# 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 10 is an Awareness Document

- Not a standard…

### First developed in 2003

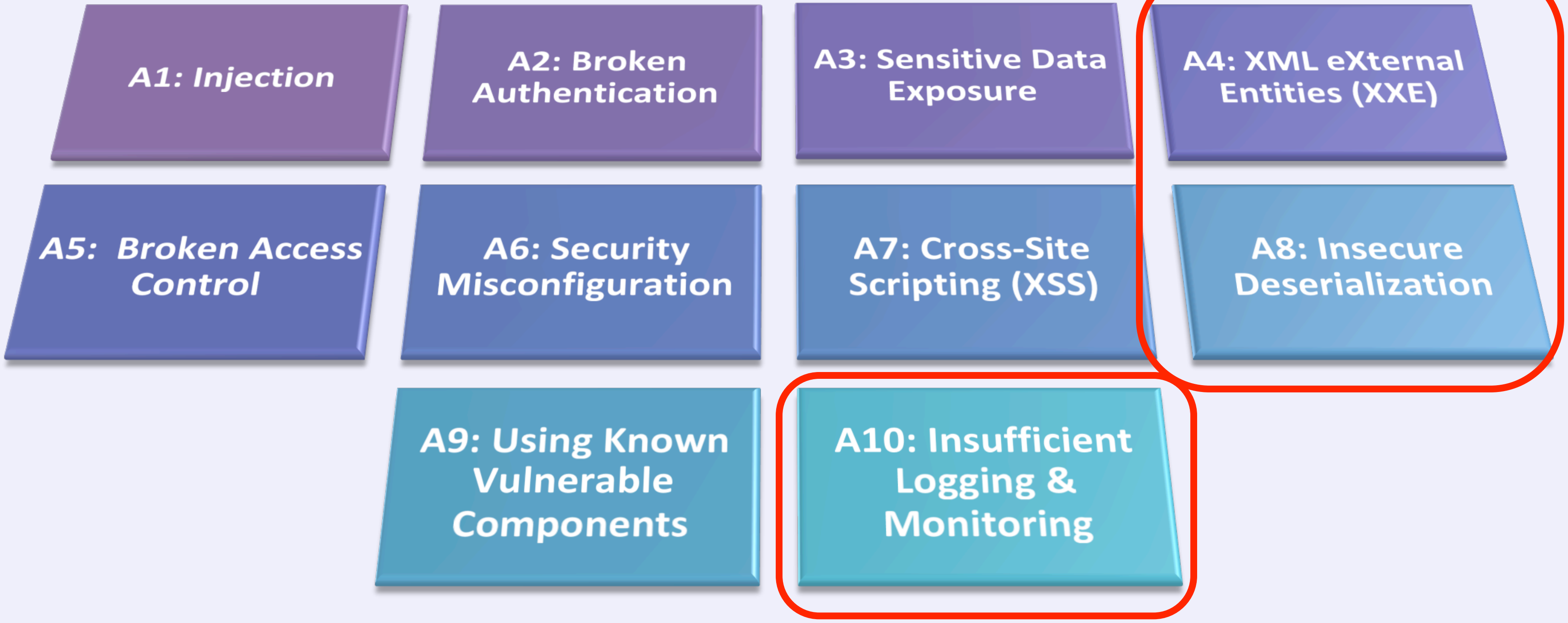
- Was probably 3rd or 4th OWASP project, after
  - Developers Guide
  - WebGoat
  - Maybe WebScarab ??

### Released

- 2003, 2004, 2007, 2010, 2013, 2017

# OWASP Top Ten (2017 Edition)

OWASP
The Open Web Application Security Project



A1: Injection

A2: Broken Authentication

A3: Sensitive Data Exposure

A4: XML eXternal Entities (XXE)

A5: Broken Access Control

A6: Security Misconfiguration

A7: Cross-Site Scripting (XSS)

A8: Insecure Deserialization

A9: Using Known Vulnerable Components

A10: Insufficient Logging & Monitoring

## What Didn't Change

OWASP
The Open Web Application Security Project

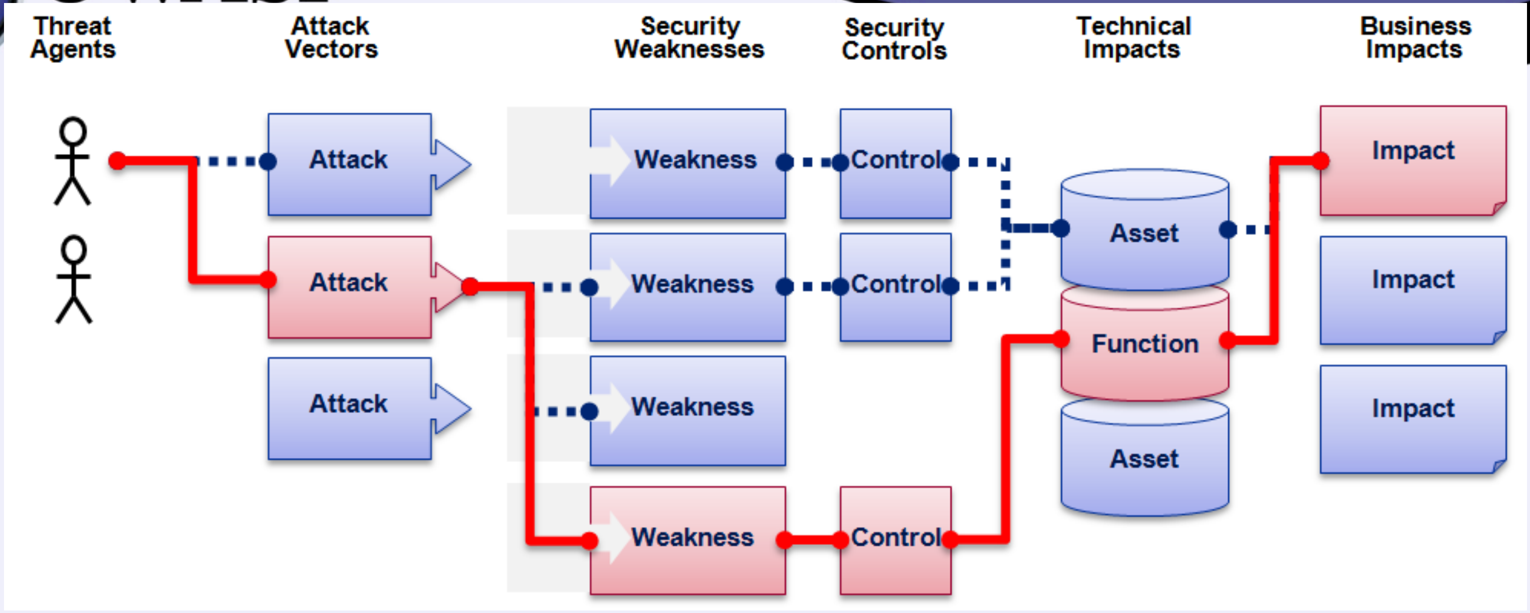
### It's About Risks, Not Just Vulnerabilities

- Title is: "The Top 10 Most Critical Web Application Security Risks"

### OWASP Top 10 Risk Rating Methodology

- Based on the OWASP Risk Rating Methodology, used to prioritize Top 10

# 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

### Risks Added, Risks Merged, Risks Reordered

- Added: 3
- Merged:  2 merged into 1
- Reordered: 3

### Development Methodology For 2017

- Significantly broader, public, data call (two actually)
  - For both Vuln Data AND Industry Opinions
- Far more data analysis
- Initial draft by original Top 10 team. Final by new Top 10 Team.

