OWASP Top-10 2013

Dave Wichers
OWASP Top 10 Project Lead
OWASP Board Member
COO/Cofounder, Aspect Security

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

OWASP Top Ten (2013 Edition)

OWASP
The Open Web Application Security Project

- *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

What Didn’t Change

It’s About Risks, Not Just Vulnerabilities

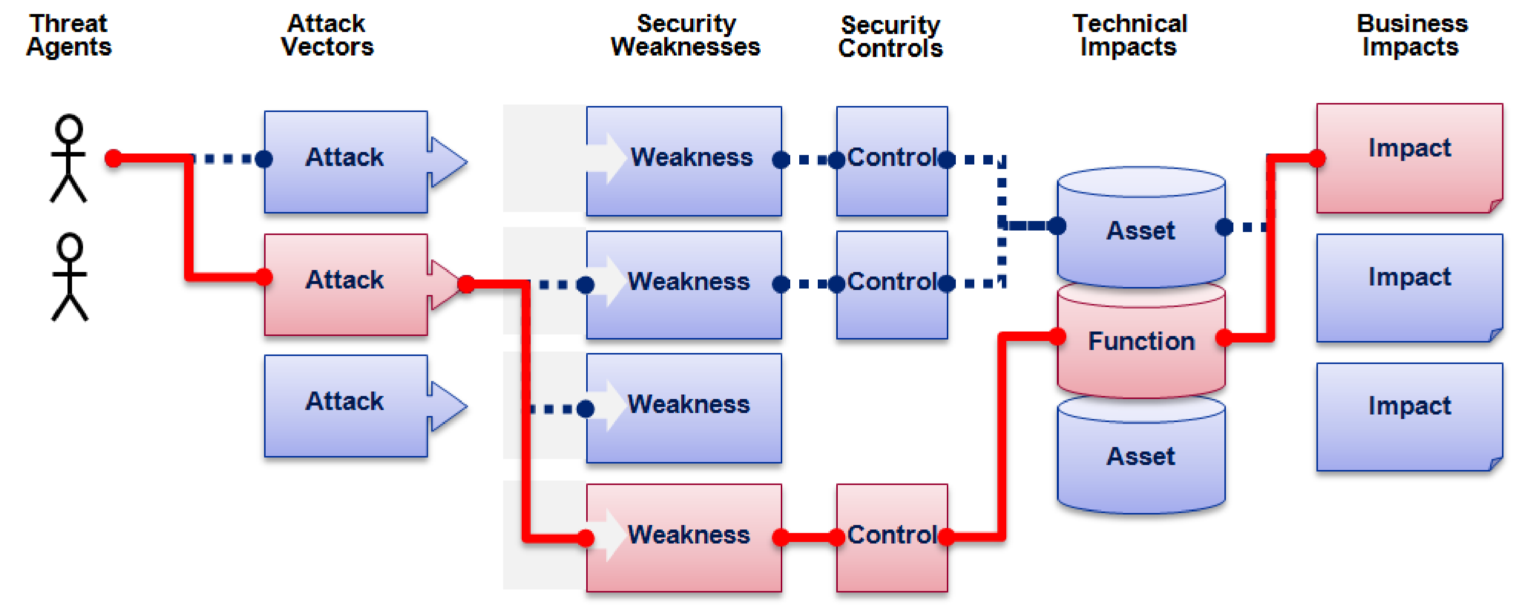
- Title is: “The Top 10 Most Critical Web Application Security Risks”

OWASP Top 10 Risk Rating Methodology

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

OWASP Top 10 Risk Rating Methodology

OWASP



Threat Agents

Attack Vectors

Security Weaknesses

Security Controls

Technical Impacts

Business Impacts

Attack

Attack

Attack

Weakness

Weakness

Weakness

Weakness

Control

Control

Control

Asset

Function

Asset

Impact

Impact

Impact

Threat Agent		Attack Vector	Weakness Prevalence	Weakness Detectability	Technical Impact	Business Impact
?	1	Easy	Widespread	Easy	Severe	?
	2	Average	Common	Average	Moderate	
	3	Difficult	Uncommon	Difficult	Minor	
		1	2	2	1	
			1.66	*	1	

Injection Example

1.66 weighted risk rating

What's Changed?

Risks Added, Risks Merged, Risks Reordered

- Reordered: 7
- Added: 1
- Merged: 2 merged into 1
- Broadened: 1

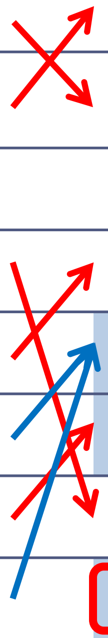
Development Methodology For 2013

- Same as 2010, but
- Used more sources of vulnerability data
- All vulnerability data made public by each provider

Development Methodology for Next Version?

- More transparency
- Requested vulnerability data format
- Earlier community involvement

Mapping from 2010 to 2013 Top 10



OWASP Top 10 – 2010 (old)	OWASP Top 10 – 2013 (New)
2010-A1 – Injection	2013-A1 – Injection
2010-A2 – Cross Site Scripting (XSS)	2013-A2 – Broken Authentication and Session Management
2010-A3 – Broken Authentication and Session Management	2013-A3 – Cross Site Scripting (XSS)
2010-A4 – Insecure Direct Object References	2013-A4 – Insecure Direct Object References
2010-A5 – Cross Site Request Forgery (CSRF)	2013-A5 – Security Misconfiguration
2010-A6 – Security Misconfiguration	2013-A6 – Sensitive Data Exposure
2010-A7 – Insecure Cryptographic Storage	2013-A7 – Missing Function Level Access Control
2010-A8 – Failure to Restrict URL Access	2013-A8 – Cross-Site Request Forgery (CSRF)
2010-A9 – Insufficient Transport Layer Protection	2013-A9 – Using Known Vulnerable Components (NEW)
2010-A10 – Unvalidated Redirects and Forwards (NEW)	2013-A10 – Unvalidated Redirects and Forwards
3 Primary Changes:	▪ Merged: 2010-A7 and 2010-A9 -> 2013-A6
▪ Added New 2013-A9: Using Known Vulnerable Components	▪ 2010-A8 broadened to 2013-A7

