd},3011
|         |         |-{rsyslogd},3012
|         |         `--runsv,3005 nginx
|         |         `--run,3007 ./run
|         |         `--nginx,3009
|         |         `--nginx,3013
|--{docker-current},2555
|--{docker-current},2556
|--{docker-current},2557
|--{docker-current},2558
|--{docker-current},2559
|--{docker-current},2560
|--{docker-current},2613
|--{docker-current},2692
|--{docker-current},2696
|--{docker-current},2697
`--{docker-current},2815

```

Привилегии этого работающего контейнера могут быть проверены через статус псевдофайла для заданного идентификатора процесса (PID) (в примере он обозначен как 2955):

```

~# grep Cap /proc/2955/status
CapInh: 00000000a00425fb
CapPrm: 00000000a00425fb
CapEff: 00000000a00425fb
CapBnd: 00000000a00425fb

```

Поскольку мы здесь имеем битовую последовательность, описывающую привилегии (в поле CapEff показываются те привилегии, которые являются не только эффективными, но и действующими в настоящий момент), то на самом деле довольно сложно сделать вывод о том, что отсутствует разрешение для выполнения системного вызова `mknod`. Но можно использовать команду `pscap` (являющуюся частью пакета `libcap-ng-utils` в RHEL или `sys-libs/libcap-ng` в Gentoo), для того чтобы показать текущие привилегии в более удобном для прочтения формате:

```

~# pscap | grep 2955
2554 2955 root boot.sh chown, dac_override, fowner, fsetid,
kill, setgid, setuid, setpcap, net_bind_service, net_raw,
sys_chroot, audit_write, setfcap

```

Администраторы могут также сбросить все привилегии у контейнера и добавить только те, которые необходимы:

```

~# docker run -d --cap-drop all --cap-add chown --cap-add dac_override ...

```

Конечно, может возникнуть желание адаптировать SELinux еще и для работы с новым перечнем привилегий. В конце концов, тот тип, присваиваемый по умолчанию, с которым работают контейнеры (имеющие поддержку sVirt), будет также иметь какое-то количество подключенных привилегий.



Существует несколько логических параметров (*booleans*) SELinux (только в операционной системе Red Hat Enterprise Linux), которые влияют на поддерживаемые привилегии, но эти логические параметры применяют ко всем контейнерам, работающим на одной и той же основной системе (*host*). К ним относятся следующие:

- `virt_sandbox_use_sys_admin` – включает привилегию `CAP_SYS_ADMIN` для контейнеров. Дает много-много системно-административных прав, поэтому лучше его отключать;
- `virt_sandbox_use_mknod` – включает привилегию `CAP_MKNOD` для контейнеров;
- `virt_sandbox_use_audit` – включает привилегию `CAP_AUDIT_WRITE` для контейнеров;
- `virt_sandbox_use_all_caps` – включает все возможные привилегии для контейнеров. Это позволяет запускать изолированную программную среду (*sandbox*) с полными правами пользователя `root` (в масштабах SELinux), и его следует использовать только тогда, когда контейнеры на основной системе являются полностью надежными.

Более тщательный подход к защите заключается в применении отдельных SELinux-типов для отдельных контейнеров, которым требуется настраивать дополнительные возможности. В следующем пункте описывается, как установить различные параметры безопасности SELinux для контейнеров Docker.

### 6.3.5. Применение различных параметров безопасности для контейнеров

Контейнеры Docker могут запускаться с разными параметрами безопасности SELinux, если указать параметр `-security-opt`. Параметры защиты, которые должны быть назначены, имеют следующий синтаксис из ключевых слов, обозначаемых как `label` (метка), `key` (ключ) и `value` (значение):

```
label:<key>:<value>
```

Могут быть использованы следующие варианты ключей:

- `user` устанавливает пользователя SELinux для контейнера;
- `role` устанавливает роль SELinux для контейнера;
- `type` устанавливает тип [допустимого набора функций] SELinux для контейнера;
- `level` устанавливает мандатную конфигурацию SELinux (информацию MLS/MCS) для контейнера;
- `disable` (когда значение у ключа отсутствует) снимает ограничения безопасности для контейнера.

К примеру, для того чтобы контейнер запускался с типом `custom_lxc_net_t`, вместо того чтобы использовать тип по умолчанию `svirt_lxc_net_t`, следует использовать такую команду:

```
~# docker run --security-opt label:type:custom_lxc_net_t ...
```

Аналогичным образом нужно применять ее, когда контейнеру необходимо работать с особым набором категорий:

```
~# docker run --security-opt label:level:s0:c3.c5,c128 ...
```

Те типы, которым разрешено преобразование из одного в другой (в рамках SELinux), контролируются политикой. По умолчанию тип `docker_t` может быть преобразован к типам гостевых систем, которые имеют установленный атрибут `svirt_sandbox_domain`:

```
~# seinfo -asvirt_sandbox_domain -x
svirt_sandbox_domain
svirt_kvm_net_t
svirt_lxc_net_t
openshift_initrc_t
svirt_qemu_net_t
```

Если планируется использовать какие-то другие типы (в т. ч. создаваемые пользователями), то администратору политики SELinux необходимо быть уверенным, что тип `docker_t` может быть преобразован к этим типам.

### 6.3.6. Перемаркировка подключенного тома данных

Docker имеет замечательную функцию, называемую **подключение тома данных** (*volume mounts*). Ее использование сводится к тому, что части файловой системы, имеющейся на основной системе, подключаются в файловую систему контейнера, позволяя программному обеспечению контейнера читать или даже записывать информацию в удаленные ресурсы. Важной является и та особенность контейнеров, что необходимо сохранять данные при перезагрузке контейнера, поскольку по умолчанию контейнеры не сохраняют свои изменения в своей собственной среде: каждый раз, когда контейнер вновь загружается, он остается таким же, каким был изначально при создании.

Вот пример того, как подключается (*mounts*) каталог, находящийся на основной системе `/srv/web/tomcat1`, в каталог контейнера `/srv/web`:

```
~# docker run -v /srv/web/tomcat1:/srv/web -d --name wastomcat1 ...
```

Но при включенной защите SELinux подключение томов может быть и недоступно контейнеру. А еще контейнер, запущенный с ограниченным типом (`svirt_lxc_net_t`), может и не иметь права на запись в область размещения `/srv/web/tomcat1` (которое может иметь тип `httpd_sys_content_t`).

В принципе, администраторы могут перемаркировать ресурсы вручную, но Docker, к счастью, умеет это делать тоже.

Если добавить параметр `:z` команде подключения тома данных, то Docker будет перемечать объекты типом `svirt_sandbox_file_t`, который является типом с полными правами управления для контейнеров, поддерживающих технологию sVirt:

```
~# docker run -v /srv/web/tomcat1:/srv/web:z -d --name wastomcat1 ...
```

При добавлении параметра `:Z` к команде подключения тома (именно заглавной Z) Docker не только перемаркирует объекты типом `svirt_sandbox_file_t`, но еще и присвоит им также набор категорий конкретному контейнеру. В результате файлы будут управляемы только этим конкретным контейнером (и не смогут дополнительно использоваться другими контейнерами):

```
~# docker run -v /srv/web/tomcat1:/srv/web:Z -d --name wastomcat1 ...
```

Если возникнет необходимость подключить какую-то область размещения [файлов] с уровнем доступа «только чтение», то администраторам будет нужно перемаркировать файлы вручную. У Docker нет (пока) опции для автоматической перемаркировки объектов с правом «только для чтения».

### 6.3.7. Понижение контроля со стороны SELinux для специальных контейнеров

Некоторые контейнеры Docker требуют работы с большими привилегиями. Такие контейнеры используются для размещения утилит управления основными системами и часто для управления самими контейнерами. Эти контейнеры размещаются на устойчивых [к сбоям] системах, которые содержат только необходимый минимум для размещения контейнеров Docker (например, основанные на операционной системе CoreOS).

Когда администраторы запускают контейнеры с выключенными ограничениями SELinux (используя ключ `disable` с параметром `security-opt`), то Docker будет запускать эти контейнеры с гораздо более привилегированным типом, называемым `src_t`. Название этого типа выбрано потому, что на самом деле является сокращением от **super-privileged container** (суперпривилегированный контейнер):

```
~# docker run --security-opt label:disable ...
```

Это по-прежнему ограничивает запуск контейнера для системы контейнеризации Docker, но при этом он больше не ограничивается с точки зрения системы защиты SELinux. Чтобы снять к тому же еще и ограничения, установленные на уровне системы Docker, используйте опцию `--privileged`:

```
~# docker run --privileged ...
```

### 6.3.8. Изменение параметров безопасности, установленных по умолчанию

Технология sVirt поддерживает возможность изменения параметров по умолчанию. Docker не прописывает жестко в коде программы SELinux-типы, которые назначаются процессам и ресурсам, а получает их значения из файла конфигурации `lxc_contexts`, который хранится в каталоге `/etc/SELinux/targeted/contexts`.

Следующий отрывок показывает исходно установленное содержимое этого файла и определяет параметры безопасности, используемые для различных целей:

```
process = "system_u:system_r:svirt_lxc_net_t:s0"  
content = "system_u:object_r:virt_var_lib_t:s0"  
file = "system_u:object_r:svirt_sandbox_file_t:s0"  
sandbox_kvm_process = "system_u:system_r:svirt_qemu_net_t:s0"  
sandbox_lxc_process = "system_u:system_r:svirt_lxc_net_t:s0"
```

Администраторы могут менять эти параметры, установленные по умолчанию, но им стоит помнить, что эти файлы являются частью пакета политики SELinux и могут быть перезаписаны, когда будет развернута обновленная политика. Поэтому рекомендуется придерживаться особенностей маркировки самого Docker.

## 6.4. ЗАКЛЮЧЕНИЕ

В этой главе мы рассмотрели виртуализацию и риски, связанные с ней. Мы обсудили, как некоторые из этих рисков могут быть снижены при помощи того же набора методов контроля, который предлагает SELinux, в частности таких, как применение типов (ограничивающих, что может делать гостевая система) и категорий (изолирующих гостевые системы друг от друга).

Затем мы изучили, что система libvirt поддерживает несколько различных технологий виртуализации на платформах Linux и что она включает в себя технологию sVirt, которая выполняет интеграцию с SELinux, обеспечивая изоляцию гостевых систем и контроль доступа. Мы увидели, как администраторы могут управлять логикой sVirt с помощью libvirt, например используя разные типы или наборы категорий.

Наконец, мы рассмотрели еще и систему Docker, популярную технологию контейнеризации, и то, как здесь тоже sVirt может создавать ограничения контейнера и с помощью метода контроля доступа (ограничивая возможности по эксплуатации уязвимостей и прерываний работы), и с помощью изоляции (защищая один контейнер от действий других). Еще здесь мы увидели, как различные методы контроля SELinux могут настраиваться администраторами, от определения типа контейнера до маркировки подключенных томов данных.

В следующей главе мы посмотрим на другую пару поддерживающих SELinux технологий: D-Bus и systemd.

# Глава 7

## D-Bus и systemd

Службы, контролирующие систему, такие как D-Bus и systemd, являются компонентами ядра операционной системы Linux, в настоящее время играющими роль в управлении даже большую, чем когда-либо раньше. В то время как D-Bus предлагает межсервисное взаимодействие в рамках сессии и системы, а также управление жизненным циклом процесса, systemd является фоновым процессом уровня ядра, предлагающим множество функций. Обе службы используют SELinux для дальнейшего усиления защиты своих действий и позволяют администраторам точную настройку применяемых средств контроля доступа.

В этой главе мы узнаем о следующем:

- как реализуется политика SELinux для D-Bus и systemd;
- как настраивать службу контроля доступа в D-Bus;
- управление разрешениями на доступ для служб.

Мы закончим главу разъяснением того, как D-Bus может использовать SELinux в качестве источника своей политики для ужесточения авторизации в своих службах.

### 7.1. ФОНОВЫЙ ПРОЦЕСС СИСТЕМЫ (SYSTEMD)

**Systemd** – это компонент ядра многих дистрибутивов Linux. Со времени его появления в 2010 году данный компонент был постепенно принят в качестве системы инициализации уровня ядра, ответственной за управление службами и действиями по загрузке.

В процессе своего развития systemd приобрел функциональные изменения:

- в systemd была включена система межпроцессного взаимодействия D-Bus, которая предоставила свои системную и сессионную шины для взаимодействия между приложениями;
- в systemd был включен менеджер устройств уровня ядра Udev, который предоставил для использования приложение гибкого управления файлами устройств (*device nodes*);
- возможности входа в систему (*login*) были добавлены в systemd, они позволяют выполнять предметное управление сессиями пользователей;

- был добавлен фоновый процесс `journal` для обеспечения нового подхода к системной и сервисной регистрациям, заместивший некоторую функциональность стандартных систем регистрации событий;
- фоновый процесс `timerd` обеспечивает поддержку для выполнения заданий, зависящих от времени, замещая некоторую функциональность стандартных фоновых процессов (`cron`), предназначенных для выполнения заданий в определенное время;
- сетевыми настройками можно управлять при помощи программы `systemd-networkd`.

Такой сложившийся подход поглощения нескольких системных служб в единый комплекс приложений не остался незамеченным и не является бесспорным. Некоторые дистрибутивы даже отказываются иметь `systemd` в качестве системы инициализации по умолчанию.

Gentoo Linux дает пользователям право выбора системы инициализации: `OpenRC` или `systemd`. В RHEL (начиная с 7-й версии и далее) `systemd` является единственным доступным вариантом выбора.

Проект `systemd` включил поддержку SELinux для большинства своих служб. Особенности этой поддержки SELinux детально описаны далее.

## 7.2. Способ поддержки в SYSTEMD СЛУЖБ

Главная способность системного фонового процесса, известная большинству, — его поддержка для системных служб. В отличие от традиционных SysV-совместимых систем инициализации, `systemd` не использует скрипты для управления службами.

Вместо этого она использует декларативный подход для различных служб, документируя желаемое состояние и параметры конфигурации, при этом применяя свою собственную логику для обеспечения своевременного запуска нужного набора служб.

### 7.2.1. Введение понятия модульных файлов

`Systemd` использует модульные файлы (*unit files*) для описания допустимого поведения службы. Эти модульные файлы имеют стиль INI-файла в части синтаксиса описаний, поддерживают разделы и пары ключ/значение. Служба может иметь целое множество модульных файлов, которые в значительной мере влияют на ее работу. При этом важно помнить, что различные модульные файлы все связаны, если относятся к одной и той же службе:

- модульные файлы `*.service` определяют, как системная служба должна быть запущена, каковы ее зависимости, как `systemd` следует исправлять внезапные ошибки и т. д.;
- модульные файлы `*.socket` определяют, какой(ие) сокет(ы) следует создать и какие разрешения нужно ему (им) присвоить. Это используется для служб, которые могут запускаться по запросу, а не непосредственно при загрузке;

- модульные файлы `*.timer` определяют, когда и с какой регулярностью следует запускать службу. Они используются для служб, которые необязательно работают как фоновый процесс, но требуют выполнения определенной логики в определенные интервалы.

Другие виды модульных файлов тоже существуют, хотя имеют больше общего с основными настройками системы (такими как определения концепции иерархического управления ресурсами группы процессов (`*.slice`), пути и установки автоматического подключения (`*.automount`)) и меньше общего имеют с конкретными службами.

Когда в систему устанавливается некоторое приложение, оно размещает свои модульные файлы внутри используемого по умолчанию каталога `/usr/lib/systemd/system`, откуда они забираются системой `systemd`. Когда выполняются обновления, то эти файлы могут быть размещены в каталоге `/run/systemd/system`, а уже из него файлы будут перераспределяться в каталог, указанный для размещения по умолчанию. Системные администраторы могут изменять настройки путем размещения модульных файлов в каталоге `/etc/systemd/system`. Такие модульные файлы будут влиять на изменение настроек в каталогах `/run/systemd/system` и `/usr/lib/systemd/system`.

В качестве примера рассмотрим модульный файл службы `dnsmasq`, установленный по умолчанию:

```
# cat /usr/lib/systemd/system/dnsmasq.service
[Unit]
Description=DNS caching server
After=network.target

[Service]
ExecStart=/usr/sbin/dnsmasq -k

[Install]
WantedBy=multi-user.target
```

Этот модульный файл описывает запуск службы обеспечения доменными именами `dnsmasq` в небольших сетях и информирует систему инициализации `systemd` о том, что данной службе следует запускаться после того, как будет выполнен модуль `network.target`. Условие последовательности выполнения является важным моментом при загрузке процессов, позволяющим правильно обрабатывать зависимости. В частности, в приведенном примере запуск сервиса `dnsmasq` выполняется до загрузки модуля `multi-user.target`. Такая последовательность является эквивалентной уровню выполнения по умолчанию, когда используются службы инициализации в стиле `SysV`.

## 7.2.2. Установка параметров безопасности SELinux для какой-либо службы

Среди модульных файлов есть те, которые отвечают за запуск служб (`service unit file`). Они имеют расширение `.service`. В них предусмотрено поле, отвечающее за запуск, и оно называется запись `ExecStart=`. Запись в этом поле

определяет команду, которую надо выполнить системе инициализации, чтобы служба начала работать. При необходимости преобразование типа для запускаемой службы будет производиться согласно правилам политики SELinux.

**i** Когда используется инициализирующий скрипт в стиле SysV, тогда служба запускается с типом `initrc_t`. В случае с `systemd` тип запускаемой службы соответствует типу самого фонового процесса, инициализирующего запуск, обычно это `init_t`. Не все дистрибутивы операционной системы Linux имеют политику SELinux, которая уже изменена в соответствии с методом службы `systemd` и может допускать преобразования типов от значения `init_t` к значениям, присвоенным различным служебным фоновым процессам.

Разработчики пакетов и системные администраторы могут тем не менее обновлять модульные файлы, выполняющие запуск служб, для того чтобы запускать эту службу в с правами того SELinux-типа, который они сами укажут. Для этого раздел `[Service]` модульного файла может быть расширен с помощью конфигурационной записи `SELinuxContext=`.

Например, для обеспечения того, чтобы служба `dnsmasq` запускалась с параметрами безопасности `dnsmasq_t:s0:c0.c128`, следует сделать так:

```
[Unit]
Description=DNS caching server
After=network.target

[Service]
ExecStart=/usr/sbin/dnsmasq -k
SELinuxContext=system_u:system_r:dnsmasq_t:s0:c0.c128

[Install]
WantedBy=multi-user.target
```

Конечно, также возможно использовать эту запись для того, чтобы запустить какую-нибудь службу с совершенно другими параметрами безопасности, которые могут быть использованы, когда разрабатывается пользовательская политика для фоновых процессов.

### 7.2.3. Использование переходных служб

`Systemd` также может использоваться для запуска приложений, как если бы они были службами, и для того, чтобы они находились под контролем `systemd`. Такие приложения тогда называются переходными службами (*transient services*), поскольку они испытывают недостаток в модульных файлах, которые обычно определяют, как `systemd` должна с ними работать.

Транзитные службы запускаются через приложение `systemd-run`:

```
# systemd-run bittorrent-sync
Running as unit run-2603.service
```

Поскольку переходные службы не имеют управляющих модульных файлов для управления, изменение параметров безопасности SELinux должно происходить через командную строку. Конечно, это требуется тогда, когда стандарт-



ные переходы типов, определенные в политике, не имеют варианта решения для желаемого поведения.

Приложение `systemd-run` поддерживает это (начиная с версии `systemd v230`) посредством опции `--property` или `-p`:

```
# systemd-run -p SELinuxContext=system_u:system_r:bittorrent_sync_t:s0 \
  bittorrent-sync
Running as unit run-6523.service
```

## 7.2.4. Требование включения или отключения SELinux для конкретной службы

Некоторым службам приходится работать только при включенной или только при выключенной защите SELinux. В `systemd` это может быть определено с помощью условных параметров (*conditional parameters*).

Модульный файл, запускающий службу, может содержать какое-то число условий, которые должны выполняться, перед тем как система инициализации начнет взаимодействовать с этой службой. Эти условия могут указывать на тип системы (виртуализированная или нет), на параметры загрузки (*kernel command-line parameters*), на файлы, которые существуют или не существуют, и т. д. Одно из условий, которое нас сейчас интересует, – это `ConditionSecurity` (<Условие защиты>), которое имеет значение `true`, если данная система защиты включена.

Например, рассмотрим модульный файл `rhel-autorelabel.service`, который перемаркирует все файловые системы, если это необходимо (согласно описанию в соответствующем поле `Description`):

```
# cat /usr/lib/systemd/system/rhel-autorelabel.service
[Unit]
Description=Relabel all filesystems, if necessary
DefaultDependencies=no
Requires=local-fs.target
Conflicts=shutdown.target
After=local-fs.target
Before=sysinit.target shutdown.target
ConditionSecurity=SELinux
ConditionKernelCommandLine=|autorelabel
ConditionPathExists=|/.autorelabel

[Service]
ExecStart=/lib/systemd/rhel-autorelabel
Type=oneshot
TimeoutSec=0
RemainAfterExit=yes
StandardInput=tty
```

Этот модульный файл декларирует три условия:

- условие `ConditionSecurity=SELinux` обеспечивает запуск службы только при активной системе защиты SELinux;

- условие `ConditionKernelCommandLine=|autorelabel` информирует `systemd` о том, что должны быть проверены параметры загрузки, указанные в командной строке, и если они содержат значение `autorelabel` (<автоматическая перемаркировка>), то желаемый сервис, который, собственно, и выполняет перемаркировку файловых систем, должен быть вызван (префикс `|` делает это условие активизирующим);
- условие `ConditionPathExists=|/.autorelabel` информирует `systemd` о том, что файл `/.autorelabel` должен быть найден, и если он существует, то является вторым необходимым условием для вызова целевой службы.

Аналогично работает и модульный файл `rhel-autorelabel-mark.service`, предоставляемый в дистрибутиве операционной системы Red Hat Enterprise Linux. Эта служба гарантирует, что если система была загружена без активной системы защиты SELinux и файл `/.autorelabel` не существует, то этот необходимый файл будет создан, чтобы обеспечить операцию перемаркировки после перезагрузки системы с поддержкой SELinux.

Мы можем увидеть эту проверку, взглянув на файл `rhel-autorelabel-mark.service`, который определяет необходимость перемаркировки после перезагрузки (согласно описанию в поле `Description`):

```
# cat /usr/lib/systemd/rhel-autorelabel-mark.service
[Unit]
Description=Mark the need to relabel after reboot
DefaultDependencies=no
Requires=local-fs.target
Conflicts=shutdown.target
After=local-fs.target
Before=sysinit.target shutdown.target
ConditionSecurity=!SELinux
ConditionPathIsDirectory=/etc/SELinux
ConditionPathExists=!/.autorelabel

[Service]
ExecStart=/bin/touch /.autorelabel
Type=oneshot
RemainAfterExit=yes
```

## 7.2.5. Перемаркировка файлов во время запуска службы

Из-за своего более декларативного подхода к управлению некоторые службы труднее контролируются через `systemd`, чем если бы использовались служебные скрипты в стиле Sys-V. Это, конечно же, из-за открытости и гибкости используемых скриптов, когда надо выполнить любые подготовительные действия, характерные для службы. Чего сложнее добиться с помощью `systemd`.

Одним из действий, которое требуют службы, является подготовка отдельных каталогов выполнения для соответствующих служб, например `/run/httpd` для службы веб-сервера Apache. Фоновый процесс `systemd` решает эту проблему, используя каталог `tmpfiles.d`. В нем размещаются файлы, которые не-

обходимо предоставить или немедленно обновить (во время загрузки), но не помещать в файловую систему на постоянное хранение.

Например, программный пакет, который предоставляет фоновый процесс веб-сервера Apache, устанавливает следующие определяющие правила (*definition*) по умолчанию в системе:

```
# cat /usr/lib/tmpfiles.d/httpd.conf
d /run/httpd 710 root apache
d /run/httpd/htcacheclean 700 apache apache
```

По аналогии с модульными файлами systemd файлы, содержащие эти установки, должны быть объявлены в одном из следующих мест размещения. Каждое из этих размещений отменяет установки предыдущего:

- по умолчанию место размещения программных пакетов представляет собой каталог `/usr/lib/tmpfiles.d`;
- исполняемые объявления (*runtime declarations*) могут быть размещены в каталоге `/run/tmpfiles.d`;
- объявления локального администрирования системы (*local sysadmin-provided declarations*) размещаются в `/etc/tmpfiles.d`.

Определяющие правила могут выходить далеко за рамки простого создания каталогов. С помощью `tmpfiles.d` определяющие правила могут быть установлены для создания файлов, пустых каталогов (с целью последующего использования), создания подтомов (*subvolumes*), управления специальными файлами, такими как символические ссылки или блочные устройства, установки расширенных атрибутов и т. д.

Определяющие правила также позволяют устанавливать режим файла и его принадлежность какому-либо владельцу. С помощью них можно задать необходимость восстанавливать параметры безопасности SELinux у файла (если использовать опцию `z`) или рекурсивно для всех объектов, находящихся в области размещения (`Z`). Восстановление может пригодиться, если параметры безопасности верно установлены в политике, но фактические значения неверно присвоены файлам.

Например, Red Hat определяют действия следующим образом:

```
# cat /usr/lib/tmpfiles.d/selinux-policy.conf
z /sys/devices/system/cpu/online - - -
Z /sys/class/net - - -
z /sys/kernel/uevent_helper - - -
w /sys/fs/SELinux/checkreqprot - - - 0
```

Перемаркировка внутри каталога `/sys` требуется, потому что оно будет промаркировано по умолчанию типом `sysfs_t`, в то время как некоторые его файлы должны иметь другой тип. Файл `/sys/devices/system/cpu/online`, например, должен быть помечен типом `cpu_online_t`:

```
# matchpathcon /sys/devices/system/cpu/online
/sys/devices/system/cpu/online system_u:object_r:cpu_online_t:s0
```

Определяющие правила гарантируют, что этот (псевдо)файл перемаркируется при загрузке, так что все другие процессы, зависящие от этого файла, будучи помечены типом `cpu_online_t`, смогут спокойно продолжать работу.

Другие аргументы в файле `selinux-policy.conf` помечены дефисом (-), что означает просто отсутствие каких-либо иных параметров. Они могут использоваться для установки режима, UID, GID, времени жизни и аргумента, связанного с правилом.

Примером конфигурации, использующей некоторые из этих и других параметров с состоянием `z` или `Z`, является файл `systemd.conf`:

```
# grep ^[zZ] /usr/lib/tmpfiles.d/systemd.conf
z /run/log/journal 2755 root systemd-journal - -
Z /run/log/journal/%m ~2750 root systemd-journal - -
z /var/log/journal 2755 root systemd-journal - -
z /var/log/journal/%m 2755 root systemd-journal - -
```

Для большей информации о формате определяющих правил см. страницы руководства для `tmpfiles.d`.

## 7.2.6. Использование активизации, основанной на сокетах

Системный фоновый процесс тоже поддерживает активизацию, основанную на сокетах (*socket-base activation*). После настройки `systemd` создаст сокет, на котором фоновый процесс обычно обрабатывает входящие соединения (*listens*), и запустит его при использовании сокета. Это позволяет системам быстро загружаться (поскольку многим фоновым процессам не требуется быть запущенными немедленно).

Когда клиент размещает информацию только в сокете (таким как сокет `/dev/log`), то ему не нужно даже ждать, когда фоновый процесс активизируется для работы с сокетом. Данные хранятся в буфере до тех пор, пока фоновый процесс не сможет прочитать их из буфера. Только в том случае, когда буфер заполнен, операция будет заблокирована до момента, пока фоновый процесс не очистит буфер.

Взгляните на модульный файл поддержки сокет-активации (socket unit file) `systemd-journald`:

```
# cat /usr/lib/systemd/system/systemd-journald.socket
[Unit]
Description=Journal Socket
DefaultDependencies=no
Before=sockets.target
IgnoreOnIsolate=yes

[Socket]
ListenStream=/run/systemd/journal/stdout
ListenDatagram=/run/systemd/journal/socket
ListenDatagram=/dev/log
SocketMode=0666
PassCredentials=yes
```

```
PassSecurity=yes
ReceiveBuffer=8M
```

Если используется один из упомянутых в скобках сокетов (`/run/systemd/journal/stdout`, `/run/systemd/journal/socket` или `/dev/log`), тогда модульный файл `systemd-journald.service` применяется для запуска службы.

Внутри раздела `[Socket]` может быть определена специальная запись, относящаяся к системе защиты SELinux: `SELinuxContextFromNet=true`. Когда эта запись установлена, тогда информация об уровнях доступа и категориях (MLS/MCS) выделяется из параметров безопасности клиента (того приложения, которое подключено к сокету) и добавляется к параметрам безопасности службы.

### 7.2.7. Управление доступом к операциям с модулями

До этого момента мы рассматривали настройки, связанные с поддержкой SELinux системой инициализации `systemd`. Но `systemd` также использует SELinux для контроля доступа к службам, определенным в модульных файлах. Когда какой-нибудь пользователь хочет провести конкретную операцию по отношению к некоему модулю (такую как запуск службы или проверка состояния работающей службы), `systemd` запрашивает политику SELinux о том, разрешена ли такая операция.

Системный фоновый процесс использует служебный класс, чтобы проверить, разрешена операция или нет. Например, чтобы подтвердить, позволено ли пользователю с типом `user_t` узнавать статус службы, связанной с модульным файлом `sshd.service`, он проверяет параметры безопасности этого файла (такой как `sshd_unit_file_t`), а потом оценивает, предоставлено ли разрешение для чтения статуса:

```
# ssearch -s user_t -t sshd_unit_file_t -c service -p status -A
```

В этом случае результат выполнения команды отсутствует, поэтому пользователь не имеет разрешений для получения этой информации. Другими поддерживаемыми разрешениями являются: `disable` (<отключить>), `enable` (<включить>), `kill` (<прекратить>), `load` (<загрузить>), `reload` (<перезагрузить>), `start` (<запустить>) и `stop` (<остановить>).

Когда политика SELinux запрещает выполнение операции, то это будет показано так:

```
# systemctl status sshd
Failed to issue method call: SELinux policy denies access
```



В русскоязычном варианте вывод команды может выглядеть следующим образом:

```
# systemctl status sshd
Неудача в выполнении вызова метода: политика SELinux запрещает доступ
```

В журнале регистрации событий безопасности это будет доступно через сообщение отказа в доступе типа `USER_AVC`:

```

type=USER_AVC msg=audit(1348750450.105:135): pid=1 uid=0
  auid=4294967295 ses=4294967295 subj=system_u:system_r:init_t:s0
  msg='avc: denied { status } for auid=3267 uid=0 gid=0
    path="/usr/lib/systemd/system/sshd.service"
    cmdline="/bin/systemctl status sshd.service"
    scontext=user_u:user_r:user_t:s0-s0:c0.c1023
    tcontext=system_u:object_r:sshd_unit_file_t:s0
    tclass=service exe="/usr/lib/systemd/systemd" sauid=0
    hostname=? addr=? terminal=?'
```

Здесь важно знать, что политика SELinux содержит правила доступа, относящиеся к этим действиям, но не отвечает за их соблюдение. Это делает уже непосредственно systemd, основываясь на содержании политики (в отличие от операций файловых систем, которые управляются ядром Linux и применяются также через ядро).

## 7.3. РЕГИСТРАЦИЯ СОБЫТИЙ С ПОМОЩЬЮ SYSTEMD

Как было упомянуто ранее, systemd отвечает не только за управление службами: она также занимается некоторыми другими задачами. Одной из таких задач является управление процессом регистрации событий, который традиционно реализуется через системную программу регистрации (*system logger*).

Хотя systemd до сих пор поддерживает работу с традиционным системным регистратором, в настоящее время она предлагает использовать программу *systemd-journald*, запускаемую как фоновый процесс. Одним из преимуществ этой фоновой программы ведения журналов (*journal daemon*) является то, что она не ограничивается только текстовыми, однострочными сообщениями. Фоновые процессы могут сейчас использовать бинарный вид, не уступающий многострочным сообщениям, как часть регистрационных возможностей.

Эта программа ведения журналов также регистрирует информацию об отправляющем процессе параллельно с самими регистрационными сообщениями. Эта дополнительная информация содержит сведения о принадлежности данных (владелец процесса), включая параметры безопасности для отправляющего процесса.

### 7.3.1. Получение информации, относящейся к SELinux

Традиционный метод получения информации, относящейся к SELinux (исключая события безопасности, которыми мы занимались ранее), предполагает использование команды *grep* по отношению к зарегистрированной информации. С помощью программы журналирования это достигается следующим образом:

```
# journalctl -b | grep -i SELinux
```

Опция *-b*, переданная программе, информирует фоновый процесс регистрации событий, что нас интересуют только сообщения, сформированные с момента последней загрузки системы.

### 7.3.2. Запрос событий, содержащих параметры безопасности SELinux

Удобной возможностью этой программы является использование в запросе косвенной информации, связанной с имеющимися в журнале сообщениями. Например, мы можем запросить только те сообщения, которые порождаются процессом с параметрами безопасности, включающими тип `udev_t`:

```
# journalctl _SELINUX_CONTEXT=system_u:system_r:udev_t:s0-s0:c0.c1023
-- Logs begin at Sat 2020-09-24 05:04:58 EDT,
   end at Sat 2020-09-24 10:35:04 EDT. --
Sep 24 05:04:59 SELinuxtest systemd-udevd[429]: starting version 219
Sep 24 05:04:59 SELinuxtest systemd-udevd[429]: Network interface
   NamePolicy= disabled on kernel command line, ignoring.
Sep 24 07:00:37 SELinuxtest systemd-udevd[4507]: starting version 219
Sep 24 07:00:37 SELinuxtest systemd-udevd[4507]: Network interface
   NamePolicy= disabled on kernel command line, ignoring.
Sep 24 07:00:38 SELinuxtest systemd-udevd[4507]: Network interface
   NamePolicy= disabled on kernel command line, ignoring.
```

### 7.3.3. Интеграция диагностики неисправностей с журналом

В дистрибутиве Red Hat Enterprise Linux программа диагностики неисправностей также интегрируется с `systemd-journald`. Любой сигнал тревоги о неисправности, приходящий от программы `setroubleshootd`, также доступен и через программу регистрации событий.

Это помогает администраторам, поскольку они могут при обнаружении проблем быстро видеть отказы в доступе, относящиеся к SELinux. Например, когда веб-сервер Apache не работает должным образом, рассмотрение статуса службы быстро выявит, что политка SELinux предотвратила некоторые действия:

```
# systemctl status httpd
* httpd.service - The Apache HTTP Server
   Loaded: loaded (/usr/lib/systemd/system/httpd.service; disabled;
          vendor preset: disabled)
   Active: active (running) since Sat 2020-09-24 13:47:23 EDT;
          8min ago
     Docs: man:httpd(8)
          man:apachectl(8)
  Main PID: 3183 (httpd)
   Status: "Total requests: 9; Current requests/sec: 0;
          Current traffic: 0 B/sec"
   CGroup: /system.slice/httpd.service
           └─3183 /usr/sbin/httpd -DFOREGROUND
           └─3184 /usr/sbin/httpd -DFOREGROUND
           └─3185 /usr/sbin/httpd -DFOREGROUND
           └─3186 /usr/sbin/httpd -DFOREGROUND
           └─3187 /usr/sbin/httpd -DFOREGROUND
```

```
└─3188 /usr/sbin/httpd -DFOREGROUND
└─3336 /usr/sbin/httpd -DFOREGROUND
```

```
Sep 24 13:47:23 SELinuxtest systemd[1]: Starting The Apache HTTP Server...
```

```
Sep 24 13:47:23 SELinuxtest systemd[1]: Started The Apache HTTP Server.
```

```
Sep 24 13:48:12 SELinuxtest python[10112]: SELinux is preventing
/usr/sbin/httpd from read access on the file
/srv/web/localhost/htdocs/dokuwiki/doku.php
```

Для получения большей информации о сообщении следует использовать `journalctl`:

```
# journalctl -r -o verbose -u httpd.service
-- Logs begin at Sat 2020-09-24 12:04:39 EDT, end at
   Sat 2020-09-24 13:59:34 EDT. --
Sat 2020-09-24 13:48:12.382005 EDT
[s=0e131d2da8174be59dc9dd50b5aa7aa9;i=772;b=e0177f4aefb046ab807e22725b85086
d;m=16f715bc2;t=53d447cbcf970;x=72b681410ee05ceb]
  PRIORITY=6
  _UID=0
  _GID=0
  _BOOT_ID=e0177f4aefb046ab807e22725b85086d
  _MACHINE_ID=02f1ddb1415c4feba9880b2b8c4c5925
  _HOSTNAME=SELinuxtest
  SYSLOG_FACILITY=3
  SYSLOG_IDENTIFIER=systemd
  _TRANSPORT=journal
  _PID=1
  _COMM=setroubleshootd
  _EXE=/usr/bin/python27
  _CMDLINE=/usr/bin/python27 -Es /usr/sbin/setroubleshootd -f
  _CAP_EFFECTIVE=1fffffffff
  _SYSTEMD_CGROUP=/
  _SELINUX_CONTEXT=system_u:system_r:setroubleshootd_t:s0-s0:c0.c1023
  MESSAGE_ID=39f53479d3a045ac8e11786248231fbf
  RESULT=done
  UNIT=httpd.service
  MESSAGE= SELinux is preventing /usr/sbin/httpd from read access on the
file /srv/web/localhost/htdocs/dokuwiki/doku.php

**** Plugin restorecon (92.2 confidence) suggests
*****

If you want to fix the label.
/srv/web/localhost/htdocs/dokuwiki/doku.php default label should be
httpd_sys_content_t.
Then you can run restorecon.
Do
# /sbin/restorecon -v /srv/web/localhost/htdocs/dokuwiki/doku.php

**** Plugin catchall_boolean (7.83 confidence) suggests
*****

...

```



## 7.4. ИСПОЛЬЗОВАНИЕ КОНТЕЙНЕРОВ SYSTEMD

Другой возможностью, которую поддерживает systemd, является `systemd-nspawn`. Эта служба представляет возможность контейнеризации в systemd и позволяет systemd управлять этими контейнерами. Она использует те же базовые компоненты построения, что и проект LXC и Docker. Но при этом программное обеспечение, работающее внутри контейнера, не будет иметь верного представления о состоянии SELinux (как в случае с Docker).

В отличие от Docker и libvirt, метод `systemd-nspawn` не поддерживает технологию sVirt, которую мы рассмотрели в предыдущей главе. Другими словами, он не будет динамически перенастраивать параметры безопасности SELinux используемых файлов и не будет искать свободную пару категорий для сопоставления с файлами и процессами.

### 7.4.1. Инициализация контейнеров systemd

Чтобы создать контейнер systemd, сначала создайте корневую файловую систему, в которой развернуто программное обеспечение, запускающее контейнер. Рекомендуется использовать размещение `/var/lib/machines` с подкаталогом на контейнер, поскольку эти каталоги будут размещением для контейнеров по умолчанию и использоваться для дальнейшей автоматизации внутри `systemd-nspawn`.

Например, контейнер `nginx` будет размещать свою корневую файловую систему в каталоге `/var/lib/machines/nginx`.



Корневая файловая система контейнера должна выглядеть подобно дереву каталогов операционной системы, поскольку иначе systemd будет отказываться запускать ее. Используйте инструменты, такие как `debootstrap` или `dnf`, для построения минимальной среды окружения операционной системы внутри этого размещения.

Далее для запуска контейнера применяйте команду `systemd-nspawn` таким образом:

```
# systemd-nspawn -D /var/lib/machines/nginx
```

Если контейнер содержит относительно полный дистрибутив, тогда используйте параметр `-b`.

```
# systemd-nspawn -bD /var/lib/machines/nginx
```

### 7.4.2. Использование специальных параметров безопасности SELinux

По аналогии с Docker `systemd-nspawn` тоже позволяет администраторам передавать параметры безопасности SELinux, с которыми процессы контейнера должны выполняться, а также параметры безопасности, которые следует использовать для файлов контейнера.

Для достижения этого `systemd-nspawn` поддерживает два следующих параметра:

- 1) `--selinux-context=` позволяет администратору определять параметры безопасности SELinux для выполняемых процессов контейнера;
- 2) `--SELinux-apifs-context=` позволяет администратору определять параметры безопасности SELinux для файлов и файловой системы контейнера.


Вот пример, запускающий контейнер с типом `svirt_lxc_net_t`, файлы с типом `svirt_image_t` и применяющийся и к контейнеру, и к файлам категории `c32, c42`:

```
# systemd-nspawn \
--SELinux-context=system_u:system_r:svirt_lxc_net_t:s0:c32,c42 \
--SELinux-apifs-context=system_u:object_r:svirt_image_t:s0:c32,c42
-D /var/lib/machines/nginx
```

## 7.5. УПРАВЛЕНИЕ ФАЙЛАМИ УСТРОЙСТВ

Linux имеет долгую историю управления устройствами. Изначально администраторам требовалось сделать так, чтобы узлы устройств были уже представлены в файловой системе (каталог `/dev` был частью постоянной файловой системы). Постепенно стали использоваться более динамичные методы управления устройствами.

Теперь файлы устройств управляются посредством сочетания псевдофайловой системы (`devtmpfs`) и средства управления устройствами пользовательского окружения, называемого **udev**. Такое средство управления устройствами (*device manager*) было включено в `systemd` и в результате стало компонентом `systemd-udev`.

 Существуют проекты, такие как **eudev**, предоставляющие функциональность `udev` без требования установить и включить систему `systemd`.

Средство управления устройствами ожидает (*listens*) касающихся его событий ядра в сокете уровня ядра (*kernel socket*). Эти события информируют его о выявлении или подключении устройств (может быть, об их удалении) и позволяют средству управления выполнить подходящее действие. Для `udev` эти действия описаны в правилах `udev`.

### 7.5.1. Использование правил `udev`

Основная конфигурация `udev` управляется при помощи правил `udev`. Эти правила представляют собой единую строку, которая содержит одну часть, описывающую сопоставления, а другую часть с выполняемыми действиями.

Часть с сопоставлениями состоит из проверок, которые выполняются для событий, получаемых средством управления `udev` от ядра Linux. Она основана на парах «ключ/значение», которые предоставляются событиями и включают:

- имя устройства, назначаемое ядром (KERNEL);
- подсистему устройства (SUBSYSTEM);

- драйвер ядра (DRIVER);
- специальные атрибуты (ATTR)
- и действующие переменные окружения (ENV).

Ядро Linux будет также информировать средство управления устройствами об иерархической структуре устройства. Это позволяет определять правила на основе, например, USB-контроллера, через который подключается USB-устройство. Информация, выстроенная иерархически, предоставляется при помощи пар «ключ/значение», чьи ключи описываются во множественном числе: SUBSYSTEMS вместо SUBSYSTEM, DRIVERS вместо DRIVER и т. д.

Например, для того чтобы сопоставить конкретную веб-камеру USB, соответствующие пары могли бы выглядеть так:

```
KERNEL=="video[0-9]*", SUBSYSTEM=="video4linux", \
SUBSYSTEMS="usb", ATTR{idVendor}=="05a9", ATTR{idProduct}=="4519"
```

Второй частью правила udev является действие, которое следует выполнить. Наиболее привычным действием является создание символической ссылки к созданному файлу устройства, обеспечивающей обращение приложений всегда к одному и тому же устройству через одну и ту же символическую ссылку, даже тогда, когда с точки зрения ядра устройство названо иначе. Предыдущий пример может выглядеть, например, так в этом случае:

```
KERNEL=="video[0-9]*", SUBSYSTEM=="video4linux", \
SUBSYSTEMS="usb", ATTR{idVendor}=="05a9", \
ATTR{idProduct}=="4519", SYMLINK+="webcam1"
```

Конечно же, приложение udev поддерживает множество других действий, кроме определения символических ссылок. Оно может сопоставлять владельца (OWNER) или членство в группе (GROUP) для устройства, при этом контролировать, кто может получать доступ к устройствам.

Udev может также устанавливать переменные окружения (ENV) и быть настроенным так, чтобы выполнять команду (RUN). Вероятно, имеет смысл выполнять команду только тогда, когда устройство добавлено, в таком случае должно быть добавлено такое соответствие, как ACTION=="add".

**i** ENV может рассматриваться и как ключ сопоставления, и как ключ действия. Разница лишь в выполняемой операции (знак равенства = одиночный или двойной). ENV{envvar}=="value" – это пример сопоставления (действительно ли переменная окружения envvar соответствует значению value?). ENV{envvar}=value – это действие присваивания значения value.

Правила udev по умолчанию располагаются в каталоге /usr/lib/udev/rules.d. Это место, где дистрибутивы, приложения и драйверы по умолчанию будут хранить свои правила. Дополняющие правила или правила переопределения могут быть размещены в каталоге /etc/udev/rules.d.

Важно помнить, что udev будет продолжать процесс обработки правила даже тогда, когда соответствующее правило будет найдено и выполнено. Это можно

изменить для каждого правила с помощью действия `OPTIONS`, сходного с этим `OPTIONS+= "last_rule"`, информирующего таким образом `udev`, что он может прекратить обработку дальнейших правил для этого конкретного события.

## 7.5.2. Назначение метки SELinux на узле устройства

Одним из действий, которое поддерживает `udev`, является присвоение конкретных параметров безопасности SELinux узлу устройства. Это делается при помощи действия `SECLABEL{SELinux}`:

```
KERNEL=="fd0", ..., \  
SECLABEL{SELinux}="system_u:object_r:my_device_t"
```

Заметим, что это действие применимо только для узла устройства, которое создано. Если правило устанавливает также символическую ссылку, тогда, собственно, символическая ссылка остается без изменений (и будет наследовать по умолчанию параметр `device_t`).

## 7.6. ВЗАИМОДЕЙСТВИЕ С ШИНОЙ СООБЩЕНИЙ D-BUS

Фоновый процесс системы D-Bus предоставляет канал межпроцессного взаимодействия (*inter-process communication* – IPC) между приложениями. В отличие от традиционных IPC-методов, D-Bus – это высокоуровневый канал взаимодействия, который предлагает больше, чем простая сигнализация или совместное использование памяти.

Приложения, желающие общаться через D-Bus, соединяются с одной из множества D-Bus-совместимых библиотек, таких как `libdbus`, `sd-bus` (являющейся частью `systemd`), `GDBus` или `QtDBus`. Система D-Bus является частью комплекта приложений системы `systemd`.

### 7.6.1. Представление о взаимодействии между процессами D-Bus

В основном Linux поддерживает два типа взаимодействия между процессами через шину сообщений D-Bus:

- системная шина сообщений, представляющая собой основной экземпляр исполняемой программы D-Bus, который используется для системных коммуникаций. Многие службы и фоновые процессы будут связываться с системным процессом D-Bus, для того чтобы другие службы и процессы взаимодействовали с ними через D-Bus;
- сессионная шина сообщений, представляющая собой отдельную копию исполняемой программы D-Bus для каждого вошедшего в систему пользователя. Этот тип взаимодействия обычно используется графическими приложениями для взаимодействия друг с другом в рамках одного сеанса пользователя.

Исполняемая программа D-Bus называется `dbus-daemon`. Для системной шины сообщений D-Bus ее процесс будет запускаться с параметром `--system`, а для сессионной шины – с параметром `--session`.

Приложения регистрируют себя в D-Bus через определенное пространство имен. Условно это пространство имен основывается на доменном имени соответствующего проекта (*project domain name*). Например, система `systemd` использует пространство имен `org.freedesktop.systemd1`, тогда как система шины сообщений D-Bus – это `org.freedesktop.DBus`.

Перечень приложений, которые взаимодействуют с системной шиной, может быть получен при помощи клиентской программы D-Bus. Данная клиентская программа реализует командный интерфейс и называется `qdbus` (предоставляется с `qt`-пакетом):

```
# qdbus --system
:1.1
  org.freedesktop.login1
:1.186
:1.2
  org.fedoraproject.FirewallD1
:1.3
  com.redhat.ifcfgrh1
  org.freedesktop.NetworkManager
:1.4
:1.5
  fi.epitest.hostap.WPASupplicant
  fi.w1.wpa_supplicant1
:1.6
  org.freedesktop.PolicyKit1
:1.65
  org.freedesktop.systemd1
:1.66
  com.redhat.tuned
org.freedesktop.DBus
```

Затем каждое приложение предоставляет объекты в шину сообщений, которые тем самым могут быть доступны другим приложениям (при условии что у них есть на это права). Эти объекты определяются некой записью, по стилю похожей на путь, а доменное имя проекта тоже используется, но в качестве префикса к этой записи.

Например, для вывода списка объектов, в настоящее время связанных с приложением, зарегистрированным как `org.freedesktop.systemd1`, следует использовать такую команду:

```
# qdbus --system org.freedesktop.systemd1
/
/org/freedesktop/systemd1/unit/firewalld_2eservice
/org/freedesktop/systemd1/unit/machines_2etarget
/org/freedesktop/systemd1/unit/sys_2ddevices_2dnpn0_2d00_3a05_2dttty_2dtttyS0_2edevice
```

```
/org/freedesktop/systemd1/unit/getty_2etarget
/org/freedesktop/systemd1/unit/lvm2_2dlvmpolld_2eservice
...
```

Приложения могут задействовать методы, относящиеся к этим объектам, или с помощью этих методов отправлять сообщения [другим] приложениям, которые связаны с этими объектами.

Например, чтобы прочитать все свойства, предоставляемые службой управления именем вычислительного ресурса (*hostname service*), мы вызываем метод `org.freedesktop.DBus.Properties.GetAll` объекта, связанного с пространством имен `/org/freedesktop/hostname1` и предоставляемого пространством имен `org.freedesktop.hostname1`. Мы передаем строку `org.freedesktop.hostname1` в ЭТОТ метод:

```
# dbus-send --system --print-reply --type=method_call \
--dest=org.freedesktop.hostname1 \
/org/freedesktop/hostname1 \
org.freedesktop.DBus.Properties.GetAll \
string:"org.freedesktop.hostname1"
method return sender=:1.192 -> dest=:1.193 reply_serial=2
array [
  dict entry(
    string "Hostname"
    variant string "SELinuxtest"
  )
...
  dict entry(
    string "KernelName"
    variant string "Linux"
  )
  dict entry(
    string "KernelRelease"
    variant string "3.10.0-327.13.1.el7.x86_64"
  )
  dict entry(
    string "KernelVersion"
    variant string "#1 SMP Thu Mar 31 11:10:31 CDT 2020"
  )
  dict entry(
    string "OperatingSystemPrettyName"
    variant string "Red Hat Enterprise Linux Server 7.2 (Maipo)"
  )
  dict entry(
    string "OperatingSystemCPENAME"
    variant string \
      "cpe:/o:redhat:enterprise_linux:7.2:GA:server"
  )
]
```

## 7.6.2. Контроль получения доступа к службам с помощью SELinux

Приложение D-Bus, так же как `systemd`, будет запрашивать политику SELinux, для того чтобы определить, разрешена ли конкретная операция. И снова само приложение D-Bus контролирует выполнение политики, а не подсистема ядра Linux.

Первое, что можно сделать для повышения уровня контроля при работе с D-Bus, – это обеспечение того, что только строго определенные приложения смогут получать конкретные объекты через шину сообщений. Без этого контроля вредоносный код мог бы, к примеру, зарегистрировать себя как `org.freedesktop.login1` и действовать с шиной сообщений как системный процесс. Или же приложения могли бы ошибочно отправлять информацию, подлежащую защите, другим приложениям.

Приложения хранят правила политик в файлах, размещенных в каталоге `/etc/dbus-1/system.d/`. Служба управления учетными записями пользователей (*login service*), например, содержит в своем конфигурационном файле такой фрагмент политики безопасности:

```
# cat /etc/dbus-1/system.d/org.freedesktop.login1.conf
<?xml version="1.0"?>
<!DOCTYPE busconfig PUBLIC
  "-//freedesktop//DTD D-BUS Bus Configuration 1.0//EN"
  "http://www.freedesktop.org/standards/dbus/1.0/busconfig.dtd">
<busconfig>
  <policy user="root">
    <allow own="org.freedesktop.login1" />
    <allow send_destination="org.freedesktop.login1" />
    <allow receive_sender="org.freedesktop.login1" />
  </policy>
  <policy context="default">
    ...
  </policy>
</busconfig>
```

Поскольку процесс этой службы запускается с установленным типом `systemd_logind_t`, то мы могли бы усилить эту конфигурацию следующим образом:

```
<?xml version="1.0"?>
<!DOCTYPE busconfig PUBLIC
  "-//freedesktop//DTD D-BUS Bus Configuration 1.0//EN"
  "http://www.freedesktop.org/standards/dbus/1.0/busconfig.dtd">
<busconfig>
  <SELinux>
    <associate
      own="org.freedesktop.login1"
      context="system_u:system_r:systemd_logind_t:s0" />
    </SELinux>
</busconfig>
```

И тогда D-Bus будет проверять, имеет ли приложение (которое, как мы предполагаем, работает с параметром безопасности `systemd_logind_t`) разрешение `acquire_svc` (класса `dbus`) для типа `systemd_logind_t`. По умолчанию политика SELinux не имеет такого разрешения, и поэтому регистрация не состоится:

```
systemd-logind[538]: Failed to register name: Permission denied
systemd-logind[538]: Failed to fully start up daemon: Permission denied
```

В журналах аудита мы замечаем следующее запрещение:

```
time->Sat Sep 24 11:53:23 2020
type=USER_AVC msg=audit(1474732403.120:404): pid=521 uid=81
  auid=4294967295 ses=4294967295
  subj=system_u:system_r:system_dbusd_t:s0-s0:c0.c1023
  msg='avc: denied { acquire_svc } for
    service=org.freedesktop.login1 spid=2313
    scontext=system_u:system_r:systemd_logind_t:s0
    tcontext= system_u:system_r:system_dbusd_t:s0
    tclass=dbus exe="/usr/bin/dbus-daemon" sauid=81
    hostname=? addr=? terminal=?'
```

Когда мы добавим следующее правило в политику SELinux (о ней более подробно будет рассказано в следующей главе), регистрация службы `systemd-logind` пройдет успешно:

```
allow systemd_logind_t self:dbus acquire_svc;
```

Ограничивая число назначаемых процессам типов, которые могут использовать данную службу, мы гарантируем использование только проверенных приложений. Непроверенные приложения обычно не запускаются с типом известного приложения (не могут инициировать преобразование к такому домену и конечные пользователи), даже если они получили права пользователя `root` (что является еще одной проверкой, которую проводит D-Bus по отношению к службе управления учетными записями, как показано в первом фрагменте `busconfig`).

Администраторы могут улучшить конфигурацию D-Bus без внесения изменений в существующие файлы конфигурации. Например, ранее упомянутый управляемый SELinux фрагмент `busconfig` мог бы быть успешно сохранен как другой файл.

### 7.6.3. Управление потоками сообщений

Второй проверкой, которую выполняет D-Bus, является определение того, каким приложениям разрешено взаимодействовать друг с другом. Это не настраивается через конфигурационные параметры службы, но является исключительно политикой управления SELinux.

Когда исходное приложение вызывает метод целевого приложения, D-Bus проверяет разрешение `send_msg` между двумя типами, назначенными исходному и целевому приложениям.



Например, при взаимодействии через D-Bus между типом, сопоставленным с пользовательскими процессами (`sysadm_t`), и типом службы (`systemd_logind_t`) будут проверяться следующие разрешения:

```
allow sysadm_t systemd_logind_t : dbus send_msg;  
allow systemd_logind_t sysadm_t : dbus send_msg;
```

Если эти разрешения не будут работать, тогда D-Bus не позволит данное взаимодействие.

Если в какой-то момент параметры безопасности приложения не могут быть получены (что невозможно при использовании сокетов домена UNIX, но может произойти, если D-Bus со временем поддержит другие методы коммуникации), тогда будут использоваться параметры безопасности фонового процесса шины сообщений.

## 7.7. ЗАКЛЮЧЕНИЕ

В этой главе мы начали с введения в `systemd` и рассмотрели возможности управления службами, которые предлагает `systemd`. Мы изучили, как запускать службы с заданными параметрами безопасности SELinux, а также как дополнительные файлы могут быть промаркированы должным образом уже после загрузки.

Наряду с управлением службами с помощью модульных файлов системы `systemd` эта глава также раскрыла возможности переходных служб и то, как можно быстро связывать их с необходимыми параметрами безопасности SELinux.

Другие возможности и службы системы инициализации `systemd` также были охвачены. Мы увидели, что параметры безопасности SELinux были зарегистрированы как часть журнала `systemd`, и выяснили, как можно запросить те события, которые эти параметры безопасности включают. Была рассмотрена интеграция фонового процесса регистрации событий с фоновым процессом SELinux, ответственным за диагностику неисправностей.

Мы изучили, как `systemd` поддерживает контейнеры и что администраторы могут сделать, для того чтобы настроить параметры безопасности SELinux, связанные с контейнером. В заключение мы кратко рассмотрели средства управления устройствами пользовательского окружения `udev` и то, как его правила могут быть использованы для помощи администраторам в управлении устройствами. Одной из его функциональных возможностей является установка параметров безопасности SELinux для файлов устройств.

Мы закончили главу кратким обзором D-Bus, как SELinux может быть использован для управления связей приложений со службами и как D-Bus использует разрешение `send_msg` для проверки связи по своим каналам.

В следующей главе мы будем изучать, как может быть настроена политика SELinux и даже то, как политики SELinux, создаваемые пользователями, могут быть разработаны и загружены.

# Глава 8

## Работа с политиками SELinux

До сих пор мы работали с уже существующими политиками SELinux, настраивая нашу систему таким образом, чтобы она могла работать с необходимыми параметрами безопасности SELinux, которые присваивались в том числе файлам, каталогам и даже сетевым портам. В этой главе мы будем:

- управлять условными правилами политик (*conditional policy rules*) SELinux посредством их включения и выключения;
- изучать, как создаются новые модули политик (*custom policy*) SELinux, доступные для формирования пользователям системы защиты;
- разрабатывать новые типы допустимых наборов функций как для пользователей, так и для приложений;
- заменять существующие политики новыми, кем-либо созданными.

Мы закончим эту главу изучением примеров нескольких политик, создаваемых пользователями системы, которые расширяют возможности нашего SELinux, и донстроим политику в соответствии с требованиями безопасности, которые необходимы администратору.

### 8.1. ЛОГИЧЕСКИЕ ПАРАМЕТРЫ SELINUX

Одним из способов управления политиками SELinux является переключение логических параметров (*SELinux boolean*). Уже в главе 2 «Режимы работы и регистрация событий», в которой мы использовали логический параметр `secure_mode_policyload`, эти возможности появлялись в содержании данной книги. Принимая одно из двух состояний, «включено» или «выключено», они подключают или исключают части политики SELinux. Администраторы политик используют логические параметры SELinux для управления составляющими политики, которые не всегда нужны (или даже нежелательны), но все же в общем случае применяются.

### 8.1.1. Вывод списка логических параметров

Некоторый обзор логических параметров SELinux может быть получен при вызове команды `semanage` с опцией `boolean`. Обычно в системе мы легко можем найти более сотни таких параметров, поэтому будет разумно сразу выделить описание тех логических параметров, которые мы конкретно ищем:

```
# semanage boolean -l | grep policyload
secure_mode_policyload (off, off)
```

Этот логический тип предназначен для определения, разрешены ли в системе загрузка политики, установка принудительного режима (*enforce mode*) и изменение значений у логических параметров. Установите его в `true`, и вам потребуется перезагрузить компьютер, чтобы вернуть значение обратно.

**P** Если вы не хотите, чтобы злоумышленники могли переключиться обратно в рекомендательный режим (даже при наличии у них необходимых привилегий), включите этот логический параметр настройки SELinux. Тогда принудительный режим отключить будет нельзя без перезагрузки системы.

Данный вариант вывода списка параметров дает нам не только краткое описание логического параметра, но и его текущее значение (на самом деле дает нам два значения: одно является ожидающим решения по изменению политики, а другое – текущее значение, но почти всегда они совпадают).

Другой способ получения текущего значения логического параметра заключается в вызове команды `getsebool` следующим образом:

```
# getsebool secure_mode_policyload
secure_mode_policyload --> off
```

Если имя логического параметра, о котором мы хотим узнать, не удалось вспомнить целиком, то мы можем запросить перечень всех логических параметров (и их значений) и выделить среди них тот, который нам необходим:

```
# getsebool -a | grep policy
secure_mode_policyload --> off
```

Для того чтобы вывести логический параметр с его описанием, можно воспользоваться командой `sepolicy booleans`:

```
# sepolicy booleans -b secure_mode_policyload
secure_mode_policyload=(_("Boolean to determine whether the system
permits loading policy, setting enforcing mode, and changing
boolean values. Set this to true and you have to reboot to
set it back."))
```

**P** Для русскоязычного варианта данное сообщение могло бы быть представлено следующим образом:

```
# sepolicy booleans -b secure_mode_policyload
secure_mode_policyload=(_("Логический параметр определяет, разрешено ли в системе
загружать политику, настраивать принудительный режим защиты и изменять логические
параметры. Установите его в значение true, и вам потребуется перезагрузить
систему, для того чтобы вернуть его обратно."))
```

При этом можно заметить, что данная команда не отражает текущего значения логического параметра.

В конечном счете логические параметры представлены также в каталоге `/sys/fs/SELinux` файловой системы:


```
# cat /sys/fs/SELinux/booleans/secure_mode_policyload
0
```

Здесь `booleans` (логические параметры) могут быть прочитаны, как если бы они были обычными файлами, и они возвращают значение 0 (ноль), если выключены, и значение 1 (единица), если включены.

### 8.1.2. Изменение значений логических параметров

Мы можем изменять значение логического параметра, используя команду `setsebool`. В качестве примера изменения мы можем использовать логический параметр `httpd_can_sendmail` (который включает и выключает правила политики, позволяющие веб-серверам отправлять сообщения по электронной почте):

```
# setsebool httpd_can_sendmail on
```

 В системе Gentoo Linux существует другая команда, `togglesebool`, которая просто переключает текущее значение логического параметра. Эта команда предоставляется библиотекой `libselinux`, но отсутствует в Red Hat Enterprise Linux.

Логические параметры SELinux имеют некое значение по умолчанию, определенное администратором политики. Изменение значения с помощью утилиты `setsebool` обновляет текущее состояние подсистемы контроля доступа, но оно не сохраняется после перезагрузки.

Для того чтобы сохранить изменения постоянными, надо добавить опцию `-P` (от англ. *persist* – сохранять) к команде `setsebool` следующим образом:

```
# setsebool -P httpd_can_sendmail on
```

В фоновом режиме обновленное значение логического параметра включается в хранилище политик: файл текущей политики пересобирается и загружается. В результате регенерируется файл политики (допустим, файл `policy.29`, находящийся внутри каталога `/etc/selinux/targeted/policy/`).

Другой путь изменения и последующего сохранения значений для логических параметров – это использование команды `semanage boolean` следующим образом:

```
# semanage boolean -m --on httpd_can_sendmail
```

В данном случае значение логического параметра изменено (`-m`) на «включено» (`--on`).

Сохранение изменений займет некоторое время, пока политика SELinux будет пересобирается (те изменения, которые не носят постоянного характера, вносятся почти мгновенно). Чем больше политика безопасности SELinux, тем дольше времени это займет.

### 8.1.3. Проверка влияния логического параметра

Для того чтобы выяснить, на какие правила политики влияет логический параметр, обычно достаточно посмотреть его описание, но иногда возникает необходимость узнать, какие правила SELinux изменяются, когда задействуется параметр безопасности.

С помощью приложения `sesearch` мы можем запросить политику SELinux вывести на экран правила, затрагиваемые логическим параметром. Чтобы получить детальную информацию, используйте опцию `-b` (для логического параметра – *boolean*), опцию `-A` (для вывода разрешающих – *allow* – правил) и опцию `-C` (для вывода условных правил – *conditional rules*):

```
# sesearch -b httpd_can_sendmail -AC
Found 46 semantic av rules:
DT allow httpd_sys_script_t bin_t : dir { getattr search open } ;
  [ httpd_can_sendmail ]
DT allow httpd_sys_script_t bin_t : lnk_file { read getattr } ;
  [ httpd_can_sendmail ]
DT allow system_mail_t httpd_suexec_t : process sigchld ;
  [ httpd_can_sendmail ]
DT allow system_mail_t httpd_suexec_t : fd use ;
  [ httpd_can_sendmail ]
DT allow system_mail_t httpd_suexec_t : fifo_file { ioctl ... } ;
  [ httpd_can_sendmail ]
DT allow httpd_t bin_t : dir { getattr search open } ;
  [ httpd_can_sendmail ]
DT allow httpd_t bin_t : lnk_file { read getattr } ;
  [ httpd_can_sendmail ]
DT allow httpd_t smtp_client_packet_t : packet { send recv } ;
  [ httpd_can_sendmail ]
...
```

В примере мы можем видеть, что правилам предшествуют два символа: DT. Они информируют нас о включенном или выключенном состоянии логического параметра в политике (первый символ) и о включенном или выключенном состоянии самого правила SELinux (второй символ).



При использовании 4-й версии инструмента `setools` опция `-C` больше недоступна при команде `sesearch`. Когда логический параметр выбирается с использованием опции `-b`, будут показаны только те правила, которые зависят от этого логического параметра. Вывод [на экран] тоже немного отличается, отображая лишь правила, имеющие активное состояние (**true** – *правда* или **false** – *ложь*), т. е. не будет двухсимвольного обозначения состояния, как показано в предыдущем примере.

Состояние отражает, включено (E – *enabled*) или выключено (D – *disabled*) правило политики SELinux в настоящий момент, а также когда правило должно сработать, если логический параметр является истинным (T – *true*) или ложным (F – *false*). Логический параметр считается истинным, если он имеет значение on (в *действии*), и ложным, если off (*бездействия*). Таким образом, обозначение

DT говорит, что правило на текущий момент выключено, но станет действующим, если логический параметр будет переключен в состояние on.

При запросе политики SELinux имеет смысл всегда добавлять условную опцию, чтобы можно было легко увидеть, поддерживает ли политика определенный доступ, основанный на одном или, может быть, большем количестве логических параметров.

Рассмотрим SELinux-тип `httpd_t`, назначенный веб-серверу. Для этого типа существует множество правил политики, управляемых при помощи логических параметров. Для того чтобы узнать, какие есть разрешающие правила для взаимодействия между процессами веб-сервера (`httpd_t`) и объектами пользователя (`user_home_t`), надо выполнить следующую команду:

```
# sestatus -s httpd_t -t user_home_t -AC
Found 7 semantic av rules:
  allow daemon user_home_t : file { getattr append } ;
  allow httpd_t file_type : filesystem getattr ;
  allow httpd_t file_type : dir { getattr search open } ;
DT allow httpd_t user_home_type : file { ioctl read getattr lock open }
; [ httpd_read_user_content ]
DT allow httpd_t user_home_type : dir { getattr search open } ;
  [ httpd_enable_homedirs ]
DT allow httpd_t user_home_type : dir
  { ioctl read getattr lock search open } ;
  [ httpd_read_user_content ]
DT allow httpd_t user_home_type : lnk_file { read getattr } ;
  [ httpd_enable_homedirs ]
```

## 8.2. УСИЛЕНИЕ ПОЛИТИК SELINUX

Не все ситуации могут быть безукоризненно сразу определены разработчиками политик. Иногда возникает необходимость внесения каких-либо изменений в политику SELinux. Поскольку изменения предполагают добавление правил, то мы можем создавать дополнительные модули SELinux для усиления политики. Если изменения предлагают большую глубину, то может потребоваться удалить существующий модуль SELinux и заменить его обновленным.

### 8.2.1. Список модулей политики

Модули политик SELinux, как было упомянуто в начале данной книги, содержат правила SELinux, которые могут быть загружены и выгружены. Файлы модулей, имеющие расширение `.pp` или `.cil`, могут быть загружены или выгружены, в зависимости от того, как это нужно администратору. При загрузке модуль политики помещается в хранилище политик SELinux и будет загружен после перезагрузки системы.

Для вывода списка модулей политик SELinux рекомендуется использовать команду `semodule`. В зависимости от версии пользовательских инструментов SELinux (в данном случае версии пакета `polyscoreutils`) в перечне модулей бу-

дуг также отображаться их версии (в более ранних инструментах) или только имена модуля (в более новых). В Red Hat Enterprise Linux 7.2 все еще применяется вывод версий вместе с названиями модулей:

```
# semodule -l
abrt      1.4.1
accountsd 1.1.0
acct      1.6.0
afs       1.9.0
...
```

Более поздние версии пользовательских программ SELinux больше не используют версии модулей, но вместо этого включают понятие приоритетов. Модули могут быть загружены с более высоким приоритетом, превалирующим над предыдущими модулями, или с более низким приоритетом (в этом случае модуль загружается, но не становится действующим). В Gentoo Linux, например, мы получаем следующий вывод, который показывает и приоритет, и формат **модуля политики** (*policy module*):

```
# semodule --list-modules=full
400 also      pp
400 android   pp
400 application pp
400 archi     pp
...
```

Утилиты SELinux будут копировать действующие модули политик в специальное место для их размещения. Поэтому отображение содержимого этого каталога также дает представление о текущих загруженных модулях:

```
# ls /etc/selinux/targeted/modules/active/modules/
abrt.pp  cockpit.pp  gitosis.pp  lvm.pp  oracleasm.pp
...
```

В более ранних версиях пользовательских программ SELinux таким местом для размещения является каталог `/etc/selinux`, тогда как более поздние используют каталог `/var/lib/selinux`.

## 8.2.2. Загрузка и удаление модулей политики

В последующих частях данной главы мы изучим, как создавать новые модули политик. Когда они уже созданы, то надо, чтобы они могли быть загружены и, при необходимости, удалены. Это делается также при помощи утилиты `semodule`, независимо от того, в каком формате политика представлена (`.pp` или `.cil`):

```
# semodule -i screen.pp
```

При использовании более новых утилит SELinux администраторы могут установить приоритет. Это позволит администраторам загрузить обновленный модуль с большим приоритетом, сохраняя при этом более старый в не-

активном состоянии. Например, для того чтобы установить модулю `archi.cil` приоритет 500, надо выполнить следующую команду:

```
# semodule -i archi.cil -X 500
```

Удаление модулей осуществляется при помощи параметра `--remove` или `-r`. В этом случае мы учитываем не столько формат пакета, сколько имеющийся загруженный модуль, поэтому указывать расширение не требуется:

```
# semodule -r screen
```

Помимо этого, в более новых утилитах SELinux есть возможность удалить модуль из конкретного приоритета. Вот пример удаления модуля из приоритета 300:

```
# semodule -r archi -X 300
```

Наконец, можно оставить модуль, но выключить его. При этом модуль сохраняется в хранилище политик, но выключает все правила политики SELinux внутри себя. Для того чтобы это сделать, надо использовать параметр `--disable` или `-d`:

```
# semodule -d archi
```

Чтобы обратно включить, надо использовать параметр `--enable` или `-e`:

```
# semodule -e archi
```

### 8.2.3. Создание политик с использованием программы `audit2allow`

Как мы уже знаем, когда SELinux предотвращает конкретные действия, то она регистрирует соответствующее запрещение в журнале событий безопасности. Это запрещение может быть использовано в качестве источника данных для создания политики SELinux, которая позволяет такое действие.

Рассмотрим следующие запреты, которые появились, когда пользователь вызвал `setkey` (команду управления в системе защиты IPsec), после того как активировал роль SELinux `sysadm_r` с помощью команды `newrole`:

```
type=AVC msg=audit(1373121736.897:6882): avc: denied { use } for
pid=15069 comm="setkey" path="/dev/pts/0" dev="devpts" ino=3
scontext=root:sysadm_r:setkey_t:s0-s0:c0.c1023
tcontext=root:staff_r:newrole_t:s0-s0:c0.c1023
tclass=fd permissive=0
type=AVC msg=audit(1373121736.907:6883): avc: denied { search }
for pid=15069 comm="setkey" name="/" dev="dm-4" ino=2
scontext=root:sysadm_r:setkey_t:s0-s0:c0.c1023
tcontext=system_u:object_r:var_t:s0
tclass=dir permissive=0
```

Если нет другого решения, предлагаемого `sealert`, кроме запуска `audit2allow`, и быстрый анализ свидетельствует о том, что отсутствуют такие логиче-



ские параметры, которые можно было бы переключить, и это позволило бы получить необходимые разрешения, тогда мы имеем в запасе всего несколько вариантов. Мы можем отказаться рассматривать это решение, сообщая пользователю о необходимости запуска команды `setkey` другим способом (без переключения роли SELinux). Но если мы уверены, что это действие ничего не нарушает и нет какого-либо несоответствия параметров безопасности любого вида, тогда можно разрешить действия, в которых было отказано на текущий момент.

Приложение `audit2allow` преобразует отказ (или набор отказов) в разрешающие правила (`allow`) SELinux. Эти правила могут быть сохранены в файл, для того чтобы можно было собрать их в модуль политики SELinux, который мы затем сможем загрузить в память.

Чтобы сгенерировать разрешающие правила из запрещающих событий, надо направить зарегистрированные записи журнала об этом в приложение `audit2allow`:

```
# grep setkey /var/log/audit/audit.log | audit2allow
#===== setkey_t =====
allow setkey_t newrole_t:fd use;
allow setkey_t var_t:dir search;
```

В результате сформированы два разрешающих правила, основанных на запрещениях, рассмотренных выше. Чтобы сразу и модуль собрать из этих правил, мы можем дополнить данную команду следующим образом:

```
# grep setkey /var/log/audit/audit.log | audit2allow -M localpolicy
***** IMPORTANT *****
To make this policy package active, execute:
semodule -i localpolicy.pp
```

Файл с именем `localpolicy.pp` будет доступен в текущем каталоге, и мы можем этот файл загрузить в память, используя эту же команду.

Если произошедшие запреты носят все же внешний характер (имеется в виду, что система функционирует, как ожидалось), можно использовать `audit2allow` для генерации правил `dontaudit`, блокирующих возникновение событий безопасности, вместо того чтобы создавать разрешающие правила. В таком случае запреты больше отмечаться не будут, даже когда действие не разрешено:

```
# grep setkey /var/log/audit/audit.log | audit2allow -D -M localpolicy
***** IMPORTANT *****
To make this policy package active, execute:
semodule -i localpolicy.pp
```

Вполне возможно, что даже после включения необходимых правил (при условии что эти правила являются разрешающими, т. е. `allow`) для того действия, которое должно было выполняться, оно все еще не будет доступно. Оно просто не станет работать на какой-то другой стадии выполнения, до которой раньше дело не доходило. Пока предыдущие запреты AVC-типа еще доступны

в журналах регистрации событий безопасности, достаточно регенерировать политику и продолжить работу над устранением препятствий. Тогда утилита `audit2allow` будет принимать в расчет все запрещения AVC-типа, с которыми она сталкивается, даже те, которые существовали до загрузки новой политики.

Другой способ предполагает переключение системы (или специфического SELinux-типа, назначенного субъекту доступа) в рекомендательный режим (*permissive mode*) для формирования и заполнения журналов событий безопасности всевозможными запретами AVC-типа, относящимися к выполняемому действию. Несмотря на то что при этом формируется больше запретов AVC, с которыми надо работать, это также может привести и к *ошибочным* решениям, которые может обеспечить программа `audit2allow`. Поэтому всегда проверяйте запреты перед созданием новых правил политики!

Когда в журнале событий уже не остается запретов AVC-типа, следует сгенерировать новый модуль политики, чтобы не получилось так, что ранее успешные попытки доступа снова станут неудачными из-за того, что какие-то ранее существующие разрешающие правила не попали в политику, загруженную на одном из этапов решения проблемы с доступом. Когда новая политика загружается, старая перестает действовать.

#### 8.2.4. Использование говорящих за себя наименований для модулей политики

В предыдущем примере была описана команда `audit2allow` для создания модуля политики с именем `localpolicy` (<локальная политика>). Тем не менее это плохая практика.

Когда создается (в виде исполняемого модуля) политика (файл `localpolicy.pp`), то администраторам довольно непросто выяснить, какие правила составили этот модуль. И хотя существует возможность распаковать файл `.pp` (при помощи программы `semodule_unpackage`) и потом выполнить обратное проектирование для полученного файла `.mod` в файл `.te`, все равно это требует, во-первых, программного обеспечения, которое пока не доступно в большинстве дистрибутивов (приложение `dismod`, являющееся частью программных средств `checkpolicy`, которые редко включены в состав поставляемых средств). И во-вторых, чтобы только проанализировать правила, являющиеся частью модуля, понадобится сложный и затратный по времени метод.

В системах с недавно разработанными программами SELinux содержание модуля может быть в какой-то мере получено из сгенерированного кода на обобщенно-промежуточном языке **CIL** (*Common Intermediate Language*). Например, действующий модуль для экранных приложений `screen` будет иметь код, доступный в файле с именем `cil`, размещенный в каталоге `/var/lib/SELinux/mcs/active/modules/400/screen`. Необходимость погружаться в правила, для того чтобы узнать, о чем в действительности говорит `localpolicy`, – это не только плохая практика, но и действие, требующее достаточных привилегий для чтения необходимых файлов.

Вместо этого много лучше давать наименования созданным модулям в соответствии с их целевым назначением. Тот модуль политики SELinux, который позволил преодолеть запреты системы, возникшие, когда команда управления `setkey` была выполнена после переключения роли, при помощи команды `newrole`, лучше всего назвать `custom_setkey_newrole`.

Также рекомендуется присваивать приставки (или окончания) выполненным по специальным требованиям политикам (*custom policies*) в виде строки, которая бы говорила, что этот модуль был добавлен администратором (или организацией), а не через распространяемые политики. При наличии всех таких политик с обозначением `custom_` много проще увидеть, какие из установленных в настоящий момент политик являются выполненными по специальным требованиям:

```
# semodule -l | grep ^custom_
custom_setkey_newrole
custom_sysadmin_powertop
custom_debug_xorg
custom_alsa_qemu
```

## 8.2.5. Использование макрокоманд посреднической политики с программой `audit2allow`

Проект посреднической политики (*reference policy*) обеспечивает дистрибутивы и разработчиков политик набором функций, которые упрощают разработку политик SELinux. В качестве примера давайте посмотрим, что макрос может сделать в предыдущей ситуации:

```
# grep setkey /var/log/audit/audit.log | audit2allow -R
require {
    type setkey_t;
    type newrole_t;
    class fd use;
}

#===== setkey_t =====
allow setkey_t newrole_t:fd use;
files_search_var(setkey_t)
```

Поскольку команда `audit2allow -R` использует некий автоматический метод для поиска потенциальных функций, необходимо внимательно проанализировать результаты. Иногда она выбирает такой метод, который создает гораздо больше привилегий для SELinux-типа, назначаемого субъектам доступа, чем вообще требуется.

Одно из правил в примере указано так: `files_search_var(setkey_t)`. По сути, это макрокоманда посреднической политики (*reference policy macro*), которая представляет конкретное правило SELinux (или набор правил) в более удобном для понимания формате чтения. В данном случае она разрешает субъектам типа `setkey_t` осуществлять поиск в каталогах, имеющих параметр безопасности `var_t`.

Все основные дистрибутивы основывают свои SELinux-политики на макрокомандах и содержании, предоставленных посреднической политикой. Список методов, которые можно вызвать в процессе построения политик SELinux, доступен на сайте <http://oss.tresys.com/docs/refpolicy/api/>, но может быть также установлен и на локальную файловую систему в каталог `/usr/share/doc/SELinux-base-*` (для Gentoo, с включенной опцией `USE="doc"` во время сборки пакета `sec-policy/SELinux-base`) или в каталог `/usr/share/doc/SELinux-policy` (для Red Hat Enterprise Linux, после установки пакета `selinux-policy-doc`).

Эти именованные методы объединяют набор правил, связанных с функциями, которые администраторы политик SELinux хотят запустить. Например, метод `storage_read_tape()` позволяет нам расширить политику для заданного SELinux-типа правом чтения накопителей на магнитной ленте.

## 8.2.6. Использование скрипта `selocal`

В Gentoo доступен скрипт с именем `selocal`, позволяющий администраторам добавлять к политике простые, однострочные правила. Затем они становятся частью модуля политики, управляемого при помощи `selocal` (по умолчанию он называется `selocal`).

Например, чтобы разрешить всем SELinux-типам (доменам), предназначенным для процессов, отправлять и получать немаркированные пакеты, можно выполнить команду `selocal` таким образом:

```
# selocal -a "allow domain unlabeled_t:packet { send recv };" -Lb
```

Для более сложного примера давайте вернемся к запретам, относящимся к типу `setkey_t`, с которыми мы столкнулись ранее. В том примере субъект с типом `setkey_t` пытался использовать файловый дескриптор с типом `newrole_t`. Если мы более внимательно посмотрим на тип `newrole_t`, то при помощи команды `seinfo` увидим относящиеся к нему атрибуты. Среди них можно отметить атрибут с именем `privfd`:

```
$ seinfo -tnewrole_t -x
newrole_t
  privfd
  mlsprocsetl
  can_change_object_identity
  kernel_system_state_reader
  ...
```

Одним из доступных методов посреднической политики является `domain_use_interactive_fds()`, который разрешает SELinux-типам процессов (доменам) использовать файловые дескрипторы объектов, имеющих тип, с установленным атрибутом `privfd`.

Можно разрешить это для объектов с типом `setkey_t`, используя команду `selocal`:

```
# selocal -a "domain_use_interactive_fds(setkey_t)" \
  -c "Get output of setkey after newrole" -L -b
```

**P** В этой команде применяется параметр `-c`, встраивающий комментарий (*comment*) к добавляемому правилу. В данном случае комментарий предполагает следующий дословный перевод: «Получить вывод `setkey` после `newrole`».

Сам метод `domain_use_interactive_fds()`, согласно официальному описанию на сайте [http://oss.tresys.com/docs/refpolicy/api/kernel\\_domain.html](http://oss.tresys.com/docs/refpolicy/api/kernel_domain.html), позволяет конкретному типу процесса наследовать и использовать файловый дескриптор от типов процессов, относящихся к интерактивным программам (взаимодействующих с пользователями). В примере, который приведен в п. 8.2.3, к интерактивной относится команда смены роли пользователя (см. п. 3.3.3). А конкретный тип процесса (называемый еще доменом) указан для программы управления `setkey` – `setkey_t`. Поэтому по смыслу комментарий предполагает, что в результате данного правила пользователь сможет *получить результат от выполнения программы setkey, если он ее запустит после того, как получит необходимые привилегии, путем смены своей роли с помощью команды newrole*.

**i** Понимание того, какой и когда метод вызвать, является основным вопросом принципов разработки SELinux. В этой главе мы касаемся основных аспектов разработки политик системы защиты. Тем не менее более подробное изучение разработки политик SELinux выходит за границы данной книги. Для этого я рекомендую книгу *SELinux Cookbook*, о которой подробнее сказано на странице <https://www.packtpub.com/networking-and-servers/selinux-cookbook>, являющуюся другим проектом издательства Packt. Эта книга ориентирована на интенсивную разработку политик SELinux.

Приложение `selocal` по умолчанию обеспечивает работу только одного модуля политики SELinux, в отличие от `audit2allow`, где нам необходимо регулярно создавать новые модули политик системы защиты. Приложение `selocal` выполняет сборку этого одного модуля по требованию (`-b`) и загружает его в память (`-L`).

Для того чтобы получить список текущих правил SELinux в политике, управляемой при помощи `selocal`, можно использовать команду `selocal -l`:

```
# selocal -l
23: files_mountpoint(portage_tmp_t) # Mount tmpfs on /var/tmp/portage
24: domain_use_interactive_fds(setkey_t)
    # Get output of setkey after newrole
```

Для удаления какой-либо определенной строки надо использовать номер, указанный в выводе списка правил. Например, чтобы удалить ранее добавленную строку, используйте такую команду:

```
# selocal -d 24
Removing line 24 from module selocal (/root/.selocal/selocal.te)
Removed following line: domain_use_interactive_fds(setkey_t) \
    # Get output of setkey after newrole
```

## 8.3. СОЗДАНИЕ МОДУЛЕЙ ПОЛИТИК ПО СПЕЦИАЛЬНЫМ ТРЕБОВАНИЯМ

SELinux позволяет создавать собственные модули политик. Для этого нам необходимо иметь как минимум файл с расширением `.te` (которое обозначает

type enforcement – *требования типов*). Дополнительно используются следующие файлы:

- с параметрами безопасности файлов (file context) – .fc;
- файл интерфейса (interface file) – .if.

Когда же мы используем формат новой политики, тогда нужен файл с расширением .cil (см. ниже). Все эти файлы должны иметь одно и то же базовое имя, которое будет использоваться в дальнейшем как имя модуля.

Существует несколько форматов, в которых могут быть написаны модули политики SELinux:

- первый формат называется **исходным форматом SELinux** (SELinux native). Он не использует макросы посреднической политики (*reference policy*), но при этом до сих пор является основным способом разработки основной политики. Проектирование посреднической же опирается на этот формат для построения собственного набора правил;
- второй формат – **посреднический стиль политики** (*reference policy style*). В нем создаются макрокоманды, которые упрощают разработку политик SELinux и в то же время поддерживают большинство синтаксических правил построения, которые применяются в первом формате;
- третий формат – это формат **обобщенно-промежуточного языка** (CIL – *Common Intermediate Language*). Это довольно новый язык для разработки политик SELinux, но, конечно же, все равно соотносимый с хорошо известными конструкциями языка SELinux. Последующие пользовательские программы SELinux будут переводить первые два формата в формат CIL в прозрачном для пользователя режиме.

Использование модулей, созданных по специальным требованиям (вместо модулей, которые сформированы с помощью программы audit2allow), более предпочтительно, поскольку дает администратору возможность в большей степени контролировать добавленные правила политик. Они также позволяют администраторам отслеживать обновления политик, включая комментарии внутри правил политики, объясняющие необходимость добавления правил.

Мы кратко рассмотрим три метода создания модулей, основанных на следующих нескольких подразделах.

### 8.3.1. Создание модулей SELinux с помощью исходного языка описания политик

Модуль, созданный на исходном языке описания политики SELinux, начинается со строки, описывающей имя самого модуля, за которой следует набор требований (типы или атрибуты, классы и разрешения), а затем уже собственно правила.

Примером является следующий файл политики:

```
# cat localpolicy.te
module localpolicy 1.0;
require {
```

```

type setkey_t;
type newrole_t;
class fd { use };
}
allow setkey_t newrole_t:fd use;

```

Файл `localpolicy.te` может быть преобразован в промежуточный модульный файл, который будет называться `localpolicy.mod`. Это достигается путем использования команды `checkmodule` следующим образом:

```
$ checkmodule -M -m -o localpolicy.mod localpolicy.te
```

После этого мы можем получить модуль политики SELinux, который можно уже загрузить, и для этого применяется команда `semodule_package`:

```
$ semodule_package -o localpolicy.pp -m localpolicy.mod
```

Загружается файл `localpolicy.pp` в память с помощью приложения `semodule`.

### 8.3.2. Создание модулей SELinux с помощью посреднического стиля описания политик

В случае когда мы имеем дело с модулем посреднической политики, используется похожая структура, что в формате описания на исходном языке SELinux, но только с привлечением функций, предоставленных различными модулями политик, содержащих определения и описания этих функций. И опять же создаваемый модуль будет начинаться с объявления имени модуля. За ним будут следовать описания требуемых типов (или других объектов SELinux). А в конце указывается конкретный набор правил политики, которые содержит этот модуль.

Что характерно, первая строка вызывает макрокоманду. Она представлена методом, который называется `policy_module()`:

```

# cat localpolicy.te
policy_module(localpolicy, 1.0)
gen_require('
    type setkey_t;
')
domain_use_interactive_fds(setkey_t)

```

Когда этот файл готов `localpolicy.te`, он может быть собран при помощи команды `Makefile` из проекта посреднической политики. Во время сборки функции, имеющиеся в файле политики, будут преобразованы к исходной форме описания SELinux. Затем будут построены пакеты политик (*policy packages*) – файлы с расширением `.pp`.

В операционной системе Gentoo программа `Makefile` размещается в каталоге `/usr/share/SELinux/targeted/include`, тогда как в Red Hat Enterprise Linux она размещается в каталоге `/usr/share/SELinux/devel`:

```

$ make -f /usr/share/SELinux/devel/Makefile localpolicy.pp
Compiling targeted localpolicy module
/usr/bin/checkmodule: loading policy configuration from

```

```

tmp/localpolicy.tmp
/usr/bin/checkmodule: policy configuration loaded
/usr/bin/checkmodule: writing binary representation (version 17) to
tmp/localpolicy.mod
Creating targeted localpolicy.pp policy package
rm tmp/localpolicy.mod.fc tmp/localpolicy.mod

```

Файл `localpolicy.pp` может быть загружен в память с помощью приложения `semodule`.

### 8.3.3. Создание модулей SELinux с помощью обобщенно-промежуточного языка

Формат CIL использует некий отличный стиль разработки политик, который может являться более простым для разбора программным обеспечением, но иногда более требовательным к вниманию пользователей в процессе разработки. Тем не менее существует некоторое число преимуществ у файлов формата CIL, которые делают данный метод весьма популярным. Правда, его поддерживают только относительно новые пользовательские программы SELinux. Например, Red Hat Enterprise Linux версии 7.2 не поддерживает CIL-формат.

В следующем файле (`localpolicy.cil`) содержание аналогично такому же файлу, но созданному на исходном языке SELinux (см. п. 8.3.1):

```

# cat localpolicy.cil
(allow setkey_t privfd (fd (use)))

```

Одним из преимуществ применения файлов в формате CIL является то, что не надо использовать команды преобразования в промежуточные форматы. Поэтому созданный файл можно загружать в память сразу же:

```

# semodule -i localpolicy.cil

```

### 8.3.4. Добавление описаний для параметров безопасности файла

Модули политик могут еще содержать описания параметров безопасности (*context definitions*), которые сообщают пользовательским программам, какие метки присвоены для ресурсов файловой системы. К примеру, можно присвоить тип `httpd_sys_content_t` содержимому каталога `/opt/dokuwiki/htdocs`.

Хотя можно использовать команду `semanage fcontext` для назначения параметров безопасности этому месту размещения в файловой системе, все же использование описаний внутри модулей дает некоторое преимущество в том, что эти описания становятся частью основной политики (и таким образом перепроверяются с помощью специальных правил, как было описано в главе 4).

В исходном формате описания политики необходимое присвоение типа выглядит так:

```

# cat localpolicy.fc
/opt/dokuwiki/htdocs(/.*)?
\

```



```

system_u:object_r:httpd_sys_content_t:s0
# checkmodule -M -m -o localpolicy.mod localpolicy.te
# semodule_package -o localpolicy.pp -m localpolicy.mod -f localpolicy.fc

```

В стиле посреднической политики это же присваивание выглядит так:

```

# cat localpolicy.fc
/opt/dokuwiki/htdocs(/.*)?
    gen_context(system_u:object_r:httpd_sys_content_t,s0)
# make -f /usr/share/SELinux/devel/Makefile localpolicy.pp

```

А на обобщенно-промежуточном языке – так:

```

# cat localpolicy.cil
(filecon "/opt/dokuwiki/htdocs(/.*)?" any
 (system_u object_r httpd_sys_content_t ((s0) (s0)))
)

```

## 8.4. СОЗДАНИЕ РОЛЕЙ И ПОЛЬЗОВАТЕЛЬСКИХ ТИПОВ ДОПУСТИМОГО НАБОРА ФУНКЦИЙ

Одной из наилучших особенностей SELinux является его способность ограничивать конечных пользователей и обеспечивать их только теми правами, которые им требуются для выполнения своей работы. Чтобы достичь этого, нам потребуется создать некий тип допустимого набора функций, который будет сопоставлен с какими-либо SELinux-пользователями для ограничения их действий (как сразу, так и после переключения их стандартной роли на более привилегированную).

Такие пользовательские типы и роли должны быть созданы при помощи расширенных функциональных возможностей политики SELinux (*policy enhancements*). Эти возможности тем не менее требуют глубокого понимания проверок доступных разрешений, макрокоманд посреднической политики и, сверх того, такого понимания, которое можно получить только опытным путем.

### 8.4.1. Создание файла `pgsql_admin.te`

Сначала давайте рассмотрим файл политики, включающий в себя правила, которые связаны с пользователями. Каждая строка снабжена комментарием, продублированным для удобства чтения на русском языке.

Файл `pgsql_admin.te` выглядит так:

```

# cat pgsqldb_admin.te
policy_module(pgsqldb_admin, 1.0)

# Define the pgsqldb_admin_r role
# Объявление роли pgsqldb_admin_r
role pgsqldb_admin_r;

# Create a pgsqldb_admin_t type that has minimal rights a regular
# user domain would need in order to work on a Linux system

```

```

# Создание пользовательского типа допустимого набора функций pgsqldb_admin_t,
# который имеет минимальные права, основанные на стандартном пользовательском типе,
# необходимом для работы в операционной системе Linux
userdom_base_user_template(pgsqldb_admin)

# Allow the pgsqldb_admin_t type to execute regular binaries
# such as id
# Разрешение типу pgsqldb_admin_t запускать обычные исполняемые файлы,
# такие как id (команда вывода подлинных и действующих UID и GID)
corecmd_exec_bin(pgsqldb_admin_t)

# Allow the user domain to read its own SELinux context
# Разрешение пользовательскому типу допустимого набора функций
# получать сведения о своих собственных параметрах безопасности SELinux
SELinux_getattr_fs(pgsqldb_admin_t)