# 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] | ∪ | 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] | ∪ | 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

## Injection means…

- Tricking an application into including unintended commands in the data sent to an interpreter

## Interpreters…

- Take strings and interpret them as commands
- SQL, OS Shell, LDAP, XPath, Hibernate, etc…

## SQL injection is still quite common

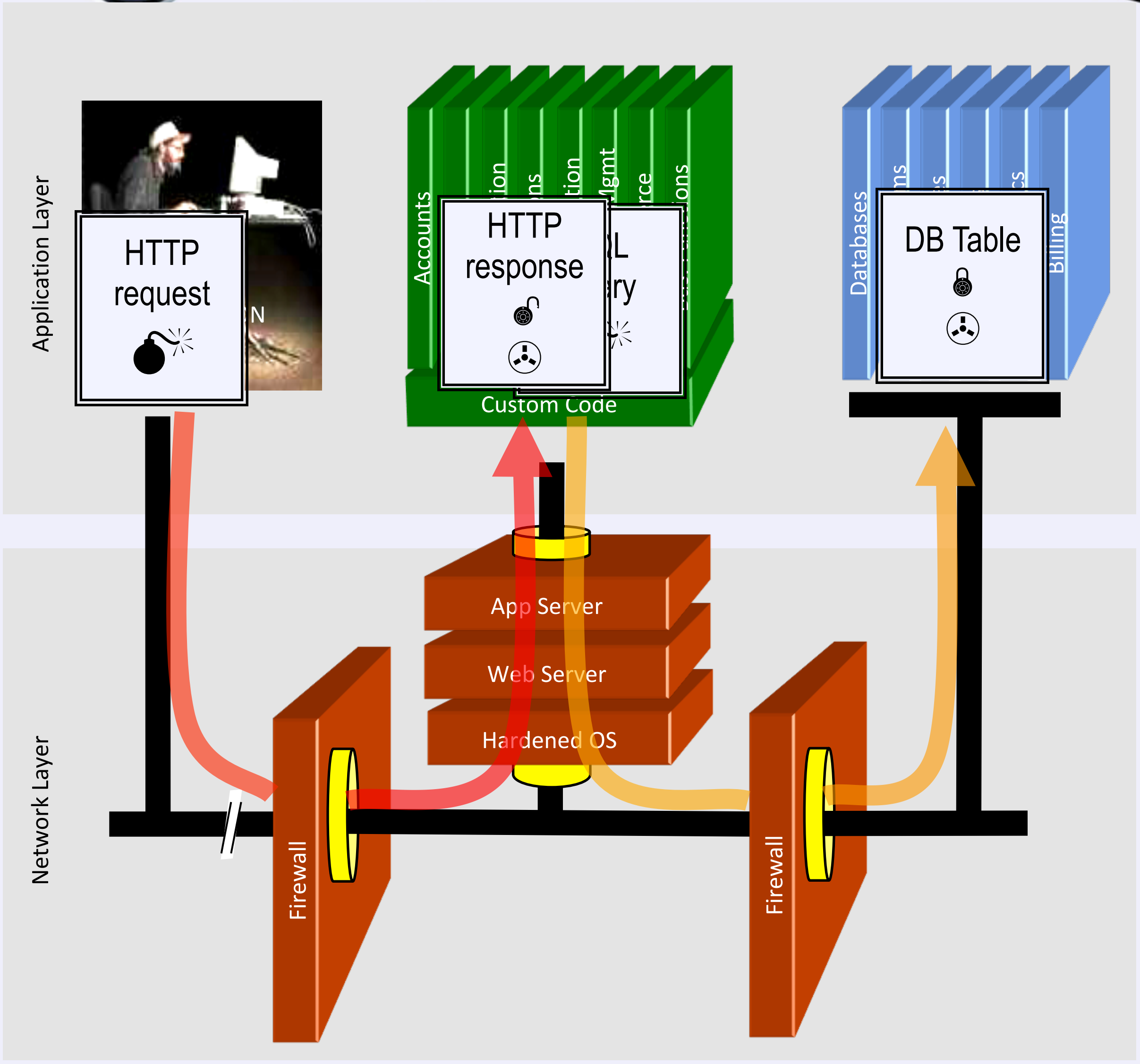
- Many applications still susceptible (really don't know why)
- Even though it's usually very simple to avoid

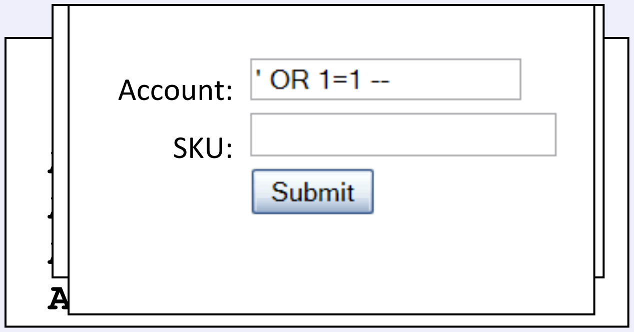
## Typical Impact

- Usually severe. Entire database can usually be read or modified
- May also allow full database schema, or account access, or even OS level access

# SQL Injection – Illustrated

OWASP
The Open Web Application Security Project





Account: ' OR 1=1 --
SKU:

**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

### Recommendations

- Avoid the interpreter entirely, or
- Use an interface that supports bind variables (e.g., prepared statements, or stored procedures),
  - Bind variables allow the interpreter to distinguish between code and data
- Encode all user input before passing it to the interpreter
- Always perform ‘white list’ input validation on all user supplied input
- Always minimize database privileges to reduce the impact of a flaw

### Follow the guidance from

- https://www.owasp.org/index.php/Injection_Prevention_Cheat_Sheet
- https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet
- https://www.owasp.org/index.php/OS_Command_Injection_Defense_Cheat_Sheet
- https://www.owasp.org/index.php/LDAP_Injection_Prevention_Cheat_Sheet

# 2017-A2 – Broken Authentication

OWASP
The Open Web Application Security Project

## How strong is initial user authentication

- What is your authentication scheme?
- Are you incorporating two-factor authentication?
- How safely do you store user credentials?
- Some form of credentials have to go with every request (initial auth, then session ID)
- Should use SSL for everything requiring authentication

## Session management flaws

- SESSION ID used to track state since HTTP doesn't
  - and it is just as good as credentials to an attacker
- SESSION ID is frequently exposed on the network, in browser, in logs, …

## Beware the side-doors

- Change my password, remember my password, forgot my password, secret question, logout, email address, etc…

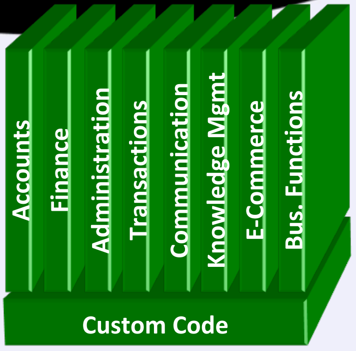
## Typical Impact

- User accounts compromised or user sessions hijacked

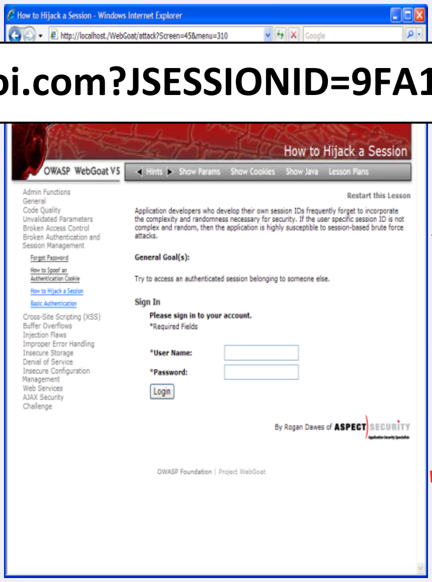
# Broken Authentication Illustrated

OWASP
The Open Web Application Security Project

1 User sends credentials



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



Application developers who develop their own session IDs frequently forget to incorporate the complexity and randomness necessary for security. If the user specific session ID is not complex and random, then the application is highly susceptible to session-based brute force attacks.