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…

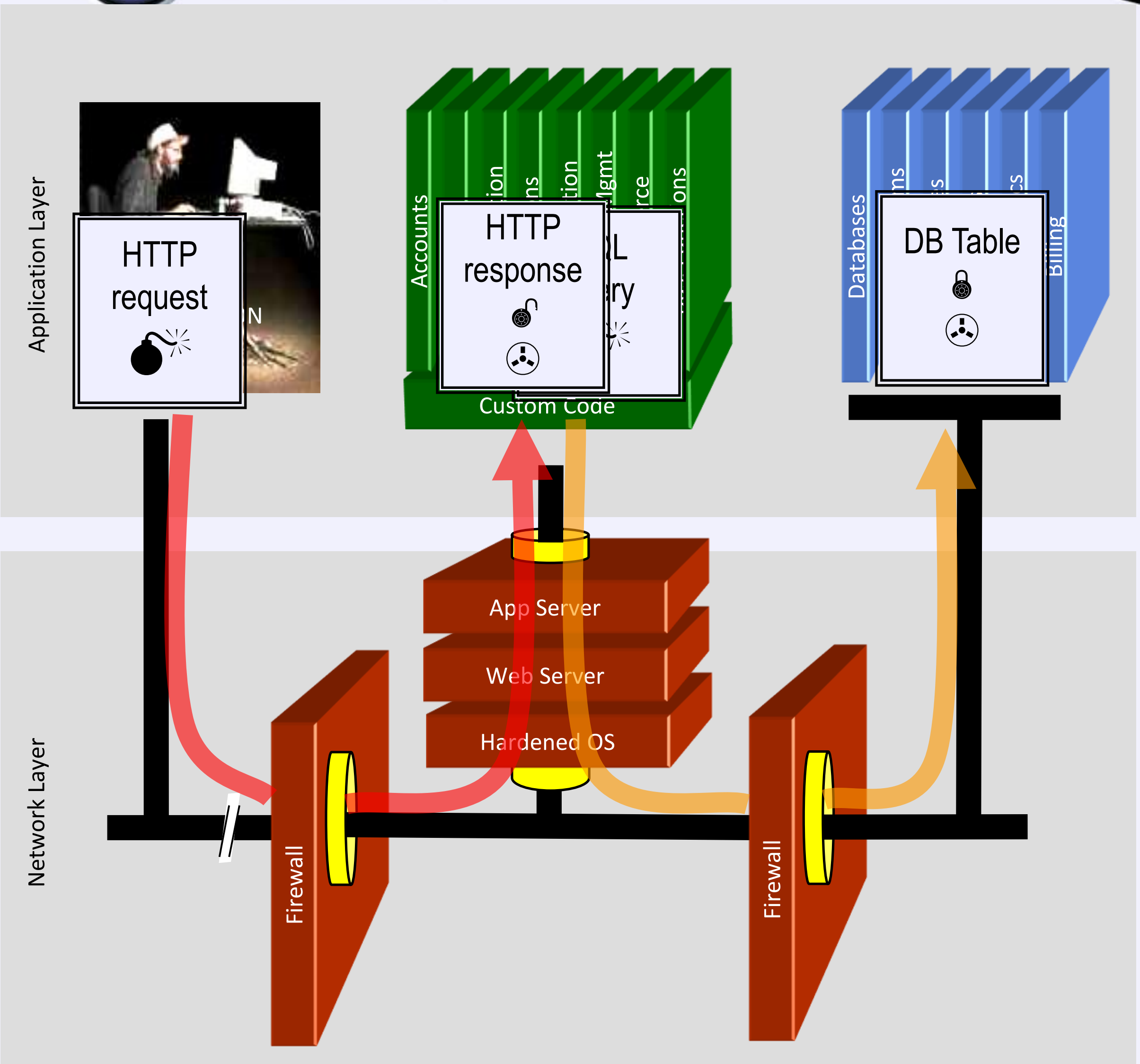
SQL injection is still quite common

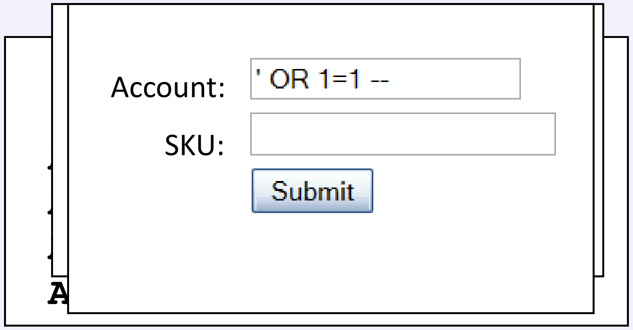
- Many applications still susceptible (really don't know why)
- Even though it's usually very simple to avoid

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

A1 – Avoiding Injection Flaws

OWASP
The Open Web Application Security Project

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

References

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

OWASP
The Open Web Application Security Project

2013-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
  - and it 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

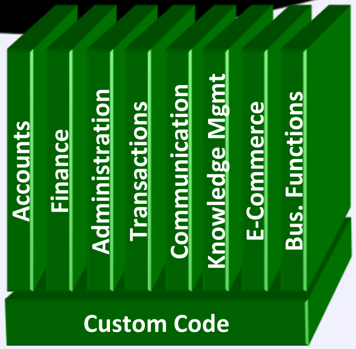
- Change my password, remember my password, forgot my password, secret question, logout, email address, etc…

Typical Impact

- User accounts compromised or user sessions hijacked

Broken Authentication Illustrated

1 User sends credentials

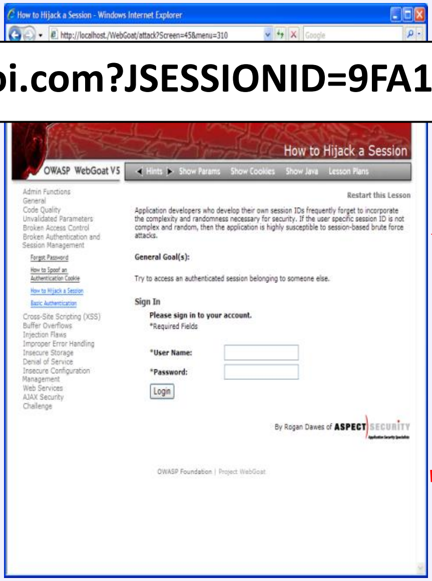


www.boi.com?JSESSIONID=9FA1DB9EA...

Site uses URL rewriting

(i.e., put session in URL)

2



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:**

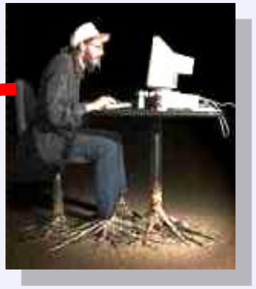
By Rogan Dawes of ASPECT SECURITY

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

A2 – Avoiding Broken Authentication and Session Management

Verify your architecture

- Authentication should be simple, centralized, and standardized
- 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
- Verify that logoff actually destroys the session
- Use OWASP's WebScarab to test the implementation

Follow the guidance from

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

OWASP

The Open Web Application Security Project

2013-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)

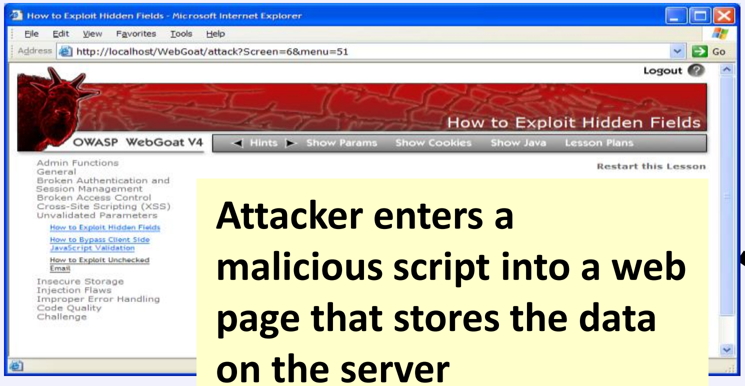
Typical Impact

- 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

OWASP
The Open Web Application Security Project

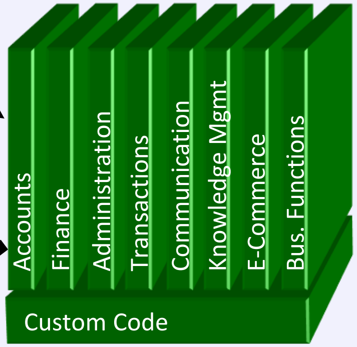
1 Attacker sets the trap – update my profile


