Securing Web Applications from Top Ten Security Threats

Key Security Concerns from Developer's Perspective

- **Authentication (Identity verification)**
  - Identify users, assign credentials to them
  - Making **sure that a user is who he claims to be**
- **Authorization (Access and Privileges)**
  - Assign rights and privileges to roles (sometimes to users)
- **Confidentiality and Data Integrity (Tamper-proofing)**
  - Secure data, only authorized users can access it
  - Encryption to protect the sensitive data from prying eyes in transit or in storage
    - Keys are used to sign, encrypt, decrypt information

Threats

- Web defacement

  ⇒ loss of reputation (clients, shareholders)

- Information disclosure (lost data confidentiality)

  e.g. business secrets, financial information, client database, medical data, government documents

- data loss (or lost data integrity)
- unauthorized access

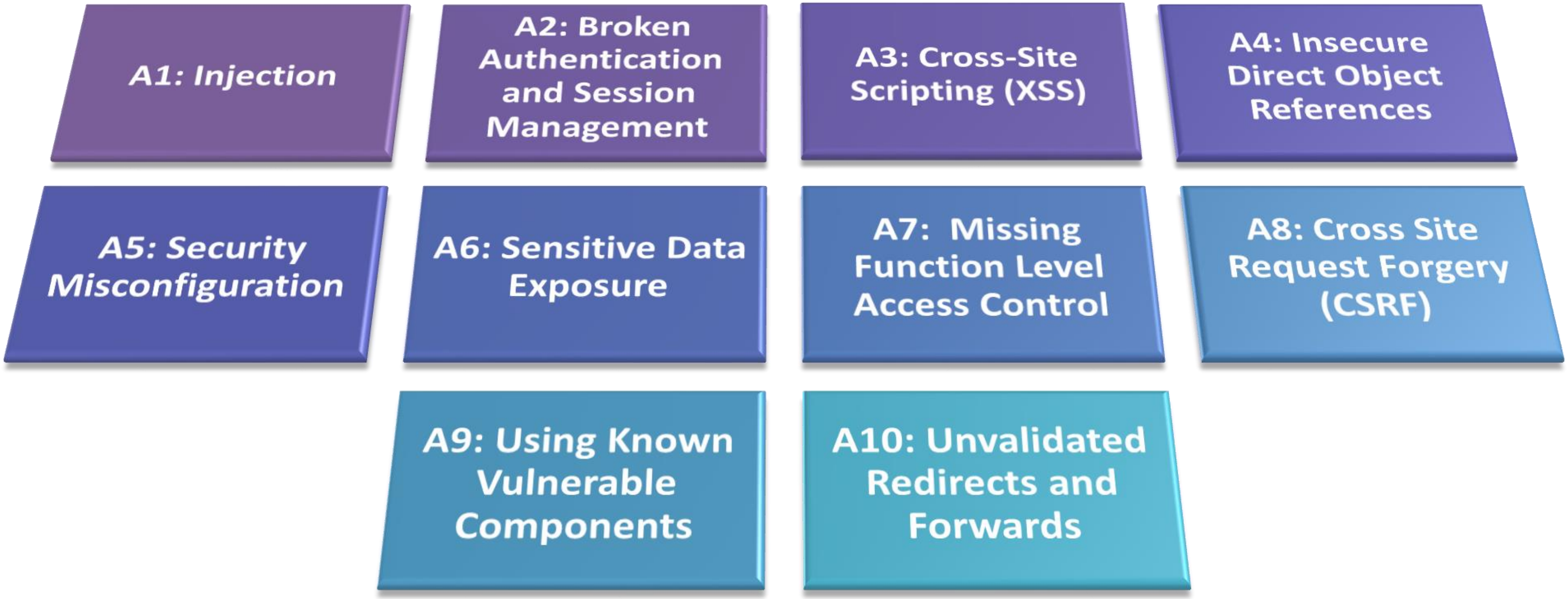
  ⇒ functionality of the application abused

- denial of service

  ⇒ loss of availability or functionality (and revenue)