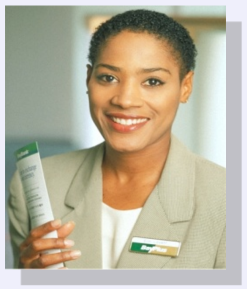
General Goal(s):

Try to access an authenticated session belonging to someone else.

Sign In

Please sign in to your account.
*Required Fields

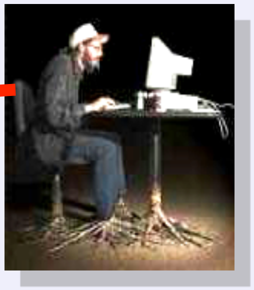
*User Name:
*Password:



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

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

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



5 Hacker uses JSESSIONID and takes over victim's account

OWASP
The Open Web Application Security Project

## A2 – Avoiding Broken Authentication

### Verify your architecture

- Authentication should be simple, centralized, and standardized
- User passwords need to be STRONGLY hashed before storage
- Use the standard session id provided by your container
- Be sure SSL protects both credentials and session id at all times

### Verify the implementation

- Forget automated analysis approaches
- Check your SSL certificate
- Examine all the authentication-related functions (particularly password storage)
- Verify that logoff actually destroys the session

### Follow the guidance from

- https://www.owasp.org/index.php/Authentication_Cheat_Sheet
- https://www.owasp.org/index.php/Forgot_Password_Cheat_Sheet
- https://www.owasp.org/index.php/Choosing_and_Using_Security_Questions_Cheat_Sheet
- https://www.owasp.org/index.php/Credential_Stuffing_Prevention_Cheat_Sheet
- https://www.owasp.org/index.php/Session_Management_Cheat_Sheet

## 2017-A3 – Sensitive Data Exposure

OWASP
The Open Web Application Security Project

### Storing and transmitting sensitive data insecurely

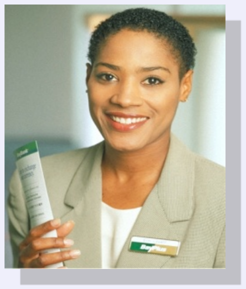
- Failure to identify all sensitive data
- Failure to identify all the places that this sensitive data gets stored
  - Databases, files, directories, log files, backups, etc.
- Failure to identify all the places that this sensitive data is sent
  - On the web, to backend databases, to business partners, internal communications
- Failure to properly protect this data in every location

### Typical Impact

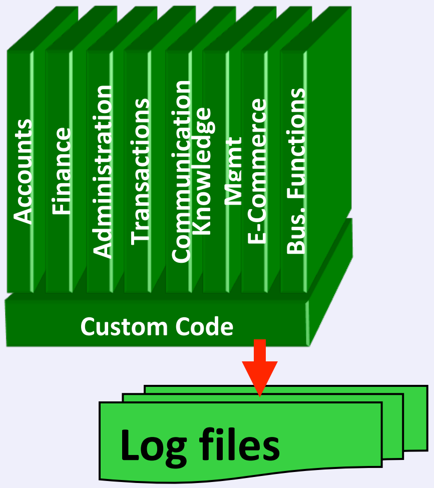
- Attackers access or modify confidential or private information
  - e.g., credit cards, health care records, financial data (yours or your customers)
- Attackers extract secrets to use in additional attacks
- Company embarrassment, customer dissatisfaction, and loss of trust
- Expense of cleaning up the incident, such as forensics, sending apology letters, reissuing thousands of credit cards, providing identity theft insurance
- Business gets sued and/or fined

# Insecure Cryptographic Storage Illustrated

OWASP
The Open Web Application Security Project

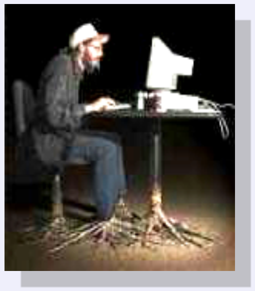


1 Victim enters credit card number in form



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

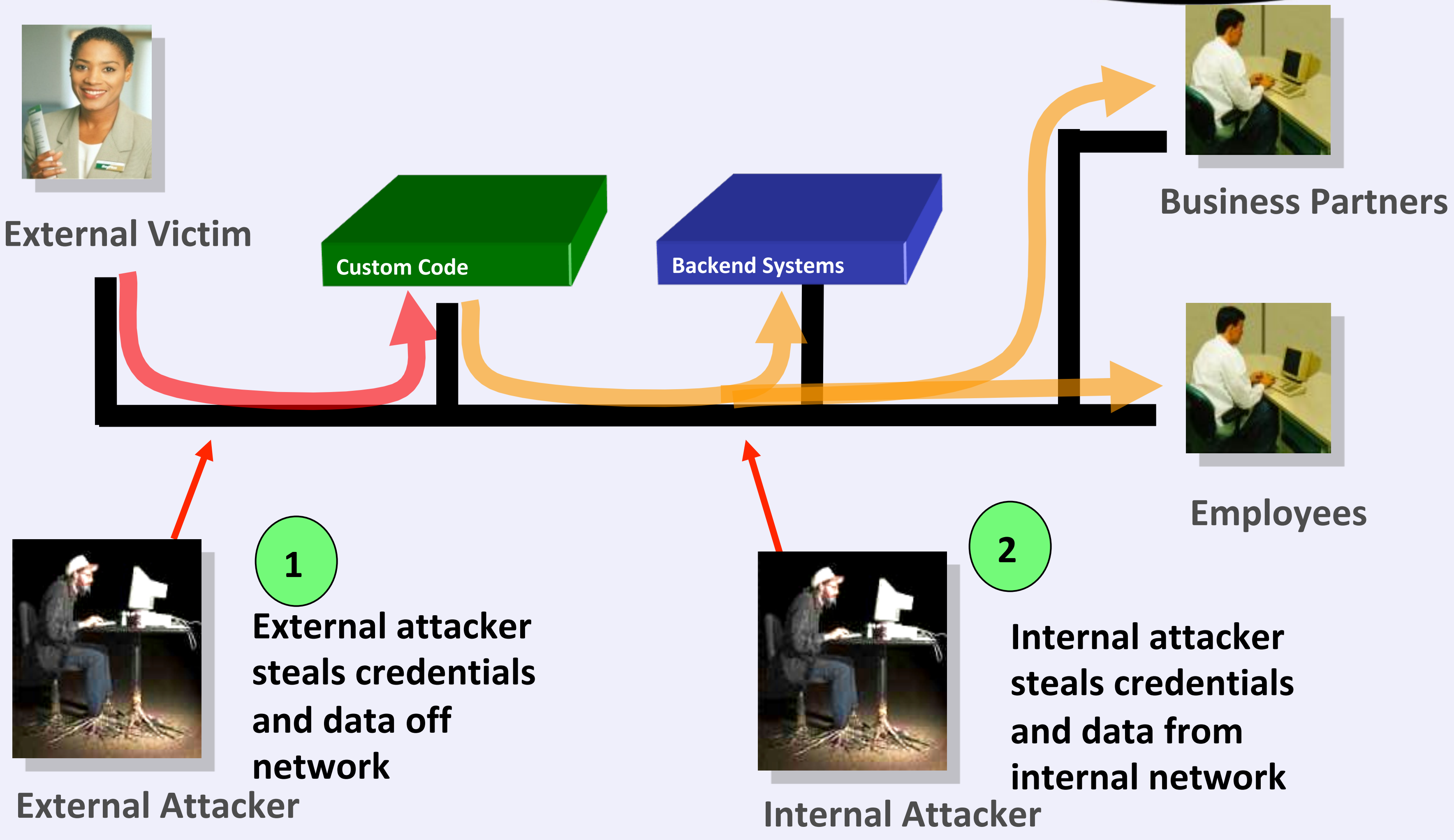
- Verify your architecture
  - Identify all sensitive data
  - Identify all the places that data is stored
  - Ensure threat model accounts for possible attacks
  - Use encryption to counter the threats, don't just 'encrypt' the data