How to Exploit Hidden Fields - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address
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

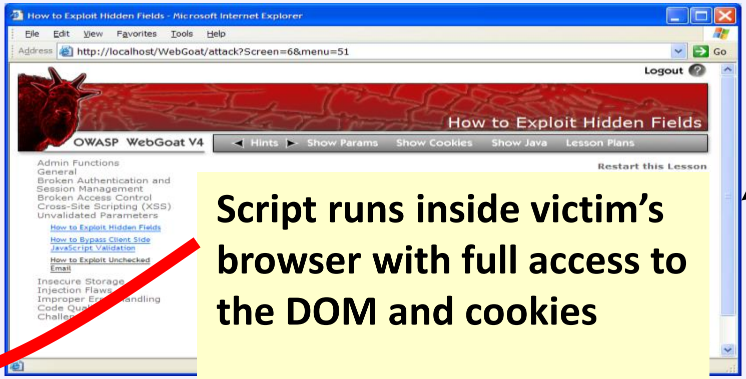
http://localhost/WebGoat/attack?Screen=6&menu=51

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


How to Exploit Hidden Fields - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address
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 Er andling
Code Qua
Challe

http://localhost/WebGoat/attack?Screen=6&menu=51

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

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 ESAPI or Java Encoders to output encode)**

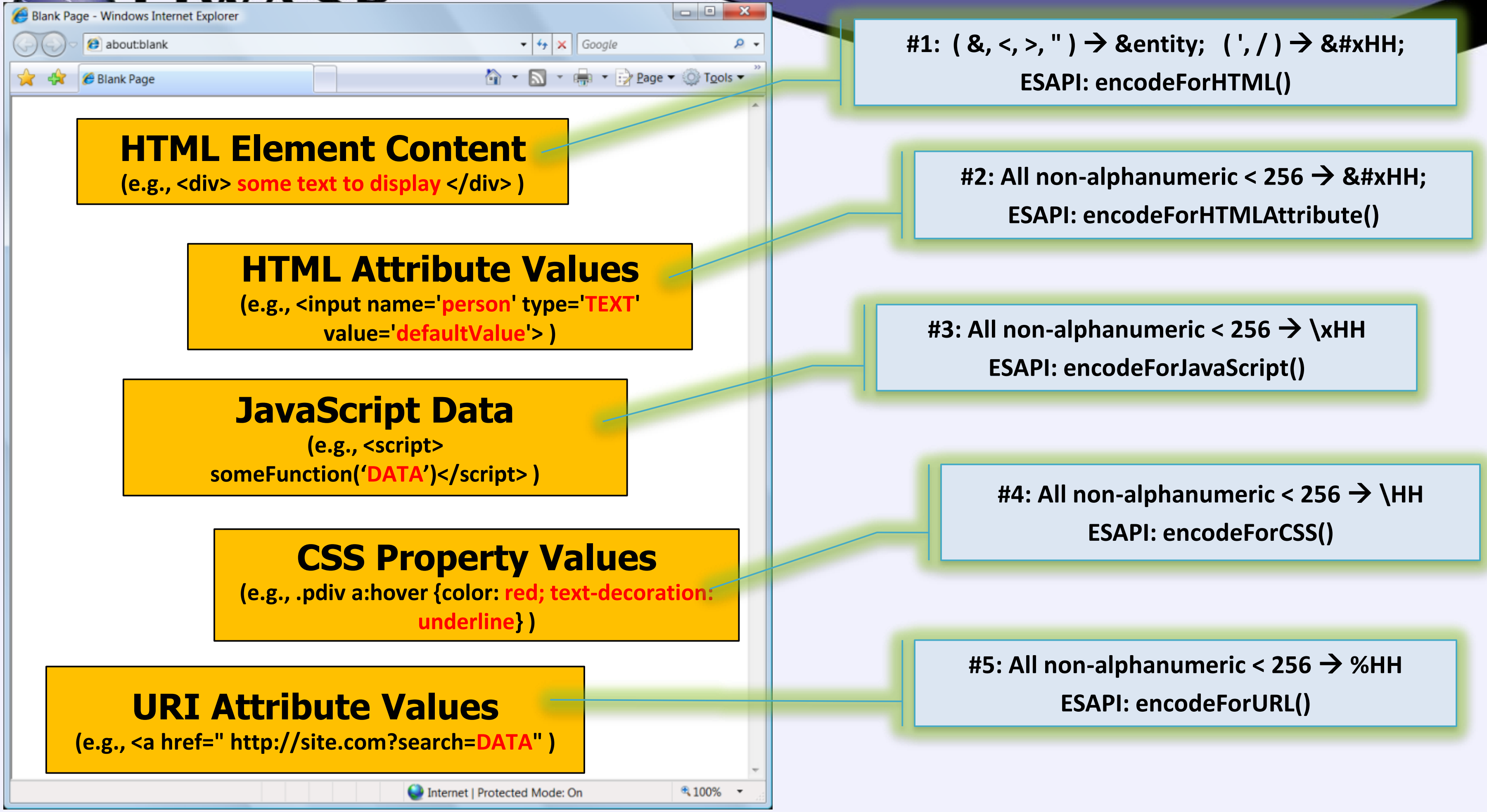
      https://www.owasp.org/index.php/ESAPI

      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 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