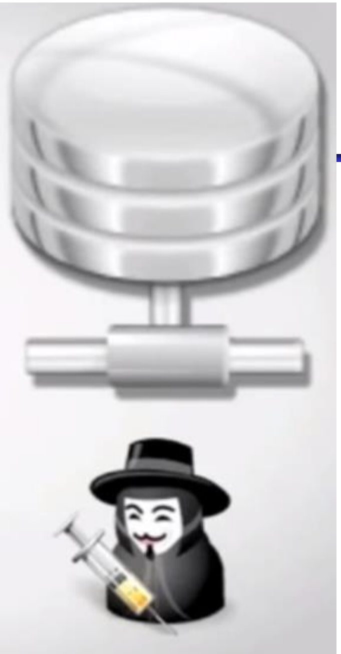
OWASP Top Ten (2013 Edition)
(Open Web Application Security Project)

http://owasp.org/index.php/Category:OWASP_Top_Ten_Project

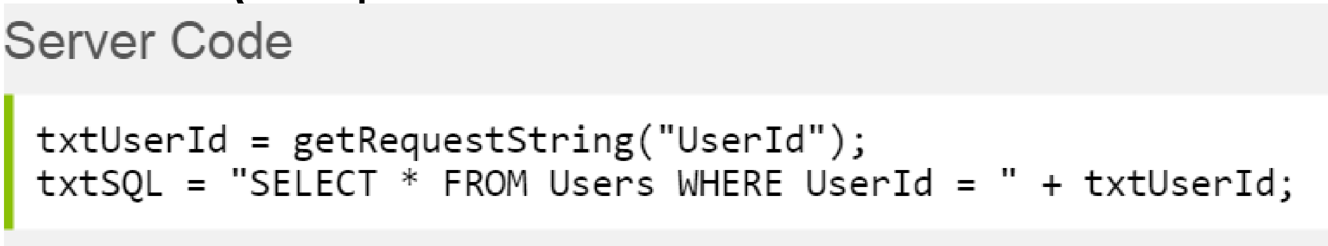


- A1: Injection
- A2: Broken Authentication and Session Management
- A3: Cross-Site Scripting (XSS)
- A4: Insecure Direct Object References
- A5: Security Misconfiguration
- A6: Sensitive Data Exposure
- A7: Missing Function Level Access Control
- A8: Cross Site Request Forgery (CSRF)
- A9: Using Known Vulnerable Components
- A10: Unvalidated Redirects and Forwards

A1: Injection flaws

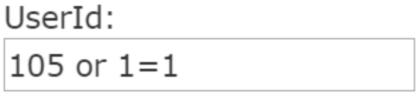


- Executing code provided (injected) by attacker
  - SQL injection



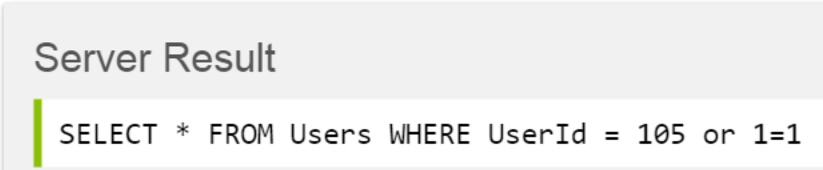
Server Code

```
txtUserId = getRequestString("UserId");
txtSQL = "SELECT * FROM Users WHERE UserId = " + txtUserId;
```



UserId:

105 or 1=1



Server Result

```
SELECT * FROM Users WHERE UserId = 105 or 1=1
```

- Solutions:
  - validate user input
  - escape values (use escape functions)
  - use parameterized queries (SQL)
  - enforce least privilege when accessing a DB, OS etc.

`'` -> `\'`

A2: Broken Authentication & session Management

Hacker uses flaws in the authentication or session management functions to **impersonate** other users:

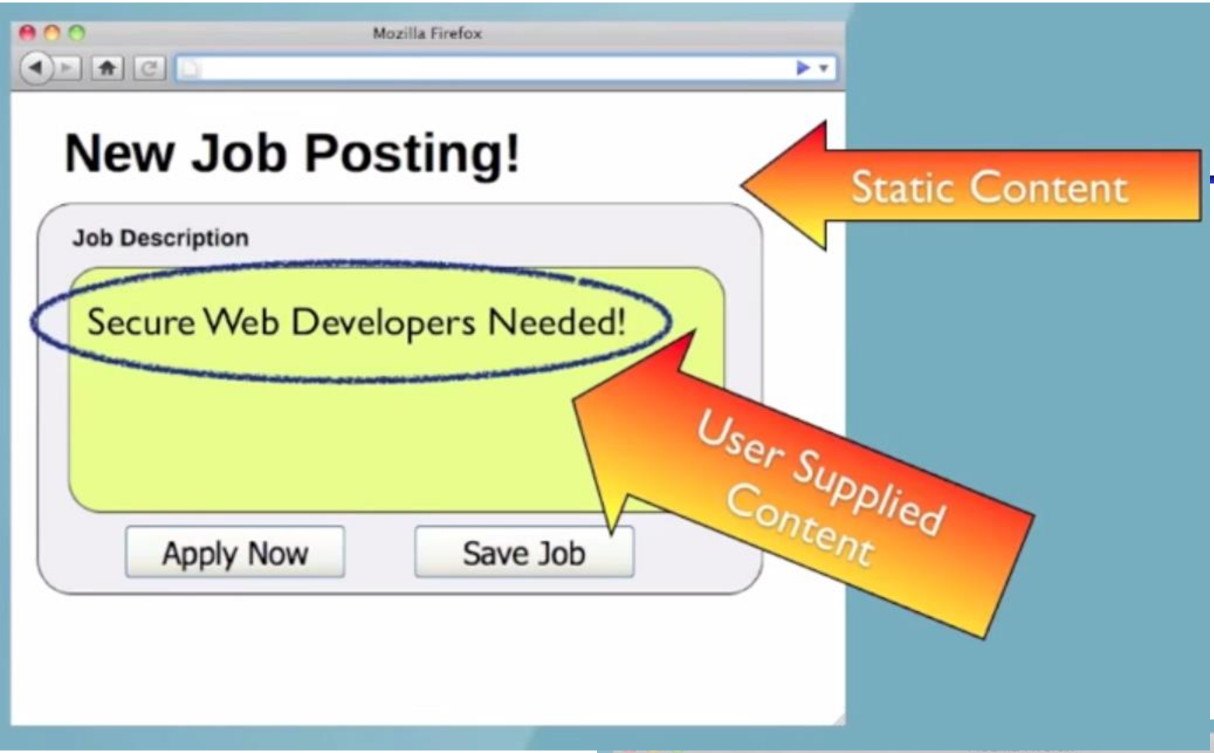
- **Scenario #1:** Session timeout isn’t set properly. Instead of selecting “logout” the user simply closes the browser tab and walks away. Attacker uses the same browser an hour later, and that browser is still authenticated.
- **Scenario #2:** Session hijacking by stealing session id (e.g., using eavesdropping if not https) then hacker uses this authenticated session id to connect to application without needing to enter user name and password.
- **Scenario #3:** Hacker gains access to the system's password database. User passwords are not properly hashed, exposing every users' password to the attacker.
- **Scenario #4:** Brute force password guessing, more likely using tools that generate random passwords.

A2 - Solutions

- User authentication credentials should be **protected when stored** using hashing or encryption.
- Session IDs should not be exposed in the URL, **use cookies for storing session ID**
- Session IDs should **timeout**. User sessions or authentication tokens should get properly invalidated during logout.
- Session IDs should be regenerated after successful login.
- Passwords, session IDs, and other credentials should not be sent over unencrypted connections (**use https** for login)
- Enforce **minimum passwords length and complexity** makes a password difficult to guess
- Enforce **account disabling** after an established number of invalid login attempts (e.g., three attempts is common).

A3: Cross-site scripting (XSS)

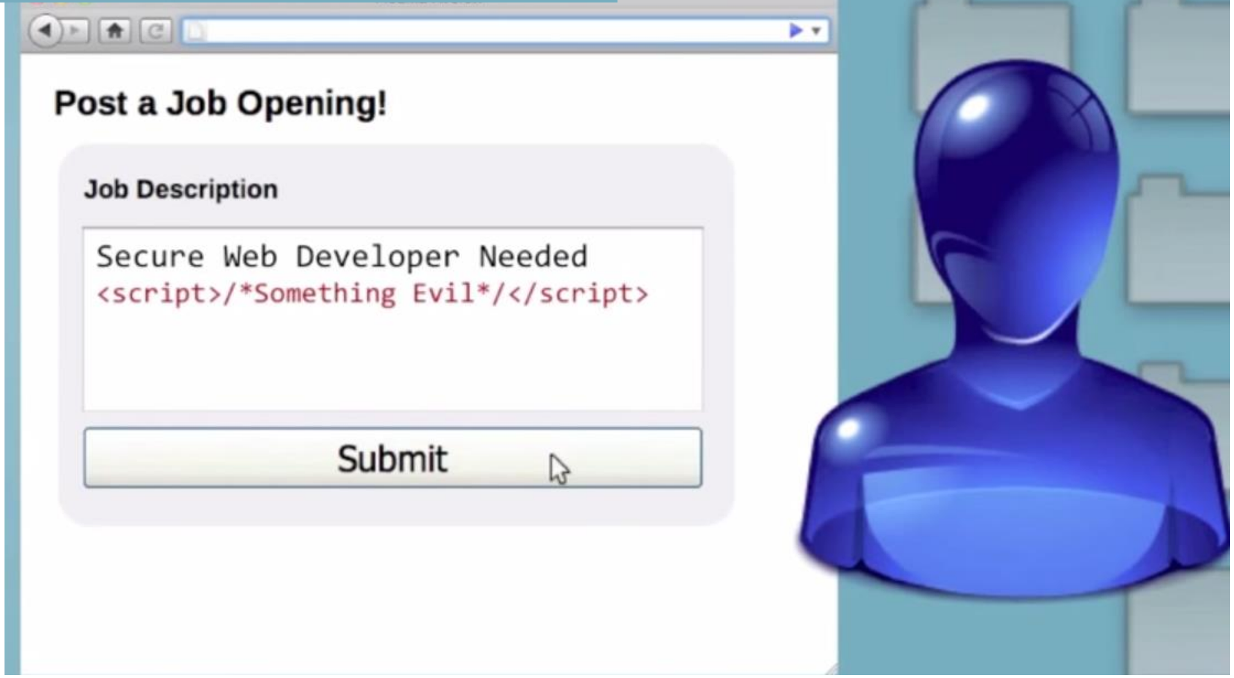
- Cross-site scripting (XSS) vulnerability
  - an application takes user input and sends it to a Web browser without validation or encoding
  - attacker can execute JavaScript code in the victim's browser
  - to hijack user sessions, deface web sites etc.
- Solution: validate user input, encode HTML output