- 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
  - Analyze encryption code for common flaws

# Insufficient Transport Layer Protection Illustrated



External Victim
Custom Code
Backend Systems
Business Partners
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

### What is it?

- An XML external entity is a URL, typically to a local file or web service, or a local variable within the XML document
- Many XML Parsers have XXE enabled by default
  - Particularly Java XML Parsers

### A common mistake …

- Developers don't even know XML documents support external entities
- They accept an XML document from an untrusted source
- Process the XML document with XML parser that has XXE enabled by default

### Typical Impact

- Attackers able to access unauthorized files (e.g., /etc/password) or resources (back end web services)
- Denial of Service (consume all available memory)

## 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 }
```

OWASP
The Open Web Application Security Project

## A4 – Avoiding XXE

### Verify your architecture

- Are you even processing XML at all?
- If so, which XML parsers are you using?
- Do they have XXE enabled by default?
- Are your XML document sources trustworthy?

### Verify the implementation

- Verify all the types of XML parsers being used, if any.
- For each, verify each XML parser has either
  - a) XXE disabled by default (and not enabled), or
  - b) XXE is disabled explicitly, or
  - c) Is replaced with an XML parser of type a) or b)

### Follow the guidance from

- https://www.owasp.org/index.php/XML_External_Entity_(XXE)_Prevention_Cheat_Sheet

## 2017-A5 – Broken Access Control

OWASP
The Open Web Application Security Project

### How do you protect access to specific functions and specific data elements?

- Each function and data reference needs to verify user is authorized to access in manner requested (read, write, delete, create, etc.)

### A common mistake …

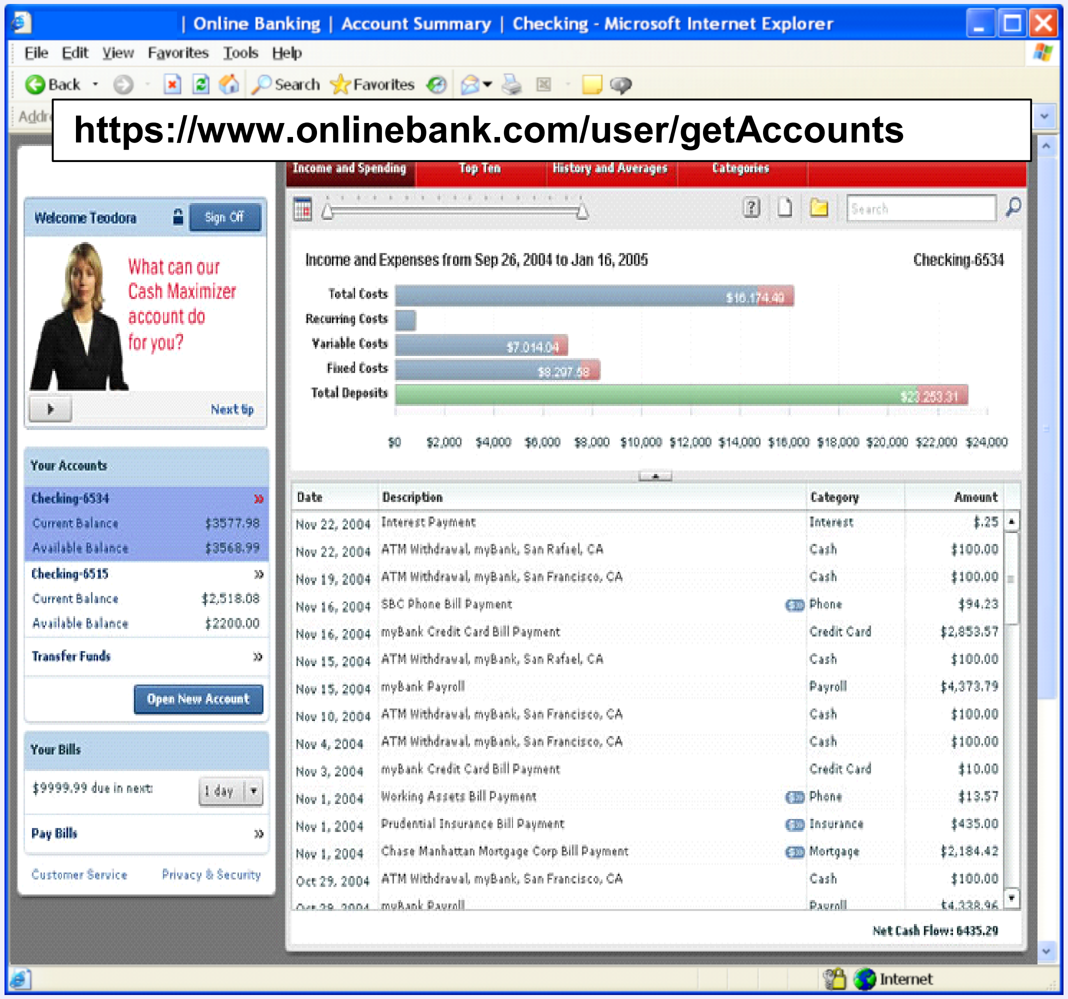
- Displaying only authorized links and menu choices
- This is called presentation layer access control, and doesn't work
- Attacker simply forges direct access to 'unauthorized' functions and data

### Typical Impact

- Attackers invoke functions and services they're not authorized for
- Access other user's accounts and data
- Perform privileged actions

# Missing Function Level Access Control Illustrated

OWASP
The Open Web Application Security Project



https://www.onlinebank.com/user/getAccounts

Income and Expenses from Sep 26, 2004 to Jan 16, 2005 Checking-6534

| | |
|---|---|
| **Your Accounts** | |
| **Checking-6534** | » |
| Current Balance | $3577.98 |
| Available Balance | $3568.99 |
| **Checking-6515** | » |
| Current Balance | $2,518.08 |
| Available Balance | $2200.00 |
| **Transfer Funds** | » |
| **Your Bills** | |
| $9999.99 due in next: | 1 day |
| **Pay Bills** | » |

| Date | Description | Category | Amount |