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

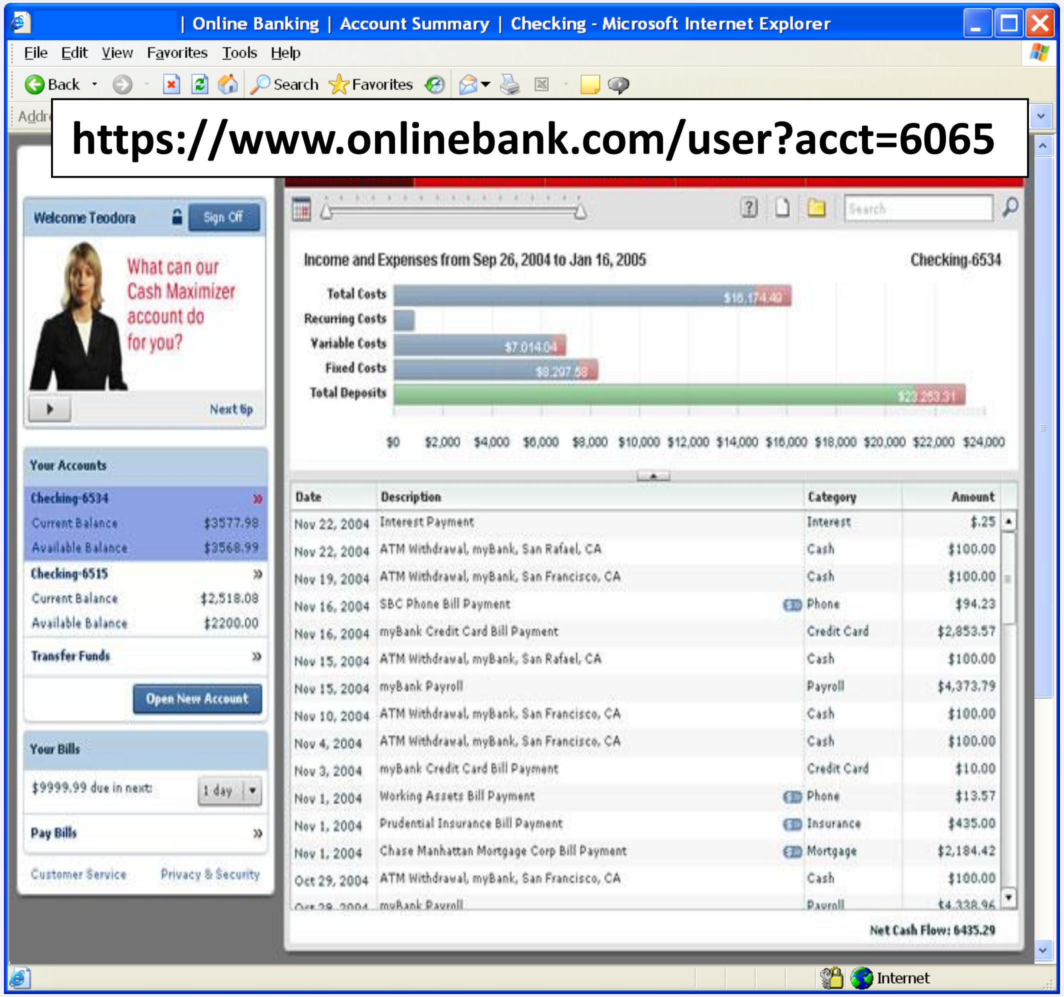
A common mistake …

- 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

Insecure Direct Object References Illustrated


| Online Banking | Account Summary | Checking - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Search
Favorites
Address
Search
Internet

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

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

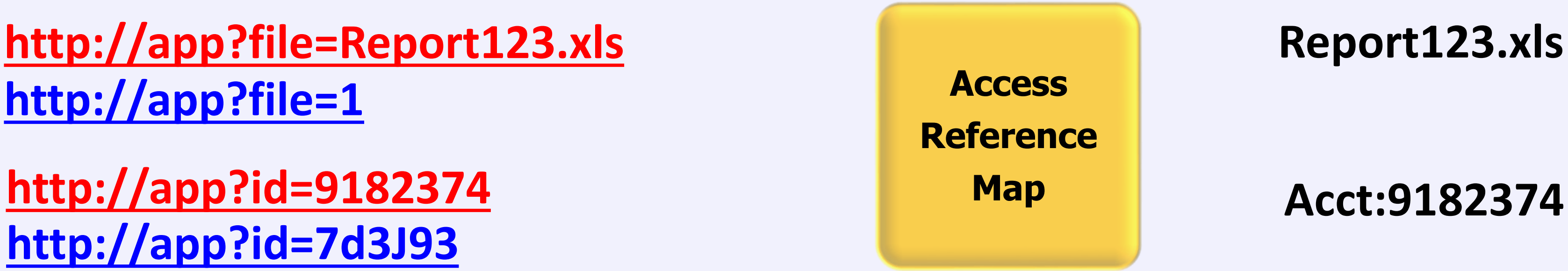
- He modifies it to a nearby number

  ?acct=6066

- Attacker views the victim's account information

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?file=1

http://app?id=9182374
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
    - Query constraints work great!
  - Verify the requested mode of access is allowed to the target object (e.g., read, write, delete)

2013-A5 – 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

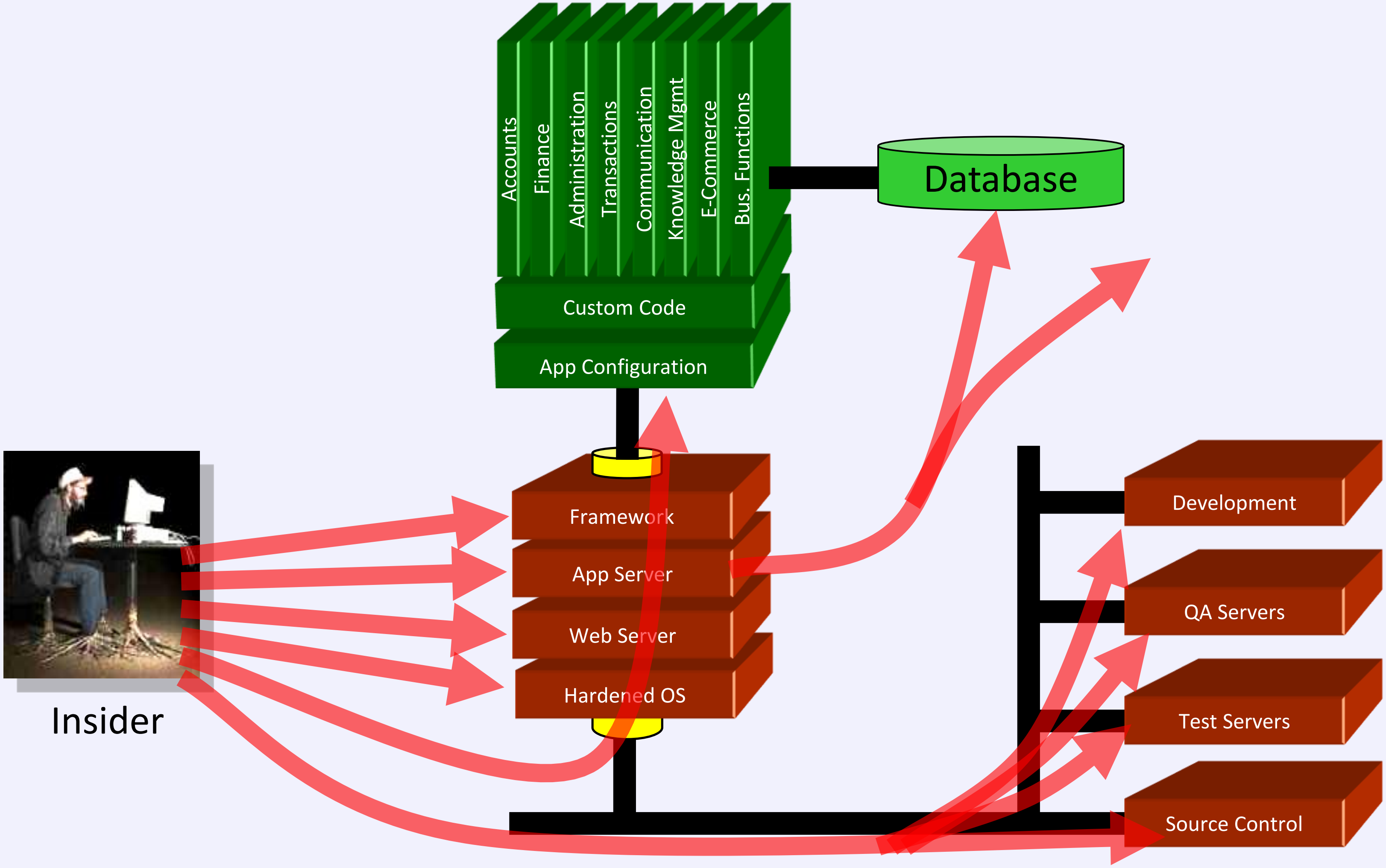
CM must extend to all parts of the application

