OWASP Top-10 2017

Dave Wichers

Previous OWASP Top 10 Project Lead (2003 thru 2017)

Former OWASP Board Member (2003 thru 2013)

CoFounder and COO, Aspect Security

which is now EY

OWASP

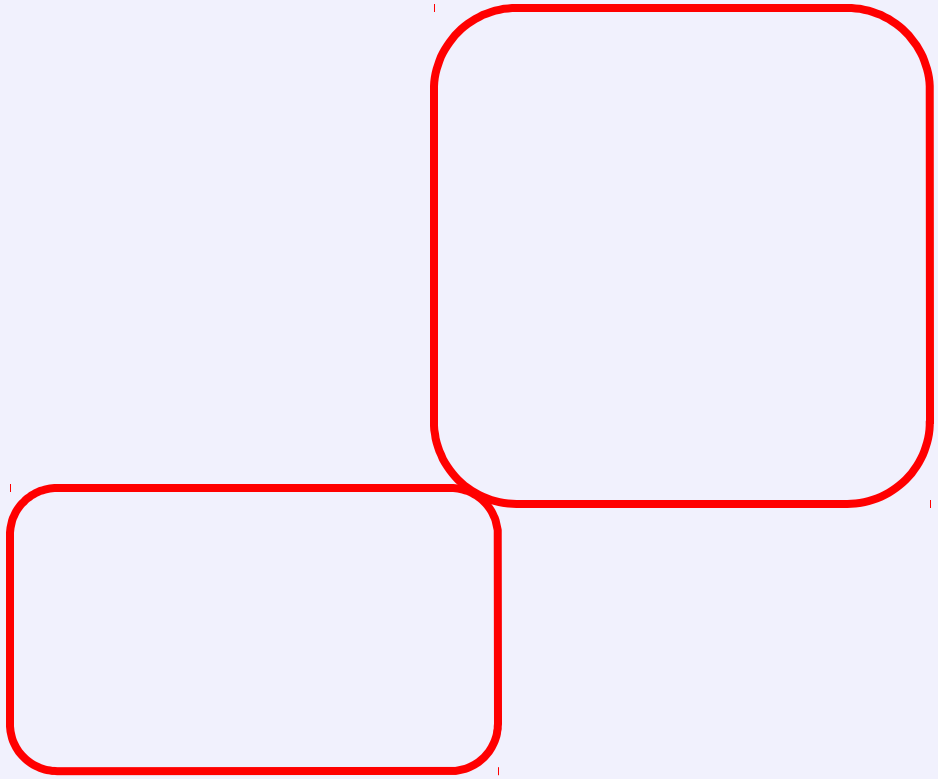
The Open Web Application Security Project

About the OWASP Top 10

OWASP
The Open Web Application Security Project

OWASP Top Ten (2017 Edition)

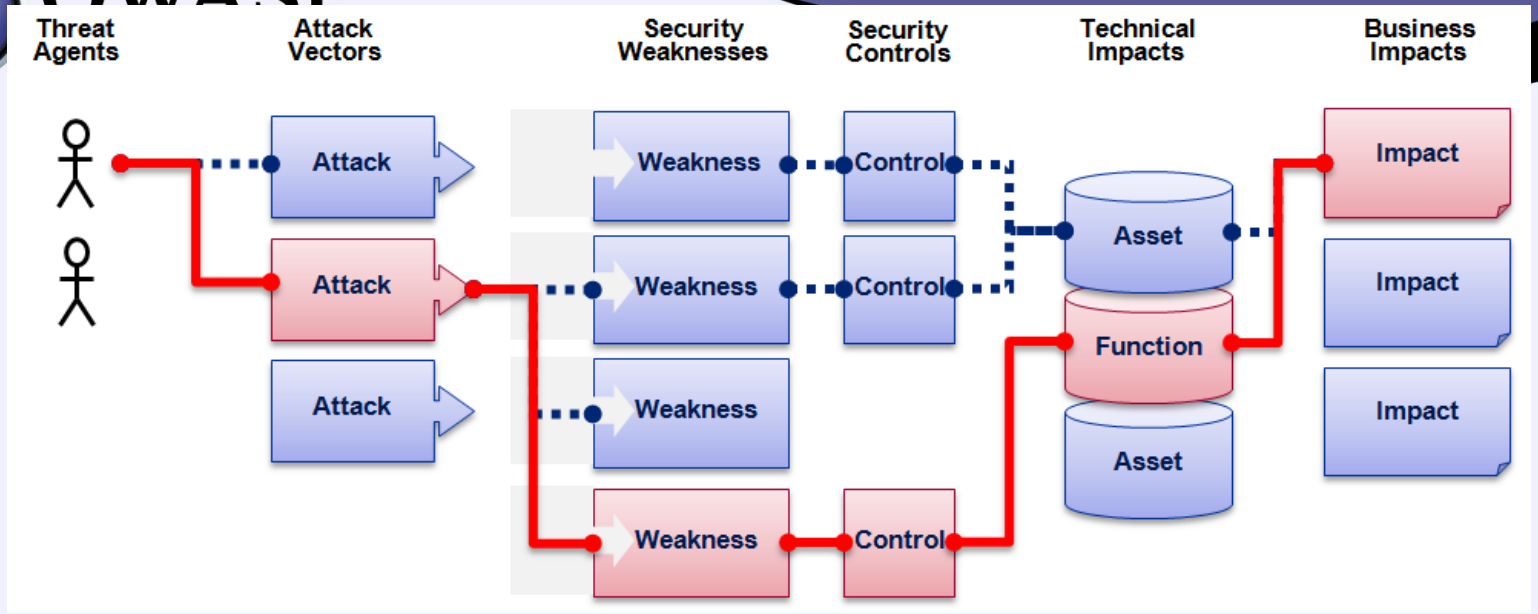
OWASP
The Open Web Application Security Project



What Didn't Change

OWASP
The Open Web Application Security Project

OWASP Top 10 Risk Rating Methodology



Threat Agent	Attack Vector	Weakness Prevalence	Weakness Detectability	Technical Impact	Business Impact
	3 Easy	Widespread	Easy	Severe	
?	2 Average	Common	Average	Moderate	?
	1 Difficult	Uncommon	Difficult	Minor	
	3	2	3	3	
		2.66	*	3	

Injection Example

8.00 weighted risk rating

What's Changed?