# Allow the user to administer postgresql, but do not fail
# if no postgresql SELinux module is loaded yet
# Позволяет пользователю администрировать систему управления БД postgresql,
# но не выдавать ошибку, если модуль SELinux, созданный для управления
# безопасностью postgresql, еще не загружен в системе
optional_policy(`
    postgresql_admin(pgsqldb_admin_t, pgsqldb_admin_r)
`)

# Allow transition from staff_r to pgsqldb_admin_r
# Позволяет преобразование из роли staff_r в роль pgsqldb_admin_r
gen_require(`
    role staff_r;
`)

allow staff_r pgsqldb_admin_r;

```

Теперь файл политики может быть собран (используя метод посреднической политики) и загружен в память.

## 8.4.2. Создание прав пользователя

Вместе с загруженной политикой становятся доступными роль `pgsqldb_admin_r` и тип `pgsqldb_admin_t`. После чего мы создаем SELinux-пользователя, называем его `pgsqldb_admin_u` и разрешаем доступ к возможностям ролей:

- `staff_r` (для непривилегированных действий);
- `system_r` (для управления службы PostgreSQL);
- `pgsqldb_admin_r` (для администрирования файлов PostgreSQL и выполнения соответствующих команд).

Как было показано в главе 3 «Управление учетными записями пользователей», мы можем сделать это с помощью команды `semanage user`:

```
# semanage user -a -R staff_r -R system_r -R pgsqldb_admin_r pgsqldb_admin_u
```

В той же главе мы узнали, как SELinux-пользователя сопоставить с пользователем операционной системы Linux. Предположим, что пользователя Linux

зовут `marija`, тогда, чтобы назначить ей в соответствие SELinux-пользователя `pgsql_admin_u`, надо выполнить следующую команду:

```
# semanage login -a -s pgsqql_admin_u marija
```

После этого нам нужно перегрузить параметры безопасности пользователя, т. к. параметры безопасности всех файлов теперь необходимо изменить. Для этого будем использовать команду `restorecon`:

```
# restorecon -RvF /home/marija
```

И в конце необходимо отредактировать файл `sudoers` так, чтобы каждая команда, которую бы пользователь запускал через `sudo`, содержала в параметрах безопасности роль `pgsql_admin_r` и запускалась бы с типом допустимого набора функций `pgsql_admin_t`.

Следующего фрагмента файла `/etc/sudoers` для этого должно быть достаточно:

```
marija ALL=(ALL) ROLE=pgsql_admin_r TYPE=pgsql_admin_t ALL
```

Этих изменений достаточно, чтобы пользователь смог пройти авторизацию и управлять системой PostgreSQL. По умолчанию `marija` будет оставаться с правами, которые предоставляет служебная роль `staff_r`, и с типом `staff_t`, которые позволят работать большинству пользовательских команд. Когда же ей понадобится выполнять более привилегированные действия, то `marija` будет использовать команду замены пользователя `sudo`. А так как пользователь не входит в группу `wheel`, то использование команды `su` для получения командной оболочки администратора `root` не представляется возможным.

Тип допустимого набора функций `pgsql_admin_t` имеет достаточно прав для управления системой PostgreSQL. К примеру, `marija` может перезапустить службу и даже отредактировать конфигурационный файл:

```
$ sudo rc-service postgresql-9.2 start
* Starting PostgreSQL... [ ok ]
$ sudo vim /etc/postgresql-9.2/pg_hba.conf
```

Поскольку дополнительные права когда-нибудь, скорее всего, понадобятся, то, для того чтобы это сделать, администратору надо лишь должным образом обновить файл `pgsql_admin.te`, пересобрать политику и загрузить ее. Такой подход позволяет адаптировать допустимые наборы функций у типа `pgsql_admin_t`, для того чтобы более точно выполнять требования, которые пользователи предъявляют для обеспечения безопасного состояния системы.

### 8.4.3. Предоставление доступа для взаимодействия с командным интерфейсом

Со временем пользователи могут захотеть запросить доступ к строчному командному интерфейсу (*shell*). Они могут получить доступ либо косвенно, при помощи команды `sudo`, либо, возможно, сразу после входа в систему (так чтобы

пользователь мог получить роль `pgsql_admin_r` без промедлений). В SELinux это сделать несложно, даже если пользователю были предоставлены привилегии командного интерфейса пользователя `root`: система защиты все равно предотвратит действия пользователя, которые ему не разрешены.

Наиболее широко используемый подход для обеспечения взаимодействия со строчным командным интерфейсом – это вместе с какой-либо ролью использовать метод `userdom_login_user_template()` вместо вызова метода `userdom_base_user_template()`. Если роль является более привилегированной администраторской ролью, то может быть даже лучше использовать метод `userdom_admin_user_template()`. При помощи переключения шаблонов (*template*), которые вызываются в файле политики (в нашем случае это файл `pgsql_admin.te`), подключаются дополнительные правила SELinux, предназначенные для большего взаимодействия.

Если требуется, чтобы пользователь, авторизовавшись, получал сразу новый тип [допустимого набора функций], то необходимо внести некоторые изменения.

Во-первых, надо создать для пользователя файл с параметрами безопасности, используемыми по умолчанию (в каталоге `/etc/selinux/mcs/contexts`). Допустим, мы можем работать из копии (для пользователя `staff_u`) и с заменой роли `staff_r` на `pgsql_admin_r`. Тогда этот файл будет говорить SELinux, что установленный по умолчанию тип должен быть тогда, когда учетная запись является управляемой с помощью одного из упомянутых параметров безопасности.

Затем файл `/etc/selinux/mcs/default_type` необходимо обновить, для того чтобы сообщить SELinux, что тип `pgsql_admin_t` является типом по умолчанию для роли `pgsql_admin_r` (как запасной вариант).

Сделав эти изменения, можно следующим образом обновить сопоставления ролей для пользователя, чтобы они были только `pgsql_admin_r` и `system_r` (и надо не забыть перезагрузить после этого параметры безопасности для пользовательских файлов):

```
# semanage user -m -R "pgsql_admin_r system_r" pgsqldb_admin_u
```

#### 8.4.4. Формирование структуры файлов пользовательской политики

Утилиты пользовательского окружения SELinux предлагают инструмент, который формирует структуру файлов для политик, создаваемых по специальным требованиям. Он называется `sepolgen` (или `sepolgen generate`) и предоставляется в составе пакета `polyscoreutils-devel` (в Red Hat Enterprise Linux) или в пакете `sys-apps/polyscoreutils` (для Gentoo).

Для того чтобы сформировать структуру файла политики для роли `pgsql_admin`, можно использовать параметр `--term_user`, чтобы сгенерировать код для взаимодействующих через терминальный доступ пользователей:

```
# sepolgen --term_user -n pgsqldb_admin
```

В первой строке вывода результатов команды указывается, что создаются следующие файлы, и приводится их перечисление:

```
pgsql_admin.te # Type Enforcement file # Файл с типами функциональных ограничений
pgsql_admin.if # Interface file # Файл с интерфейсами
pgsql_admin.fc # File Contexts file # Файл с параметрами безопасности файлов
pgsql_admin_SELinux.spec # Spec file # Спецификационный файл для создания RPM-пакетов
pgsql_admin.sh # Setup Script # Установочный командный файл
```

Первые три файла подобны тем, которые создавались ранее. Два дополнительных файла позволяют администраторам быстро ввести созданные политики в свои системы:

- файл `pgsql_admin_selinux.spec` используется для сборки файлов RPM (оригинальное название Red Hat Package Manager – *Менеджер пакетов Red Hat*), который позволяет администраторам устанавливать политики при помощи стандартной системы управления жизненным циклом программ;
- командный файл, который собирает политику, загружает ее в систему, генерирует стандартную страницу пользовательского руководства для модуля, обновляет необходимые файлы в системе, для того чтобы приспособить нового пользователя, и в конце собирает RPM-пакет (применяя файл `.spec`, указанный выше).

Использование `sepolgen` (или `sepolgen generate`) позволяет администраторам легко начать с общего набора файлов политики.

Другие поддерживаемые пользовательские шаблоны в `sepolgen` являются следующими:

- `--admin_user` (<административный\_пользователь>) для администрирования, привилегированных пользовательских типов допустимого набора функций;
- `--confined_admin` (<ограниченный\_в\_правах\_администратор>) для администрирования, но в то же время ограниченных пользовательских типов;
- `--desktop_user` (<пользователь\_рабочего\_стола>) для стандартного типа конечного пользователя;
- `--x_user` (<пользователь\_графической\_подсистемы\_X\_Server>) для низкопривилегированных типов конечных пользователей, которые могут использовать X server.

## 8.5. СОЗДАНИЕ НОВЫХ ТИПОВ ДЛЯ ПРИЛОЖЕНИЙ

По умолчанию дистрибутивы Linux распространяются со многими предварительно упакованными типами допустимого набора функций для приложений. Однако может возникнуть ситуация, когда нам понадобится собрать свою собственную политику для приложения или включить политику, созданную по специальным требованиям, но предлагаемую сторонними средствами.

В отличие от пользователей и ролей, типы для приложений обычно имеют файл контекстно связанной с ним информации.

### 8.5.1. Создание файлов `mojomoho.*`

Следующая политка SELinux создана для приложения `mojomoho`, некоторого свободного ПО с открытым исходным кодом – `wiki`, основанного на платформе `Catalyst`.

**P** `MojoMojo` – это приложение, предназначенное для представления информационного материала с привлечением пользователей к наполнению и многократной перепроверке (реализация концепции `Web 2.0 wiki`), применяющее подход асинхронной загрузки необходимых пользователю данных (`AJAX – Asynchronous JavaScript u XML`) для предварительного просмотра в процессе взаимодействия. С иерархической структурой, тегами, отличиями, подключаемым синтаксисом, правами разграничения доступа на основе дискреционного метода, вложениями, `RSS`-лентами, фотогалереей, решением конфликтов с редактированием с помощью трехстороннего слияния, встроенным полнотекстовым поиском и другими возможностями [<https://github.com/mojomoho/mojomoho/wiki>].

Код довольно легкий, поскольку является относительно простым веб-приложением (с точки зрения инфраструктуры). В нем мы вызываем метод `apache_content_template()`, который предоставляет большинство необходимых правил:

```
# cat mojomoho.te
policy_module(mojomoho, 1.1.0)

# Create all types based on the apache content template
# Создание всех типов, основанных на шаблоне содержимого веб-сервера apache
apache_content_template(mojomoho)

# Only call creation of alias on Red Hat Enterprise Linux systems
# Вызов создания альтернативного наименования для системы Red Hat Enterprise Linux
ifdef(`distro_Red Hat Enterprise Linux', `
    apache_content_alias_template(mojomoho,mojomoho)
')

# Needed by the mojomoho application
# Требуется приложением mojomoho
allow httpd_mojomoho_script_t httpd_t:unix_stream_socket
    rw_stream_socket_perms;

# Network connectivity
# Подключение к сети
corenet_sendrecv_smtp_client_packets(httpd_mojomoho_script_t)
corenet_tcp_connect_smtp_port(httpd_mojomoho_script_t)
corenet_sendrecv_smtp_client_packets(httpd_mojomoho_script_t)

# Additional File system access
# Дополнительный доступ к файловой системе
files_search_var_lib(httpd_mojomoho_script_t)

# Networking related activities (name resolving & mail sending)
# Действия, связанные с сетью (определение имени и отправка почты)
sysnet_dns_name_resolve(httpd_mojomoho_script_t)
mta_send_mail(httpd_mojomoho_script_t)
```

Можно видеть, что принципиально файл не сильно отличается от модуля с типом допустимого набора функций для пользователя, который был создан чуть раньше. Очевидно, что много отличающихся вызовов, но подход такой же.

Посмотрим на описание параметров безопасности для файлов (`mojomomo.fc`):

```
# catmojomomo.fc
/usr/bin/mojomomo_fastcgi\ .pl --
    gen_context(system_u:object_r:httpd_mojomomo_script_exec_t,s0)
/usr/share/mojomomo/root(/.*)?
    gen_context(system_u:object_r:httpd_mojomomo_content_t,s0)
/var/lib/mojomomo(/.*)?
    gen_context(system_u:object_r:httpd_mojomomo_rw_content_t,s0)
```

Первый столбец (задающий путь к объектам) аналогичен по формату тому, что мы использовали раньше с командой `semanage fcontext`. Параметр `--` в первой строке указывает политике SELinux, что регулярное выражение предназначено только для обычного файла – так же, как и в случае с `semanage fcontext`.

Второй столбец представляет собой макрокоманду посреднической политики. Данные макрокоманды генерируют параметры безопасности, основанные на целевых политиках. Если целевая политика включает многоуровневую защиту (MLS), тогда уровень доступа тоже применяется и используется (в данном случае он имеет значение `s0`), в другом случае – пропускается.

## 8.5.2. Создание интерфейсов политик

Когда мы собираем какую-нибудь политику для приложения, которое предназначено для конечного пользователя, то в итоге нам понадобится узнать у SELinux, какие существующие (и новые) роли и типы являются разрешенными для выполнения этим новым приложением. И хотя мы можем сделать это через стандартные правила SELinux, куда удачнее создать для этого некий интерфейс. Обычные правила, которые относятся к нескольким типам, нарушают изоляцию, предоставляемую модулями политик SELinux. Интерфейсы позволяют нам группировать правила согласованно.

В качестве примера давайте посмотрим на интерфейсы модуля `zosremote` (в файле `zosremote.if`), который может быть найден в подкаталоге `contrib/`, находящемся в каталоге `/usr/share/selinux/devel/include/` (для Red Hat Enterprise Linux) или в каталоге `/usr/share/selinux/targeted/include/` (для Gentoo Linux). Если пропустить комментарии, то содержание этого файла будет следующим:

```
# cat zosremote.if
interface(`zosremote_domtrans',`
    gen_require(`
        type zos_remote_t, zos_remote_exec_t;
    `)
    corecmd_search_bin($1)
    domtrans_pattern($1, zos_remote_exec_t, zos_remote_t)
`)
```

```
interface(`zosremote_run',`
    gen_require(`
        attribute_role zos_remote_roles;
    ')
    zosremote_domtrans($1)
    roleattribute $2 zos_remote_roles;
')
```

Данный файл предоставляет следующие интерфейсы:

- `zosremote_domtrans` разрешает заданному типу процесса преобразовать при запуске к типу `zosremote_t` тот процесс, чей файл исходно имеет тип `zos_remote_exec_t`;
- `zosremote_run` позволяет заданному типу процесса преобразовать в тип `zosremote_t`, но также обеспечивает, чтобы данный тип `zosremote_t` был разрешен для заданной роли.

Разница заключается в применении: интерфейс `zosremote_domtrans` будет использован для преобразований между приложениями, в то время как `zosremote_run` будет использован для пользователей (и ролей пользователей). Например, для того чтобы разрешить пользователю системы PostgreSQL запускать приложения, имеющие тип `zosremote` (удаленные службы, относящиеся к проприетарной 64-битной серверной операционной системе z/OS, разработанной компанией IBM для мейнфреймов), необходимо включить следующий код правил политики в файл `pgsql_admin.te`:

```
zosremote_run(pgsql_admin_t, pgsql_admin_r)
```

Когда собираются специально разработанные файлы с интерфейсами, то этим файлам (таким как `mojomomo.if`) необходимо быть доступными либо в текущем каталоге (где собираются другие модули политик), либо в подкаталоге `contrib/` (или `apps/`), размещающемся в каталоге `/usr/share/selinux/devel/include`. Если это так не будет, то политики, которые должны использовать интерфейсы, допустим, модуля `mojomomo`, не смогут найти описания этих интерфейсов.

### 8.5.3. Создание структуры файлов политики для приложений

Подобно тому как мы делали с файлами пользовательских политик, можно использовать утилиту `sepolgen` для создания политик, ориентированных на приложения. В случае с `mojomomo` можно использовать параметр создания политик для веб-приложений `--cgi`:

```
# sepolicy generate --cgi -n momomomo /usr/bin/momomomo_fastcgi.pl
Loaded plugins: fastestmirror
Created the following files:
mojomomo.te # Type Enforcement file # Файл с типами функциональных ограничений
mojomomo.if # Interface file # Файл с интерфейсами
mojomomo.fc # File Contexts file # Файл с параметрами безопасности файлов
mojomomo_SELinux.spec # Spec file # Спецификационный файл для создания пакета RPM
mojomomo.sh # Setup Script # Установочный командный файл
```



Для приложений команда `sepolity generate` требует, чтобы команда `main` была передана в качестве аргумента. Она будет использоваться для создания элементарного файла с параметрами безопасности (`.fc`) файлов.

Ниже перечислены параметры, которые также поддерживаются командой `sepolity generate` и связаны с какими-либо приложениями:

- 1) `--application` для генерации стандартной политики командно-строчного приложения;
- 2) `--dbus` для создания политики приложений, применяющих систему межпроцессного взаимодействия D-Bus;
- 3) `--inetd` для создания политики, применяемой к диспетчеру управления службами интернета (Internet Services Daemon);
- 4) `--init` для создания политики, управляющей действиями стандартной подсистемы инициализации в операционной системе, которая запускает все остальные процессы.

## 8.6. ЗАМЕНА СУЩЕСТВУЮЩИХ ПОЛИТИК

Когда применяются политики SELinux, выполненные по специальным требованиям пользователей, то все, что могут сделать фактические пользователи, – это добавлять разрешающие (`allow`) правила. SELinux не имеет правил запрещения (`deny rule`), которые могут быть использованы для блокировки или удаления текущих правил, разрешающих доступ из действующей политики.

Если текущая политика является слишком уж много позволяющей с точки зрения администратора, то он будет вынужден обновить политику, а не просто усилить ее. А это означает, что администратор имеет доступ к использованию правил текущей политики SELinux.

Процесс замены существующей политики зависит от имеющихся утилит из числа пользовательских программ SELinux (более новые поддерживают приоритетную загрузку) и от источника текущей политики. Давайте дальше рассмотрим два метода: один для Red Hat Enterprise Linux, а другой для Gentoo Linux.

### 8.6.1. Замена политик Red Hat Enterprise Linux

Для того чтобы заменить какую-либо действующую политику Red Hat, необходимо скачать исходный RPM-пакет с политикой SELinux и применить к нему утилиту `rpmbuild` для извлечения из этого пакета файлов. После извлечения мы обновим файлы политик, пересоберем их и затем установим собранную политику в систему.

Для начала разберемся, какая есть текущая версия политики SELinux:

```
# rpm -qi selinux-policy
Name       : selinux-policy
Version    : 3.13.1
Release    : 60.el7_2.9
```

```

Architecture: noarch
Install Date: Sat 24 Sep 2020 07:00:07 AM EDT
Group        : System Environment/Base
Size         : 180
License      : GPLv2+
Signature    : DSA/SHA1, Thu 15 Sep 2020 11:05:48 AM EDT, Key ID b0b4183f192a7d7d
Source RPM   : SELinux-policy-3.13.1-60.el7_2.9.src.rpm
Build Date   : Wed 14 Sep 2020 01:19:26 PM EDT
Build Host   : sl7.fnal.gov
Relocations  : (not relocatable)
Packager     : Scientific Linux
Vendor       : Scientific Linux
URL          : http://oss.tresys.com/repos/refpolicy/
Summary      : SELinux policy configuration
Description  :
SELinux Reference Policy - modular.
Based off of reference policy: Checked out revision 2.20091117

```

**P** В результате запроса информации о пакете `rpm -qi selinux-policy` предоставляются следующие сведения:

```

# rpm -qi selinux-policy
Наименование      : selinux-policy
Версия             : 3.13.1
Выпуск продукта   : 60.el7_2.9
Архитектура       : noarch
Дата установки    : Суббота 24 Сентябрь 2020 07:00:07 AM EDT
Группа            : System Environment/Base
Размер            : 180
Лицензия          : GPLv2+
Подпись           : DSA/SHA1, Четверг 15 Сентября 2020 11:05:48, Ключевой
                   идентификатор b0b4183f192a7d7d
Пакет для сборки RPM : SELinux-policy-3.13.1-60.el7_2.9.src.rpm
Дата сборки       : Среда 14 Сентября 2020 01:19:26 PM EDT
Место сборки      : sl7.fnal.gov
Перемещение       : (неперемещаемый)
Составитель программы: Scientific Linux
Поставщик         : Scientific Linux
Интернет-адрес    : http://oss.tresys.com/repos/refpolicy/
Итоговые данные   : конфигурация политики SELinux
Описание:
SELinux посредническая политика - modular.
Основано на посреднической политике: Проверочная ревизия 2.20091117

```

Затем попытаемся получить пакет для сборки RPM (*source RPM*), указанный в списке выше. Если система не может получить его в рамках основной поддержки, то можно воспользоваться репозиториями третьих лиц, как предлагает CentOS. Когда пакет и там трудно найти, то можно поискать на сайте <https://rpmfind.net>.

В общем, надо скачать этот пакет для сборки и установить в системе:

```
# rpm -i SELinux-policy-3.13.1-60.el7_2.src.rpm
```

После чего применить утилиту `rpm-build` (являющуюся частью пакета `rpm-build`) для извлечения файлов, составляющих RPM-пакет:

```
# rpm-build -bp ~/rpm-build/SPECS/SELinux-policy.spec
```

Когда это закончится, то исходный код политик SELinux будет располагаться внутри каталога `~/rpm-build/BUILD/serfepolicy-3.13.1`. Например, файл `screen.te` может быть найден в подкаталоге `./policy/modules/contrib`.

Эти файлы политик теперь могут быть безопасно скопированы для дальнейших манипуляций с ними и сборки, для замены существующей политики. Чтобы это сделать, надо, во-первых, удалить старый модуль политики и вставить новый модуль (но с таким же именем). Так будет произведена замена.

## 8.6.2. Замена политик в Gentoo

Для того чтобы заменить SELinux-политику в Gentoo Linux, во-первых, понадобится скачать политики через распределенную систему управления версиями Git, а с помощью команды `checkout` переключиться в ветку репозитория, содержащую необходимую версию политики. После этого мы уже сможем копировать файлы политик, обновлять их и устанавливать им более высокий приоритет.

Место, где хранятся и поддерживаются политики SELinux для операционной системы Gentoo Linux, называется `hardened-refpolicy.git`:

```
# git clone https://anongit.gentoo.org/git/proj/hardened-refpolicy.git
Cloning into 'hardened-refpolicy'...
remote: Counting objects: 23027, done.
remote: Compressing objects: 100% (7186/7186), done.
remote: Total 23027 (delta 18788), reused 19384 (delta 15768)
Receiving objects: 100% (23027/23027), 3.98 MiB | 3.38 MiB/s, done.
Resolving deltas: 100% (18788/18788), done.
```

**P** В результате выполнения команды `clone` (которая создает локальный каталог и определяет его как новое место для хранения и поддержки файлов, т. е. репозиторий, затем добавляет для взаимодействия удаленный репозиторий и скачивает из него всю отсутствующую информацию в локальное размещение) появляется следующая информация:

```
# git clone https://anongit.gentoo.org/git/proj/hardened-refpolicy.git
Клонирование в 'hardened-refpolicy'...
Удаленный репозиторий: вычисление числа объектов: 23027, выполнено.
Удаленный репозиторий: сжатие объектов: 100% (7186/7186), выполнено.
Удаленный репозиторий: общее число 23027 (различия 18788),
                          повторное использование 19384 (различия 15768)
Получение объектов: 100% (23027/23027), 3.98 Мебибайт | 3.38 Мебибайт/с, выполнено.
Принятие решений по различиям: 100% (18788/18788), выполнено.
```

Теперь найдем текущую версию политики, которая уже установлена:

```
# qlist -ICv SELinux-base-policy
sec-policy/SELinux-base-policy-2.20151208-r4
```

После того как мы узнали, какая нам нужна ветка репозитория (обозначенная как 2.20151208-r4), можем на нее переключиться с помощью следующей команды:

```
# git checkout tags/2.20151208-r4
```

Исходный код политики может быть теперь скопирован, им можно управлять, и его можно собрать. Когда модуль будет собран, то надо загрузить его с более высоким приоритетом, чем установленная политика по умолчанию (например, приоритет может быть 500):

```
# semodule -i screen.pp -X 500
```

## 8.7. ДРУГИЕ ВАРИАНТЫ УСИЛЕНИЯ ПОЛИТИКИ БЕЗОПАСНОСТИ

На протяжении всей этой книги мы рассматривали довольно много технологических возможностей SELinux. Создавая свои собственные политики SELinux, мы можем получить еще больше возможностей.

### 8.7.1. Создание типов SECMARK по специальным требованиям

Если мы собираем свою собственную политику и создаем специальный тип для маркировки сетевых пакетов методом SECMARK, это нам гарантирует, что конкретный тип допустимого набора функций, назначаемый процессам, будет именно тем, которому разрешено управлять этим сетевым взаимодействием.

Следующие правила SELinux создают тип `invalid_packet_t` (для таких пакетов, которые не должны быть переданы куда-то конкретно: например, при взаимодействии в рамках работы со службой PostgreSQL пакеты должны быть направлены в интернет, а не во внутреннюю сеть).

```
# cat custom_packets.te
policy_module(custom_packets, 1.0)

type invalid_packet_t;
corenet_packet(invalid_packet_t)

type intranet_packet_t;
corenet_packet(intranet_packet_t)
```

Вместе с этими правилами мы можем теперь создавать правила SECMARK для маркировки пакетов типами `invalid_packet_t` и `intranet_packet_t`.

Следующий шаг позволяет конкретным типам, назначаемым процессам, отправлять и принимать пакеты типа `intranet_packet_t`. Для типа `nginx_t`, назначаемого промежуточному серверу (*reverse proxy application*), представляющемуся клиенту как исходный сервер, нужно использовать следующее правило:

```
allow nginx_t intranet_packet_t:packet { send recv };
```

Кроме того, можно создать интерфейсы выполнения этих действий:

```
# cat custom_packets.if
interface('corenet_sendrecv_intranet_packets',`
    gen_require(`
        type intranet_packet_t;
    `)
    allow $1 intranet_packet_t : packet { send recv };
`)
```

Используя эти интерфейсы, можно в одну строку указать правило политики, определяющее, что сервер Nginx должен работать в усиленном режиме защиты:

```
corenet_sendrecv_intranet_packets(nginx_t)
```

### 8.7.2. Регистрация попыток доступа в журнале событий

Некоторые приложения имеют привилегии, о которых мы бы хотели получать уведомления, когда они вдруг используются. Подсистема регистрации событий безопасности в Linux имеет широкие возможности, для того чтобы сообщать о различной активности в системе, и SELinux способен усилить эти возможности, используя правило `auditallow`.

Это правило имеет аналогичный синтаксис, что и выражение `allow`. Но, в отличие от него, сообщает SELinux, не что доступ разрешен, а что факт доступа должен регистрироваться в журнале событий безопасности, если он разрешен.

К примеру, для того чтобы была зарегистрирована запись в файл, который имеет тип допустимого набора функций `etc_runtime_t`, необходимо записать следующее правило:

```
auditallow domain etc_runtime_t : file { write };
```

Когда сам доступ на запись выполнен, можно видеть запись AVC-типа с фиксацией выполненного разрешения на запись (`granted`) – в отличие от сообщений с запретами (`denial`):

```
type=AVC msg=audit(1373135944.183:209339): avc: granted { write }
for pid=23128 comm="umount" path="/etc/mtab" dev="md3" ino=135500
scontext=pgsql_admin_u:sysadm_r:mount_t
tcontext=root:object_r:etc_runtime_t
tclass=file permissive=0
```

Из этого сообщения можно сделать вывод о том, что пользователь SELinux `pgsql_admin_u` вызвал команду `umount`, которая в результате выполнила изменения в `/etc/mtab`.

### 8.7.3. Создание типов, соответствующих специальным требованиям

Для формирования типа по специальным требованиям надо создать описание типа (*type definition*) в SELinux (который является обычным типом допустимых наборов функций для файла), определить разрешения для доступа пользова-

телей (и приложений) к этому типу и затем зарегистрировать как настраиваемый под специальные требования тип (*customizable type*) – так, чтобы операция перемаркировки (*relabel operation*) не привела к исчезновению данного типа.

Допустим, что понадобился некий отдельный тип для внутренних файлов базы данных, с которыми взаимодействует пользователь через команду `sqlite3` (которая не запускается с собственным типом, назначенным ей, но запускается с типом вызывающего процесса, таким как `user_t` или `staff_t`). При использовании отдельного типа какой-то другой доступ к файлу (непривилегированными приложениями, которые запущены с другим типом допустимого набора функций) является по умолчанию закрытым. Даже когда эти другие приложения имеют доступ к стандартному типу `user_home_t`.