- 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



Avoiding Security Misconfiguration

- 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

2013-A6 – Sensitive Data Exposure

OWASP
The Open Web Application Security Project

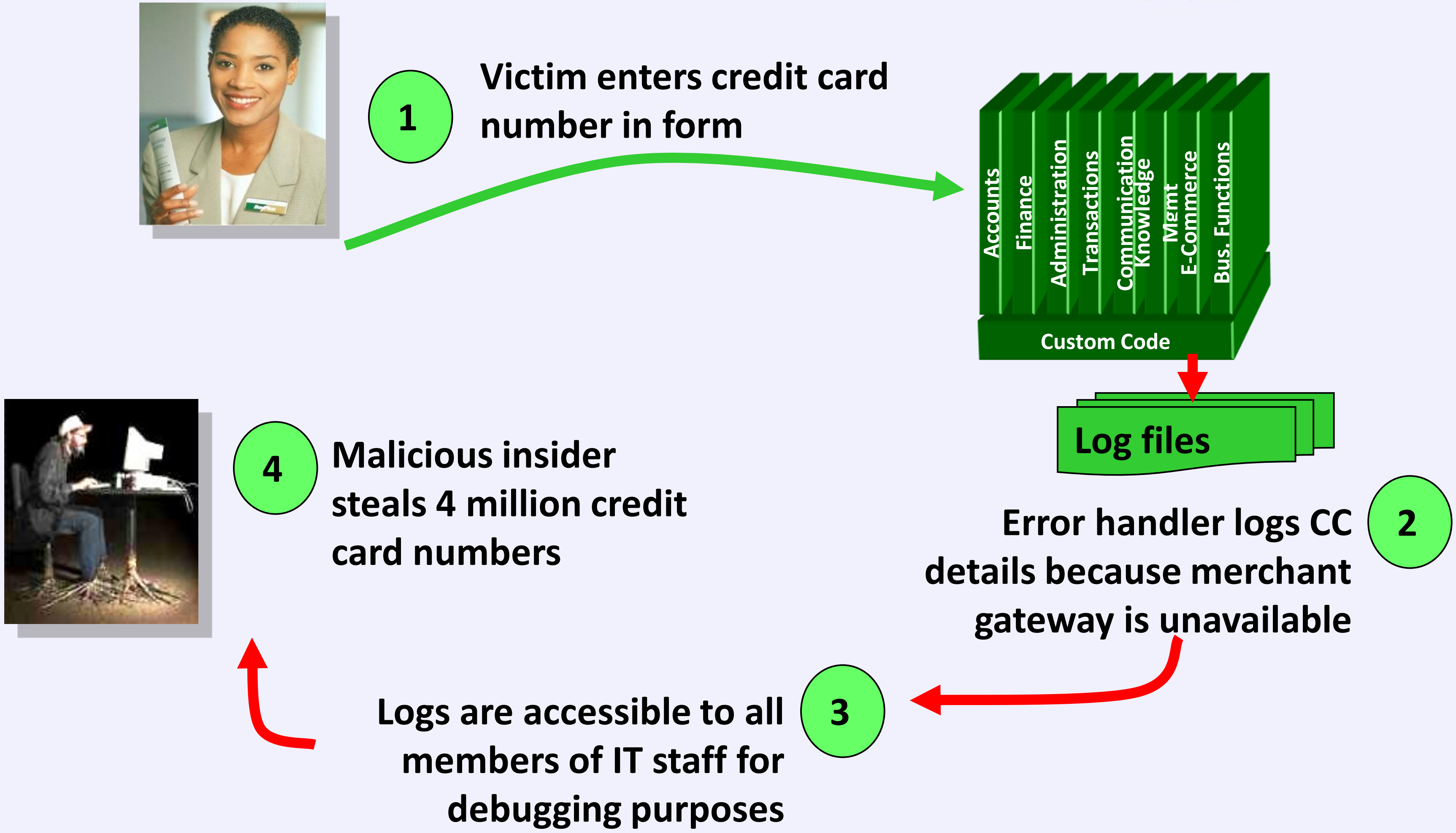
Storing and transmitting sensitive data insecurely

- 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



1 Victim enters credit card number in form

2 Error handler logs CC details because merchant gateway is unavailable

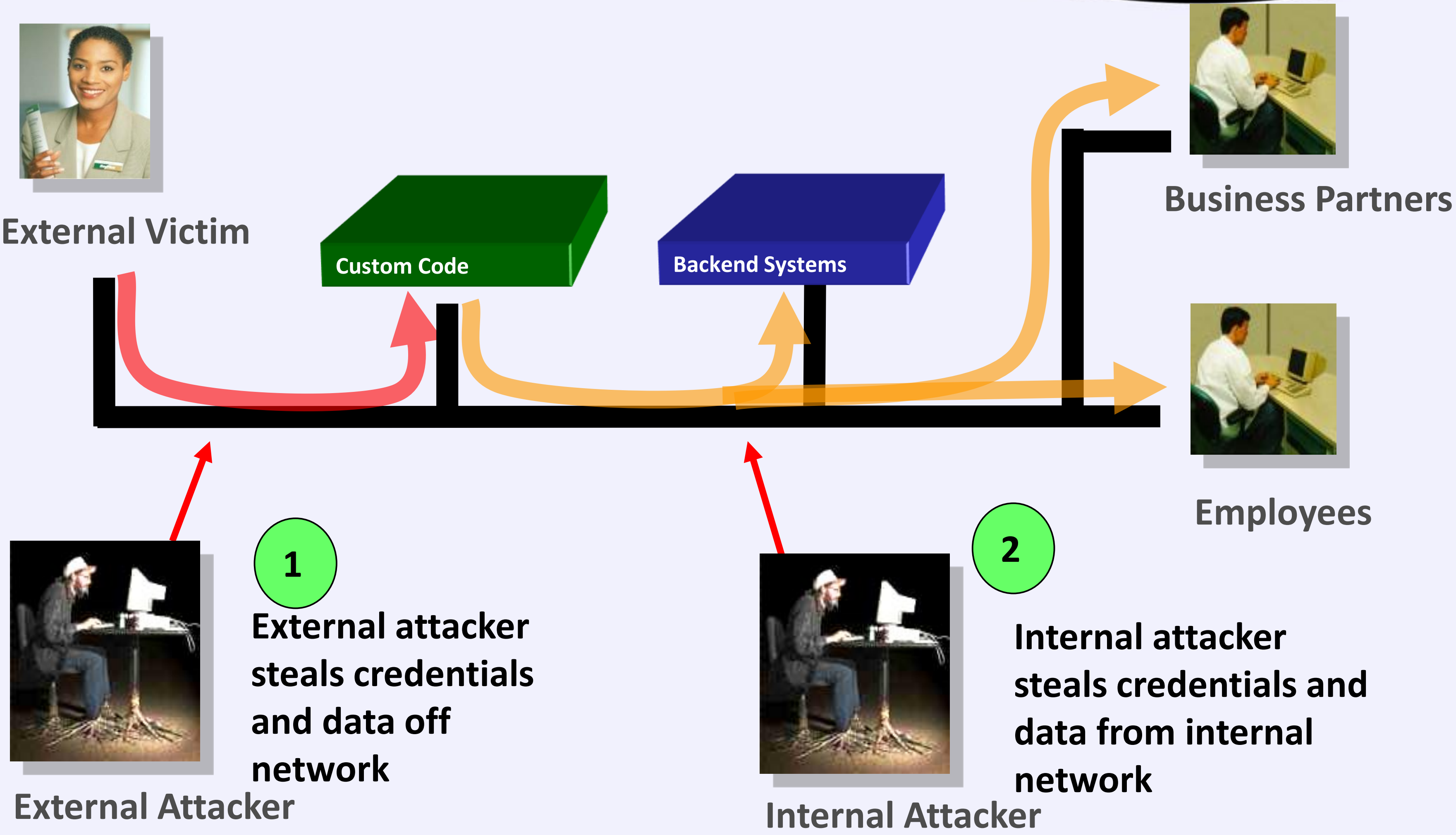
3 Logs are accessible to all members of IT staff for debugging purposes