OWASP
The Open Web Application Security Project

Mapping from 2013 to 2017 Top 10

OWASP
The Open Web Application Security Project

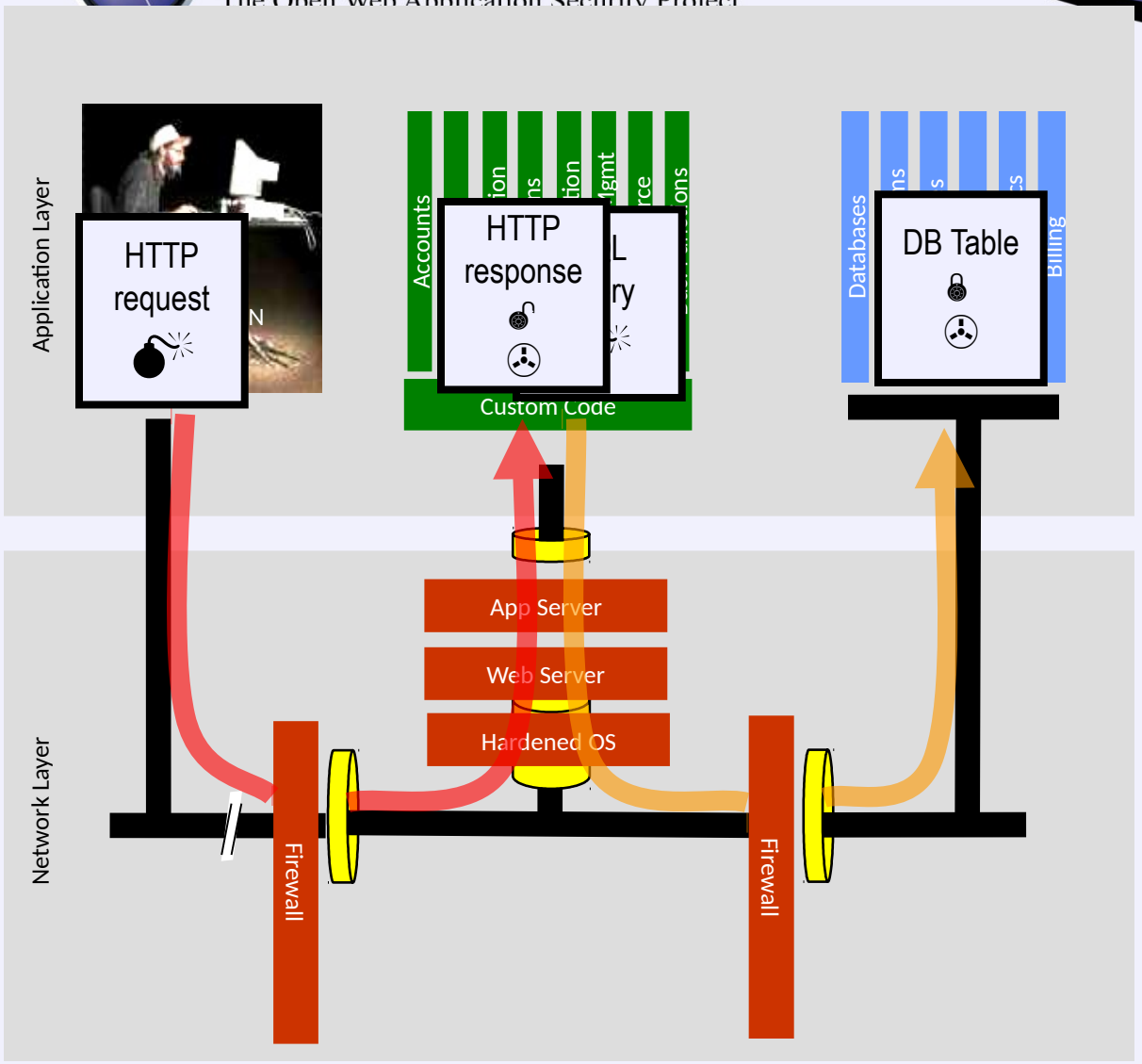
OWASP Top 10 - 2013	→	OWASP Top 10 - 2017
A1 – Injection	→	A1:2017-Injection
A2 – Broken Authentication and Session Management	→	A2:2017-Broken Authentication
A3 – Cross-Site Scripting (XSS)	↘	A3:2017-Sensitive Data Exposure
A4 – Insecure Direct Object References [Merged+A7]	U	A4:2017-XML External Entities (XXE) [NEW]
A5 – Security Misconfiguration	↘	A5:2017-Broken Access Control [Merged]
A6 – Sensitive Data Exposure	↗	A6:2017-Security Misconfiguration
A7 – Missing Function Level Access Contr [Merged+A4]	U	A7:2017-Cross-Site Scripting (XSS)
A8 – Cross-Site Request Forgery (CSRF)	☒	A8:2017-Insecure Deserialization [NEW, Community]
A9 – Using Components with Known Vulnerabilities	→	A9:2017-Using Components with Known Vulnerabilities
A10 – Unvalidated Redirects and Forwards	☒	A10:2017-Insufficient Logging&Monitoring [NEW,Comm.]