```
# cat custom_mydb_embedded.te
policy_module(custom_mydb_embedded, 1.0)

type mydb_embedded_t;
files_type(mydb_embedded_t)

gen_require(`
    type user_t;
`)

admin_pattern(user_t, mydb_embedded_t, mydb_embedded_t)
```

Затем надо отредактировать файл `/etc/selinux/targeted/contexts/customizable_types` и добавить тип `user_home_t` в него.

После того как все эти шаги будут выполнены, все пользователи (которые имеют назначенный тип `user_t`) смогут применять команду `chcon` для маркировки какого-либо файла как `mydb_embedded_t` и даже использовать данный файл в программе `sqlite` (или с другим приложением, которое запускается с таким SELinux-типом пользователя).

## 8.8. ЗАКЛЮЧЕНИЕ

Итак, мы рассмотрели, как управлять логическими параметрами SELinux при помощи такого инструмента, как `setsebool`, и как получать больше информации об этих параметрах, включая описания (используя команду `semanage boolean`) и зависящие от них правила (используя команду `sesearch`).

Затем мы рассмотрели, как модули специально создаваемых политик SELinux могут быть загружены и удалены, а также какие различные типы форматов разработки могут применяться при создании таких политик. Мы создали свои собственные модули политик для усиления штатных средств SELinux, применяя различные примеры, такие, которые позволили выполнить описания пользовательских типов, типов для веб-приложений и типов для метода маркировки сетевых пакетов `SECMARK`.

Также мы выяснили, как уже существующие политики могут быть заменены без добавления дополнительных правил. Тем более что как раз замена полити-

ки – это чуть ли не единственный способ, для того чтобы ее как-то ограничить в имеющихся разрешениях.

В следующей главе мы будем использовать различные инструменты для анализа существующей политики SELinux. Это необходимо, чтобы администратор мог проверить, что политика действительно поддерживает те правила безопасности, которые по его предположению должны работать, и что пользователи, на которых наложены какие-то ограничения в правах, не могут выйти за границы своего типа допустимого набора функций.

# Глава 9

## Анализ

### поведения политики

Несмотря на то что политики SELinux обеспечивают необходимое поведение системы, знание того, как она будет себя вести в различных ситуациях, очень поможет администратору. Оно позволит выполнить оценку и анализ основополагающих причин для тех или иных действий. В этом разделе мы будем:

- изучать, как получать сведения о деталях политик;
- использовать множество инструментов для определения преобразований процессов;
- анализировать информационные потоки.

Завершим эту главу рассмотрением нескольких небольших инструментов, выполняющих анализ различий между двумя файлами политик.

#### 9.1. Одноступенчатый анализ

В предыдущих разделах мы рассмотрели несколько методов анализа политик SELinux с помощью командно-строчных утилит, таких как `seinfo` и `sesearch`. Эти утилиты могут помочь пользователям в выполнении одноступенчатого анализа. Такие утилиты либо представляют информацию об объектах SELinux сразу же (в основном подобные сведения получаем от `seinfo`), либо запрашивают правила SELinux (что характерно для программы `sesearch`).

**i** Эти утилиты предоставляются в рамках пакета `setools`. Данный пакет регулярно обновляется, и последняя версия может оказаться не включенной, например, в Red Hat Enterprise Linux. Пакет программ предлагает новые возможности, но при этом могут различаться и результаты выполнения в зависимости от той среды, в которой эти программы выполняются. Отображаемые результаты работы, которые приводятся в этой главе, не будут объединяться с теми предупреждениями, которые могут появляться в зависимости от операционной системы.

Далеко не все возможности, которые предоставляют утилиты `seinfo` и `sesearch`, могут быть сейчас рассмотрены. В следующих нескольких подразделах



разберемся в том, как они могут быть использованы для получения необходимых данных о политиках SELinux и их анализа.

### 9.1.1. Использование различных файлов политик SELinux

Утилиты `seinfo` и `sesearch` могут быть применены как по отношению к текущей загруженной политике, так и к какому-либо избранному файлу политики. Последнее позволяет разработчикам запросить политики систем, к которым они не имеют прямого доступа или доступ к которым требует большого количества действий (например, на мобильных устройствах, где операционная система Android имеет свою политику SELinux, доступную в виде файла `/sepolicy`).

На примере той же системы Android анализ политики, имеющей наименование `sepolicy`, выполняется командой

```
$ seinfo sepolicy
```

Когда не удастся так обратиться к файлу политики, то программы `seinfo` и `sesearch` будут пытаться выполнить запрос к текущей активной политике (которая может быть и не последней установленной политикой) через псевдофайл `/sys/fs/selinux/policy`.

### 9.1.2. Отображение информации об объектах политики

Основной целью программы `seinfo` является отображение информации об объектах SELinux. Эта информация связана с типами объектов, которые SELinux поддерживает (и `seinfo` соответственно тоже). Различные варианты типов объектов поддерживаются в SELinux, начиная с хорошо известных (таких как типы допустимых наборов функций, атрибуты, роли, пользователи) и заканчивая специфическими объявлениями `fs_use_*` или выражениями `genfscon`.

Полный список поддерживаемых типов объектов (и связанных с ними параметров `seinfo`) может быть получен на страницах руководства (*manual page*) `seinfo` или если вызвать саму утилиту с параметром `--help`:

```
$ seinfo --help
```

```
usage: seinfo [-h] [--version] [-x] [--flat] [-v] [--debug] [-a [ATTR]]
              [-b [BOOL]] [-c [CLASS]] [-r [ROLE]] [-t [TYPE]] [-u [USER]]
              [--category [CAT]] [--common [COMMON]] [--constrain [CLASS]]
              [--default [CLASS]] [--fs_use [FS_TYPE]] [--genfscon
[FS_TYPE]]
              [--initialsid [NAME]] [--netifcon [DEVICE]] [--nodecon
[ADDR]]
              [--permissive [TYPE]] [--polcap [NAME]]
              [--portcon [PORTNUM[-PORTNUM]]] [--sensitivity [SENS]]
              [--typebounds [BOUND_TYPE]] [--validatetrans [CLASS]] [--all]
              [--ioportcon] [--iomemcon] [--pcidevicecon] [--pirqcon]
              [--devicetreecon]
              [policy]
```

```
...
```

Невзирая на тип объекта, который интересен пользователю, `seinfo` имеет три основных образа действия.

В **первом режиме** программа выводит список объектов заданного вида. Для этого должен быть указан только соответствующий параметр без какой-либо дополнительной информации. Например, для получения перечня классов, доступных в политике, надо выполнить следующую команду:

```
$ seinfo --class
Classes: 83
  appletalk_socket
  association
  blk_file
  capability
  capability2
  ...
```

Во **втором режиме** программа может подтвердить наличие (или, наоборот, отсутствие) экземпляра объекта. Чтобы это выполнить, надо добавить имя экземпляра в продолжение команды. Для проверки, является ли класс `memprotect` доступным в политике, надо набрать следующую команду:

```
$ seinfo --class memprotect
Classes: 1
  memprotect
```

К сожалению, если указанный экземпляр не доступен, то он будет показан только как часть выводимого сообщения на экран. Возвращаемый код приложения является таким же, несмотря на то, был найден экземпляр объекта или нет. Это делает его менее интересным для использования в скриптах, где рекомендуется использовать утилиту `grep`:

```
$ seinfo --class | grep -q -E "^[ ]*memprotect$"
```

В **третьем режиме** выводится расширенная информация о выбранном экземпляре. И хотя не все объекты поддерживают расширенный набор (*expanded set*) данных, многие все-таки его имеют. Эта информация в основном показывает список (различных) объектов, которые связаны с изначальным запросом.

Например, для класса расширенная информация покажет поддерживаемые разрешения:

```
$ seinfo --class memprotect -x
Classes: 1
  class memprotect
  {
    mmap_zero
  }
```

Наконец, `seinfo` может показать всю информацию сразу, если будет использоваться параметр `--all`. Правда, без расширенных сведений:

```
$ seinfo --all
```

### 9.1.3. Применение утилиты `sesearch`

В то время как приложение `seinfo` предоставляет информацию об объектах SELinux, утилита `sesearch` применяется для получения правил SELinux и информации о взаимодействии между исходным и целевым ресурсами.

Мы уже использовали эту утилиту для получения стандартных разрешающих (`allow`) правил, когда контролировали доступ через типы допустимых наборов функций и когда имели дело с логическими параметрами SELinux для этих разрешающих правил. Утилита `sesearch` позволяет нам не просто запросить правила, основанные на типе, но еще и отобразить те правила, которые сравнимы с заданным исходным ресурсом после параметра `--source (-s)` и/или целевым ресурсом после параметра `--target (-t)`.

✔ Утилита `sesearch` может иметь дело с косвенными сведениями об исходном ресурсе взаимодействия и целевом. Например, когда запрашивается информация, связанная с атрибутом `java_domain`, то могут быть выведены правила всех типов, которые имеют эти атрибуты. В более старых версиях `setools` такое поведение могло быть отключено с помощью параметра `-d`. В более новых версиях можно выбрать, какая информация не нужна, – об исходном ресурсе (тогда надо применить параметр `-ds`) или о целевом (тогда `-dt`).

Так как эти сведения составляют основной объем поведения SELinux, то давайте рассмотрим различные правила и их влияние на систему.

### 9.1.4. Запрос разрешающих правил

Первая группа правил – это разрешающие правила (*allow rules*) для взаимодействия субъекта доступа с объектом доступа с учетом класса ресурса, над которым выполняются действия:

```
$ sesearch --allow -s guest_t -t cgroup_t -c dir
allow guest_usertype cgroup_t:dir { search read lock ... open };
allow guest_usertype filesystem_type:dir { getattr open search };
```

i В более новых политиках SELinux поддерживаются (в ядре Linux) версии `setools`, которые включают правила `allowxperm`, расширяющие правила `allow` тем, что позволяют учитывать дополнительную информацию. Поэтому и такое название – от сочетания слов расширенный (*extended*) и решения (*permission*). Это правило используется для более тщательного контроля, связанного с текущими операциями ввода/вывода, но могут быть добавлены и другие возможности в ближайшем будущем.

Связанными с правилом `allow` являются правила `auditallow` (определяющие, какие срабатывания разрешающих правил должны быть зарегистрированы в журнале регистрации событий безопасности) и правила `dontaudit` (определяющие, какие действия, запрещенные политикой, не должны попадать в журнал событий в результате возникновения соответствующего события).

### 9.1.5. Запрос сведений о правилах преобразования типов

Следующий набор правил связан с правилами преобразования типов. Они определяют, в результате каких действий (таких как создание новых файлов,

директорий или даже процессов) изменяется параметр безопасности. Получаемые сведения позволяют проанализировать, какие преобразуются типы и в каких случаях:

```
$ sestatus -T -s guest_t -c process
type_transition guest_t abrt_helper_exec_t:process abrt_helper_t;
type_transition guest_t chfn_exec_t:process chfn_t;
...
```

В этом примере вывода списка правил мы можем увидеть, что даже для гостевого субъекта доступа (`guest_t`) имеется какое-то количество правил, которые позволяют ему преобразовывать одни типы допустимого набора функций процессов к другим.

Эти виды анализа будут использованы позже, когда мы будем рассматривать анализ преобразования SELinux-типов, назначаемых процессам (*domain transition analysis*).

### 9.1.6. Запрос правил для других типов

Кроме правил преобразования типов, существуют два других вида правил, связанных с преобразованиями, которые являются частью политики SELinux. При этом они используются приложениями, которые поддерживают SELinux и запрашивают эти правила.

Первое правило называется `type_change` (<изменение\_типа>), оно сообщает поддерживающему SELinux приложению, что когда оно запросит переназначение параметров безопасности (*relabel*) какого-то конкретного ресурса (*target*), работая при этом с каким-то известным SELinux-типом (*source*), тогда операция переназначения должна в результате присвоить ресурсу определенный тип допустимого набора функций. Это используется, когда какой-то ресурс был создан неким родительским процессом (*parent*), который работал с отличным типом, а управляться должен другим процессом, имеющим также и другой тип допустимого набора функций (*source*). В этом случае родительский процесс будет вызывать функции SELinux, для того чтобы обеспечить созданному ресурсу получение корректных прав доступа.

Утилита `sestatus` запускается с параметром `--type_change`:

```
$ sestatus --type_change -s guest_t
type_change guest_t ajaxterm_devpts_t:chr_file user_devpts_t;
type_change guest_t console_device_t:chr_file user_tty_device_t;
...
```

Второе правило называется `type_member` (<представитель\_типа>), которое используется для многоэкземплярных ресурсов. Здесь снова родительское приложение, которое иницирует множество экземпляров, должно быть из числа поддерживающих SELinux и, в свою очередь, вызывать необходимые SELinux функции, для того чтобы обеспечить получение необходимых параметров безопасности экземплярами такого ресурса.

В этом случае утилиту `sesearch` надо запускать с параметром `--type_member`:

```
$ sesearch --type_member -s guest_t
type_member guest_t tmp_t:dir user_tmp_t;
type_member guest_t user_home_dir_t:dir user_home_dir_t;
```

### 9.1.7. Запрос правил, связанных с ролями

Предыдущий набор правил был строго связан с типами. Однако SELinux имеет также правила, связанные с деятельностью ролей. С помощью утилиты `sesearch` можно выяснить, какие роли разрешены для доступа другим ролям и когда роли преобразуются (как при переключении от пользовательской роли к роли системной).

Применение параметра `--role_allow` (<разрешение\_роли>) позволит увидеть, какие роли разрешены:

```
$ sesearch --role_allow -s webadm_r
allow webadm_r system_r;
```

Применение параметра `--role_trans` позволит выяснить, когда происходит автоматический переход от одной роли к другой:

```
$ sesearch --role_trans -s webadm_r
role_transition webadm_r httpd_initrc_exec_t:process system_r;
```

Анализ преобразования ролей и правил разрешения ролей помогает администратору делать выводы о том, какие роли являются более привилегированными или могут привести к потенциальным проблемам безопасности. К примеру, если есть роль администратора веб-приложений `webadm_r`, способная переключаться на системную роль `system_r` при работе с объектами типа `httpd_initrc_exec_t`, то это может позволить такой роли выполнять действия, выходящие за пределы ее допустимых функций, если у нее есть права на изменение ресурсов, имеющих тип `httpd_initrc_exec_t`.

И следующий запрос дает возможность увидеть, что таких прав у нее нет:

```
$ sesearch -s webadm_t -t httpd_initrc_exec_t -A
allow webadm_t httpd_initrc_exec_t:file { read open ... execute };
```

Однако недостаточно просто посмотреть на основной тип пользователя. Неплохо было бы проанализировать все типы, которые доступны для роли `webadm_r`. Этот глубокий и многоитерационный анализ является предметом рассмотрения в следующих подразделах главы.

### 9.1.8. Отображение данных с помощью графической программы `arol`

Подходящим инструментом для выполнения анализа политики является программа `arol`, распространяемая через пакет `setools`. Она имеет графический интерфейс, позволяющий аналитикам и администраторам выполнять множество аналитических действий в отношении политики SELinux.

После запуска первое действие, которое надо сделать в программе `apol`, – это загрузить предназначенную для анализа политику (либо текущую активную политику, либо файл, скопированный из другой системы). Это можно сделать при помощи кнопки **Open Policy** (*Открыть политику*) или из главного меню **File | Open Policy** (*Файл | Открыть политику*).

Затем инструмент отобразит основные данные загруженной политики, показанные на рис. 9.1.

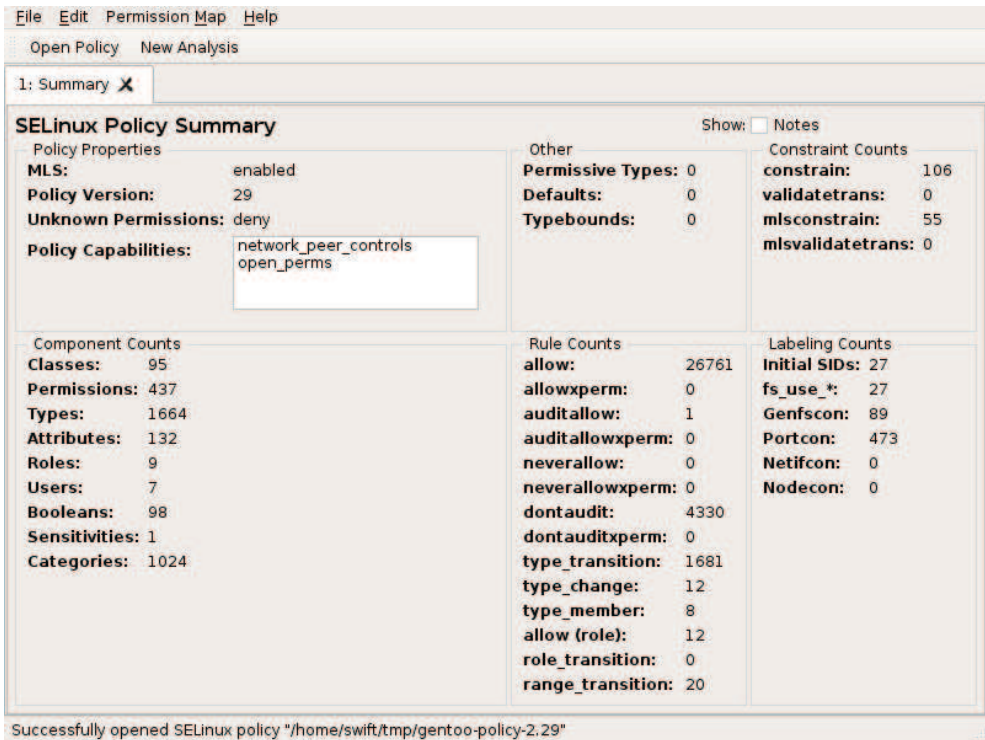


Рис. 9.1 ❖ Приложение `apol` после загрузки файла политики

**i** Многие функции анализа, реализованные в программе `apol`, поддерживаются и 3-й, и 4-й версиями пакета программ `setools`. И хотя сам графический интерфейс претерпел некоторые изменения, но изображения интерфейса приводятся здесь для версии 4.

После того как политика будет загружена, следует нажать кнопку **New Analysis** (*Новый анализ*), для того чтобы инициировать доступные функции для анализа политики, как показано на рис. 9.2.

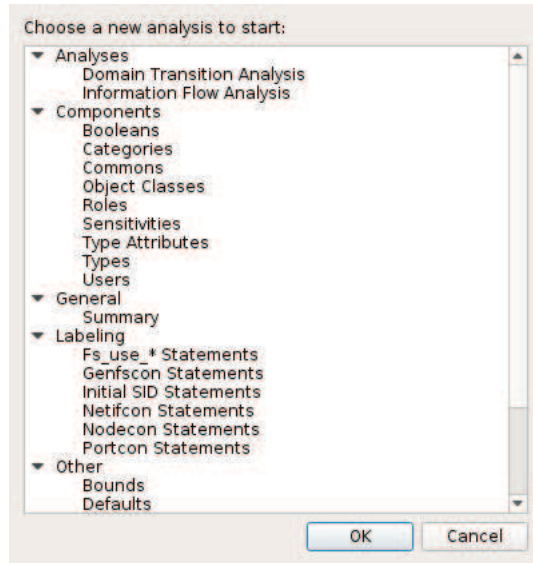


Рис. 9.2 ❖ Отображение поддерживаемых методов анализа в программе apol

В представленном списке методов анализа давайте выберем строку **Types** (*Типы*), в результате чего можно будет увидеть следующий экран (см. рис. 9.3), отображающий доступные типы. А затем выберем атрибут, для того чтобы увидеть, какие типы допустимого набора функций с ним связаны.

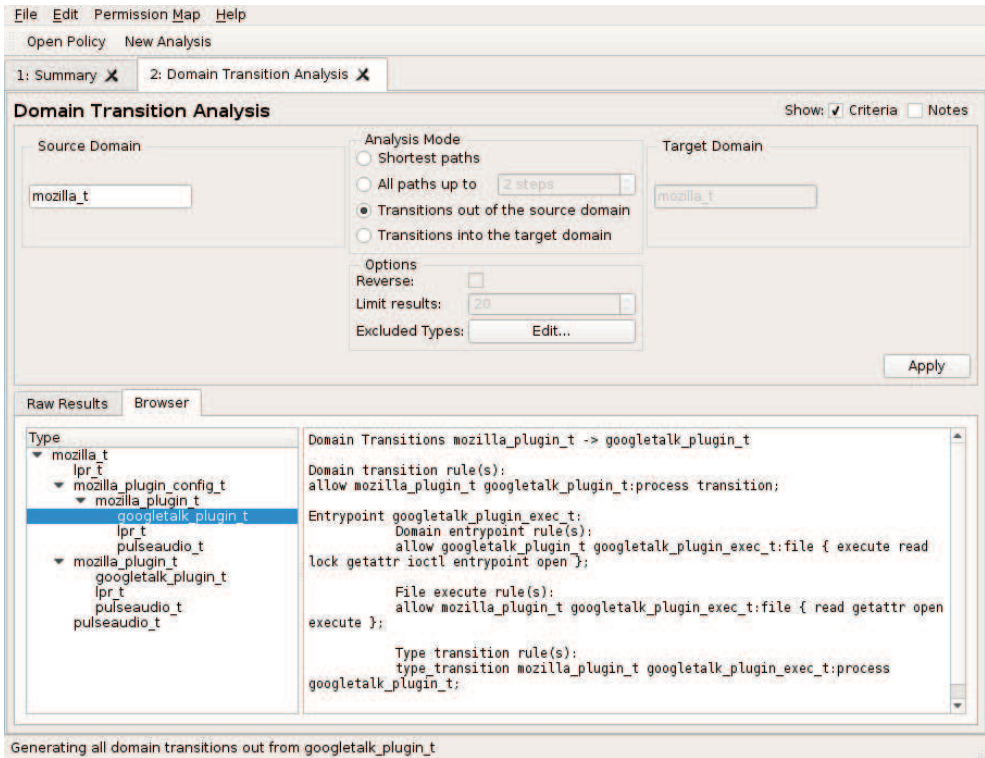


Рис. 9.3 ❖ Отображение типов в приложении apol.

В отдельном окне показано, какие типы сопоставлены с атрибутом alsadomain

Аналогичным образом анализируются правила для типов функциональных ограничений (*TE Rules*). Может быть выполнен анализ, аналогичный тому, который делался при помощи программы *sesearch* (рис. 9.4).



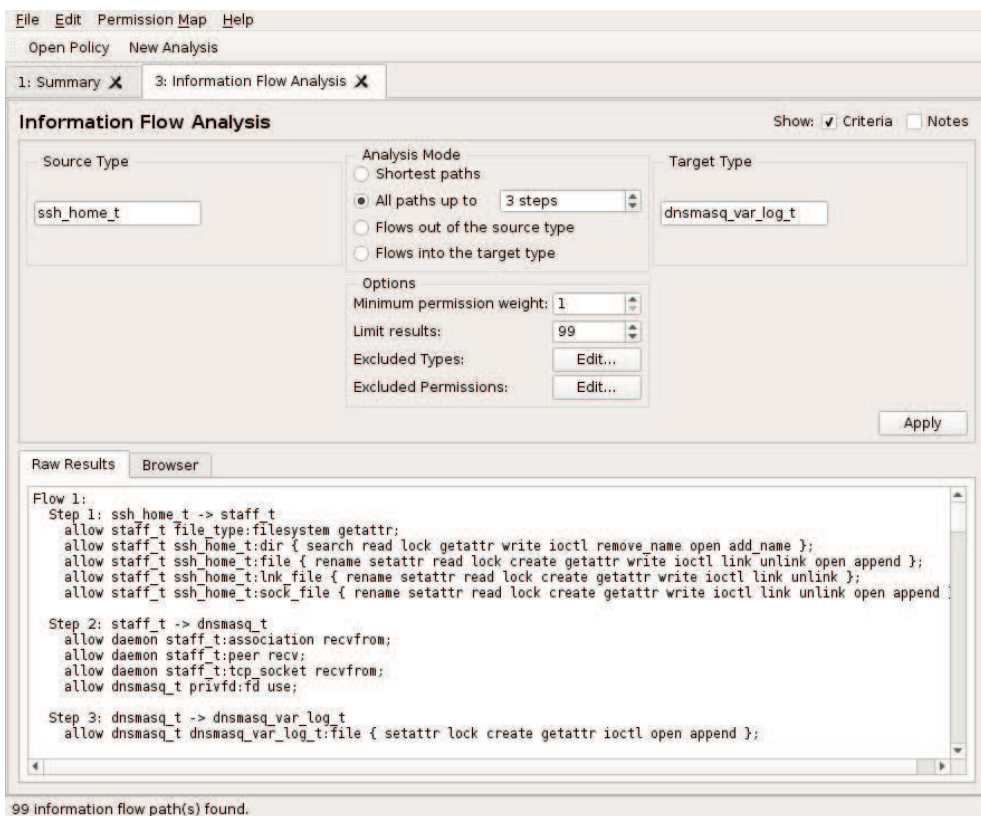


Рис. 9.4 ❖ Пример запуска при помощи программы arol запроса правил для типов функциональных ограничений, в которых участвуют два типа допустимого набора функций

## 9.2. АНАЛИЗ ПРЕОБРАЗОВАНИЙ ТИПОВ ПРОЦЕССОВ

Когда мы имеем дело с политиками SELinux, важным методом анализа является выполнение разбора преобразований типов допустимого набора функций, назначаемых процессам (*domain transition analysis*). Такие типы ограничены правилами контроля доступа, назначенными для них, но пользователи (сессии) могут переходить к другим типам, выполняя предусмотренные наборы приложений.

Поэтому правильнее было бы разобраться с условиями преобразования, которые могут происходить между двумя типами. Это позволит администраторам выполнять проверку безопасного состояния политики. Учитывая обяза-

тельный характер защиты SELinux, злоумышленники скорее сочтут довольно сложным получить возможность выполнить желаемое приложение, если анализ преобразования его типа покажет, что тип объекта, который инициирует работу желаемого приложения, не может выполнить это приложение ни прямым запуском, ни косвенным.

Использование такого анализа позволит подтвердить, является ли тип допустимого набора функций, назначаемого процессам (домен), корректно ограничен и не могут ли уязвимости, характерные для этого типа, привести к повышению привилегий.

### 9.2.1. Использование программы apol

Для того чтобы выполнить анализ преобразований типов, после запуска программы apol надо выбрать кнопку **New Analysis** (*Новый анализ*). Отобразится число возможных вариантов анализа. На самом верху будет вариант анализа для преобразований типов, назначаемых процессам (доменам), – **Domain Transition Analysis**. После того как этот вариант анализа будет выбран, на следующем интерфейсе программы будут показаны следующие доступные режимы для анализа (**Analysis Mode**):

- 1) режим кратчайших путей (**Shortest paths**) используется программой, чтобы показать преобразование типа родительского процесса к целевому типу, и останавливается в поиске после того, как найдет какой-нибудь путь преобразования;
- 2) режим нахождения всех путей за заданное количество шагов (**All paths up to**) может, в принципе, показать множество преобразований между родительским и целевым типами, но только с учетом шагов, максимальное количество которых задается в графическом интерфейсе при выборе этого режима. Прямое преобразование от родительского к целевому считается переходом за один шаг (и как раз такой результат легко получить, используя программу *sesearch*);
- 3) режим вывода преобразований типа родительского процесса (**Transitions out of the source domain**) в результате показывает, какие виды преобразований являются разрешенными для данного типа родительского процесса (*source domain*). Пользователю полученный результат может быть представлен в виде дерева, по которому он может перемещаться для получения подробной информации;
- 4) режим вывода преобразований в тип целевого процесса (**Transitions into the target domain**) показывает, какие виды преобразований приводят к переходу к типу целевого процесса. Получается анализ обратного характера.

Чтобы сделать анализ более гибким, можно использовать дополнительные параметры. Например, можно исключить определенные типы из анализа. Можно исключить те приложения, которые администратор относит к доверенным, например такие, которым назначается тип `*_sudo_t`. Тем более что такие

типы предоставят множество потенциальных шагов преобразований к большому числу типов, назначенных приложениям.

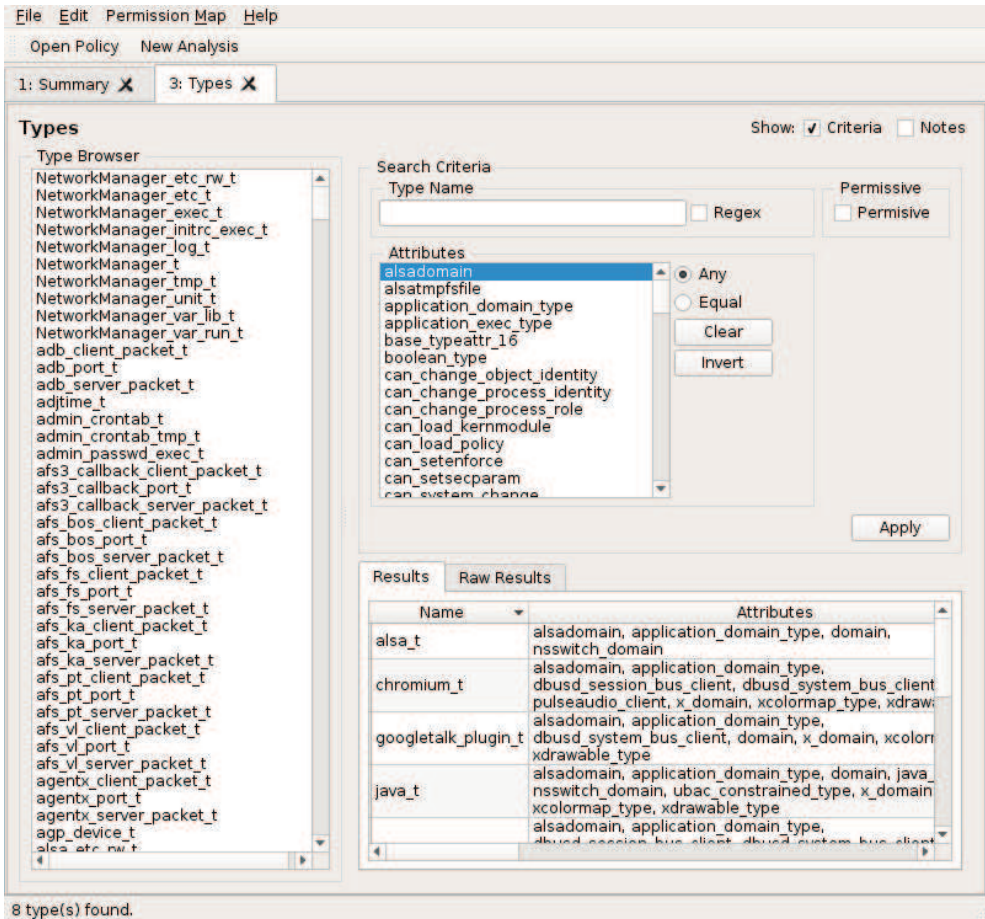


Рис. 9.5 ❖ Пример экранного вывода после запроса вывести варианты преобразований для типа родительского процесса

## 9.2.2. Использование программы `sedta`

Начиная с 4-й версии программы `setools` приложение командной строки, которое называется `sedta`, имеет возможность выполнять анализ преобразований типа без применения графического приложения, такого как `apol`.

Основные функциональные возможности, предлагаемые программой `apol`, также доступны и в `sedta`. Но такой интерактивности отображаемых данных, которая присутствует в графическом интерфейсе, конечно, в консольной программе не представлено. Администратору надо еще раз выполнить команду,

для того чтобы обновить полученные данные, подобно тому как их получается видеть в графическом окне.

Например, для того чтобы увидеть доступные преобразования для такого типа, как `mozilla_t`, надо использовать такую команду:

```
$ sedta -s mozilla_t
Transition 1: mozilla_t -> mozilla_plugin_config_t
...
Transition 2: mozilla_t -> pulseaudio_t
...
Transition 3: mozilla_t -> lpr_t

Domain transition rule(s):
allow mozilla_t lpr_t:process transition;

Entrypoint lpr_exec_t:
Domain entrypoint rule(s):
allow lpr_t lpr_exec_t:file { execute read ... entrypoint open };

File execute rule(s):
allow mozilla_t lpr_exec_t:file { read getattr open execute };

Type transition rule(s):
type_transition mozilla_t lpr_exec_t:process lpr_t;

Transition 4: mozilla_t -> mozilla_plugin_t
...
4 domain transition(s) found.
```

**P** В русскоязычном варианте этот вывод команды выглядит так:

```
$ sedta -s mozilla_t
Преобразование 1: mozilla_t -> mozilla_plugin_config_t
...
Преобразование 2: mozilla_t -> pulseaudio_t
...
Преобразование 3: mozilla_t -> lpr_t
Правило, разрешающее преобразование родительского типа mozilla_t в целевой lpr_t:
allow mozilla_t lpr_t:process transition;
Отправная точка преобразования к типу lpr_t:
Разрешающее правило для взаимодействия с типом, являющимся отправной точкой:
allow lpr_t lpr_exec_t:file { execute read ... entrypoint open };
Правила выполнения процессом файла:
allow mozilla_t lpr_exec_t:file { read getattr open execute };
Правила преобразования типа родительского процесса к типу целевого процесса:
type_transition mozilla_t lpr_exec_t:process lpr_t;
Преобразование 4: mozilla_t -> mozilla_plugin_t
...
Найдено 4 преобразования типа процесса.
```

Другой пример – выяснить, могут ли учетные записи обычных пользователей выполнять подключаемый модуль для голосового чата и видеочата **Google Talk**:

```

$ sedta -s user_t -t googletalk_plugin_t -S
Domain transition path 1:
Step 1: user_t -> googletalk_plugin_t

Domain transition rule(s):
allow user_t googletalk_plugin_t:process transition;

Entrypoint googletalk_plugin_exec_t:
Domain entrypoint rule(s):
allow googletalk_plugin_t googletalk_plugin_exec_t:file \
{ execute read lock getattr ioctl entrypoint open };