New Job Posting!

Job Description

Secure Web Developers Needed!

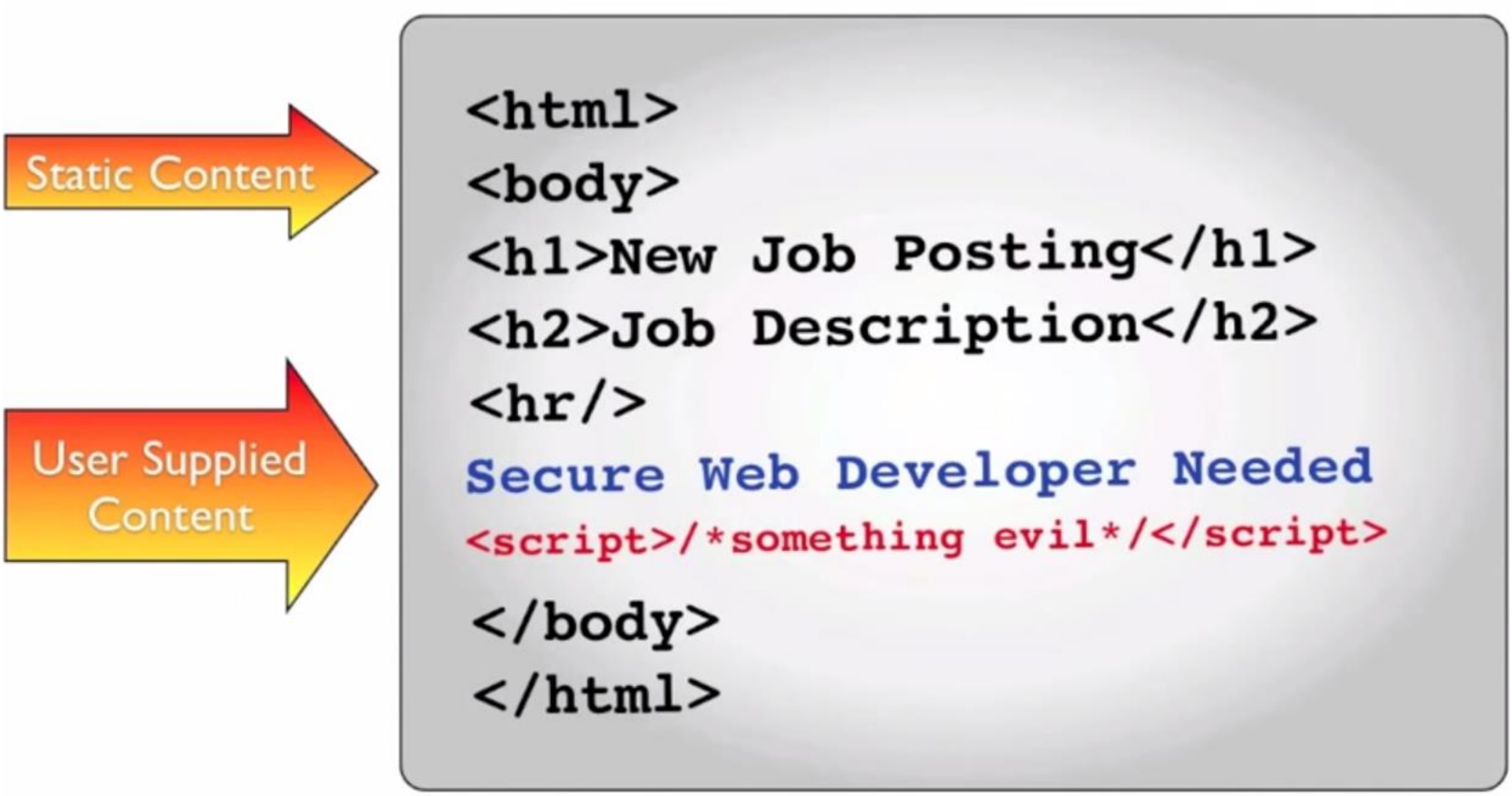


Post a Job Opening!