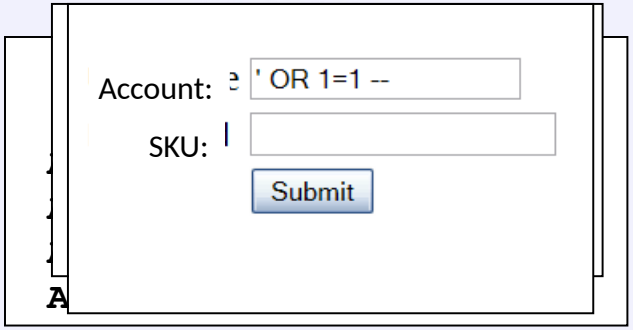
2017-A1 – Injection

OWASP
The Open Web Application Security Project

SQL Injection – Illustrated

OWASP
The Open Web Application Security Project





1. Application presents a form to the attacker
2. Attacker sends an attack in the form data
3. Application forwards attack to the database in a SQL query
4. Database runs query containing attack and sends encrypted results back to application
5. Application decrypts data as normal and sends results to the user

A1 – Avoiding Injection Flaws

OWASP
The Open Web Application Security Project

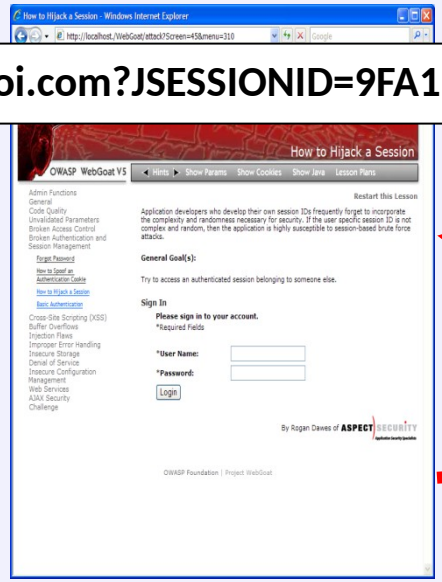
2017-A2 – Broken Authentication

OWASP
The Open Web Application Security Project

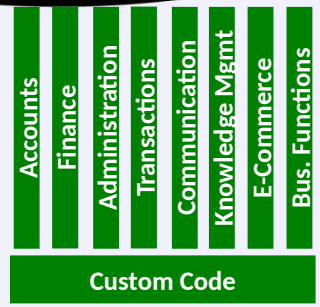
Broken Authentication Illustrated

OWASP
The Open Web Application Security Project

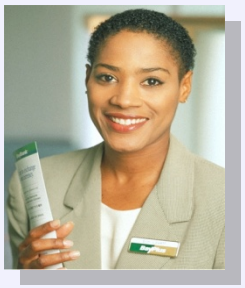
1 User sends credentials



www.boi.com?JSESSIONID=9FA1DB9EA...

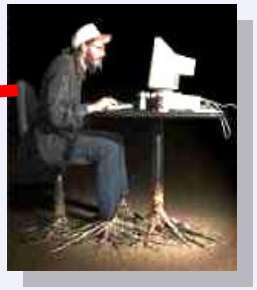


2 Site uses URL rewriting (i.e., put session in URL)



3 User clicks on a link to http://www.hacker.com in a forum

4 Hacker checks referrer logs on www.hacker.com and finds user's JSESSIONID



5 Hacker uses JSESSIONID and takes over victim's account

A2 – Avoiding Broken Authentication

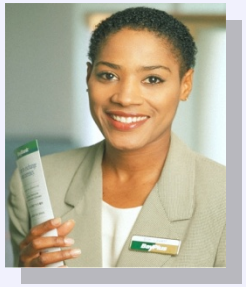
OWASP
The Open Web Application Security Project

2017-A3 – Sensitive Data Exposure

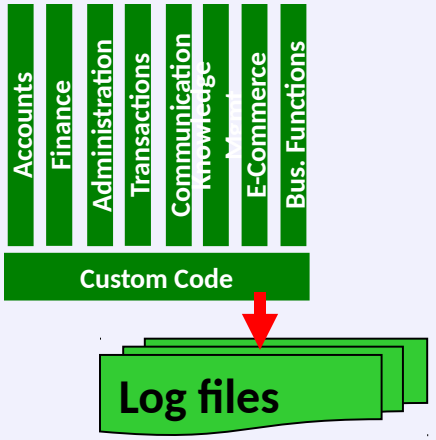
OWASP
The Open Web Application Security Project

Insecure Cryptographic Storage Illustrated

OWASP
The Open Web Application Security Project



1 Victim enters credit card number in form



2 Error handler logs CC details because merchant gateway is unavailable

3 Logs are accessible to all members of IT staff for debugging purposes

4 Malicious insider steals 4 million credit card numbers

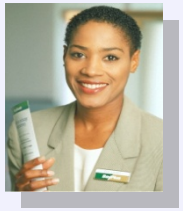
Avoiding Insecure Cryptographic Storage

OWASP
The Open Web Application Security Project

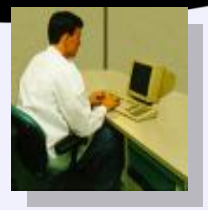
- **Verify your architecture**
  - Identify all sensitive data
  - Identify all the places that data is stored
  - Ensure threat model accounts for possible attacks
  - Use encryption to counter the threats, don't just 'encrypt' the data
- **Protect with appropriate mechanisms**
  - File encryption, database encryption, data element encryption
  - https://www.owasp.org/index.php/Password_Storage_Cheat_Sheet
- **Use the mechanisms correctly**
  - Use standard strong algorithms
  - Generate, distribute, and protect keys properly
  - Be prepared for key change
- **Verify the implementation**
  - A standard strong algorithm is used, and it's the proper algorithm for this situation
  - All keys, certificates, and passwords are properly stored and protected
  - Safe key distribution and an effective plan for key change are in place