|---|---|---|---|
| Nov 22, 2004 | Interest Payment | Interest | $.25 |
| Nov 22, 2004 | ATM Withdrawal, myBank, San Rafael, CA | Cash | $100.00 |
| Nov 19, 2004 | ATM Withdrawal, myBank, San Francisco, CA | Cash | $100.00 |
| Nov 16, 2004 | SBC Phone Bill Payment | Phone | $94.23 |
| Nov 16, 2004 | myBank Credit Card Bill Payment | Credit Card | $2,853.57 |
| Nov 15, 2004 | ATM Withdrawal, myBank, San Rafael, CA | Cash | $100.00 |
| Nov 15, 2004 | myBank Payroll | Payroll | $4,373.79 |
| Nov 10, 2004 | ATM Withdrawal, myBank, San Francisco, CA | Cash | $100.00 |
| Nov 4, 2004 | ATM Withdrawal, myBank, San Francisco, CA | Cash | $100.00 |
| Nov 3, 2004 | myBank Credit Card Bill Payment | Credit Card | $10.00 |
| Nov 1, 2004 | Working Assets Bill Payment | Phone | $13.57 |
| Nov 1, 2004 | Prudential Insurance Bill Payment | Insurance | $435.00 |
| Nov 1, 2004 | Chase Manhattan Mortgage Corp Bill Payment | Mortgage | $2,184.42 |
| Oct 29, 2004 | ATM Withdrawal, myBank, San Francisco, CA | Cash | $100.00 |
| Oct 29, 2004 | myBank Payroll | Payroll | $4,338.96 |

Net Cash Flow: 6435.29

- Attacker notices the URL indicates his role

  /user/getAccounts

- He modifies it to another directory (role)

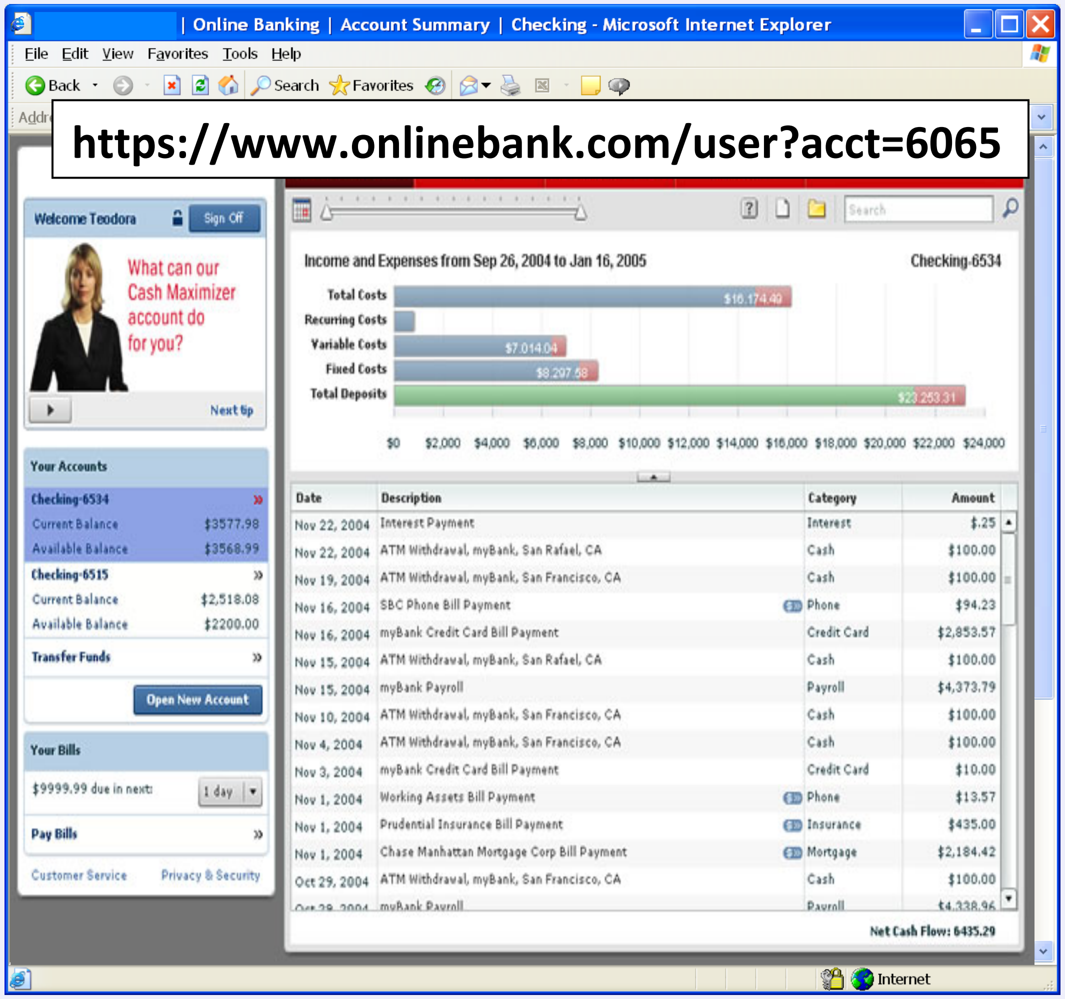
  /admin/getAccounts, or

  /manager/getAccounts

- Attacker views more accounts than just their own

# Insecure Direct Object References Illustrated

OWASP
The Open Web Application Security Project



| Online Banking | Account Summary | Checking - Microsoft Internet Explorer

File Edit View Favorites Tools Help

https://www.onlinebank.com/user?acct=6065

Welcome Teodora — Sign Off

What can our Cash Maximizer account do for you?

Next tip

Your Accounts

Checking-6534
Current Balance $3577.98
Available Balance $3568.99

Checking-6515
Current Balance $2,518.08
Available Balance $2200.00

Transfer Funds

Open New Account

Your Bills

$9999.99 due in next: 1 day

Pay Bills

Customer Service — Privacy & Security

Income and Expenses from Sep 26, 2004 to Jan 16, 2005 — Checking-6534

| | |
|---|---|
| Total Costs | $16,174.49 |
| Recurring Costs | |
| Variable Costs | $7,014.04 |
| Fixed Costs | $8,297.58 |
| Total Deposits | $23,253.31 |

| Date | Description | Category | Amount |
|---|---|---|---|
| Nov 22, 2004 | Interest Payment | Interest | $.25 |
| Nov 22, 2004 | ATM Withdrawal, myBank, San Rafael, CA | Cash | $100.00 |
| Nov 19, 2004 | ATM Withdrawal, myBank, San Francisco, CA | Cash | $100.00 |
| Nov 16, 2004 | SBC Phone Bill Payment | Phone | $94.23 |
| Nov 16, 2004 | myBank Credit Card Bill Payment | Credit Card | $2,853.57 |
| Nov 15, 2004 | ATM Withdrawal, myBank, San Rafael, CA | Cash | $100.00 |
| Nov 15, 2004 | myBank Payroll | Payroll | $4,373.79 |
| Nov 10, 2004 | ATM Withdrawal, myBank, San Francisco, CA | Cash | $100.00 |
| Nov 4, 2004 | ATM Withdrawal, myBank, San Francisco, CA | Cash | $100.00 |
| Nov 3, 2004 | myBank Credit Card Bill Payment | Credit Card | $10.00 |