4 Malicious insider steals 4 million credit card numbers

Avoiding Insecure Cryptographic Storage

- 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
- 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



1 External attacker steals credentials and data off network

2 Internal attacker steals credentials and data from internal network

Avoiding Insufficient Transport Layer Protection

- **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
- See: http://www.owasp.org/index.php/Transport_Layer_Protection_Cheat_Sheet for more details

2013-A7 – Missing Function Level Access Control

OWASP
The Open Web Application Security Project

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 …

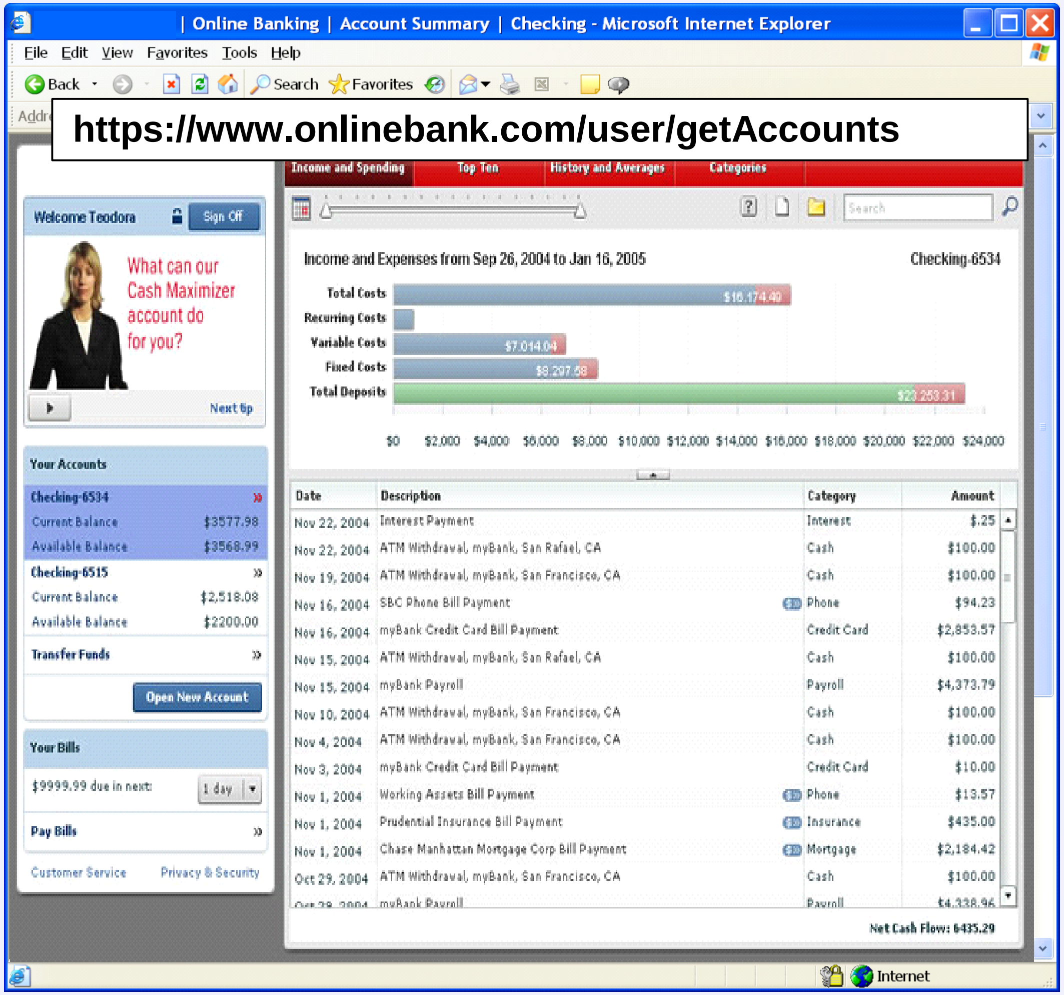
- Displaying only authorized links and menu choices
- This is called presentation layer access control, and doesn’t work
- Attacker simply forges direct access to ‘unauthorized’ pages

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

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:

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

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.)
- 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 that each URL (plus any parameters) referencing a function is protected by
    - An external filter, like Java EE web.xml or a commercial product
    - Or internal checks in YOUR code – e.g., use ESAPI's isAuthorizedForURL() method
  - Verify the server configuration disallows requests to unauthorized file types
  - Use OWASP's ZAP or your browser to forge unauthorized requests

2013-A8 – Cross Site Request Forgery (CSRF)

OWASP
The Open Web Application Security Project

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 Vulnerability Pattern

- The Problem
  - Web browsers automatically include most credentials with each request
  - Even for requests caused by a form, script, or image on another site
- All sites relying solely on automatic credentials are vulnerable!
  - (almost all sites are this way)
- Automatically Provided Credentials
  - Session cookie
  - Basic authentication header
  - IP address
  - Client side SSL certificates
  - Windows domain authentication