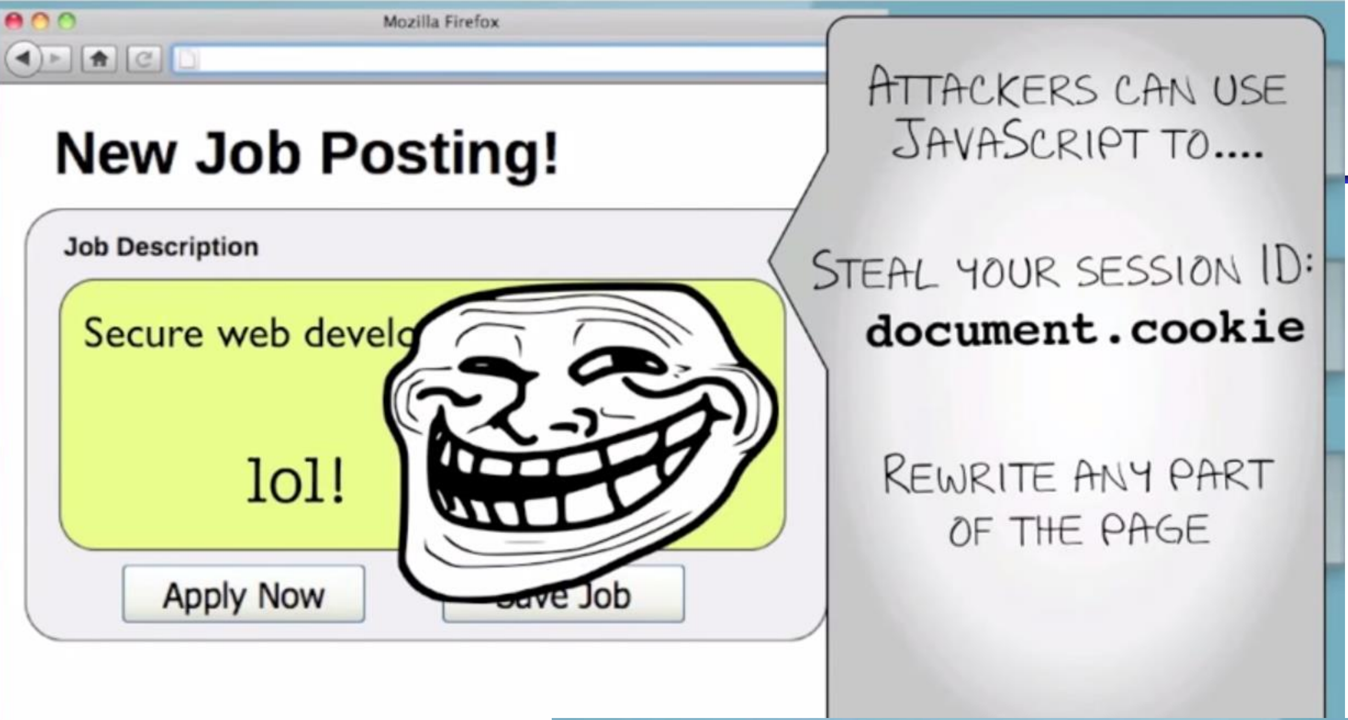
Job Description

```
Secure Web Developer Needed
<script>/*Something Evil*/</script>
```

Visitors will get the evil script



```
<html>
<body>
<h1>New Job Posting</h1>
<h2>Job Description</h2>
<hr/>
Secure Web Developer Needed
<script>/*something evil*/</script>
</body>
</html>
```



New Job Posting!