| Nov 1, 2004 | Working Assets Bill Payment | Phone | $13.57 |
| Nov 1, 2004 | Prudential Insurance Bill Payment | Insurance | $435.00 |
| Nov 1, 2004 | Chase Manhattan Mortgage Corp Bill Payment | Mortgage | $2,184.42 |
| Oct 29, 2004 | ATM Withdrawal, myBank, San Francisco, CA | Cash | $100.00 |
| Oct 29, 2004 | myBank Payroll | Payroll | $4,338.96 |

Net Cash Flow: 6435.29

- 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 <u>every</u> 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
    - Or internal checks in YOUR code – e.g., your isAuthorizedForRESOURCE() method
  - Verify the server configuration disallows requests to unauthorized file types

# 2017-A6 – Security Misconfiguration

OWASP
The Open Web Application Security Project

## Web applications rely on a secure foundation

- Everywhere from the OS up through the App Server

## Is your source code a secret?

- Think of all the places your source code goes
- Security should not require secret source code

## CM must extend to all parts of the application

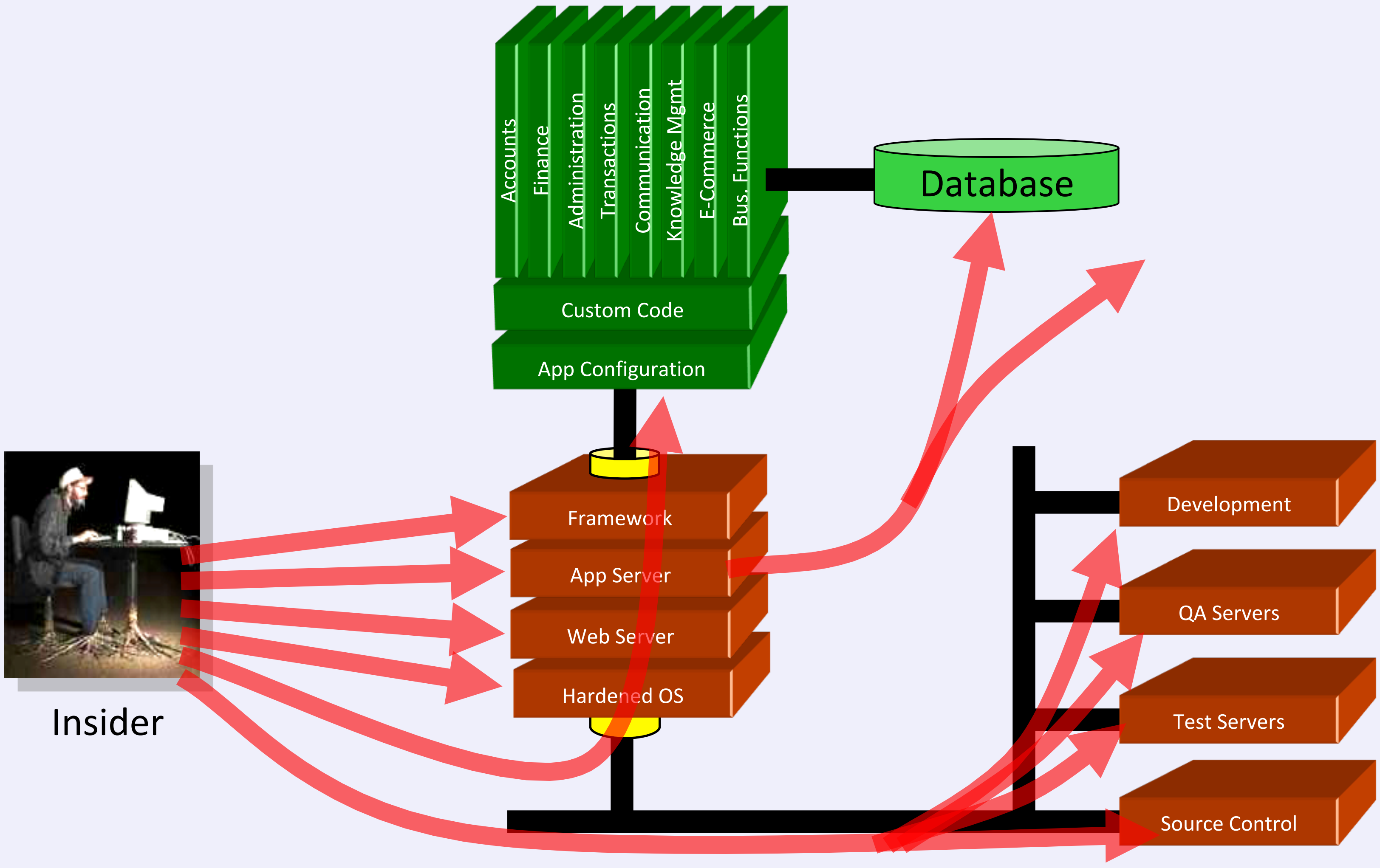
- All credentials should change in production

## Typical Impact

- Install backdoor through missing OS or server patch
- Unauthorized access to default accounts, application functionality or data, or unused but accessible functionality due to poor server configuration

# 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**

OWASP
The Open Web Application Security Project

## 2017-A7 – Cross-Site Scripting (XSS)

### Occurs any time…

- Raw data from attacker is sent to an innocent user's browser

### Raw data…

- Stored in database
- Reflected from web input (form field, hidden field, URL, etc…)
- Sent directly into rich JavaScript client

### Virtually <u>every</u> web application has this problem

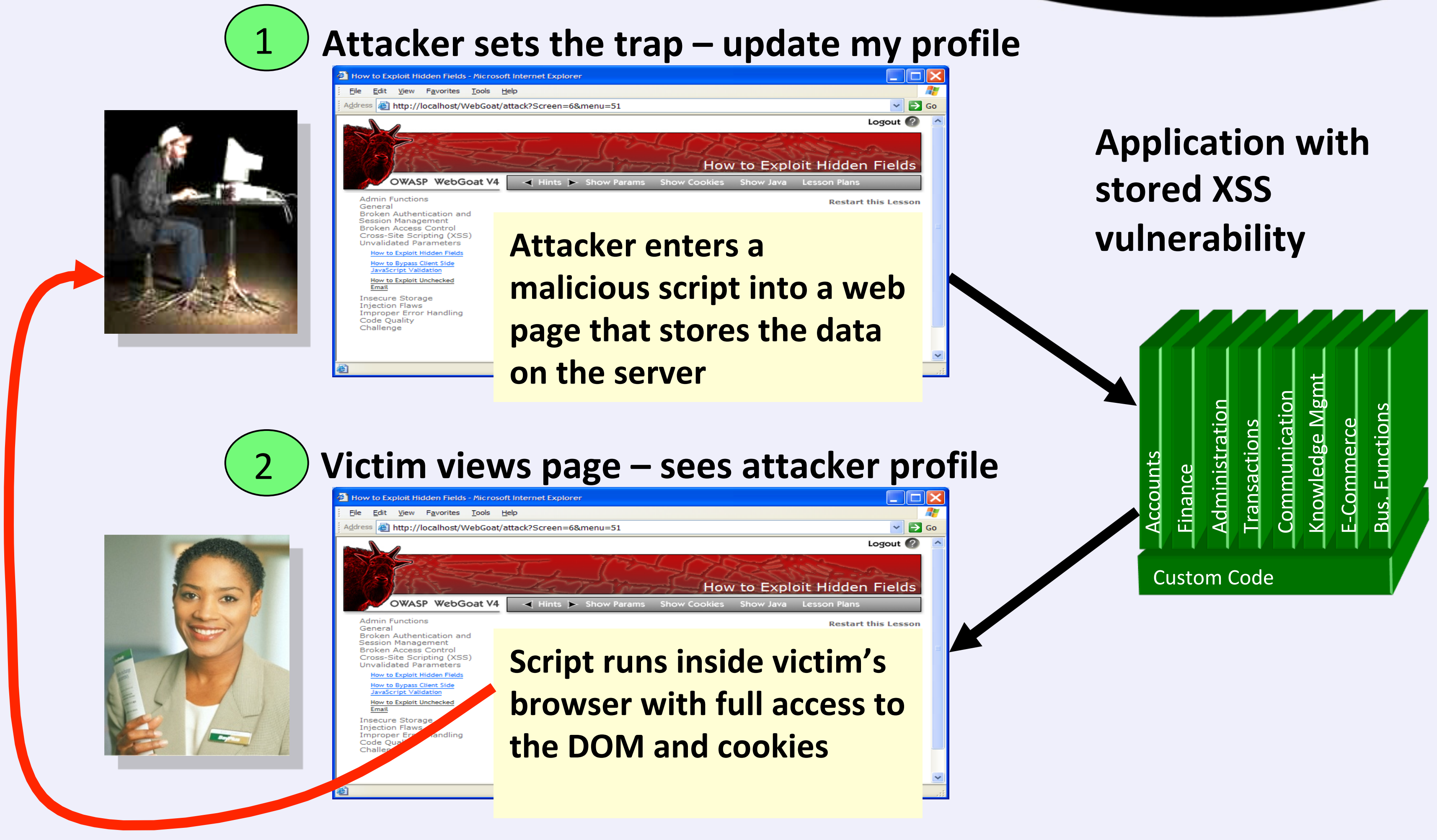
- Try this in your browser – javascript:alert(document.cookie)

### Typical Impact

- Steal user's session, steal sensitive data, rewrite web page, redirect user to phishing or malware site
- Most Severe: Install XSS proxy which allows attacker to observe and direct all user's behavior on vulnerable site and force user to other sites

# Cross-Site Scripting Illustrated

OWASP
The Open Web Application Security Project



How to Exploit Hidden Fields - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://localhost/WebGoat/attack?Screen=6&menu=51
Go
Logout
How to Exploit Hidden Fields
OWASP WebGoat V4
Hints
Show Params
Show Cookies
Show Java
Lesson Plans
Restart this Lesson
Admin Functions
General
Broken Authentication and Session Management
Broken Access Control
Cross-Site Scripting (XSS)
Unvalidated Parameters
How to Exploit Hidden Fields
How to Bypass Client Side JavaScript Validation
How to Exploit Unchecked Email
Insecure Storage
Injection Flaws
Improper Error Handling
Code Quality
Challenge
Accounts
Finance
Administration
Transactions
Communication