File execute rule(s):
allow user_t googletalk_plugin_exec_t:file \
{ read getattr open execute };
allow user_t application_exec_type:file \
{ execute read lock getattr execute_no_trans ioctl open };

Type transition rule(s):
type_transition user_t googletalk_plugin_exec_t:process googletalk_plugin_t;

1 domain transition path(s) found.

```

**P** В русскоязычном варианте этот вывод команды мог бы выглядеть следующим образом:

```

$ sedta -s user_t -t googletalk_plugin_t -S
Преобразование типа процесса по пути 1:
Шаг 1: user_t -> googletalk_plugin_t

Правило, разрешающее преобразование родительского типа в целевой:
allow user_t googletalk_plugin_t:process transition;

Отправная точка преобразования к целевому типу:
Разрешающее правило для взаимодействия с типом, являющимся отправной точкой:
allow googletalk_plugin_t googletalk_plugin_exec_t:file \
{ execute read lock getattr ioctl entrypoint open };

Правила выполнения файла:
allow user_t googletalk_plugin_exec_t:file \
{ read getattr open execute };
allow user_t application_exec_type:file \
{ execute read lock getattr execute_no_trans ioctl open };

Правило преобразования типа родительского процесса к типу целевого процесса:
type_transition user_t googletalk_plugin_exec_t:process googletalk_plugin_t;

Найдено 1 преобразование типа процесса.

```

## 9.3. АНАЛИЗ ПОТОКОВ ИНФОРМАЦИИ

Другим подходом к исследованию политики SELinux является анализ информационных потоков. В отличие от преобразований типов процессов (которые показывают, как один тип процесса может получить какой-то конкретный набор разрешений, переходя в другой тип допустимого набора функций), анализ информационных потоков позволяет увидеть, как один тип процесса может упускать (целенаправленно или не очень) информацию к процессам, имеющим другой тип допустимого набора функций.

Анализ информационных потоков (*information flow analysis*) выполняется путем просмотра всех действий, которые могут происходить между двумя типами. Объект некоего исходного типа может быть прочитан каким-либо типом процесса, который впоследствии может записать полученную информацию в объект другого типа. Этот простой метод представляет собой двухшаговый анализ потока.

В действительности это не так уж и просто – проверить операции чтения и записи (хотя и более чем возможно). Информация может несанкционированно распространяться (*leaked*) через имена файлов, файловых дескрипторов и многое другое. Анализ информационных потоков должен учитывать все эти способы.

### 9.3.1. Использование программы *apol* для анализа потоков информации

После загрузки политики SELinux следует нажать кнопку **New Analysis** (*Новый анализ*), для того чтобы инициировать доступные функции для анализа политики, как показано на рис. 9.6. В появившемся окне выбрать пункт **Information Flow Analysis** (*Анализ информационных потоков*) в качестве метода анализа. На рис. 9.6 мы видим окно программы, похожее на то, которое видели раньше при анализе преобразований типов процессов.

Как можно видеть на рис. 9.6, в результате анализа было найдено 99 путей информационных потоков, существующих между типом родительского процесса `ssh_home_t` и типом `dnsmasq_home_t` целевого объекта. Можно было найти и больше, чем это количество, но в программе установлены ограничения по числу обнаружений в поле *Limit results – Предел результатов*. На первом шаге (на рис. 9.6 отмечены как Step 1) показана связь между типом `ssh_home_t` и типом `staff_t`. Для них существуют разрешающие правила, позволяющие процессам с типом `staff_t` взаимодействовать (считывать, записывать и выполнять иные действия) с объектами (файлы, каталоги, сокет и др.) типа `ssh_home_t`. На втором шаге (Step 2) просматривается взаимодействие между типом `staff_t` и `dnsmasq_t`: процесс с типом `dnsmasq_t` способен получать информацию от объектов типа `staff_t` (поскольку `dnsmasq_t` является типом фонового процесса) и, конечно, способен добавлять (*to append*) сведения в файл регистрации, имеющий тип `dnsmasq_var_log_t`.

Для того чтобы выполнить качественный анализ информационного потока, необходимо точно определить критерии поиска, а также иметь карту разрешений (*permission map*), о которой будет сказано позже.

Среди настраиваемых режимов анализа (см. Analysis Mode на рис. 9.6) предлагаются методы, похожие на те, которые были рассмотрены для режимов преобразования типа процесса:

- режим кратчайших путей (**Shortest paths**) – определяет, что инструмент анализа завершит поиск конкретного информационного потока (между

источником, иницирующим взаимодействие, и целевым объектом), когда какой-нибудь поток будет найден. Если существует не один путь перемещения потоков, с одинаковым количеством шагов от источника до целевого объекта, то все эти пути информационных потоков будут отображаться;

- режим нахождения всех путей за заданное количество шагов (**All paths up to**) – определяет, что инструмент будет искать все информационные потоки, укладывающиеся в число шагов не больше, чем явно задано в настройке. Независимо от того, что именно происходит, ресурс считывается процессом с определенным типом или процесс пишет информацию в ресурс с заданным типом;

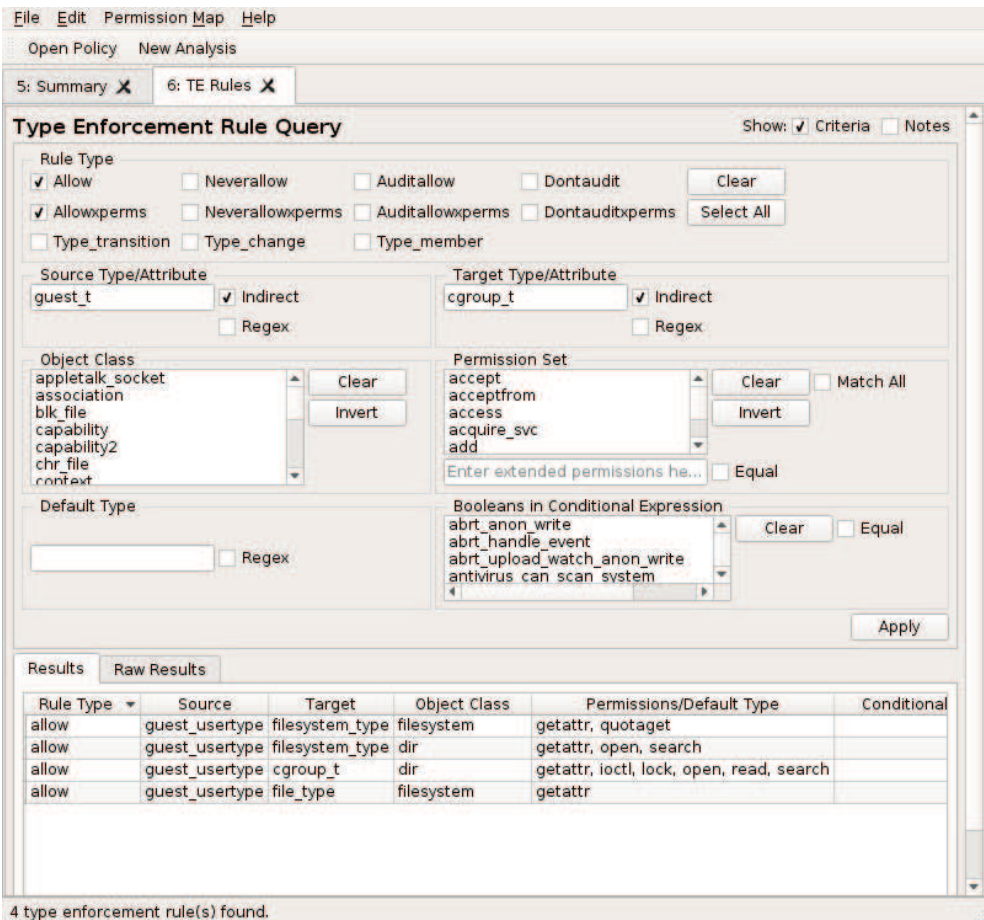


Рис. 9.6 ❖ Пример анализа потоков без обновления сопоставления привилегий

- режим вывода потоков, исходящих от типа процесса, инициирующего взаимодействие (**Flows out of the source type**), – предоставляет вывод в виде дерева, где указаны все потоки от данного типа. С помощью него пользователи могут просматривать различные типы процессов и типы объектов, к которым может поступать информация;
- режим выявления потоков, приходящих к объекту, который имеет целевой тип допустимого набора функций (**Flows into the target type**), – предоставляет обратный анализ информационных потоков, начиная с целевого типа и отображая по восходящей последовательность передачи информационных потоков к различным исходным типам процессов и типам объектов.

Существует несколько параметров, которые могут быть установлены для анализа информационных потоков:

- 1) **Minimum permission weight** (минимальное оценочное значение разрешения) – позволяет пользователям видеть только те разрешения (действия), которые имеют определенное оценочное значение (или более высокое). Каждому действию назначается некое оценочное значение в этом инструменте, начиная с наиболее низкого приоритета (такой операции, как блокировка, назначается значение 1) и заканчивая наиболее высоким (такой операции, как запись, присваивается значение 10). Эти оценочные значения определяются в карте разрешений (*permission map*), которые будут рассмотрены позже;
- 2) **Limit results** (предельное число выводимых результатов) – это тот параметр, который определяет, что средству анализа надо остановиться при достижении заданного числа обнаруженных потоков;
- 3) **Excluded Types** (исключенные типы) – позволяет пользователям удалить конкретные типы из числа подлежащих анализу. Доверенные типы вполне могут быть исключены из анализа потоков, позволяя пользователям сосредотачиваться на менее доверенных;
- 4) **Excluded Permissions** (исключенные разрешения) – позволяет пользователям удалить конкретные разрешения из анализа (такие как `ioctl`, `lock` и `listen`).

Последний параметр дает возможность пользователям управлять действующей картой разрешений. При анализе информационного потока создание достоверной карты разрешений является важным и не умаляющим значимости шагом.

По умолчанию карта разрешений доступна по пути `/usr/share/setools/perm_map`. В ней сосредоточены все классы, упоминаемые со всеми разрешениями. Для каждого разрешения карта предоставляет информацию для инструмента анализа о том, является ли данное разрешение подобным разрешению на чтение, подобным разрешению на запись, двусторонним каналом (когда возможны и запись, и чтение) или не связанным с потоками. Затем каждому из этих разрешений присваивается оценочное значение.



Для того чтобы ограничить анализ потоков заданным набором классов, надо обновить карту разрешений (либо напрямую, либо с помощью инструмента) и затем перезапустить анализ.

### 9.3.2. Использование программы `seinfoflow` для анализа потоков информации

Командно-строчная программа `seinfoflow` предоставляется в рамках пакета `setools` версии 4 и предлагает возможности для анализа потоков, сравнимые с программой `apol`.

Каждый вызов `seinfoflow` требует карты разрешений для проведения анализа. Пользователи могут указать карту разрешений, имеющуюся по умолчанию `/usr/share/setools/perm_map`, но рекомендуется вместо нее использовать карту разрешений, созданную по специальным требованиям пользователя.

К примеру, когда анализируются не связанные с сетью информационные потоки, администратор может создать карту разрешений, которая исключает все классы, связанные с сетевыми взаимодействиями. В результате анализатор не будет учитывать их из числа актуальных при разборе возможной передачи потока информации.

Давайте посмотрим на информационные потоки между типами `ssh_home_t` и `dnsmasq_var_log_t`. При анализе будем использовать специальную карту разрешений, метод определения кратчайших путей и не рассматривать оценочное значение разрешений больше 10, что указывается в команде параметром `-w 10`. Кроме того, исключим некоторые типы процессов, которые либо не применяются (такие как `nfsd_t`, потому что служба NFS не запущена), либо являются доверенными. В результате мы обнаружим два информационных потока, в каждом из которых будет по два шага:

```
$ seinfoflow -m perm_map -s ssh_home_t -t dnsmasq_var_log_t \
  -S -w 10 \
  setfiles_t restorecond_t tmpfiles_t nfsd_t kernel_t
```

Flow 1:

```
Step 1: ssh_home_t -> portage_t
  allow portage_t file_type:dir { read ... write };
  allow portage_t file_type:fifo_file { read ... write };
  allow portage_t file_type:file { read ... write };
  allow portage_t file_type:sock_file { read ... write };
  allow portage_t non_auth_file_type:dir { read ... };
  allow portage_t non_auth_file_type:file { read ... };

Step 2: portage_t -> dnsmasq_var_log_t
  allow portage_t file_type:dir { read ... write };
  allow portage_t file_type:fifo_file { read ... write };
  allow portage_t file_type:file { read ... write };
  allow portage_t file_type:sock_file { read ... write };
```

Flow 2:

```
Step 1: ssh_home_t -> sysadm_t
```

```

allow sysadm_t file_type:dir { read ... };
allow sysadm_t non_auth_file_type:dir { read ... write };
allow sysadm_t non_auth_file_type:fifo_file { read ... write };
allow sysadm_t non_auth_file_type:file { read ... write };
allow sysadm_t non_auth_file_type:sock_file { read ... write };
allow sysadm_t ssh_home_t:dir { read ... write };
allow sysadm_t ssh_home_t:file { read ... write };
allow sysadm_t ssh_home_t:sock_file { read ... write };

```

Step 2: sysadm\_t -> dnsmasq\_var\_log\_t

```

allow sysadm_t dnsmasq_var_log_t:dir { read ... write };
allow sysadm_t dnsmasq_var_log_t:fifo_file { read ... write };
allow sysadm_t dnsmasq_var_log_t:file { read ... write };
allow sysadm_t dnsmasq_var_log_t:sock_file { read ... write };
allow sysadm_t non_auth_file_type:dir { read ... write };
allow sysadm_t non_auth_file_type:fifo_file { read ... write };
allow sysadm_t non_auth_file_type:file { read ... write };
allow sysadm_t non_auth_file_type:sock_file { read ... write };

```

2 information flow(s) found.

## 9.4. ДРУГИЕ ВИДЫ АНАЛИЗА ПОЛИТИК

Есть еще два дополнительных инструмента (*sediff* и *sepolicy*), которые позволяют получить некоторое представление о текущей политике. В следующих двух пунктах этой главы они будут рассмотрены более детально.

### 9.4.1. Сравнение политик при помощи *sediff*

Инструмент *sediff*, являясь частью пакета *setools*, показывает различия между двумя файлами политик и создает отчет о них для пользователя. Данная программа не предоставляет такие же возможности по определению изменений и внесению исправлений в оригинальный файл, как известная Linux-команда *diff*. Но при этом является достаточно мощной для обнаружения и анализа небольших различий.

Наиболее частый случай использования инструмента *sediff* состоит в проверке, что политика, собранная из некоего исходного файла, является сравнимой с политикой, распространяемой в дистрибутиве. В этом случае речь идет о двоичных файлах политик. Администраторы могут убедиться в том, что исходный код, который они использовали для построения политик, является таким же, как и тот код, который распространяется как образующий для предоставленной официально политики.

В базовом варианте использования надо просто подать два файла на вход программы:

```
$ sediff distro-policy.30 selfbuilt-policy.30
```

```
Policy Properties (0 Modified)
```

```
Booleans (0 Added, 0 Removed, 1 Modified)
```

```
Modified Booleans: 1
```

```
* mcelog_exec_scripts (Modified default state)
+ True
- False
```

Еще при помощи данного инструмента можно показать различия в конкретной области (такой как допустимые типы, роли, логические параметры или правила).

Например, для того чтобы показать различия между файлом с политикой Gentoo Linux и файлом с политикой Red Hat Enterprise Linux на уровне типов, нужно использовать следующую команду:

```
$ sediff --type gentoo-policy.29 rhel-policy.29 | grep Types
Types (3220 Added, 269 Removed, 369 Modified)
Added Types: 3220
Removed Types: 269
Modified Types: 369
```

**P** В русскоязычном варианте этот вывод команды мог бы выглядеть следующим образом:

```
$ sediff --type gentoo-policy.29 rhel-policy.29 | grep Types
Типы допустимого набора функций (3220 добавленных, 269 удаленных, 369 измененных)
Добавленных типов: 3220
Удаленных типов: 269
Измененных типов: 369
```

Глядя на этот вывод, можно сказать, что политика Gentoo имеет гораздо меньше (на 3220) типов, чем политика в Red Hat Enterprise Linux. Это связано с тем, что Gentoo задействует модули политики SELinux, только когда установлены пакеты, которые используют данную политику.

Полный набор поддерживаемых полей сравнения доступен в страницах руководства для `sediff` или в сведениях из справки, которую можно получить, вызвав следующую команду:

```
$ sediff --help
```

## 9.4.2. Анализ политик при помощи `sepolicy`

Другой инструмент, распространяемый в составе пакета `polyscoreutils`, – приложение `sepolicy`. Эта программа уже раньше встречалась в тексте данной книги, когда рассматривалось ее применение для вывода основной информации о политике SELinux, такой как логические параметры. Но у этой утилиты есть и другие трюки, спрятанные, как говорится, в рукаве.

С помощью команды `sepolicy communicate` администраторы могут быстро увидеть, взаимодействуют ли два разных типа, назначаемых процессам, за один промежуточный шаг, на котором используется для взаимодействия какой-нибудь файл. Это похоже на анализ информационного потока, который был рассмотрен раньше, но с акцентом, сделанным на файлах:

```
$ sepolicy communicate -s mozilla_t -t chrome_sandbox_t
config_home_t
cifs_t
```

```
xserver_tmpfs_t
ecryptfs_t
fusefs_t
user_fonts_cache_t
cache_home_t
nfs_t
```

Другой вид анализа, который можно сделать, – это разбор возможных преобразований типов процессов. В результате его проведения можно узнать, какие преобразования могут происходить между одним и другим типами процессов. Для этого надо выполнить команду `sepolcity transition`:

```
$ sepolcity transition -s user_t -t lpr_t
user_t ... mozilla_plugin_t @ lpr_exec_t --> lpr_t
user_t ... vmtools_helper_t ... vmtools_t ... ifconfig_t \
... iptables_t ... insmod_t ... mount_t ... glusterd_t \
... initrc_t ... realmd_t ... sshd_t ... unconfined_t \
... openshift_initrc_t ... apmd_t ... system_cronjob_t \
... munin_t @ lpr_exec_t --> lpr_t
```

## 9.5. ЗАКЛЮЧЕНИЕ

В этой главе были рассмотрены различные методы анализа политик SELinux.

Вначале был разобран одноступенчатый анализ, выполняемый при помощи таких инструментов, как `sesearch` и `seinfo`, которые уже раньше упоминались в данной книге. В результате стало понятно, что эти инструменты могут предоставить администратору, который хочет проанализировать действующую политику SELinux, довольно много сведений.

После них были рассмотрены программы `apol`, `sedta` и `seinfoflow`, позволяющие выполнить более глубокий анализ политики. Они позволяют рассмотреть последовательность преобразований типов процессов (как один тип может быть преобразован в другой) и отследить информационные потоки. Причем потоки анализируются как в ключе передачи данных от одного типа к другому, так и для поиска возможных утечек данных через косвенные разрешения передачи данных между типами процессов. Что позволяет принять администратору превентивные меры по защите информации до того, как она сможет покинуть защищаемые им границы передачи данных.

В завершение главы было рассмотрено еще несколько утилит для анализа. Одной из них стала команда `sediff`, которая показывает различия между двумя файлами политик, позволяя таким образом администраторам определить, насколько похожа активная политика в одной системе на другую политику.

В следующей и завершающей главе будут применены полученные знания из всех других глав для демонстрации того, как система защиты SELinux может быть использована для решения ряда задач обеспечения защиты информационных систем.

# Глава 10

## Частные случаи настройки защиты

Ранее описанные функции оперативного управления должны быть настроены в соответствии с целями и требованиями администраторов. В этой главе будет описано несколько ситуаций, которые научат администраторов:

- 1) обеспечивать большую защищенность веб-серверов при помощи категорий SELinux и корректной маркировки файлов;
- 2) защищать службы командного интерфейса (*shell*) с помощью разделения экземпляров SSH и различного использования службы РАМ фоновым процессом SSH;
- 3) настраивать сервер с сетевой файловой системой (NFS) для работы с различными параметрами безопасности файлов и настраивать прикладные программы для работы с файлами, предоставленными по протоколу NFS.

Завершит главу небольшое сравнение реализации защиты средствами SELinux файлового сервера Samba и конфигурации сетевой файловой системы.

### 10.1. УСИЛЕНИЕ ЗАЩИТЫ ВЕБ-СЕРВЕРОВ

Веб-серверы – это сервис общей инфраструктуры многих архитектур. Они также часто бывают доступны в сети интернет (как напрямую, так и опосредованно через промежуточный сервер (*reverse proxy*), перенаправляющий внешние запросы на различные серверы внутренней сети, образуя таким образом дополнительную подсистему защиты). И поэтому они намного более уязвимы к атакам, чем сервисы второго плана, выполняющиеся на уровне доступа к данным (*backend services*), такие как системы баз данных.

Веб-серверы способны размещать материалы различного типа – от статичных сайтов до динамичных, вплоть до веб-сервисов, которые используются в микросервисной архитектуре. Несмотря на свою направленность на приложения, SELinux способен защитить и веб-сервер тоже.

### 10.1.1. Описание условий работы

Прежде чем приступить к настройке SELinux и заняться активными изменениями, было бы разумно описать, с чем придется иметь дело. Глядя на имеющиеся условия и изучая различные технические аспекты, администраторы будут способны получить наиболее точное представление об архитектуре и принять решения, которые хорошо повлияют на защищенность системы. Зачастую мы экономим на том, чтобы описать сложившуюся ситуацию, поскольку схема часто более информативна, чем тщательно выполненное описание конкретных исходных данных.

Когда описываете подобные архитектуры, возьмите в расчет несколько аспектов. Каждый из них оказывает влияние на установки, связанные с защитой, и помогает администратору с выбором конкретного направления действий:

- 1) пользовательский аспект:
  - какие группы пользователей нуждаются в соединении с веб-сервером?
  - все ли пользовательские группы в равной степени заслуживают доверия?
  - всем ли группам пользователей требуются одинаковые возможности веб-приложения, которым они пользуются?
  - размещаются ли подключающиеся группы пользователей к серверу в одном и том же месте?
- 2) аспект администрирования веб-сервера:
  - продумать, каким образом будет управляться веб-сервер и кем. Здесь есть два главных подхода: системное администрирование и администрирование веб-сервера (которые часто требуют доступа через командную оболочку к системе). Отдельно от них существует администрирование информационной составляющей веб-сервера, которое не требует особых привилегированных прав доступа к системе;
- 3) аспект сложности функционирования:
  - проверить, требуют ли разные веб-приложения дифференцированного поведения от веб-сервера. Если одно веб-приложение имеет только статическое содержание, в то время как иное требует присоединения к базам данных и другим удаленно размещенным сервисам, тогда было бы разумно разделить эти веб-приложения и разместить их на разных системах.

Предположим, что после рассмотрения требований работы веб-сервера мы пришли к следующей ситуации: все сайты поделены между тремя отдельными программно-аппаратными серверами, которые суммарно запускают шесть экземпляров веб-сервера. **Внешние пользователи** (*public users*) подключаются через один промежуточный сервер (*reverse proxy*), выполняющий перенаправления их запросов на различные серверы внутренней сети, а **внутренние пользователи** (*internal users*) подключаются через другой – свой собственный промежуточный сервер. В зависимости от того, какие сайты доступны, промежуточные серверы определяют, к каким экземплярам веб-серверов они подключаются. Схему взаимодействия см. на рис. 10.1.

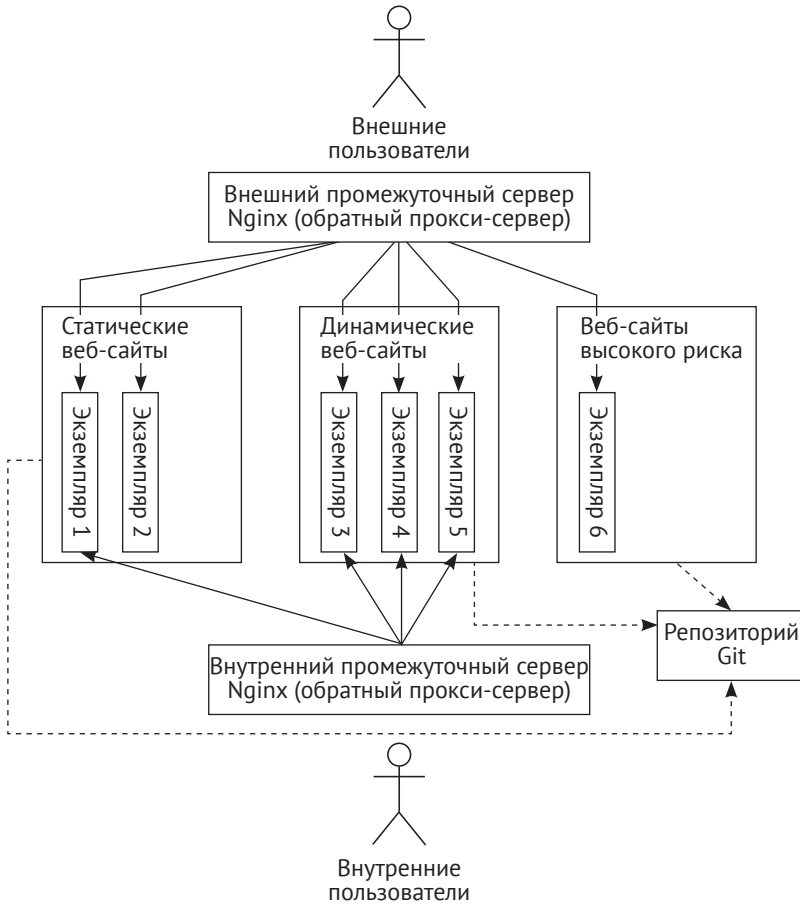


Рис. 10.1 ❖ Высокоуровневый обзор распределения веб-сайтов

Для каждого компонента такой инфраструктуры веб-сервисов (включая сервер «Nginx») рекомендуются разные конфигурации и настройки SELinux. Мы же главным образом обратим свое внимание на экземпляры, несущие основную функциональную нагрузку. Однако повышение защищенности за счет промежуточных серверов не должно быть забыто, т. к. они представляют первую линию защиты в предлагаемой архитектуре.

### 10.1.2. Настройка для установки нескольких экземпляров программ

Многие из серверов, которые были определены ранее, будут запущены множеством экземпляров «Apache» (или какими-либо другими веб-серверами). Допустимо ожидать, что эти экземпляры должны работать на различных пор-

тах (в условиях, когда не существует множества IP-адресов, связанных с этим сервером) и даже с различными категориями информации.

Во-первых, убедитесь, что конфигурации каждого из рассмотренных экземпляров размещаются в различных директориях, желательно имеющих название, аналогичное размещенному в них экземпляру. Смешивание различных конфигураций в одной и той же директории может усложнить разделение экземпляров впоследствии.

```
# mkdir /etc/apache2/instance1 /etc/apache2/instance2 ...
```

Затем обновите модульный файл (*unit file*), который описывает параметры системы инициализации `systemd`, применяемые к веб-серверу для поддержки множества экземпляров программы. Или скрипт инициализации (*init script*), если работаете в системе с SysV-совместимой подсистемой загрузки. В операционной системе Gentoo скрипт инициализации может быть обновлен для того, чтобы он поддерживал символические ссылки на скрипты инициализации. Если называть скрипты согласованно с созданными экземплярами, то система `init` (*инициализации*) сможет легко сделать вывод о том, где находится действующий файл конфигурации.