Job Description

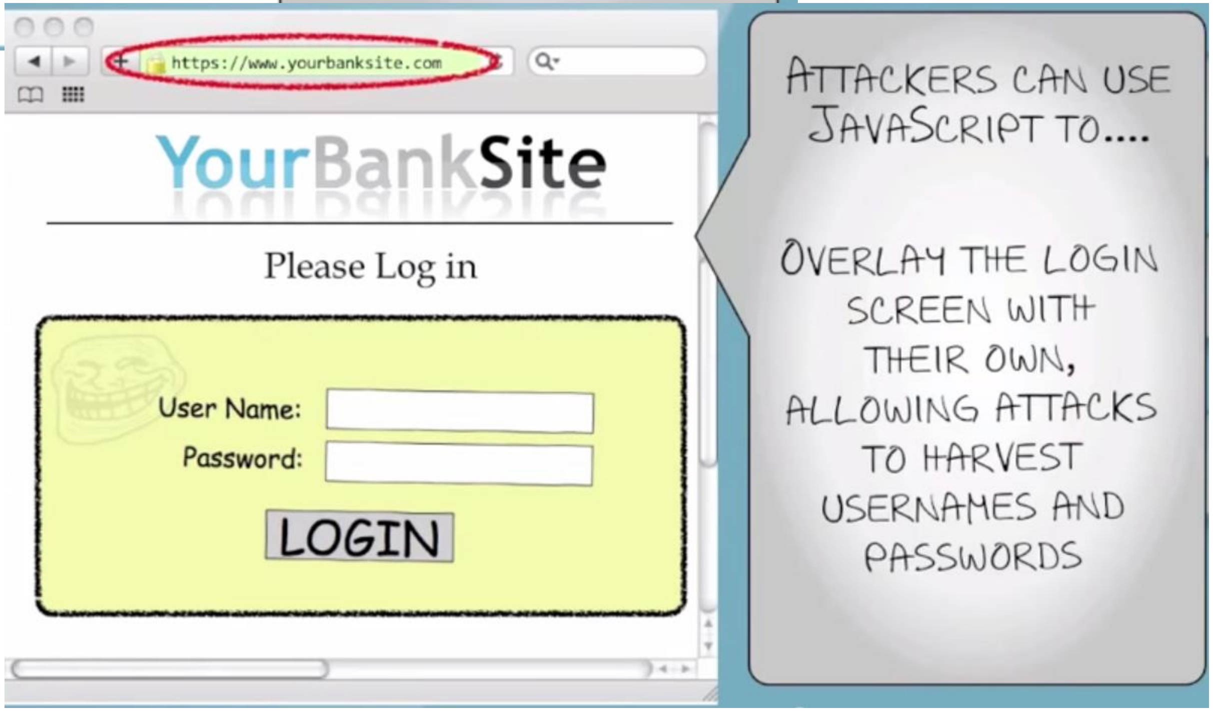
Secure web develo

lol!

ATTACKERS CAN USE JAVASCRIPT TO....

STEAL YOUR SESSION ID: **document.cookie**

REWRITE ANY PART OF THE PAGE



Please Log in

User Name:

Password:

ATTACKERS CAN USE JAVASCRIPT TO....

OVERLAY THE LOGIN SCREEN WITH THEIR OWN, ALLOWING ATTACKS TO HARVEST USERNAMES AND PASSWORDS

A4: Insecure Direct Object Reference

- Attacker manipulates the URL or form values to get unauthorized access
  - to objects (data in a database, objects in memory etc.):

    `http://shop.com/cart?id=`**`413246`** (your cart)

    `http://shop.com/cart?id=`**`123456`** (someone else's cart ?)
  - to files:

    `http://s.ch/?page=`**`home`** -> **`home`**

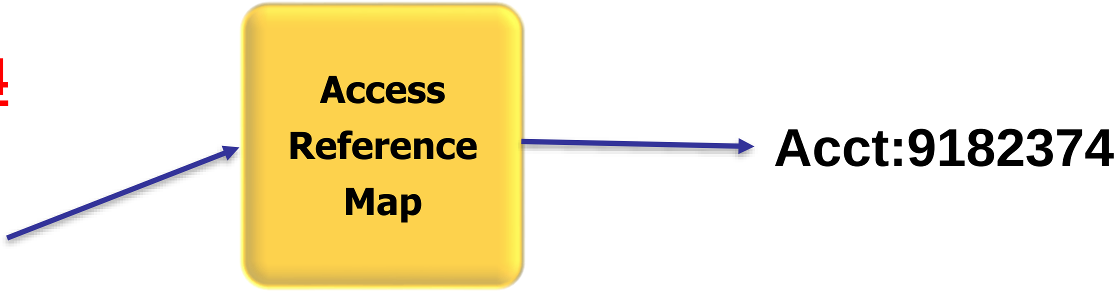
    `http://s.ch/?page=`**`/etc/passwd`** -> **`/etc/passwd`**
- Solution:
  - avoid exposing IDs, keys, filenames to users if possible
  - validate input, accept only correct values
  - verify authorization to all accessed objects (files, data etc.)