CSRF Illustrated

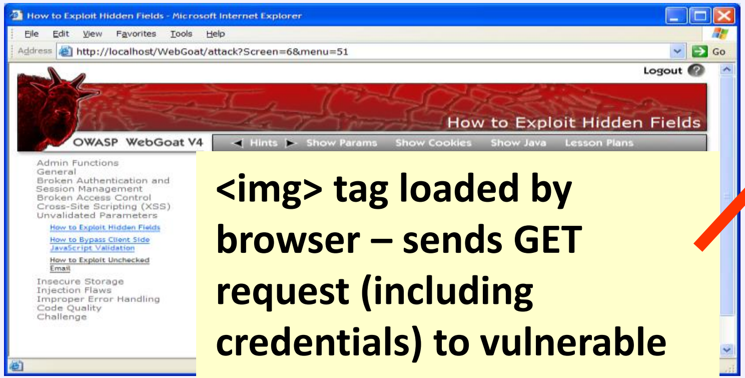
OWASP
The Open Web Application Security Project

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

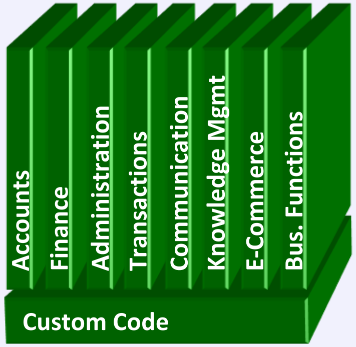
2 While logged into vulnerable site, victim views attacker site


How to Exploit Hidden Fields - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address
Go
Logout
How to Exploit Hidden Fields
OWASP WebGoat V4
Hints
Show Params
Show Cookies
Show Java
Lesson Plans
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

http://localhost/WebGoat/attack?Screen=6&menu=51

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

Application with CSRF vulnerability



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

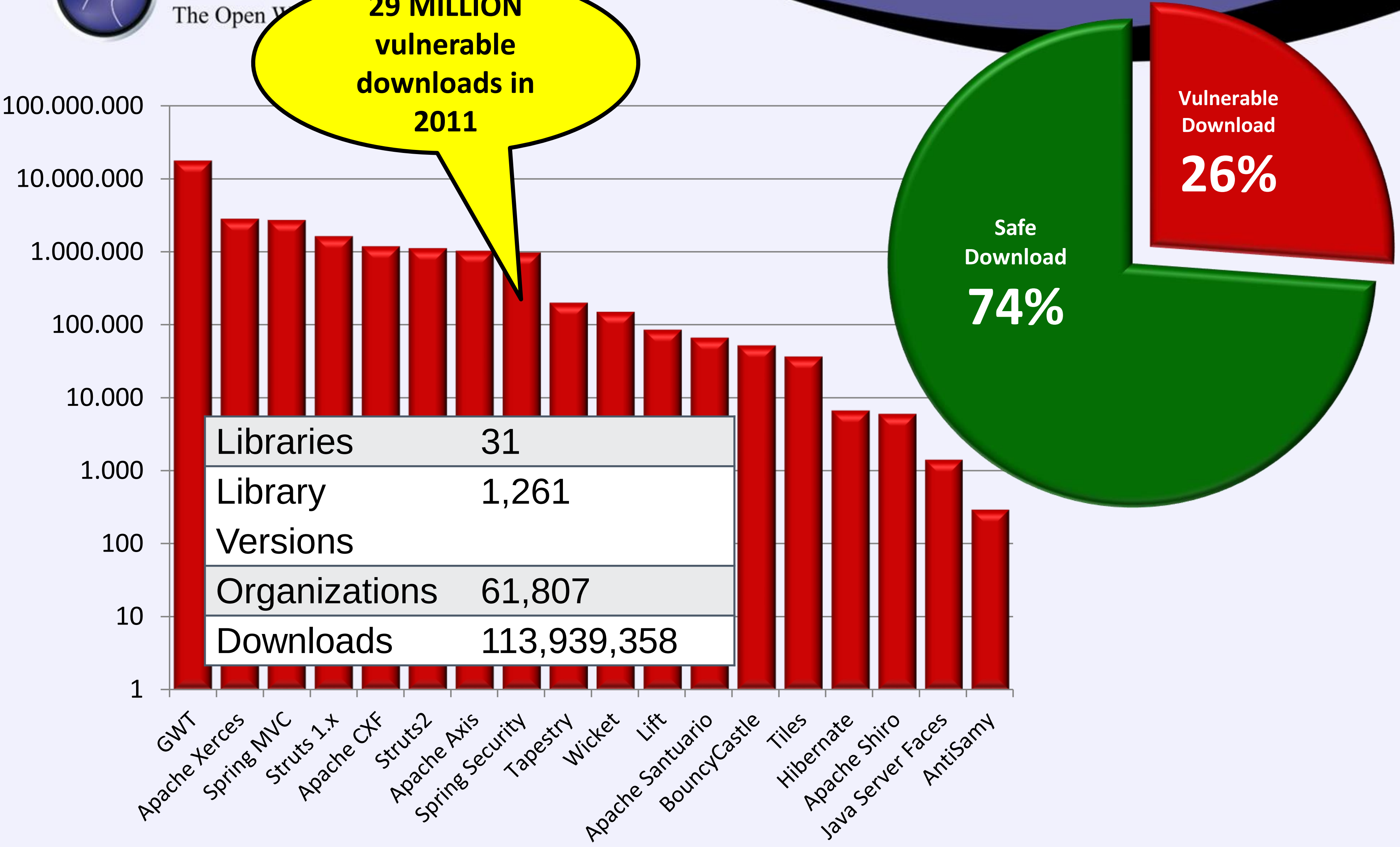
A8 – Avoiding CSRF Flaws

OWASP
The Open Web Application Security Project

- Add a secret, not automatically submitted, token to ALL sensitive requests
  - This makes it impossible for the attacker to spoof the request
    - (unless there's an XSS hole in your application)
  - Tokens should be cryptographically strong or random
- Options
  - 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` …
  - Beware exposing the token in a referer header
    - Hidden fields are recommended
  - 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)
- 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



29 MILLION vulnerable downloads in 2011

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

https://www.aspectsecurity.com/news/press/the-unfortunate-reality-of-insecure-libraries

2013-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

What Can You Do to Avoid This?