Insufficient Transport Layer Protection Illustrated

OWASP
The Open Web Application Security Project



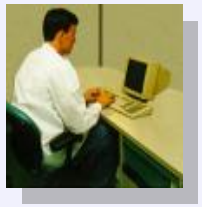
External Victim



Business Partners

Custom Code

Backend Systems



Employees

1

External attacker steals credentials and data off network

External Attacker

2

Internal attacker steals credentials and data from internal network

Internal Attacker

Avoiding Insufficient Transport Layer Protection

OWASP
The Open Web Application Security Project

- **Protect with appropriate mechanisms**
  - Use TLS on all connections with sensitive data
  - Use HSTS (HTTP Strict Transport Security)
  - Use key pinning
  - Individually encrypt messages before transmission
    - E.g., XML-Encryption
  - Sign messages before transmission
    - E.g., XML-Signature
- **Use the mechanisms correctly**
  - Use standard strong algorithms (disable old SSL algorithms)
  - Manage keys/certificates properly
  - Verify SSL certificates before using them
  - Use proven mechanisms when sufficient
    - E.g., SSL vs. XML-Encryption

- https://www.owasp.org/index.php/Transport_Layer_Protection_Cheat_Sheet
- https://www.owasp.org/index.php/HTTP_Strict_Transport_Security_Cheat_Sheet

2017-A4 – XML eXternal Entity (XXE) Attack

OWASP
The Open Web Application Security Project

XXE Attack Examples

OWASP
The Open Web Application Security Project

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE meh [<!ENTITY xxeFun SYSTEM "file:///etc/passwd"> ]>>
<someStuff>
  <isHere>
    Hi! &xxeFun;
  </isHere>
</someStuff>
```

If This XML document is

- **received from an external provider,**
- **evaluated, then**
- **returned to the user**

The contents of /etc/passwd are returned to the attacker

```
<?xml version="1.0"?>
<!DOCTYPE kaboom [
  <!ENTITY a "aaaaaaaaaaaaaaaaaa..."> ]>
<kaboom>&a;&a;&a;&a;&a;&a;&a;&a;&a;...</kaboom>
```

What happens this time?

XXE Defense Examples

OWASP
The Open Web Application Security Project

Defense 1: Disable Entity inclusion. The XML Validator will throw a Fatal Exception if such an entity is included.

Xerces Example:

```
DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
dbf.setNamespaceAware(true);
try {
 dbf.setFeature("http://apache.org/xml/features/disallow-doctype-decl", true);
 // Use DBF here to parse XML (safely)
} catch (ParserConfigurationException e) { //handle error }
```

Defense 2: If entities need to be allowed, disable expansion of external entities.

Xerces Example:

```
DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
dbf.setNamespaceAware(true);
try {
 dbf.setFeature("http://xml.org/sax/features/external-general-entities", false);
 dbf.setFeature("http://xml.org/sax/features/external-parameter-entities", false);
 // Use DBF here
} catch (ParserConfigurationException e) { //handle error }
```

A4 – Avoiding XXE

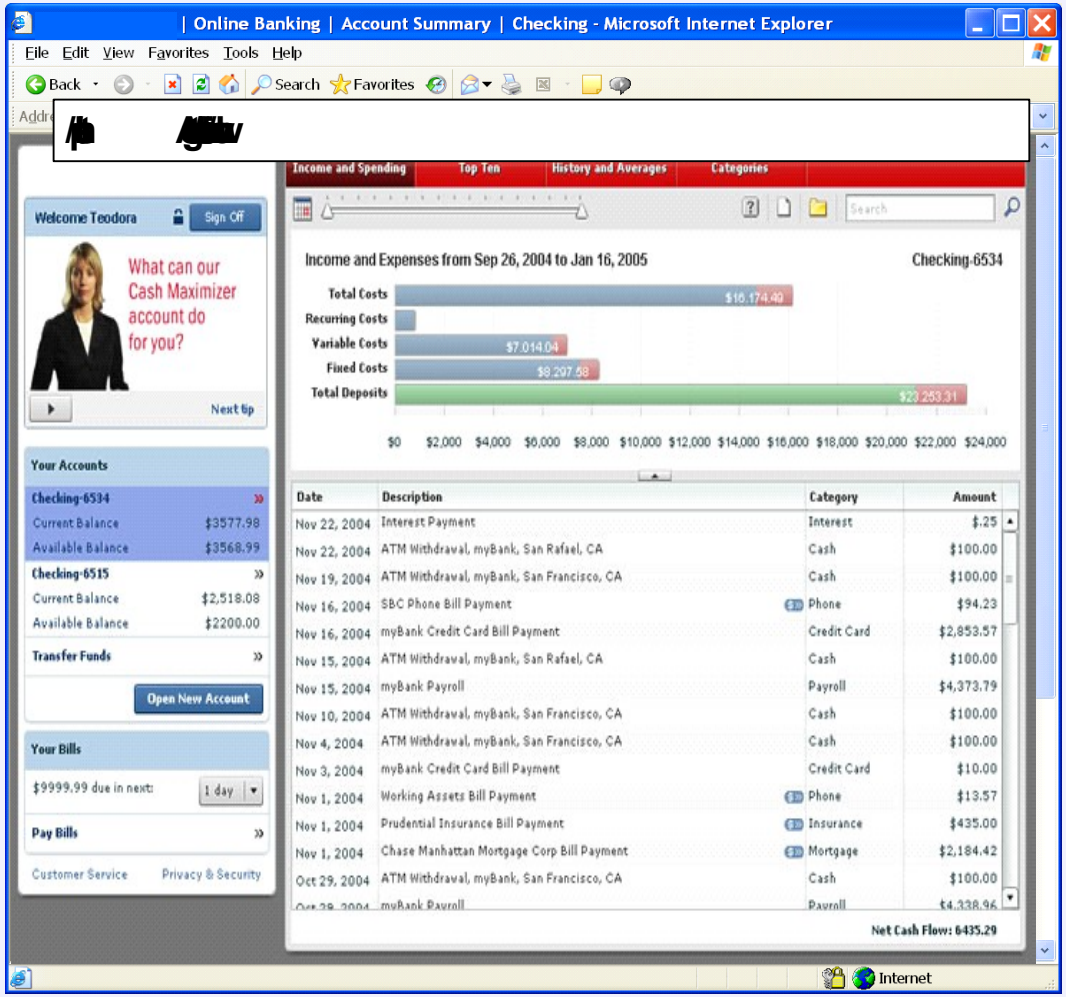
OWASP
The Open Web Application Security Project

2017-A5 – Broken Access Control

OWASP
The Open Web Application Security Project

Missing Function Level Access Control Illustrated

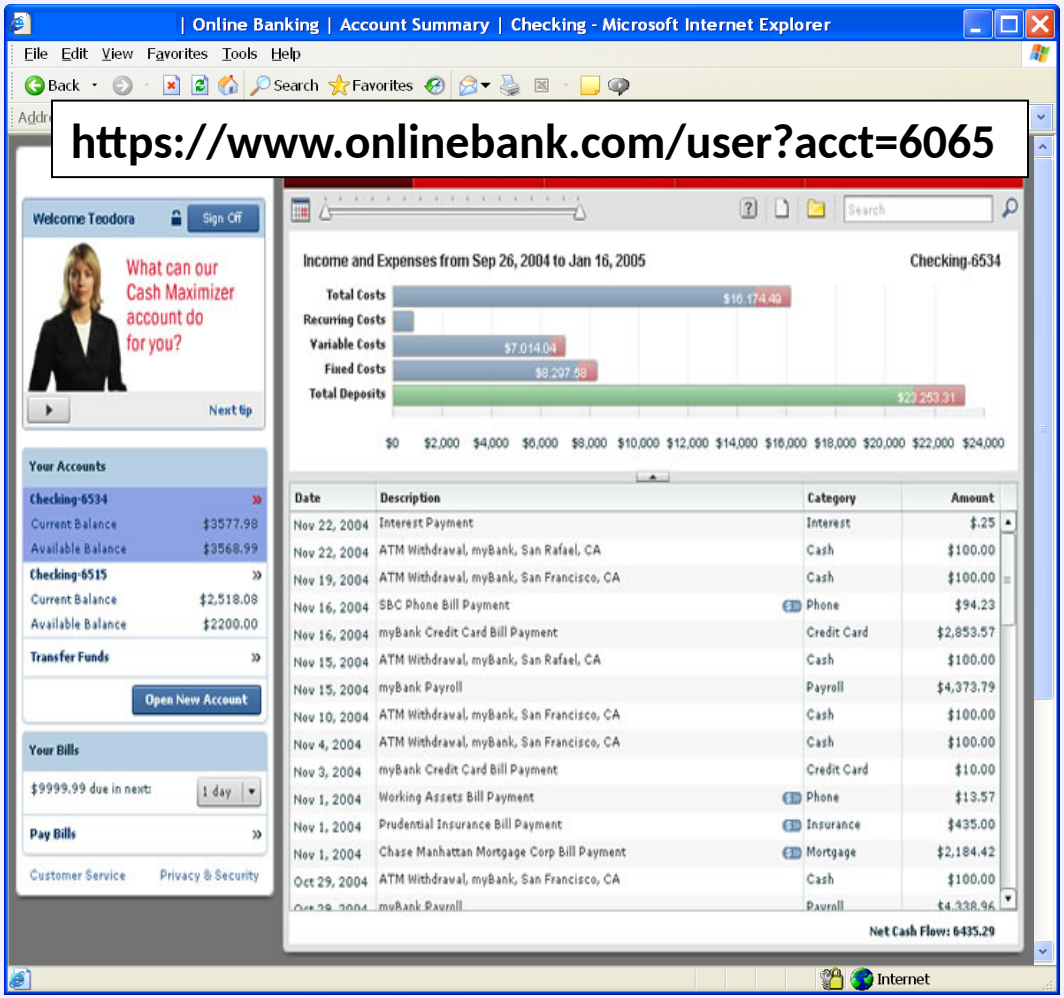
OWASP
The Open Web Application Security Project



- Attacker notices the URL indicates his role
  /user/getAccounts
- He modifies it to another directory (role)
  /admin/getAccounts, or
  /manager/getAccounts
- Attacker views more accounts than just their own

Insecure Direct Object References Illustrated

OWASP
The Open Web Application Security Project



- Attacker notices his acct parameter is 6065

  ?acct=6065

- He modifies it to a nearby number

  ?acct=6066

- Attacker views the victim's account information

Avoiding Broken Access Control

OWASP
The Open Web Application Security Project

- For a function, a site needs to do at least these things
  - Restrict access to authenticated users (if not public)
  - Enforce any user or role based permissions (if private)

- For data, a site needs to verify
  - User has required role to see that data, or
  - User has been granted access (i.e., is data owner, is in associated group, etc.)
  - User has the TYPE of access being used (Read, Write, Delete, etc.)

- Verify your architecture
  - Use a simple, positive model at every layer
  - Be sure you actually have a mechanism at every layer

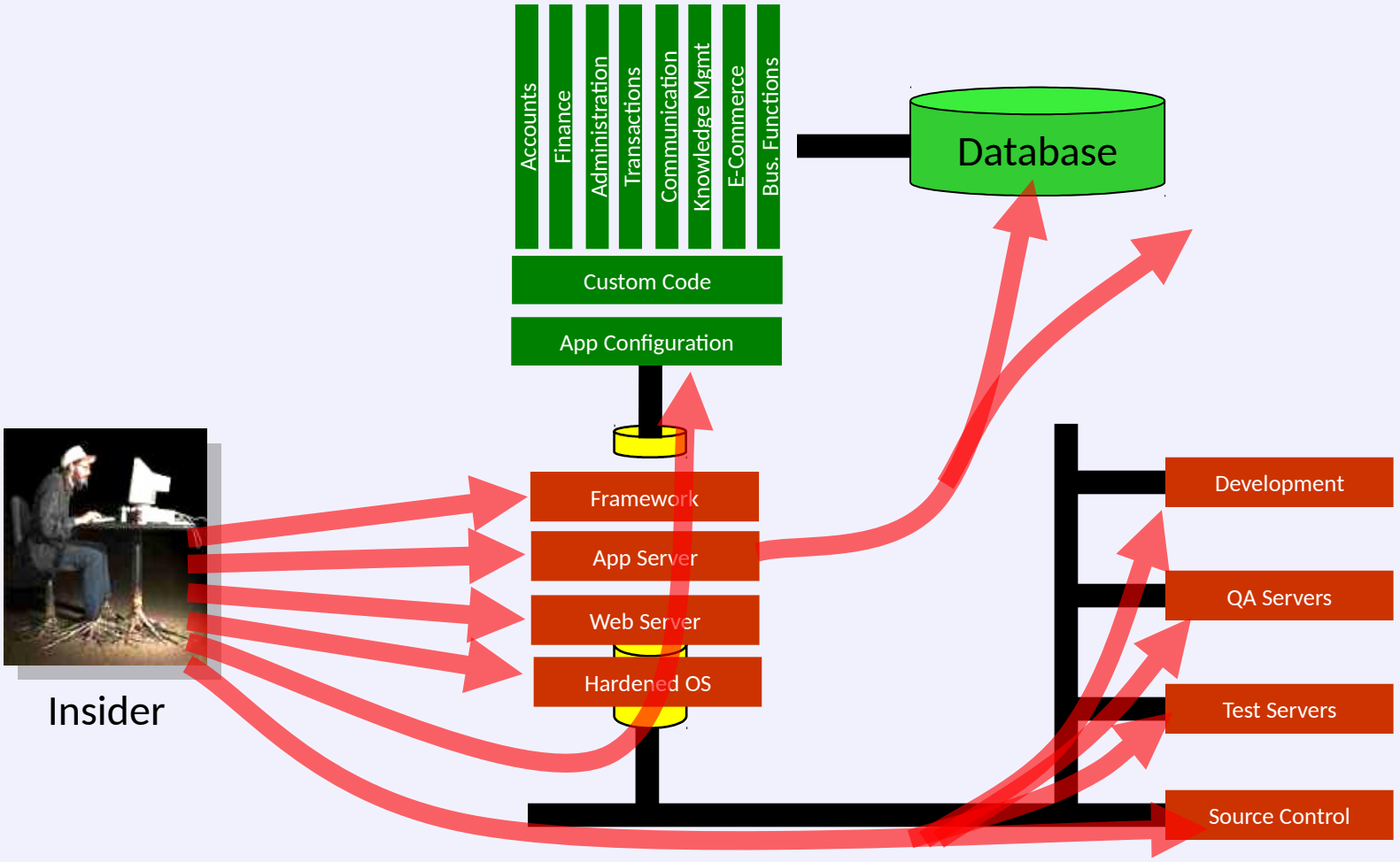
- Verify the implementation
  - Forget automated analysis approaches
  - Verify each URL (plus any parameters) referencing a function or data is protected by
    - An external filter, like Java EE web.xml or a commercial product

2017-A6 – Security Misconfiguration

OWASP
The Open Web Application Security Project

Security Misconfiguration Illustrated

OWASP
The Open Web Application Security Project