```
# cat /etc/systemd/service/
[Unit]
Description=Apache web server
ConditionPathExists=/etc/apache2/%i/httpd.conf
After=network.target

[Service]
Type=forking
EnvironmentFile=/etc/sysconfig/httpd.%i
PIDFile=/run/apache2/%i.pid
ExecStart=/usr/sbin/apache2 -f /etc/apache2/%i/httpd.conf
ExecReload=/usr/sbin/httpd -k restart -f /etc/apache2/%i/httpd.conf
ExecStop=/usr/sbin/httpd -k stop -f /etc/apache2/%i/httpd.conf
SELinuxContext=system_u:system_r:httpd_t:%i
Restart=always

[Install]
WantedBy=multi-user.target
```

При помощи этого метода каждый экземпляр будет сопоставлен со своим собственным файлом конфигурации и со своей собственной SELinux-категорией.

### 10.1.3. Создание категорий SELinux

Для поддержки именованных категорий (к примеру, `instance1` и `instance2`) необходимо включить сервис `mctransd` и настроить категории в файле `setrans.conf`, как было описано в главе 3 «Управление учетными записями пользователей»:

```
# cat /etc/SELinux/targeted/setrans.conf
s0-s0:c0,c101.c106=WebAdmin
```



```
s0:c101=instance1
s0:c102=instance2
...
```

Это необходимо сделать, т. к. модульный файл (*unit file*) или скрипт инициализации (*init script*) будет ссылаться на имя экземпляра как на переменную, имеющую значение конкретного диапазона категорий. Постоянные значения в этих файлах использовать в скриптах не всегда возможно, и в них достигается таким образом большая гибкость выполнения.

#### 10.1.4. Выбор необходимых параметров безопасности

Веб-серверы имеют в своем распоряжении множество параметров безопасности SELinux. Следует указать верные параметры безопасности для информационного наполнения веб-сайта, поскольку это обеспечит правильную обработку веб-сервером файлов, даже когда дискреционный механизм контроля доступа мог бы разрешить большие права доступа:

- тип `httpd_sys_content_t` следует использовать для режима «только чтение» по отношению к статическому содержанию сайта. Целесообразно использовать его для изображений, каскадных таблиц стилей (CSS-файлов), HTML-файлов, PHP-файлов и других, по крайней мере до тех пор, пока веб-серверу не требуется их модифицировать;
- тип `httpd_sys_rw_content_t` следует использовать для режима «чтение/запись» информационного наполнения веб-сайта. К примеру, система `wiki` (которая обрабатывает публикуемую пользователем информацию и использует отдельный каталог `data/` для хранения страниц) будет использовать этот тип допустимого набора функций по отношению к этому каталогу, в то время как остальное содержание веб-сайта (такое как файл конфигурации) работает с типом `httpd_sys_content_t`;
- тип `httpd_sys_ra_content_t` следует использовать для информационного наполнения веб-сайта, подлежащего считыванию и пополнению (*append*). Его можно использовать для файлов, которые не очищаются при записи в них, например журналов регистрации событий;
- тип `httpd_sys_htaccess_t` предназначен для файлов дополнительной конфигурации веб-сервера с разрешением `.htaccess` и, возможно, для файлов, содержащих пароли для доступа к веб-ресурсам `.htpasswd`. Эти файлы не предназначаются для демонстрации пользователям, но используются для чтения веб-сервером;
- тип `httpd_sys_script_exec_t` следует использовать для CGI-скриптов, позволяющих веб-серверу выполнять сценарии;
- типы `httpd_sys_script_rw_t`, `httpd_sys_script_ra_t` и `httpd_sys_script_t` используются для файлов, которые обрабатываются только CGI (и другими, вызываемыми веб-сервером) сценариями. Они могут быть с правами на чтение/запись, только на добавление или только на чтение;

- типы `httpd_user_*_t`, в принципе, подобны типам для системных скриптов `httpd_sys_*_t`, но в данном случае предназначены для информационного содержания веб-сайта, определяемого пользователем. Веб-сервер может поддерживать работу с пользовательскими директориями (например, при помощи директивы `UserDir` веб-сервера Apache, которая устанавливает фактическую директорию в домашнем каталоге пользователя для использования, когда будет получен запрос на какой-нибудь документ пользователя). И в этом случае будут использоваться типы `httpd_user_*_t`;
- тип `public_content_rw_t` – это особый случай. Он предназначен для файлов, которые доступны и управляются несколькими службами. Например, если веб-сервер будет содержать данные, которые предоставляются для скачивания через стандартный протокол (FTP), то вполне может иметь смысл использовать для них тип `public_content_rw_t` (поскольку FTP-сервер не будет иметь никаких прав на управление типами `httpd*_content_t`);
- некоторые веб-приложения имеют в своем распоряжении узкоспециализированные политики. Как правило, эти политики определяют необходимые типы, связанные с обрабатываемыми характерными для приложения данными (*content*) и выполняемыми сценариями (*scripts*). Например, для такой платформы, как MediaWiki, предусмотрены типы `httpd_mediawiki_content_t` и `httpd_mediawiki_script_exec_t`. Все типы, используемые для этих специфических веб-приложений, должны следовать одним и тем же правилам, поскольку они формируются с помощью основной политики веб-сервера.

Для корректной работы защиты требуется присвоить верный параметр безопасности для информационного наполнения веб-сайта. Многие администраторы, вероятно, использовали бы команду `semanage fcontext` для его установки:

```
# semanage fcontext -a -t httpd_sys_rw_content_t \
"/srv/web/instance1/htdocs/data(/.*)?"
# semanage fcontext -a -t httpd_sys_content_t \
"/srv/web/instance1/htdocs(/.*)?"
# semanage fcontext -a -t httpd_mediawiki_content_t \
"/srv/web/instance3/htdocs/wiki(/.*)?"
```

Тем не менее, чтобы можно было более быстро воспроизводить эти назначения и эффективно использовать имеющиеся в библиотеках SELinux возможности по упорядочиванию и обработке, было бы лучше создать (возможно, даже не учитывающий ничего больше) некий модуль политики безопасности SELinux, который бы выполнял сопоставление между областями размещения файлов и соответствующих им параметров безопасности.

Например, чтобы создать такую политику, можно использовать синтаксис обобщенно-промежуточного языка SELinux (CIL) следующим образом:

```
# cat custom_mediawiki.cil
(filecon "/srv/web/instance1/htdocs/data(/.*)?" any
(system_u object_r httpd_sys_rw_content_t ((s0) (s0)))
)
```

```
(filecon "/srv/web/instance1/htdocs(/.*)?" any
(system_u object_r httpd_sys_content_t ((s0) (s0)))
)
(filecon "/srv/web/instance3/htdocs/wiki(/.*)?" any
(system_u object_r httpd_mediawiki_content_t ((s0) (s0)))
)
```

Этот модуль потом может быть загружен и использован конкретно по отношению к описанным в нем объектам:

```
# semodule -i custom_mediawiki.cil
# restorecon -RvF /srv/web/instance*
```

### 10.1.5. Включение администраторов в систему защиты

Если веб-серверами будут управлять различные пользователи или команды, то может оказаться хорошей идеей сопоставить им разные роли. В главе 8 «Работа с политиками SELinux» мы увидели, как создавать дополнительные роли и типы пользователей, тогда как глава 3 «Управление учетными записями пользователей» показала нам, как сопоставлять пользователей Linux и группы с разными SELinux-пользователями.

Мы можем создать группу пользователей с названием `webadmins` и потом присвоить членам этой группы SELinux-пользователя `webadm_u`:

```
# semanage login -a -s webadm_u -r WebAdmin %webadmins
```

Администраторы веб-сайта должны быть сопоставлены с правильным диапазоном необходимых ему для работы уровней доступа и категорией информации. Имя `WebAdmin` было определено в файле `setrans.conf`, который создали выше в пункте «Управление категориями SELinux».

### 10.1.6. Управление работой веб-сервера

Когда веб-сервер используется, его работа, как правило, требует правильной настройки. Статичная веб-страница не требует какого-либо динамического контроля доступа, который мог бы быть включен в других случаях. И даже серверы динамических веб-приложений нечасто требуют полных привилегий как для доступа к файлам, так и для полноценной работы процесса.

Проект рассматриваемой системы предполагает несколько вариантов работы:

- статичные веб-сайты не будут иметь каких-либо дополнительных характерных для работы с ними правил. Потому что веб-серверы не будут, к примеру, иметь возможность присоединиться через них к другим системам;
- динамичные веб-сайты имеют общий набор характерных действующих правил. Однако правила, чувствительные к состоянию безопасности, не включены;
- веб-сайты с высоким уровнем риска (возможности понести значительный ущерб) при утечке, потере или модификации информации имеют

много правил, связанных с обеспечением защиты. Эти системы, как правило, наиболее надежные, по сравнению с теми системами, на которых размещены динамические сайты.

Если необходимо, то можно использовать несколько систем размещения веб-сайтов, надежно выполняющих обработку данных, утечка которых может привести к ощутимому ущербу. Благодаря виртуализации (в защите которой SELinux также эффективно участвует, как мы видели в 6-й главе «Поддержка *sVirt* и *Docker*») мы можем легко создать специализированные системы, облегчающие стратегию защиты.

Настройка характерного для работы контроля доступа выполняется главным образом при помощи логических параметров SELinux. Существует около 40 таких параметров SELinux, применяемых к окружению веб-сервера. Следующий их набор показывает уровень детализации и чувствительности правил наиболее точно:

- 1) логические параметры `httpd_can_*` (<веб-сервер\_может\_\*>) включают или выключают правила, относящиеся к действию, на которое логический параметр ссылается. Например:
  - `httpd_can_connect_ftp` (<веб-сервер\_может\_подключаться\_к\_FTP>) позволяет какому-либо веб-серверу соединиться с неким FTP-сервером. Это может быть необходимо, если одно из веб-приложений является FTP-клиентом, основанным на веб-технологии;
  - `httpd_can_network_connect` (<веб-сервер\_может\_выполнить\_сетевое\_соединение>) позволяет веб-серверу подключаться к любой службе, имеющей сетевой интерфейс взаимодействия, доступ к которому в целом не был бы разрешен;
  - более детально ориентированный логический параметр SELinux, `httpd_can_network_connect_db` (<веб-сервер\_может\_выполнить\_сетевое\_подключение\_к\_базе\_данных>) позволяет веб-серверам подключаться к системам баз данных, имеющим сетевой интерфейс взаимодействия, которых по крайней мере намного меньше, чем всевозможных сетевых служб.

Эти логические параметры SELinux будут отключены на статических веб-сайтах. Хорошо проработанные в деталях логические параметры SELinux используются для динамических веб-сайтов, а основные на сайтах с высоким уровнем возможности несения потерь в случае нарушения конфиденциальности, целостности или доступности обрабатываемых данных;

- 2) параметр `httpd_anon_write` (<веб-серверу\_разрешена\_анонимная\_запись>) позволяет веб-серверам записывать данные в файлы, которым назначен тип `public_content_rw_t` (<тип\_общие\_данные\_читать\_и\_записывать>). Этот тип может быть использован, когда информационная составляющая веб-сервера обрабатывается множеством служб, таких как веб-сервер, FTP-сервер и сервер файловых ресурсов;

- 3) параметр `httpd_builtin_scripting` (<веб-сервер\_со\_встроенными\_сценариями>) должен быть включен, когда используются динамические языки программирования, такие как PHP. Он будет, как правило, выключен для статических веб-сайтов, а включен для динамических сайтов и сайтов с высоким уровнем риска в случае нарушения конфиденциальности, целостности или доступности обрабатываемых данных:
- параметры `httpd_dbus_*` (такие как, например, `httpd_dbus_sssd`, предназначенный для взаимодействия с набором фоновых программ, управляющих доступом к удаленным директориям, и механизмом аутентификации SSSD – *System Security Services Daemon*) позволяют веб-серверу взаимодействовать с другими службами через систему межпроцессного взаимодействия D-Bus. Параметр должен быть выключен для статических веб-сайтов, но может быть включен для динамических и сайтов с высоким уровнем риска;
- 4) параметры `httpd_use_*`, такие как `httpd_use_nfs` (<веб-сервер\_использует\_NFS>), позволяют веб-серверу применять конкретную службу или команду. К примеру, `httpd_use_nfs` позволяет веб-серверу обрабатывать данные, размещенные на подключенных сетевых файловых системах (NFS). А `httpd_use_gpg` позволит веб-серверу вызывать программу для шифрования информации и создания электронных цифровых подписей GnuPG.

Некоторые логические типы SELinux очень специфичны. Например, `httpd_tmp_exec` (<веб-сервер\_выполнение\_временных>) позволяет веб-серверу выполнять файлы, содержащиеся в каталоге `/tmp` (или других временных размещениях). Это считается риском для безопасности системы (так как атакующие могут повлиять на временные объекты более просто, чем на любые другие). Много логических параметров, содержащих в названии фразу `_exec` (сокр. от *execution* – выполнение), например `httpd_execmem` (<веб-сервер\_выполнение\_в\_памяти>), считаются небезопасными, и их следует включать только тогда, когда система в других отношениях достаточно защищена.

Переключение логических типов SELinux выполняется при помощи программы `setsebool`:

```
# setsebool -P httpd_use_nfs on
```

Если функциональная значимость логического параметра SELinux еще не определена, то можно включить его без сохранения состояния в хранилище политик и потом проверить, так ли влияет измененный логический параметр на поддерживаемые правила, как ожидалось:

```
# setsebool httpd_use_nfs on
```

### 10.1.7. Работа с обновлением содержания

В представленной архитектуре (на рис. 10.1) используется хранилище распределенной системы управления версиями Git для размещения информационного содержания веб-сайта. Это, конечно, не единственный вариант разме-

щения данных. Мы могли бы использовать подключение сетевой файловой системы NFS (как описано позже в этой главе) или возможность подключения пользователей через командно-строчный интерфейс (*shell*) для возможности передать свои собственные данные.

Преимущество применения репозитория Git в данном случае заключается в том, что мы можем использовать локально выполняемое пакетное задание (*batch job*), производящее обновление этого хранилища, для каждого из веб-сайтов. Администраторам, которые управляют информацией веб-сайта, нет необходимости в том, чтобы заходить в систему для обновления сайта, достаточно просто отправить (*push*) данные в соответствующую ветвь репозитория Git. После чего локально выполняемое пакетное задание сначала извлечет (*pull*) информацию из указанного репозитория, а потом вставит ее в текущие данные, таким образом гарантируя, что параметры безопасности будут корректно установлены.

Допустим, мы хотим, чтобы содержимое каталога `/srv/web/instance1` было извлечено (*pull*) из каталога репозитория Git `gitserver:/proj/instance1`. В таком случае системный администратор (или веб-администратор) может один раз создать клон (*clone*) [всех версий всех файлов] репозитория, а потом создать некий сценарий обновления. В данном случае созданный клон не требует аутентификации во время доступа (поскольку тут не потребуются каких-либо новых привилегий), который позже поможет нам в автоматизации извлечения данных из репозитория и объединения их с имеющейся рабочей версией (для которой нет необходимости предоставлять имя пользователя и пароль):

```
# cd /srv/web
# git clone https://gitserver/proj/instance1.git instance1
# restorecon -RvF instance1
```

В связи с тем, что администратор (*root*) сайта (`./instance1/htdocs`) не включил каталог `.git/` (`./instance1/.git`) в число объектов, которым назначаются специальные параметры безопасности (см. настройки, выполненные раньше), данные, составляющие информацию сайта, имеют базовый контроль безопасности, определяющий, какие из них могут быть предоставлены через сайт, а какие нет. Конечно, это подразумевает, что эта структура каталогов исходно имеет надлежащую маркировку параметрами безопасности.

Локальное выполнение следующего задания может, в свою очередь, гарантировать, что параметры безопасности (включая категории) назначены верно:

```
# cat /usr/local/bin/update-instance1.sh
#!/bin/sh
cd /srv/web/instance1 || exit 1;
git pull || exit 2;
restorecon -RvF /srv/web/instance1/ || exit 3;
```

Само по себе задание должно быть запущено с правами, достаточными для того, чтобы исполнить эти команды. По умолчанию задания программы `cron` (обеспечивающей периодическое выполнение заданий в определенное время) выполняются с типом `cronjob_t`, который уже имеет базовые права для выпол-

нения. Но права на перемаркировку ресурсов ей не предоставлено. Их можно предоставить дополнительно типу `sgonjob_t` или же создать собственный тип для процесса (домен), который будет содержать правильный набор разрешений для обновления информации на сайте.

### 10.1.8. Настройка сети и правил межсетевого экрана

Межсетевые экраны уже давно стали частью метода защиты от окружающих систем. Системы, которые размещают веб-серверы, также должны использовать правильные настройки межсетевого экрана – для обеспечения того, чтобы только с легального адреса можно бы было получить доступ к службам.

Когда размещается на сервере множество экземпляров (*instances*) веб-сайтов, то может потребоваться ограничить доступ к ним с высоким уровнем детализации этих ограничений. Веб-серверу `instance3` необходима доступность только из внутренних систем, в то время как `instance4` должен иметь доступ как из внешнего мира, так и из внутренней сети. С учетом того, что оба веб-сайта взаимодействуют с окружением через промежуточный сервер (*reverse proxy*), перенаправляющий внешние запросы на различные серверы внутренней сети, межсетевой экран должен убедиться, что только те системы, на которых размещается этот промежуточный сервер, могут подключаться к конкретному экземпляру веб-сайта.

Для этого можно включить контроль маркировки пакетов методом `SECMARK` в правилах межсетевого экрана, гарантируя тем самым, что экземпляры веб-сервера смогут получать только те сетевые пакеты, которые имеют соответствующую маркировку параметром безопасности, отвечающим за категорию информации:

```
# iptables -t security -A INPUT -p tcp --dport 8081 -j SECMARK \
  --selctx "system_u:object_r:http_server_packet_t:s0:c101"
# iptables -t security -A INPUT -p tcp --dport 8082 -j SECMARK \
  --selctx "system_u:object_r:http_server_packet_t:s0:c102"
```

Эти установки отражают только часть необходимых настроек. Более глубокое рассмотрение использования `SECMARK` было описано в главе 5 «Контроль сетевого взаимодействия».

Так как мы запускаем экземпляры веб-сайтов, работающих на различных сетевых портах, то необходимо сконфигурировать SELinux так, чтобы позволить веб-серверам использовать эти порты:

```
# semanage port -a -t http_port_t -p tcp 8081
# semanage port -a -t http_port_t -p tcp 8082
```

## 10.2. ЗАЩИТА КОМАНДНО-СТРОЧНОГО ИНТЕРФЕЙСА

Другой инфраструктурной службой, которая требует защиты, является командно-строчный интерфейс пользователя (*shell*). Службы командно-строчного интерфейса способны предоставить взаимодействие в том числе и злоумышлен-

никам, которые мечтают использовать уязвимости в системе для **удаленного выполнения команд** (RCE – remote command execution). Очевидно, что защита этих служб является стратегически важной задачей для администраторов.

### 10.2.1. Разделение SSH на несколько экземпляров

Одним из потенциальных методов повышения защиты серверов, поддерживающих службы для взаимодействия с пользователем через командный интерфейс, является метод разделения доступа для администраторов и пользователей.

Сервер, предоставляющий доступ через SSH (*Secure Shell* – защищенный командно-строчный пользовательский интерфейс), может потребовать аутентификацию по идентификатору и паролю либо на основе криптографических ключей. Служба будет по умолчанию работать через 22-й порт и, вероятно, включит изменение корневого каталога при помощи команды `chroot` таким образом, что обычные пользователи не будут иметь доступа ко всей файловой системе, а только к конкретной области файловой системы, например к каталогу `/var/jail`.

Дополнительные методы обеспечения безопасности для защиты SSH могут быть тоже подключены на стороне сервера. Например, локальная служба `fail2ban` проверяет записи в журналах регистрации для IP-адресов, определяя те, которые пытаются грубой силой атаковать SSH-сервер и потом запрещают эти IP-адреса, внося их в правила локального межсетевого экрана. Кроме того, могут быть запущены на сервере и такие программы, как `sshguard`, `sshblock` и `sshit`.

Управляемый через SSH сервер должен быть защищен в наибольшей степени. Он должен обеспечивать двухфакторную аутентификацию (и по паролю, и по ключу) и авторизацию, основанную на паролях и ключах, и какую-нибудь другую взаимосвязанную аутентификацию, а также использовать нестандартный порт (например, 4971), разрешая подключаться через него только членам группы администраторов (см. рис. 10.2).

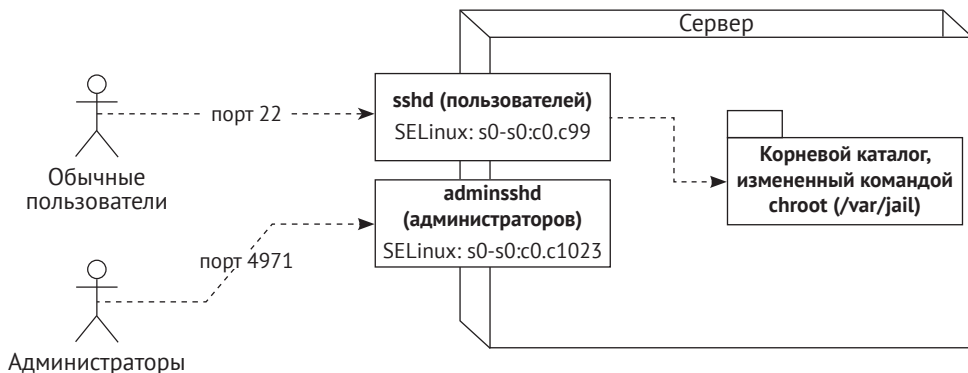


Рис. 10.2 ❖ Разделение доступа по SSH, основанное на пользовательских ролях



Конфигурация фоновой программы SSH на сервере будет храниться в файлах `/etc/ssh/user/sshd_config` и `/etc/ssh/admin/sshd_config`. Модульный файл подсистемы инициализации `systemd` или инициализирующий скрипт обновляется для того, чтобы указывать на правильный экземпляр, подобно тому методу, который использовался для веб-сервера в предыдущем подразделе.

Использование отдельных экземпляров для SSH имеет и другие преимущества, кроме мер защиты и контроля. Несложно представить ситуацию, когда потребуется запустить для взаимодействия с пользователем фоновую программу SSH на сервере с более низким уровнем доступа и ограниченным диапазоном поддерживаемых категорий защищаемой информации (`s0-s0:c0.c99`), в то время как экземпляр этой программы для взаимодействия с администратором должен иметь более высокий уровень доступа (если какая-нибудь политика многоуровневой защиты SELinux используется на сервере) или иметь полный набор категорий информации (`s0-s0:c0.c1023`). Это отличается от запуска нескольких экземпляров веб-серверов, поскольку в том случае нам не нужен был диапазон.

В данном случае пользователи могли бы быть разделены позже, так чтобы один имел доступ к информации из категории `c7`, а другой – к информации из диапазона категорий `c8.c10`. Подобное разделение будет осуществлено при помощи подключаемых модулей аутентификации PAM, но это возможно, только если программа SSH, через которую они подключаются, имеет преобладающий диапазон категорий над категориями, с которыми разрешено работать пользователям.

Отдельные экземпляры также позволяют администраторам временно блокировать службу (например, путем выключения программы SSH, работающей с пользователем), в то же время сохраняя доступ для себя.

### 10.2.2. Обновление правил работы сети

Аналогично тому, как настраивался веб-сервер, мы должны обратить внимание на правила межсетевого экрана. Но, в отличие от веб-сервера, мы намерены использовать SECMARK несколько на другом уровне (если только не стояла бы задача по обеспечению различий на уровне сетевых адресов, гарантирующих, что администраторы подключаются к системе только с имеющихся в списке узлов сети, чьи адреса включены в число доверенных).

В данном случае мы включим маркировку методом SECMARK на уровне пакетов (не обращая внимания на маркировку категориями). Эта маркировка SECMARK полезна (и даже обязательна, если уже была активирована другая метка SECMARK), для того чтобы обеспечить взаимодействие между двумя SSH-службами, получившими маркировку параметрами безопасности как взаимодействующие через SSH:

```
# iptables -t security -A INPUT -p tcp --dport 22 -j SECMARK \
  --selctx "system_u:object_r:ssh_server_packet_t:s0"
# iptables -t security -A INPUT -p tcp --dport 4971 -j SECMARK \
  --selctx "system_u:object_r:ssh_server_packet_t:s0"
```

Необходимо изменить SELinux-тип, назначенный по умолчанию для нестандартного порта 4971 системой защиты. Этот порт по умолчанию будет иметь тип `unreserved_port_t`, а надо присвоить ему тип допустимого набора функций `ssh_port_t`, характерный для работы с SSH:

```
# semanage port -a -t ssh_port_t -p tcp 4971
```

Обычно администраторы подключаются к серверу из меньшего числа рабочих мест, чем клиенты и обычные пользователи, с применением программ командно-строчного пользовательского интерфейса. Ограничить такой доступ можно при помощи нескольких настроек.

Межсетевой экран может обновить свои правила так, чтобы позволить взаимодействие с портом 4971 только из разрешенных подсетей. Это позволит обеспечить скрытость данной службы от других подсетей.

Если управляющие воздействия администратора поступают из какого-то другого сетевого интерфейса, то службу SSH можно даже настроить на сервере для ожидания запросов подключения только от этого сетевого интерфейса. При этом служба SSH, принимающая запросы пользователей, может ожидать подключения ото всех доступных интерфейсов.

### 10.2.3. Изменение корневого каталога для отдельной программы

Если на сервере фоновая программа SSH, получающая запросы от пользователей, осуществляет изменение корневого каталога (*chrooting*), предоставляя пользователям доступ к некоему подкаталогу в файловой системе, то надо сообщить системе защиты SELinux, что этот подкаталог должен иметь такой же набор параметров безопасности, как если бы доступ осуществлялся без изменений корневого каталога.

Например, для того чтобы объекты каталога `/var/jail/*` были промаркированы параметрами безопасности так же, как и корень файловой системы `/`, надо сделать так:

```
# semanage fcontext -a -e / /var/jail/
# restorecon -RvF /var/jail
```

Несмотря на то что параметры безопасности файлов теперь эквиваленты оригинальным, может не получиться обращение к домашним каталогам пользователей. Поэтому понадобится создать отдельные правила для конкретных пользователей. Если все пользователи Linux сопоставлены с одним и тем же SELinux-пользователем, тогда все дело лишь во включении следующих правил:

```
# semanage fcontext -a -t user_home_dir_t -f d /var/jail/home/*
# semanage fcontext -a -t user_home_t /var/jail/home/*/*
# restorecon -RvF /var/jail/home
```

Только изменение (*chroot*) корневого каталога на каталог `jail` (<замкнутое пространство>) не является достаточным действием, т. к. пустой каталог,

предназначенный для работы через командно-строчный интерфейс, хорошо бы наполнить какими-нибудь еще необходимыми компонентами и структурой. Каталоги, предназначенные для ограничения доступного файлового пространства системы, могут быть обеспечены необходимым окружением при помощи таких инструментов, как `debootstrap` или `jailkit`.

Например, для создания замкнутой среды с помощью комплекта программ `jailkit` требуется сначала создать область размещения, в которой планируется изолировать пользователя, а затем передать в качестве параметров утилиты `jk_init` некоторое число тех компонент, с которыми будет разрешено пользователю работать. В результате будет сформирована необходимая среда окружения с обычными исполняемыми файлами:

```
# jk_init -v /var/jail netutils basicshell jk_lsh
```

**i** Поддерживаемые программы, составляющие изолированное окружение (все наборы исполняемых файлов, которые `jailkit` может поместить в замкнутую среду), указаны в текстовом файле `/etc/jailkit/jk_init.ini`.

Когда создание изолированной среды в `jail` готово, фоновая программа `SSH`, обрабатывающая запросы пользователей, может быть перенастроена на работу с ним следующим образом:

```
# cat /etc/ssh/user/sshd_config
```

```
...
Match group sshusers
  ChrootDirectory /var/jail/
  AllowTcpForwarding no
```

Другой подход предполагает немедленную изоляцию (*jail*) пользователей всей системы (а не только при применении `SSH`). Однако это означает, что любое иное взаимодействие с системой будет в результате использовать домашние каталоги изолированного пространства, либо все входы в систему будут перенаправляться в изолированную среду. А так как нам могут потребоваться различные режимы работы, основанные на том, к какой из двух служб `SSH` подключился пользователь, то это не то решение, которое бы хотелось увидеть в этом случае.

#### 10.2.4. Предоставление параметров безопасности пользователю в зависимости от способа доступа

Этот подход рекомендуется тем администраторам, которые имеют другую учетную запись для выполнения задач по тестированию функциональных возможностей запускаемых ими служб. Такие учетные записи, предназначенные для тестирования, позволяют им проверять, что служба работает правильно и для пользователя, как и должна, несмотря на отсутствие привилегий администраторской учетной записи.

Но использование тестовых учетных записей не всегда возможно, или ситуация бывает такой, что одному и тому же пользователю требуется подклю-

чаться к двум службам (например, к службе SSH, предназначенной для подключения администраторов, и к службе SSH, к которой подключаются другие пользователи). Вместе с SELinux мы можем спокойно выполнять сопоставление различных параметров безопасности в зависимости от ситуации, в рамках которой предоставляется доступ.

В частности, в ситуации, когда подключается обычный пользователь, используется стандартный процесс `sshd` со службой обеспечения безопасности с помощью подключаемых модулей аутентификации (PAM). В то же время для взаимодействия с администраторами применяется процесс `adminsshd` с подключаемыми модулями аутентификации. После того как с ними возникнет определенность, можно заняться настройками (в т. ч. учетных записей относительно служб), которые обсуждались в главе 3 «Управление учетными записями пользователей», для того чтобы различать сопоставления с параметрами защиты SELinux.

Во-первых, надо настроить фоновую программу SSH, используемую для администрирования, так чтобы она использовала в качестве имени службы `adminsshd`. За счет этого служба будет настраиваться в конфигурационном файле `/etc/pam.d/adminsshd` для работы с подключаемыми модулями аутентификации (PAM), а не в стандартном для службы SSH файле `/etc/pam.d/sshd`. Такое разделение позволяет администраторам еще больше обеспечить защиту службы на уровне подключения модулей аутентификации (PAM).

Чтобы использовать такое имя службы, по сути надо, чтобы фоновый процесс SSH был запущен из исполняемого файла `adminsshd` (а не из файла `sshd`, который установлен по умолчанию). Для начала создадим символическую ссылку, которая позволит этого достичь:

```
# ln -s /usr/sbin/sshd /usr/local/sbin/adminsshd
```

Затем надо обновить модульный файл в подсистеме инициализации `systemd` или инициализирующий скрипт, для того чтобы указать новый исполняемый файл (точнее, символическую ссылку) как исполняемый процесс:

```
# cat /etc/systemd/system/adminsshd.service
[Unit]
Description=OpenSSH Server Daemon for Administrators
After=syslog.target network.target auditd.service

[Service]
ExecStart=/usr/local/sbin/adminsshd
ExecReload=/bin/kill -HUP $MAINPID
SELinuxContext=system_u:system_r:sshd_t:s0-s0:c0.c1023

[Install]
WantedBy=multi-user.target
```

Далее надо отредактировать или создать файл для сопоставления созданной службы с пользователем в SELinux. Например, для Linux-пользователя `alice` потребуется сделать следующее:

```
# cat /etc/SELinux/targeted/logins/alice
adminsshd: staff_u: s0-s0:c0.c1023
```

При этом исходно для пользователя `alice` выполняется сопоставление с SELinux-пользователем `user_u`:

```
# semanage login -l
Login Name      SELinux User  MLS/MCS Range  Service
%users         user_u       s0-s0:c0.c99   *
__default__    user_u       s0-s0:c0.c9     *
root           unconfined_u s0-s0:c0.c1023 *
system_u       system_u     s0-s0:c0.c1023 *

Local customizations in /etc/SELinux/targeted/logins
alice          staff_u      s0-s0:c0.c1023 adminsshd
```

**P** В русскоязычном варианте этот вывод команды предполагает следующий текст:

```
# semanage login -l
Имя пользователя  SELinux-пользователь  Диапазоны для уровней доступа  Службы
и категорий информации
%users           user_u                 s0-s0:c0.c99                    *
__default__      user_u                 s0-s0:c0.c9                      *
root             unconfined_u          s0-s0:c0.c1023                  *
system_u         system_u               s0-s0:c0.c1023                  *

Локальные настройки в /etc/SELinux/targeted/logins
alice            staff_u                s0-s0:c0.c1023                  adminsshd
```

Эта настройка говорит о том, что когда пользователь `alice` войдет в систему через службу SSH, предназначенную для администрирования, то ему будут предоставлены служебные привилегии, имеющиеся у SELinux-пользователя `staff_u`. Тогда как при доступе через службу SSH, предназначенную для входа обычных пользователей, будут предоставлены права, характерные для `user_u`.

## 10.2.5. Настройка правил для SSH

Существует несколько логических параметров SELinux, связанных с SSH, которые хорошо управляют функциональными возможностями пользователей, получаемыми за счет взаимодействия с фоновой программой SSH. Учитывая, что и служба, предназначенная для подключения обычных пользователей, и служба, запущенная для администрирования, работают с одним и тем же типом допустимого набора функций `sshd_t`, логические параметры SELinux одинаково применяются и к одной, и ко второй службе.

Если это нежелательно, тогда, может быть, и необходимо создать отдельный SELinux-тип для одной или даже для каждой из фоновых программ SSH. Мы полагаем, что в данном случае это не нужно, так как создание отдельных типов для SSH – весьма трудоемкое занятие само по себе.

Логический параметр `ssh_chroot_rw_homedirs` (<чтение/запись\_домашнего\_каталога\_при\_смене\_корневого\_каталога\_через\_ssh>) не применим, когда используется стандартное взаимодействие по SSH для изменения корневого ка-

талога. Однако если защищенный протокол передачи файлов (SFTP – Secure File Transport Protocol) использует возможность смены корневого каталога, имеющуюся у фоновой программы SSH, тогда пользователи, подключившиеся к такой системе, будут работать уже с другим типом допустимого набора функций – `chroot_user_t`. В этом случае логический параметр `ssh_chroot_gw_homedirs` позволяет таким пользователям читать из измененного командой `chroot` домашнего каталога и производить запись в него.

Таким же образом параметр `ssh_chroot_full_access` (<полный\_доступ\_при\_смене\_корневого\_каталога\_через\_ssh>) переключается, когда этим самым пользователям измененного корневого каталога (*chrooted users*) (работающим с типом `chroot_user_t`) требуется получить доступ к разным видам файлов, даже вне их исходного домашнего каталога (или в их же домашнем каталоге, но имеющем другой тип допустимого набора функций).

Если вместо этого потребуется доступ к информационному содержанию веб-сервера (к такому как `httpd_sys_content_t`, описанному в предыдущем подразделе), то тогда полный доступ – это слишком много. Вместо него может быть использован логический параметр `ssh_chroot_manage_apache_content` (<управление\_информационным\_содержанием\_в\_apache\_при\_смене\_корневого\_каталога\_через\_ssh>).

Для того чтобы позволить пользователям с типом `sysadm_t` входить в систему, должен быть включен логический параметр `ssh_sysadm_login`. Стоит заметить, что в конфигурации, описанной ранее, мы соотносили администраторские учетные записи пользователей (таких как `alice`) с SELinux-пользователем `staff_u`. В результате этим пользователям назначена служебная роль `staff_r` и служебный тип допустимого набора функций `staff_t`. И поэтому они могут в дальнейшем использовать такие команды, как `newrole` и `sudo`, для переключения к роли вроде `sysadm_r`, позволяющей получить больше администраторских прав. Поэтому данный логический параметр SELinux не должен быть включен для нашего варианта использования.

## 10.2.6. Включение многопользовательского режима использования

Наконец, если сервер, предоставляющий доступ через командно-строчный пользовательский интерфейс, является общим для нескольких групп пользователей, то можно включить многопользовательский режим работы на этом уровне.

В ранее рассмотренных командах мы выполняли сопоставление пользователей с мандатной конфигурацией `s0-s0:c0.c99`. Мы могли бы создать более детализированный набор мандатных конфигураций, наподобие того, как до этого было выполнено разделение по экземплярам веб-сервера:

```
# cat /etc/SELinux/targeted/setrans.conf
s0-s0:c1=Customer1
s0-s0:c2=Customer2
...
```

Пользователи могут затем быть объединены в соответствующие группы:

```
# getent group customer1
customer1:x:160348:andreas,bob,chelsea,dana
```

Благодаря поддержке программой SSH подключаемых модулей аутентификации (PAM) все, что нам необходимо сделать, – это настроить имена (*logins*) для этих групп. Модуль  `pam_SELinux`, который вызывается службой подключаемых модулей аутентификации для `sshd`, сам делает все остальное:

```
# semanage login -l
Login Name  SELinux User  MLS/MCS Range  Service
%customer1  user_u      s0-s0:c1      *
%customer2  user_u      s0-s0:c2      *
__default__ user_u      s0-s0:c0.c9   *
root        unconfined_u s0-s0:c0.c1023 *
system_u    system_u     s0-s0:c0.c1023 *
```

## 10.3. ОБЩИЙ ДОСТУП К ФАЙЛАМ ЧЕРЕЗ СЕТЕВУЮ ФАЙЛОВУЮ СИСТЕМУ NFS

Когда системам требуется предоставлять общий доступ к одному и тому же набору структурированных данных, то они обычно используют базы данных. Если данные не структурированы, то используется файловый сервер. Одна из наиболее популярных возможностей предоставления общего доступа к данным в Linux – применение службы сетевой файловой системы NFS (*Network File System*).

Однако по умолчанию NFS не способна обрабатывать расширенные атрибуты (необходимые для отслеживания параметров безопасности SELinux). Может быть использовано несколько возможных подходов, для того чтобы без особых сложностей включить сетевую файловую систему с поддержкой SELinux.

### 10.3.1. Базовая настройка службы NFS

Начать стоит с базовой настройки сетевой службы файловой системы для размещения тех данных, к которым планируется предоставить доступ. Например, мы хотим хранить файлы в каталоге `/export`, который, в свою очередь, содержит два каталога с названиями `instance1` и `instance2`. Эти два каталога затем будут связаны с соответствующими веб-серверами, размещенными на сервере.

В основном файле конфигурации NFS `/etc/exports`, который хранит в себе настройки экспортируемых каталогов, укажите абсолютный путь к иерархии каталогов (экспортируемой через сетевую файловую систему) и перечень клиентов (некая разновидность довольно грубого списка контроля доступа) с параметрами подключения:

```
# cat /etc/exports
/export 192.168.1.0/255.255.255.0(ro,sync)
```

Теперь можно запустить службы сетевой файловой системы и проверить, что иерархия каталогов действительно предоставляется для подключения удаленным клиентам:

```
# systemctl start nfs# exportfs  
/export 192.168.1.0/255.255.255.0
```

### 10.3.2. Включение поддержки NFS на стороне защищенного клиента

Первый и наиболее популярный способ работы с подключениями к NFS в системах, защищаемых SELinux, заключается в том, чтобы все службы, которые должны иметь дело с файлами, доступными через NFS, имели возможность обрабатывать объекты с типом `nfs_t`. Этот тип допустимого набора функций по умолчанию назначается всем подключениям к сетевой файловой системе. Это достигается при помощи логических параметров SELinux, которые надо включить на клиентских системах, а не на самом сервере NFS.

Для большинства служб такими логическими параметрами являются параметры вида `*_use_nfs` (<\*\_использует\_nfs>). Например, `cobbler_use_nfs` позволяет программному обеспечению Cobbler (это такой сервер сетевой установки Linux- и Windows-систем, позволяющий создать среду бездисковой установки, сконфигурировать и развернуть ОС) использовать файлы, размещенные в NFS. Таким же образом параметр `ftpd_use_nfs` позволяет FTP-серверам использовать подключенные сетевые файловые системы и управлять ими с целью обеспечения пользователей необходимыми возможностями.

Особенно следует упомянуть логический параметр `httpd_use_nfs`. Он позволяет веб-серверам (точнее, их типам) использовать экспортированные файловые системы NFS для хранения информации, содержащейся на веб-сайтах. Если такой NFS-сервер будет применяться в системе веб-серверами, которые рассматривались в этой главе раньше, то правильнее будет включить данный логический параметр.

Специальным логическим параметром является `use_nfs_home_dirs`. Когда этот параметр включен, то службы, которые работают с домашними каталогами пользователей, имеют право взаимодействовать с этими каталогами, размещенными в общем доступе при помощи NFS. В данном случае акцент делается на целевом объекте (домашних каталогах), а не на службах.

### 10.3.3. Настройка правил безопасности для NFS на сервере

Сам по себе сервер NFS тоже управляется набором правил SELinux. Существует три основных логических параметра SELinux, относящихся к сетевой файловой системе и применимых на стороне сервера NFS:

- параметр `nfsd_anon_write` (<службе\_NFS\_разрешена\_анонимная\_запись>) позволяет серверу NFS вносить изменения в файлы, имеющие тип допустимого набора функций `public_content_rw_t`. Он похож на логический



параметр `httpd_anon_write`, упомянутый ранее в этой главе, который применяется для работы с ресурсами, обрабатываемыми несколькими не связанными между собой службами;

- параметр `nfs_export_all_ro` (<сетевая\_файловая\_система\_экспортирует\_все\_только\_для\_чтения>) гарантирует, что NFS-сервер может обслуживать размещенные данные в режиме «только чтение». Если существует какая-нибудь уязвимость в сервере NFS, позволяющая добиться-таки записи данных, или же ошибочная конфигурация будет позволять выполнять подключения с правом на запись, тогда установка этого параметра приведет к тому, что NFS-сервер не сможет выполнять запись в экспортируемые файлы и каталоги;
- параметр `nfs_export_all_rw` (<сетевая\_файловая\_система\_экспортирует\_все\_для\_чтения\_и\_записи>) позволяет серверу NFS предоставлять в общий доступ файлы с правами на чтение и запись независимо от того, какие параметры безопасности SELinux им присвоены.

Например, для незащищаемой информации, размещенной на веб-сайте, нужно включить режим чтения/записи, поскольку динамическим сайтам может потребоваться записывать открытые данные в файловую систему:

```
# setsebool -P nfs_export_all_rw on
```

### 10.3.4. Подключение общих сетевых ресурсов с различными параметрами безопасности

В то время как области подключения сетевой файловой системы NFS имеют тип допустимого набора функций `nfs_t`, администраторы могут выбрать подключение общих сетевых ресурсов с различными параметрам безопасности. Эти параметры являются характерными для тех систем, на которых выполняется подключение сетевой файловой системы, и никак не отражаются на самих серверах, их предоставляющих:

```
# mount nfsserver:/export/web /srv/web/instance1/htdocs \
-o context="system_u:object_r:httpd_sys_content_t:s0"
```

Впрочем, если попытаться подключить на той же самой системе, но в другой каталог еще один ресурс, размещенный на том же самом сервере NFS, то возникает ошибка, сообщающая о том, что подключение закончилось неудачно из-за действий, выполняемых с одним и тем же суперблоком, который не может иметь различные настройки безопасности:

```
# mount nfsserver:/export/wiki /srv/web/instance3/htdocs/wiki \
-o context="system_u:object_r:httpd_sys_rw_content_t:s0"
kernel: SELinux: mount invalid. Same superblock,
different security settings for (dev 0:17, type nfs)
```

Это происходит из-за того, что метаданные сохраняются в специальной области памяти и препятствуют использованию разных параметров безопасности при подключении в том числе общих сетевых ресурсов. К счастью, такое

поведение может быть изменено с помощью параметра `nosharecache`, который надо указать при подключении:

```
# mount nfsserver:/export/web /srv/web/instance1/htdocs \
-o nosharecache,context="system_u:object_r:httpd_sys_content_t:s0"
# mount nfsserver:/export/wiki /srv/web/instance3/htdocs/wiki -o \
nosharecache,context="system_u:object_r:httpd_sys_rw_content_t:s0"
```

### 10.3.5. Работа с промаркированной сетевой файловой системой

Версии серверов NFS начиная с 4.2 поддерживают маркировку параметрами безопасности. Информация о параметрах безопасности SELinux сохраняется и обрабатывается NFS-серверами. Если маркировка включена на NFS-сервере, то пользователи могут применять подключение этих общих сетевых ресурсов так, будто бы они имеют дело с локальной файловой системой, поддерживающей полный набор расширенных атрибутов.

Использование маркировки требует, чтобы и NFS-сервер, и все системы клиентов, которые подключают файловые системы, находящиеся на NFS-сервере, имели одну и ту же политику SELinux. Когда используется маркированная сетевая файловая система, параметры безопасности файлов, размещенных на NFS-сервере, являются открытыми, и NFS-сервер будет передавать запросы на перемаркировку в локальное ядро системы.

Для того чтобы включить маркированную NFS, удостоверьтесь, что служба NFS запущена с параметром `-V 4.2`. Например, в операционной системе Red Hat Enterprise Linux это делается путем обновления файла `/etc/sysconfig/nfs`:

```
# cat /etc/sysconfig/nfs
# Optional arguments passed to rpc.nfsd. See rpc.nfsd(8)
# Дополнительные аргументы, передаваемые в процесс NFS, запускаемый на сервере.
# См. руководство по rpc.nfsd (8)
RPCNFSDARGS="-V 4.2"
```

В операционной системе Gentoo это выполняется в файле `/etc/conf.d/nfs`:

```
# cat /etc/conf.d/nfs
# Start with 8 threads, and disable version 2, 3, 4 and 4.1
# Запускается с 8 потоками и отключается в версиях 2, 3, 4 и 4.1
OPTS_RPC_NFSD="8 -N 2 -N 3 -N 4 -N 4.1 -V 4.2"
```

После настройки конфигурационного файла следует перезапустить службу NFS:

```
# systemctl restart nfs
```

А на стороне клиентов удостовериться, что при подключении будет использоваться версия NFS v4.2:

```
# mount -o v4.2 nfsserver:/export/web/srv/web/instance1/htdocs
```

На самом же NFS-сервере должны быть назначены правильные параметры безопасности для экспортируемой иерархии каталогов.

### 10.3.6. Сравнение файлового сервера Samba с сетевой файловой системой NFS

Другая популярная служба для обмена файлами называется Samba. Она является свободной повторной реализацией сетевого протокола **SMB/CIFS (Server Message Block / Common Internet File System)** – блок серверных сообщений / общий протокол доступа к интернет-файлам). Она позиционируется аналогично сетевой файловой системе NFS, хотя администрирование Samba немного отличается от NFS.

С точки зрения системы защиты SELinux у NFS и Samba имеется много общего.

Например, многие логические параметры SELinux, характерные для NFS, существуют и для Samba:

- параметр `allow_smbd_anon_write` (<разрешить службе Samba анонимную запись>) позволяет фоновой программе Samba записывать данные в объекты, имеющие тип допустимого набора функций `public_content_rw_t`;
- параметр `samba_create_home_dirs` (<создание службой Samba домашних каталогов>) позволяет фоновой программе Samba создавать новые домашние каталоги пользователей. Это может быть включено с помощью подключаемых модулей аутентификации (PAM) к службе Samba;
- параметр `samba_enable_home_dirs` (<служба Samba включает домашние каталоги>) позволяет службе Samba предоставлять в общий доступ домашние каталоги пользователей;
- параметры `samba_export_all_ro` (<служба Samba экспортирует все только для чтения>) и `samba_export_all_rw` (<служба Samba экспортирует все для чтения и записи>) действуют схоже с логическими параметрами `nfs_export_all_ro` и `nfs_export_all_rw`. Если они включены, то это позволяет службе Samba предоставлять пользователю (экспортировать) любой файл или каталог (независимо от его типа допустимого набора функций) либо в режиме только для чтения, либо в режиме чтения/записи;
- параметр `samba_share_nfs` (<служба Samba имеет общий доступ к сетевой файловой системе>) позволяет службе Samba получать доступ к предоставленным в общий доступ ресурсам через сетевую файловую систему NFS. Лучше бы, конечно, было обозначить этот логический параметр чуть иначе – `samba_use_nfs` (<служба Samba использует сетевую файловую службу>), но, к сожалению, это (ненаписанное) соглашение не было в свое время принято разработчиками политик SELinux.

Открытые ресурсы, предоставленные в общий доступ с помощью службы Samba, имеют тип допустимого набора функций `cifs_t`, подобно аналогичным ресурсам, предоставляемым пользователям через сетевую файловую систему, обозначаемым `nfs_t`. И для того чтобы позволить приложениям получить доступ к ресурсам, промаркированным типом `cifs_t`, существуют логические параметры SELinux, которые обычно используют в названии синтаксис `*_use_samba` (<\*\_использует\_Samba>).

Например, логический параметр `virt_use_samba` (<виртуализация\_использует\_Samba>) позволяет гостевым системам виртуальной инфраструктуры использовать данные (образы), размещенные в общем доступе службы Samba. Точно так же параметр `sanlock_use_samba` (<sanlock\_использует\_samba>) позволяет менеджеру совместного хранения данных `sanlock` управлять общедоступными ресурсами, представляемыми службой Samba.

И все-таки имеется существенное различие между NFS и Samba. Общедоступные ресурсы, предоставляемые службой Samba, необходимо маркировать типом `samba_share_t` непосредственно на самом сервере Samba. В то время как для сетевой файловой системы NFS такая специальная маркировка не является обязательным требованием.

## 10.4. ЗАКЛЮЧЕНИЕ

В этой главе мы рассмотрели ряд случаев использования SELinux и настроили систему для использования возможностей SELinux с целью улучшения безопасности служб.

В отношении веб-сервера мы провели полную настройку для управления различными профилями рисков веб-сайтов в нескольких системах, настраивая SELinux для каждой из этих систем. Мы увидели, как могут запускаться множество экземпляров веб-сайтов и каждый экземпляр со своей собственной категорией защищаемой информации. И как информационное наполнение сайтов может быть управляемо в защищенной манере. Мы также посмотрели, как разделять администраторские роли для одной и той же системы, и закончили рассмотрением сетевых настроек.

После этого было рассмотрено, как сервер, предоставляющий доступ пользователям через защищенный командно-строчный интерфейс (SSH), может повысить свою безопасность с помощью разделения серверных фоновых программ SSH по назначению. И как можно их запустить с различными категориями защищаемой информации. Кроме того, была рассмотрена детальная настройка файловой системы для доступа с изменением корневого каталога файловой системы. Даже был учтен специальный вход в систему, при котором пользователь получал различные параметры безопасности SELinux, основанные на том, через какой именно экземпляр службы SSH было выполнено подключение.

В конце были рассмотрены сервер сетевой файловой службы NFS и различные параметры ее настройки (определяемые через логические параметры SELinux). Отмечены параметры подключения, которые влияют на параметры безопасности совместно используемых ресурсов. Кроме того, в этой главе была рассмотрена возможность использования NFS с поддержкой расширенных атрибутов и то, как это может использоваться для применения параметров безопасности SELinux. Завершилось все небольшим сравнением реализаций защиты для службы обмена файлами Samba и сетевой файловой системой NFS.

# Предметный указатель

## **P**

policy store, 45

## **S**

SELinux

многокатегорийная защита, 40

многоуровневая защита, 39

Multi-category security (MCS), 40

multilevel security (MLS), 39

## **T**

transient services, 200

## **U**

unit file, socket, 204

unit files, 198

## **Б**

Библиотеки системы защиты

libselinux, 30

libsemanage, 30

linsepol, 50

pam\_namespace, 106

pam\_selinux, 105

pam\_sepermit, 106

## **Ж**

Журналы регистрации /var/log/avc.  
log, 67

Журналы событий /var/log/audit/  
audit.log, 62

## **И**

Инструмент системы защиты  
sepolicy, 267

Инструменты системы защиты  
apol, 254

audit2allow, 30

audit2why, 30, 81

autorelabel, 129

chcat, 98

chcon, 117

findcon, 123

fixfiles, 128

getpidcon, 130

matchpathcon, 121

newrole, 100

restorecon, 30

restorecond, 113

rlpkg, 128

runcon, 102

seapplet, 78

sediff, 266

sedispatch, 63, 78

sedta, 259

seinfoflow, 265

selinuxexeccon, 134

semanage, 30

semodule, 30

sepolgen, 237

setenforce, 30

setfiles, 128

setools, 266

setroubleshootd, 62

программой поиска  
неисправностей, 62

## **К**

Команды

cat, 24

checkout, 243

chmod, 24

- chroot, 282
  - diff, 266
  - ldd, 61
  - mount, 119
  - netlabelctl, 159
  - pscap, 192
  - readelf, 61
  - su, 38
  - sudo, 22
  - tar, 119
  - Команды системы защиты**
    - getenforce, 54
    - getfacl, 26, 27
    - getfattr, 111
    - getseuser, 100
    - id -Z, 32
    - load\_policy, 45
    - seinfo, 38
    - seinfo -constrain, 143
    - semanage boolean, 57
    - semanage fcontext, 121, 122
    - semanage login, 92
    - semanage user -l, 94
    - semodule, 44
    - sepolicy, 151
    - sepolicy communicate, 267
    - sepolicy generate, 241
    - sepolicy transition, 268
    - sestatus, 54
    - setenforce, 56
    - setfacl, 26
    - setsebool, 56
  - Конфигурационные файлы**
    - /etc/audit/plugins.d/syslog.conf, 65
    - /etc/audit/auditd.conf, 65
    - /etc/dbus-1/system.d/, 215
    - /etc/fstab, 27
    - /etc/jailkit/jk\_init.ini, 283
    - /etc/security/namespace.conf, 107
    - /etc/SELinux/config, 54
    - /etc/selinux/semanage.conf, 50
    - /etc/selinux/strict/contexts, 125
    - /etc/selinux/targeted/contexts/
      - /etc/selinux/targeted/contexts/customizable\_types, 246
      - /etc/SELinux/targeted/seusers, 93
      - /etc/setroubleshoot/setroubleshoot.conf, 81
      - /etc/shadow, 22, 24, 28
      - /etc/ssh/user/sshd\_config, 281
      - /sys/fs/SELinux/avc/cache\_threshold, 63
      - /sys/fs/SELinux/enforce, 54, 56
      - /sys/fs/selinux/mls, 46
      - /sys/fs/selinux/policy, 249
      - /usr/share/selinux, 44
    - current, 35
    - exec, 35
    - file\_contexts.local, 124
    - fscreate, 35
    - keycreate, 35
    - prev, 35
    - sepermit.conf, 106
    - setrans.conf, 96
    - sockcreate, 35
    - хранилище политик /targeted, 44
- М**
- Методы посреднической политики  
domain\_use\_interactive\_fds(), 228
- Модели защиты
  - модель Белла-ЛаПадула, 33, 39
  - модель наименьших прав, 28
  - least-privilege model, 28
- Модульные файлы, 198
- Модульный файл,  
сокет-активация, 204
- П**
- Параметры безопасности
  - extended attribute, 110
  - libvirt-домен, 185
  - SELinux role, 33
  - SELinux type, 33
  - SELinux user, 33
  - sensitivity level, 33
  - домен, 33, 59

- имя пользователя, 33
- мандатная конфигурация, 33
- расширенный атрибут, 110
- роль, 33
- тип допустимого набора функций, 33
- Переходные службы, 200
- Политики безопасности
  - allow rules, 251
  - booleans, 56
  - constraints, 142
  - context definitions, 232
  - customizable types, 125
  - domain transition, 131
  - named file transitions, 115
  - policy modules, 41
  - policy package, 41
  - policy rules, 41
  - policy store, 41
  - process transition, 131
  - transition rules, 131
  - type attributes, 140
  - type definition, 245
  - type transition rules, 113
  - unconfined, 29
  - атрибуты типа, 140
  - заказные типы, 125
  - именованные файловые преобразования, 115
  - логические параметры, 56
  - модули политик, 41
  - наложенные ограничения, 142
  - неограниченные, 29
  - описание параметров безопасности, 232
  - описание типа, 245
  - пакет политики, 41
  - правила политик, 41
  - правила преобразования типов, 113, 131
  - преобразование домена, 131
  - преобразование типа процесса, 131
  - разрешающие правила, 251
  - хранилище политик, 41
- Политики SELinux
  - custom policies, 227
  - reference policy, 227
  - выполненные по специальным требованиям политики, 227
  - посредническая политика, 227
- Пользователи системы
  - lisa, 26
  - root, 22, 24
- Пользователи SELinux, 38
- Привилегии и ограничения
  - allow, 46
  - auditallow, 64, 245
  - block\_suspend, 46
  - connectto, 149
  - deny, 46
  - dontaudit, 64
  - dyntransition, 134
  - forward\_in, 155
  - forward\_out, 155
  - polmatch, 164
  - recv, 159
  - recvfrom, 164
  - reject, 46
  - send\_msg, 216, 217
  - sendto, 149
  - setcontext, 164
  - virt\_sandbox\_use\_\*, 193
- Программные компоненты
  - auditd, 62
  - Database management system, 22
  - DBMS, 22
  - система управления базой данных, 22
- Программные пакеты Gentoo
  - app-admin/setools, 38
  - sys-apps/attr, 111
  - sys-apps/policycoreutils, 57, 100, 151
  - sys-libs/libcap-ng, 192
  - sys-libs/libselinux, 56
  - sys-libs/libSELinux, 54
  - sys-process/audit, 66

Программные пакеты Red Hat  
attr, 111  
audispd-plugins, 66  
libcap-ng-utils, 192  
libselinux-utils, 56  
libSELinux-utils, 54  
policycoreutils, 44, 45, 56  
policycoreutils-devel, 151  
policycoreutils-newrole, 100  
policycoreutils-python, 57  
selinux-policy-targeted, 45  
setools-console, 38, 123  
Программный пакет Red Hat  
setroubleshoot-server, 63

## Р

Режимы защиты  
disable, 54  
enforcement, 40, 54  
permissive, 54  
отключенный режим, 54  
принудительный режим, 40  
рекомендательный режим, 54

Роли в системе защиты

guest\_r, 38  
secadm\_r, 37  
staff\_r, 37  
sysadm\_r, 37  
system\_r, 37  
unconfined\_r, 37  
user\_r, 37  
xguest\_r, 38

## С

Система инициализации systemd  
socket-base activation, 204  
активизация служб через сокет, 204  
условные параметры, 201  
Системные компоненты  
audisp, 62  
audisp-remote, 65  
audit subsystem, 62  
aureport, 66  
ausearch, 68, 76  
daemons, 22

D-Bus, 197  
dbus-daemon, 36  
gid, 24  
Git, 278  
httpd, 32  
jk\_init, 283  
libvirt, 181  
Linux Security Modules, 24  
LSM, 24, 25, 29  
netlabel\_tools, 159  
netlink-сокеты, 149  
Nginx, 271  
raccoon, 165  
run\_init, 102  
setuid, 135  
SSH, 280  
systemd, 197  
systemd-journald, 206  
systemd-udev, 210  
uid, 24  
анонимные каналы, 147  
виртуализация, 173  
виртуальная файловая система, 147  
дейтаграммные сокеты, 149  
идентификатор группы, 24  
идентификатор пользователя, 24  
командно-строчный интерфейс  
пользователя, 279  
межпроцессного  
взаимодействия, 212  
механизмы взаимного  
исключения, 145  
модули безопасности Linux, 24  
неименованные каналы, 147  
подсистема контроля событий, 62  
поточные сокеты, 149  
сетевая файловая система, 287  
служба Samba, 291  
сокеты, 148  
сокеты домена UNIX, 148  
фоновые программы, 22  
Системный компонент  
Docker, 188  
prctl(), 138



setpriv, 138  
 Сообщения системы защиты  
   Permission denied, 24  
   Policy deny\_unknown status, 47  
 доступ запрещен, 24  
 Состояния системы защиты  
   disabled, 54  
   enforcing, 54  
   permissive, 54  
   отключенное, 54  
   принудительное, 54  
   рекомендательное, 54  
 Способы защиты  
   access vector cache, 63  
   ACL, 26  
   AVC, 63  
   capabilities, 28  
   context, 31  
   DAC, 22  
   Discretionary Access Control, 22  
   fallback labeling, 158  
   hash-code, 22  
   information flow analysis, 262  
   Labeled IPsec, 163  
   MAC, 23  
   Mandatory access control, 23  
   MCS, 40  
   MLS, 39  
   Multi-category security, 40  
   multilevel security, 39  
   NetLabel, 74  
   nosuid, 138  
   peer labeling, 158  
   SECMARK, 153  
   security identifier, 135  
   SID, 135  
   UBAC, 38  
   User-based access control, 38  
 авторизации доступа к  
   гипервизору, 176  
 анализ информационных  
   потоков, 261  
 анализ преобразований типов  
   процессов, 257

анализ преобразования типов  
   процессов, 252  
 дискреционный контроль  
   доступа, 22  
 идентификатор безопасности, 135  
 изоляция гостевой системы, 176  
 кеш вектора доступа, 63  
 мандатный контроль доступа, 23  
 метка, 31  
 многокатегорийная защита, 40, 179  
 многоуровневая защита, 39, 46  
 одноранговая маркировка, 158  
 одноступенчатый анализ, 248  
 параметры безопасности, 31  
 пользователь-ориентированный  
   контроль доступа, 38  
 промаркированный IPsec, 163  
 резервная маркировка, 158  
 резервное копирование, 119  
 список контроля доступа, 26  
 хеш-код, 22  
 Стандарты  
   CIPSO, 74  
   Common Criteria at Evaluation  
   Assurance, 180  
   Common Criteria (ISO/IEC 15408), 23  
   POSIX, 26, 27  
   RFC 4949 «Словарь безопасности  
   интернета», 75  
   SysV, 145  
   Trusted Computer System Evaluation  
   Criteria (TCSEC), 23  
 Сторонние системы защиты  
   AppArmor, 31  
   IPsec, 163  
   Libreswan, 167  
   pam\_namespace, 107  
   подключаемые модули  
   аутентификации, 104

## Т

Типы допустимого набора функций  
   docker\_t, 189  
   httpd\_sys\_\*, 273

httpd\_t, 32  
 http\_port\_t, 32  
 invalid\_packet\_t, 244  
 iso9660\_t, 120  
 java\_tmp\_t, 111  
 mozilla\_plugin\_t, 137  
 mysqld\_\*, 112  
 netlabel\_peer\_t, 159  
 postfix\_cleanup\_t, 87  
 rpcbind\_var\_run\_t, 111  
 staff\_dbusd\_t, 36  
 svirt\_t, 177  
 system\_dbusd\_t, 36  
 systemd\_logind\_t, 215  
 user\_home\_t, 111  
 vird\_t, 177

#### Типы событий системы защиты

MAC\_CIPSOV4\_ADD, 75  
 MAC\_CIPSOV4\_DEL, 75  
 MAC\_CONFIG\_CHANGE, 73  
 MAC\_IPSEC\_ADDDSA, 75  
 MAC\_IPSEC\_ADDSPD, 75  
 MAC\_IPSEC\_DELSA, 75  
 MAC\_IPSEC\_DELSPD, 75  
 MAC\_IPSEC\_EVENT, 75  
 MAC\_MAP\_ADD, 75  
 MAC\_MAP\_DEL, 75  
 MAC\_POLICY\_LOAD, 73  
 MAC\_STATUS, 74  
 MAC\_UNLBL\_STCADD, 75  
 MAC\_UNLBL\_STCDEL, 75  
 SELINUX\_ERR, 73  
 USER\_AVC, 72

#### Типы файлов политик

.fc, 230  
 .te, 229  
 file context, 230  
 interface file, 230  
 type enforcement, 230  
 файл интерфейса, 230  
 файл с требованиями типов, 230  
 файлы с параметрами безопасности  
 файлов, 230

## У

### Уязвимости

перехват управления, 29  
 удаленное выполнение команд, 29  
 RCE, 29  
 remote command execution, 29  
 sql-инъекции, 29

## Ф

### Файловые системы

псевдофайловая система, 34  
 xfs, 27

### Файлы политик безопасности

.fc, 43  
 .if, 43  
 .pp, 44  
 .te, 43  
 /etc/SELinux/targeted/modules/  
 active/modules, 44  
 /var/lib/selinux/mcs/active/  
 modules, 44  
 mojomajo.fc, 239  
 pgsql\_admin.te, 233  
 zosremote.if, 239

язык высокого уровня, 44

### Фоновые процессы (daemons)

systemd-journald, 204

### Форматы записи политик

CIL, 42  
 common intermediate language, 42  
 format-a human-readable, 41  
 reference policy style, 42  
 воспринимаемый человеком  
 (стандартный), 41  
 макросов M4, 42  
 обобщенно-промежуточный  
 язык, 42, 226  
 посреднический стиль, 42

### Форматы описания политик

CIL – Common Intermediate  
 Language, 230  
 reference policy style, 230  
 SELinux native, 230

исходный формат, 230  
обобщенно-промежуточный  
язык, 230  
посреднический формат, 230  
Форматы политик безопасности  
high-level language, 44  
HLL, 44

**Х**

Хранилище политик, 45  
mcs, 45  
mls, 45  
strict, 45  
target, 45

Книги издательства «ДМК ПРЕСС»  
можно купить оптом и в розницу  
в книготорговой компании «Галактика»  
(представляет интересы издательств  
«ДМК ПРЕСС», «СОЛОН ПРЕСС», «КТК Галактика»).

Адрес: г. Москва, пр. Андропова, 38;  
тел.: (499) 782-38-89, электронная почта: [books@alians-kniga.ru](mailto:books@alians-kniga.ru).

При оформлении заказа следует указать адрес (полностью),  
по которому должны быть высланы книги;  
фамилию, имя и отчество получателя.

Желательно также указать свой телефон и электронный адрес.

Эти книги вы можете заказать и в интернет-магазине: [www.a-planeta.ru](http://www.a-planeta.ru).

Свен Вермейлен

## **Администрирование системы защиты SELinux**

Главный редактор *Мовчан Д. А.*  
[dmkpress@gmail.com](mailto:dmkpress@gmail.com)  
Перевод *Верецагин В. Л., Севостьянова О. К.*  
Корректоры *Синяева Г. И.*  
Верстка *Чаннова А. А.*  
Дизайн обложки *Мовчан А. Г.*

Формат 70×100 1/16.  
Гарнитура PT Serif. Печать офсетная.  
Усл. печ. л. 24,38. Тираж 200 экз.

Веб-сайт издательства: [www.dmkpress.com](http://www.dmkpress.com)