A4 – Avoiding Insecure Direct Object References

- Eliminate the direct object reference
  - Replace them with a temporary mapping value such as a random mapping

Instead of :
http://app?id=9182374

Use:
http://app?id=7d3J93



Access Reference Map

Acct:9182374

- Validate the direct object reference
  - 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)

5 – Security Misconfiguration

- Hackers can take advantage of poor server configuration (e.g., default accounts, unpatched flaws, unprotected files and directories) to gain unauthorized access to application functionality or data
- Security misconfiguration can happen at any level of an application stack, including the platform, web server, application server, database, framework, and custom code.

Solution

- Verify your system's configuration by scanning to find misconfiguration or missing patches
- Secure configuration by "**hardening**" the servers:
  - Disable unnecessary packages, accounts, processes & services.
  - Disable default accounts
  - Patch OS, Web server, and Web applications
  - Run Web server user a regular (non-privileged) user

6 – Sensitive Data Exposure

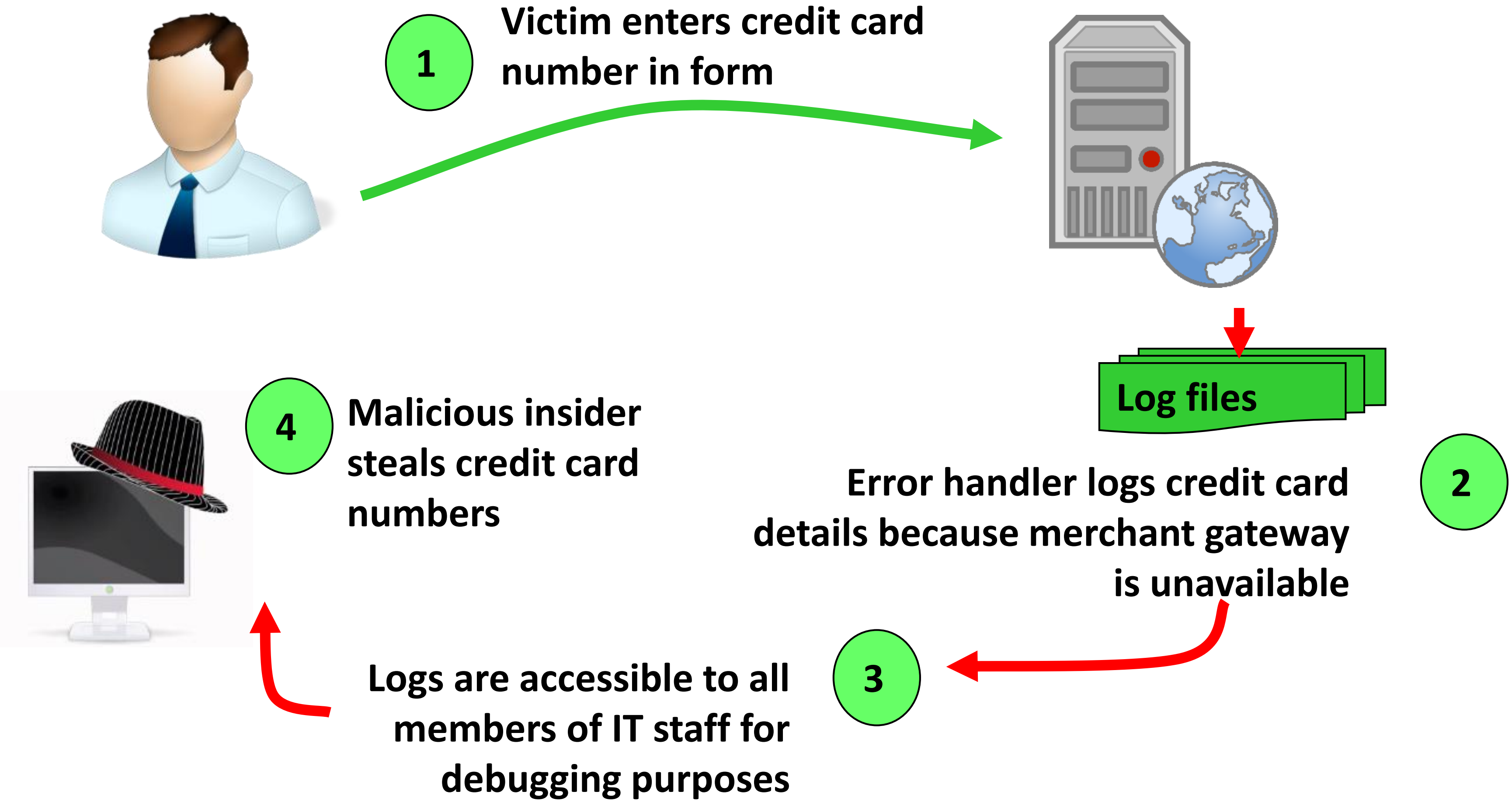
Storing and transmitting sensitive data insecurely