Avoiding Security Misconfiguration

OWASP
The Open Web Application Security Project

- Verify your system's configuration management
  - Secure configuration "hardening" guideline
    - Automation is REALLY USEFUL here
  - Must cover entire platform and application
  - Analyze security effects of changes

- Can you "dump" the application configuration
  - Build reporting into your process
  - If you can't verify it, it isn't secure

- Verify the implementation
  - Scanning finds generic configuration and missing patch problems

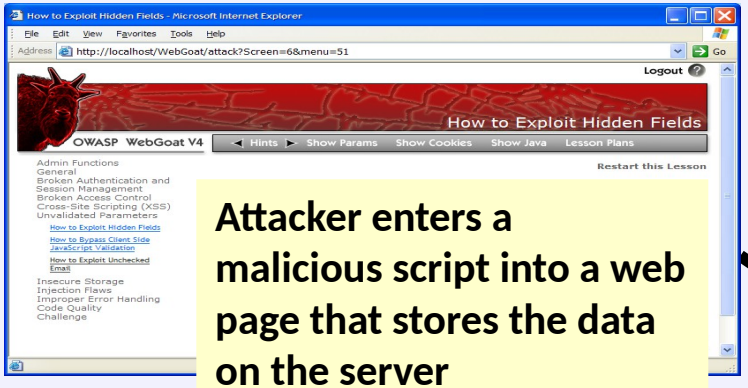
2017-A7 – Cross-Site Scripting (XSS)

OWASP
The Open Web Application Security Project

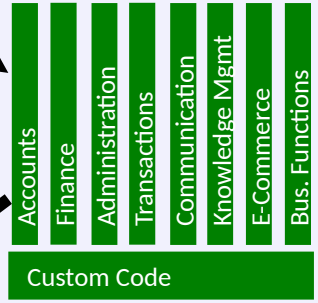
Cross-Site Scripting Illustrated

OWASP
The Open Web Application Security Project

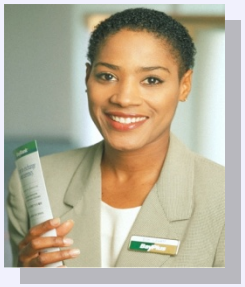
1 **Attacker sets the trap – update my profile**

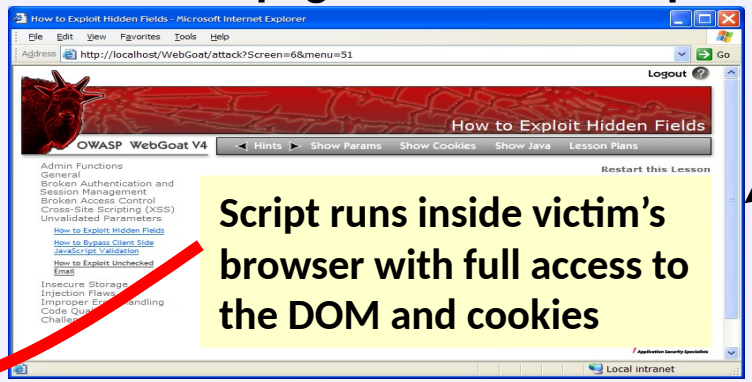


Application with stored XSS vulnerability



2 **Victim views page – sees attacker profile**





3 **Script silently sends attacker Victim's session cookie**

Avoiding XSS Flaws

OWASP
The Open Web Application Security Project

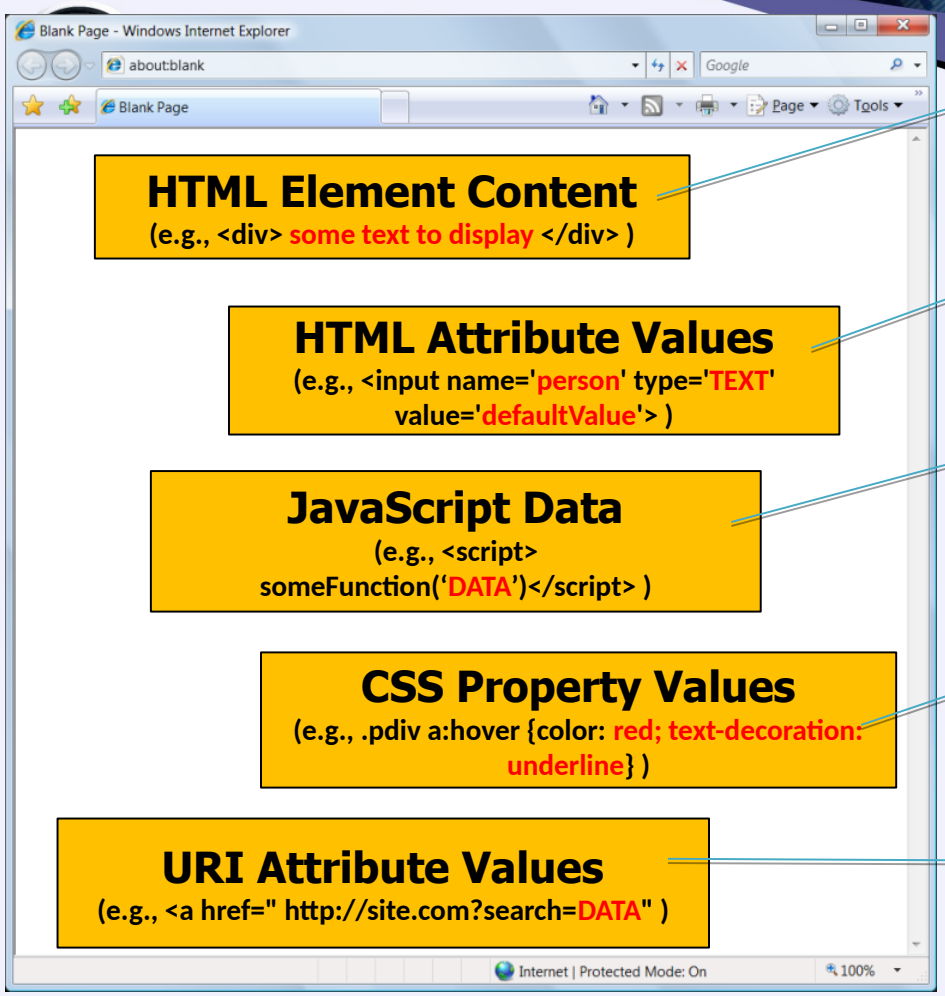
- **Recommendations**
  - **Eliminate Flaw**
    - **Don't include user supplied input in the output page**
  - **Defend Against the Flaw**
    - **Use Content Security Policy (CSP)**
    - **Primary Recommendation: Output encode all user supplied input (Use OWASP's Java Encoders to output encode)**

      https://www.owasp.org/index.php/OWASP_Java_Encoder_Project
    - **Perform 'white list' input validation on all user input to be included in page**
    - **For large chunks of user supplied HTML, use OWASP's AntiSamy to sanitize this HTML to make it safe**

      See: https://www.owasp.org/index.php/AntiSamy
- **References**
  - **For how to output encode properly, read the https://www.owasp.org/index.php/XSS_(Cross Site Scripting) Prevention Cheat Sheet**

(AntiSamy)

Safe Escaping Schemes in Various HTML Execution Contexts



#1: (&, <, >, ") → &entity; (', /) → &#xHH;
ESAPI: encodeForHTML()

#2: All non-alphanumeric < 256 → &#xHH;
ESAPI: encodeForHTMLAttribute()

#3: All non-alphanumeric < 256 → \xHH