Knowledge Mgmt
E-Commerce
Bus. Functions
Custom Code

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

**Attacker enters a malicious script into a web page that stores the data on the server**

**Application with stored XSS vulnerability**

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

**Script runs inside victim's browser with full access to the DOM and cookies**

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

# Avoiding XSS Flaws

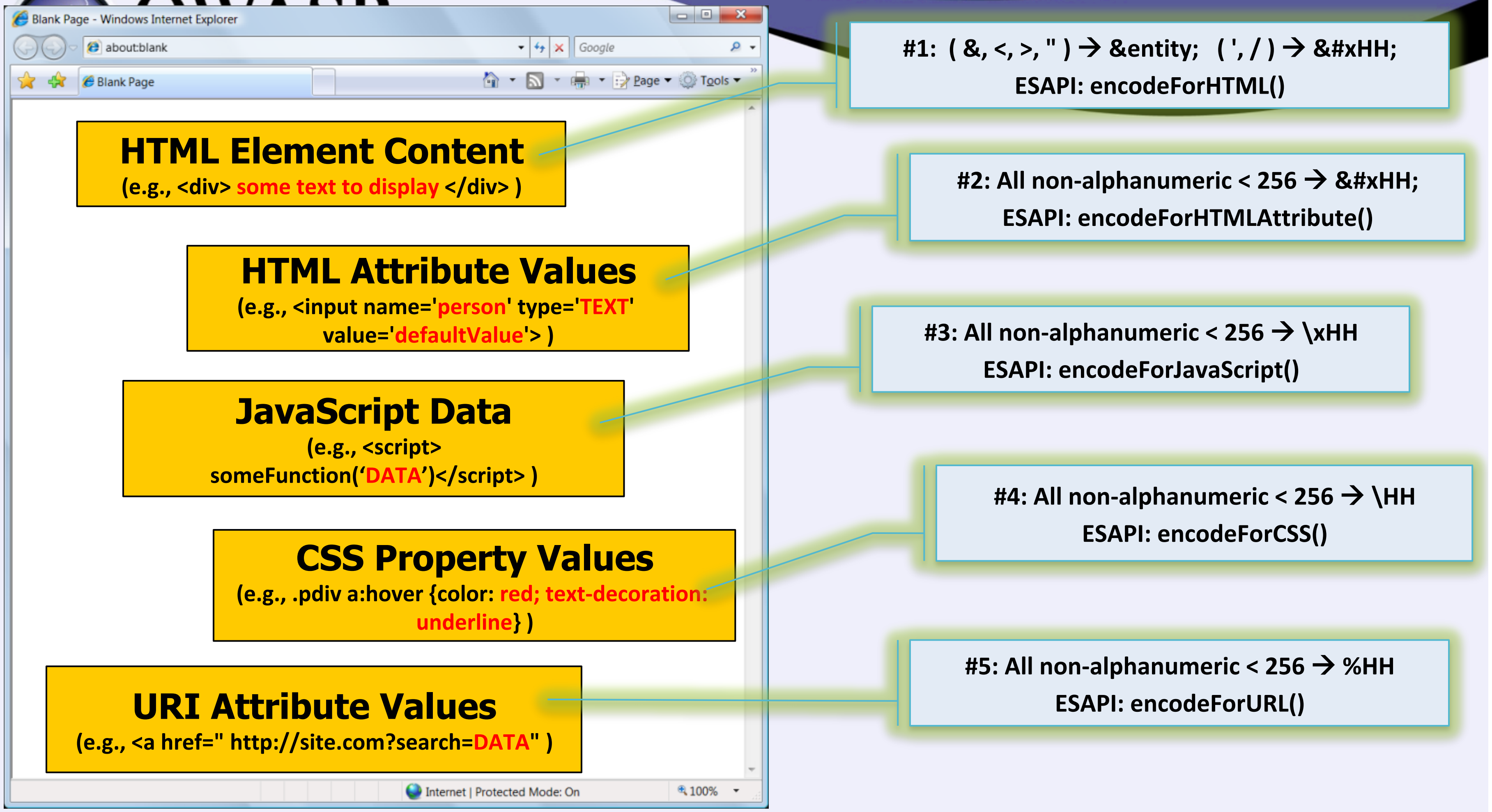
- **Recommendations**
  - **Eliminate Flaw**
    - **Don't include user supplied input in the output page**
  - **Defend Against the Flaw**
    - **Use Content Security Policy (CSP)**
    - **Primary Recommendation: Output encode all user supplied input (Use OWASP's Java Encoders to output encode)**

      **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


Blank Page - Windows Internet Explorer
about:blank
Google
Blank Page
Page
Tools
Internet | Protected Mode: On
100%

**HTML Element Content**
(e.g., <div> some text to display </div> )

**HTML Attribute Values**
(e.g., <input name='person' type='TEXT' value='defaultValue'> )

**JavaScript Data**
(e.g., <script> someFunction('DATA')</script> )

**CSS Property Values**
(e.g., .pdiv a:hover {color: red; text-decoration: underline} )

**URI Attribute Values**
(e.g., <a href=" http://site.com?search=DATA" )

#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

### Occurs any time…

- Data from attacker is deserialized back into application object

### Commonly identified in

- Major open source libraries/frameworks (e.g., Struts, Spring)
- Custom code as well (but less attractive to attackers)

### Typical Impact

- Arbitrary code execution caused as a side affect of attempt to construct application object from deserialized data
- Data tampering attacks where application object is successfully created, but object type or object values are not as expected/ not authorized for current user

## 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

## Libraries

- Stay on top of A9 – Use of known vulnerable components, as bulk of this risk involves use of such components

## Your custom code

- Ideal: Don’t send deserialized objects to untrusted users
- If you must, then
  - try to validate untrusted data BEFORE deserializing
    - Integrity seals recommended
  - Harden your deserialization mechanism to ONLY deserialize limited set of object types
  - Limit the size of such objects to avoid denial of service attacks

## Follow guidance from

- [https://www.owasp.org/index.php/Deserialization_Cheat_Sheet](https://www.owasp.org/index.php/Deserialization_Cheat_Sheet)

## 2017-A9 – Using Known Vulnerable Components

OWASP
The Open Web Application Security Project

### Vulnerable Components Are Common

- Some vulnerable components (e.g., framework libraries) can be identified and exploited with automated tools