OWASP
The Open Web Application Security Project

Ideal

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

Minimum

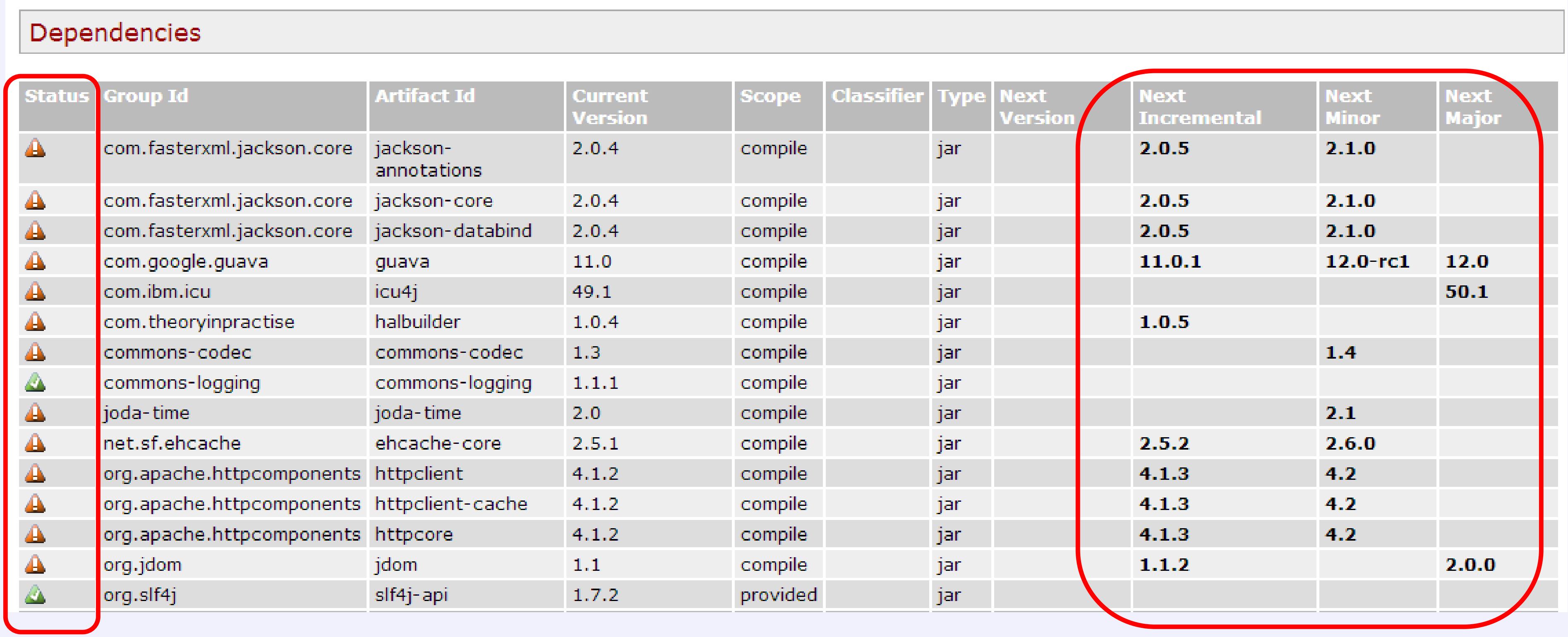
- By hand, periodically check to see if your libraries are out of date and upgrade those that are
- If any are out of date, but you really don’t want to upgrade, check to see if there are any known security issues with these out of data libraries
  - If so, upgrade those

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

OWASP
The Open Web Application Security Project

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

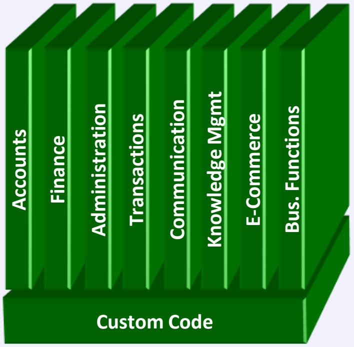
Unvalidated Redirect Illustrated

OWASP
The Open Web Application Security Project

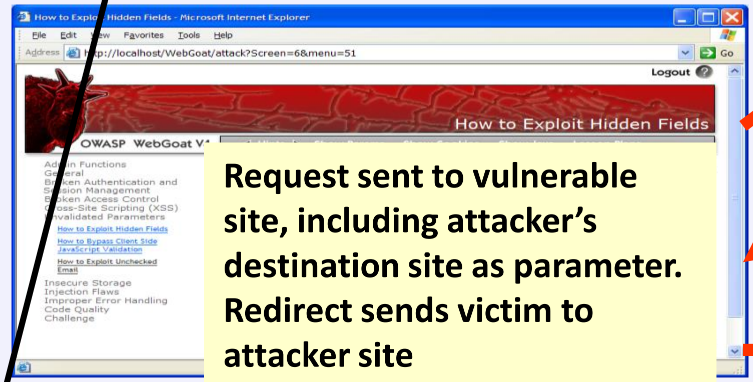
1 Attacker sends attack to victim via email or webpage

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.

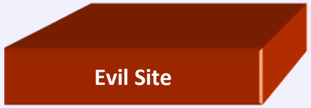
3 Application redirects victim to attacker's site



2 Victim clicks link containing unvalidated parameter


How to Explo Hidden Fields - Microsoft Internet Explorer
File Edit ew Favorites Tools Help
Address
tp://localhost/WebGoat/attack?Screen=6&menu=51
Go
Logout
How to Exploit Hidden Fields
OWASP WebGoat V
Ad in Functions
Ge eral
Br ken Authentication and
Se sion Management
B oken Access Control
oss-Site Scripting (XSS)
nvalidated 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



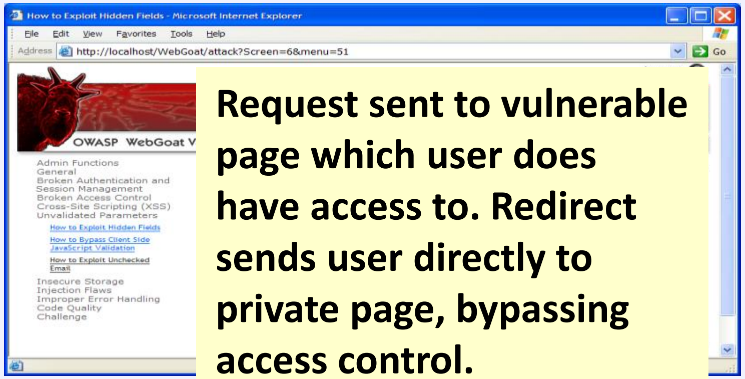
http://www.irs.gov/taxrefund/claim.jsp?year=2006 & … &dest=www.evilsite.com

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

Unvalidated Forward Illustrated

OWASP
The Open Web Application Security Project

1 Attacker sends attack to vulnerable page they have access to


How to Exploit Hidden Fields - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://localhost/WebGoat/attack?Screen=6&menu=51
Go
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 page which user does have access to. Redirect sends user directly to private page, bypassing access control.

2 Application authorizes request, which continues to vulnerable page

Filter

```
public void doPost( HttpServletRequest request,
HttpServletResponse response) {
    try {
        String target = request.getParameter( "dest" ) );
        ...
        request.getRequestDispatcher( target
        ).forward(request, response);
    }
    catch ( ...
```

```
public void sensitiveMethod(
HttpServletRequest request,
HttpServletResponse response) {
    try {
        // Do sensitive stuff here.
        ...
    }
    catch ( ...
```

3 Forwarding page fails to validate parameter, sending attacker to unauthorized page, bypassing access control

A10 – Avoiding Unvalidated Redirects and Forwards

OWASP
The Open Web Application Security Project

- 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 valid and authorized 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)

- Some thoughts about protecting Forwards
  - Ideally, you'd call the access controller to make sure the user is authorized before you perform the forward (with ESAPI, this is easy)
  - With an external filter, like Siteminder, this is not very practical
  - Next best is to make sure that users who can access the original page are ALL authorized to access the target page.

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 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

Thank you
OWASP Top-10 2013