ESAPI: encodeForJavaScript()

#4: All non-alphanumeric < 256 → \HH
ESAPI: encodeForCSS()

#5: All non-alphanumeric < 256 → %HH
ESAPI: encodeForURL()

ALL other contexts CANNOT include Untrusted Data

Recommendation: Only allow #1 and #2 and disallow all others

See: www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet
https://www.owasp.org/index.php/DOM_based_XSS_Prevention_Cheat_Sheet

2017-A8 – Insecure Deserialization

OWASP
The Open Web Application Security Project

Deserialization Examples

OWASP
The Open Web Application Security Project

- CVE-2017-5954 – “serialize-to-js package 0.5.0 for Node.js. Untrusted data passed into the deserialize() function can be exploited to achieve arbitrary code execution by passing a JavaScript Object with an Immediately Invoked Function Expression (IIFE).”
- CVE-2017-9424 – “IdeaBlade Breeze Breeze.Server.NET before 1.6.5 allows remote attackers to execute arbitrary code, related to use of TypeNameHandling in JSON deserialization.”
- CVE-2017-9805 – “REST Plugin in Struts 2.1.2 thru 2.3.33 and 2.5.x before 2.5.13 uses an XStreamHandler with an instance of XStream for deserialization without any type filtering, which can lead to Remote Code Execution when deserializing XML payloads.”
- CVE-2017-1000034 – “Akka versions <=2.4.16 and 2.5-M1 are vulnerable to a java deserialization attack in its Remoting component resulting in remote code execution”

Avoiding Deserialization Vulnerabilities

OWASP
The Open Web Application Security Project

2017-A9 – Using Known Vulnerable Components

OWASP
The Open Web Application Security Project

What Can You Do to Avoid This?

OWASP
The Open Web Application Security Project

Automation Example for Java – Use Maven 'Versions' Plugin

OWASP
The Open Web Application Security Project

Output from the Maven Versions Plugin – Automated Analysis of Libraries' Status against Central repository

Dependencies

Status	Group Id	Artifact Id	Current Version	Scope	Classifier	Type	Next Version	Next Incremental	Next Minor	Next Major
⚠	com.fasterxml.jackson.core	jackson-annotations	2.0.4	compile		jar		**2.0.5**	**2.1.0**	
⚠	com.fasterxml.jackson.core	jackson-core	2.0.4	compile		jar		**2.0.5**	**2.1.0**	