- This expands the threat agent pool beyond targeted attackers to include chaotic actors

### Widespread

- Virtually every application has these issues because most development teams don’t focus on ensuring their components/libraries are up to date
- In many cases,  the developers don’t even know all the components they are using, never mind their versions. Component dependencies make things even worse

### Typical Impact

- Full range of weaknesses is possible, including injection, broken access control, XSS ...
- The impact could range from minimal to complete host takeover and data compromise

OWASP
The Open Web Application Security Project

## What Can You Do to Avoid This?

### Ideal (Detect Known Vulnerable Libraries)

- Automation checks periodically (e.g., nightly build) to see if your libraries have <u>known vulnerabilities</u>
- Upgrade to avoid critical/exploitable vulnerabilities
  - Or mitigate in some other way
- Commercial Solutions: Numerous options now
  - Far more than in 2013 when 2013-A9 was first added
- Free: https://www.owasp.org/index.php/OWASP_Dependency_Check

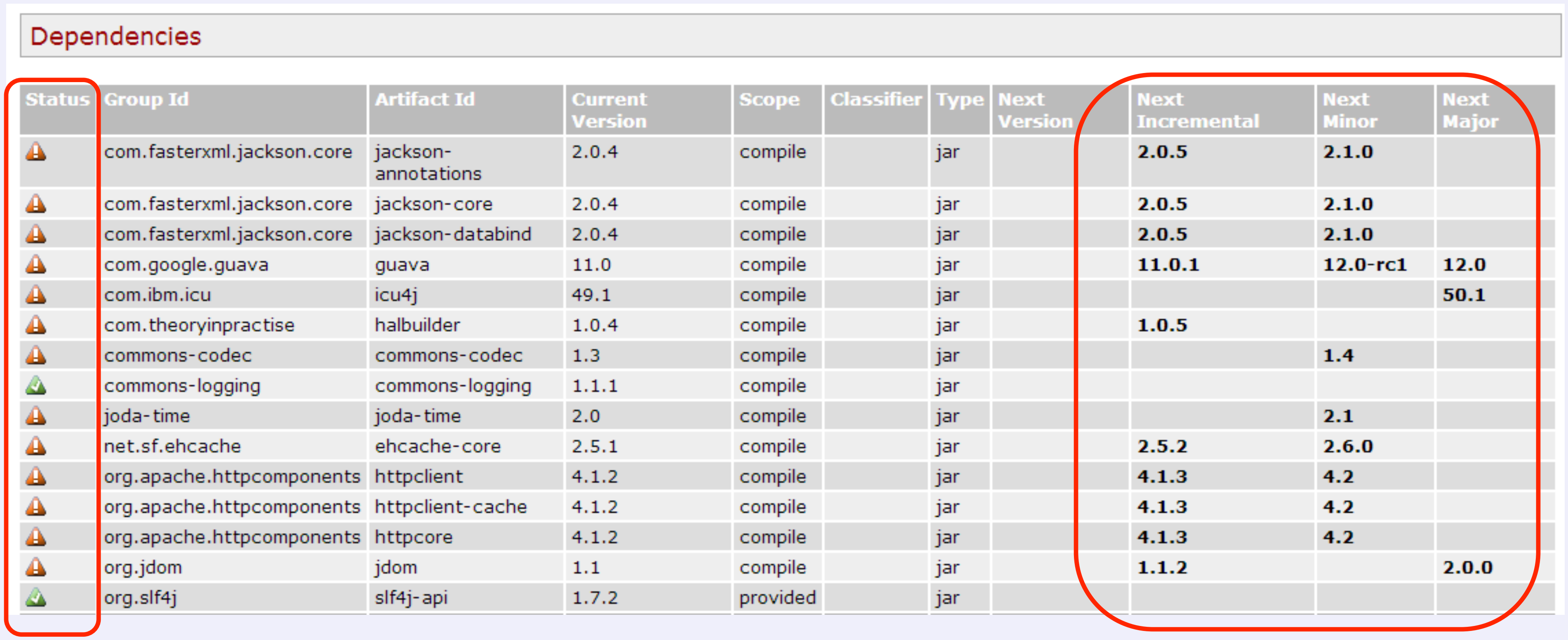
### Minimum (Identify out of date libraries)

- Automation checks periodically (e.g., nightly build) to see if your libraries are <u>out of date</u>
- If any are out of date, but you really don't want to upgrade, check to see if there are any known security issues with these out of data libraries
  - If so, upgrade those

## 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

### Web application are frequently very polite

- They usually just tell users about errors (failed attacks) and ask them to try again
- They (usually) DON'T log such attacks well
- They frequently don't monitor what they do log (or notice), and take action when unexpected behaviors are detected (Attacks!!)

### Typical Impact

- This enables attackers to try over and over until they eventually break in
- And when they do, they might not be noticed and/or how they succeeded and what they did cannot be determined

OWASP
The Open Web Application Security Project

## Providing Sufficient Logging & Monitoring

### Logging

- Do you have a standard logging system?
- Do all your standard security controls log all security critical events?
- Do all your custom security controls log all security critical events?
- Can you easily distinguish security vs. non-security log events?

### Monitoring

- Are application logs sent to a central monitoring location?
- Do you have monitoring software, that is running?
- Can it detect security critical events? Can it detect the accumulation of interesting events above defined thresholds?
- Can monitoring software raise alerts to system owners?
- Can monitoring software take action against obvious attackers?

### Follow guidance from

- https://www.owasp.org/index.php/OWASP_AppSensor_Project

## Summary: How do you address these problems?

- 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