⚠	com.fasterxml.jackson.core	jackson-databind	2.0.4	compile		jar		**2.0.5**	**2.1.0**	
⚠	com.google.guava	guava	11.0	compile		jar		**11.0.1**	**12.0-rc1**	**12.0**
⚠	com.ibm.icu	icu4j	49.1	compile		jar				**50.1**
⚠	com.theoryinpractise	halbuilder	1.0.4	compile		jar		**1.0.5**		
⚠	commons-codec	commons-codec	1.3	compile		jar			**1.4**	
✓	commons-logging	commons-logging	1.1.1	compile		jar				
⚠	joda-time	joda-time	2.0	compile		jar			**2.1**	
⚠	net.sf.ehcache	ehcache-core	2.5.1	compile		jar		**2.5.2**	**2.6.0**	
⚠	org.apache.httpcomponents	httpclient	4.1.2	compile		jar		**4.1.3**	**4.2**	
⚠	org.apache.httpcomponents	httpclient-cache	4.1.2	compile		jar		**4.1.3**	**4.2**	
⚠	org.apache.httpcomponents	httpcore	4.1.2	compile		jar		**4.1.3**	**4.2**	
⚠	org.jdom	jdom	1.1	compile		jar		**1.1.2**		**2.0.0**
✓	org.slf4j	slf4j-api	1.7.2	provided		jar				

Most out of Date!

Details Developer Needs

This can automatically be run EVERY TIME software is built!!

2017-A10 – Insufficient Logging & Monitoring

OWASP
The Open Web Application Security Project

Providing Sufficient Logging & Monitoring

OWASP
The Open Web Application Security Project

Summary: How do you address these problems?

OWASP
The Open Web Application Security Project

- **Develop Secure Code**
  - **Follow the best practices in OWASP's Guide to Building Secure Web Applications**
    - **https://www.owasp.org/index.php/Guide**
    - **And the cheat sheets: https://www.owasp.org/index.php/Cheat_Sheets**
  - **Use OWASP's Application Security Verification Standard as a guide to what an application needs to be secure**
    - **https://www.owasp.org/index.php/ASVS**
  - **Use standard security components that are a fit for your organization**
    - **Use OWASP's ESAPI to help identify what standard security components you are likely to need**
    - **https://www.owasp.org/index.php/ESAPI**

- **Review Your Applications**
  - **Have an expert team review your applications**
  - **Review your applications yourselves following OWASP Guidelines**
    - **OWASP Code Review Guide: https://www.owasp.org/index.php/Code_Review_Guide**
    - **OWASP Testing Guide: https://www.owasp.org/index.php/Testing_Guide**

Thank you
OWASP Top-10 2017