- 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

Sensitive Data Exposure – Example



1 Victim enters credit card number in form

2 Error handler logs credit card details because merchant gateway is unavailable

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

4 Malicious insider steals credit card numbers

A7: Missing Function Level Access Control

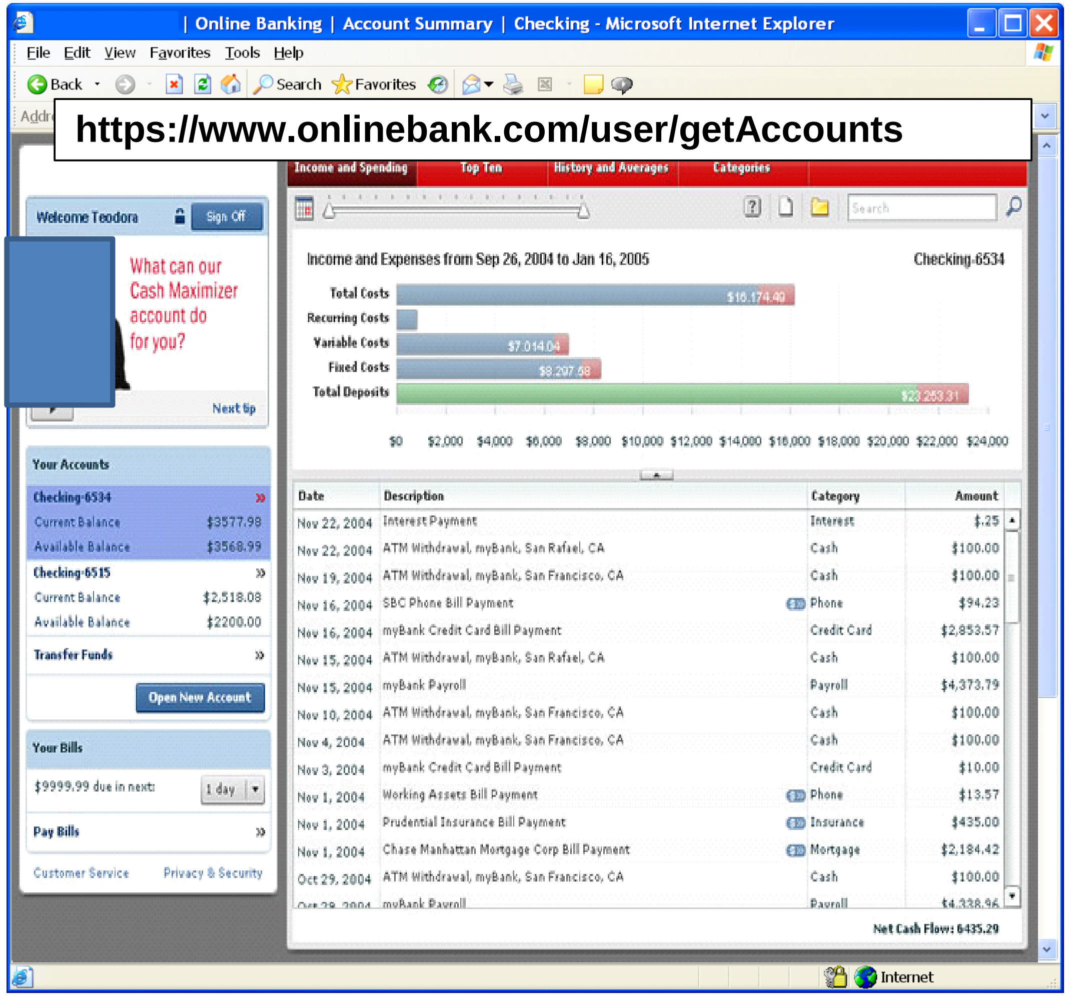
- Often Web applications verify function level access rights before making that functionality visible in the UI. However, applications need to perform the same access control checks on the server when each function is accessed.
- "Hidden" URLs that don't require further authorization
  - to actions:

    `http://site.com/`**`admin/adduser?name=x&pwd=x`**

    (even if `http://site.com/`**`admin/`** requires authorization)
  - to files:

    `http://site.com/`**`internal/salaries.xls`**
- Solution
  - Add missing authorization ☺
  - Don't rely on security by obscurity – it will not work!

Missing Function Level Access Control Illustrated



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

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

Welcome Teodora

What can our Cash Maximizer account do for you?

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 »

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 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

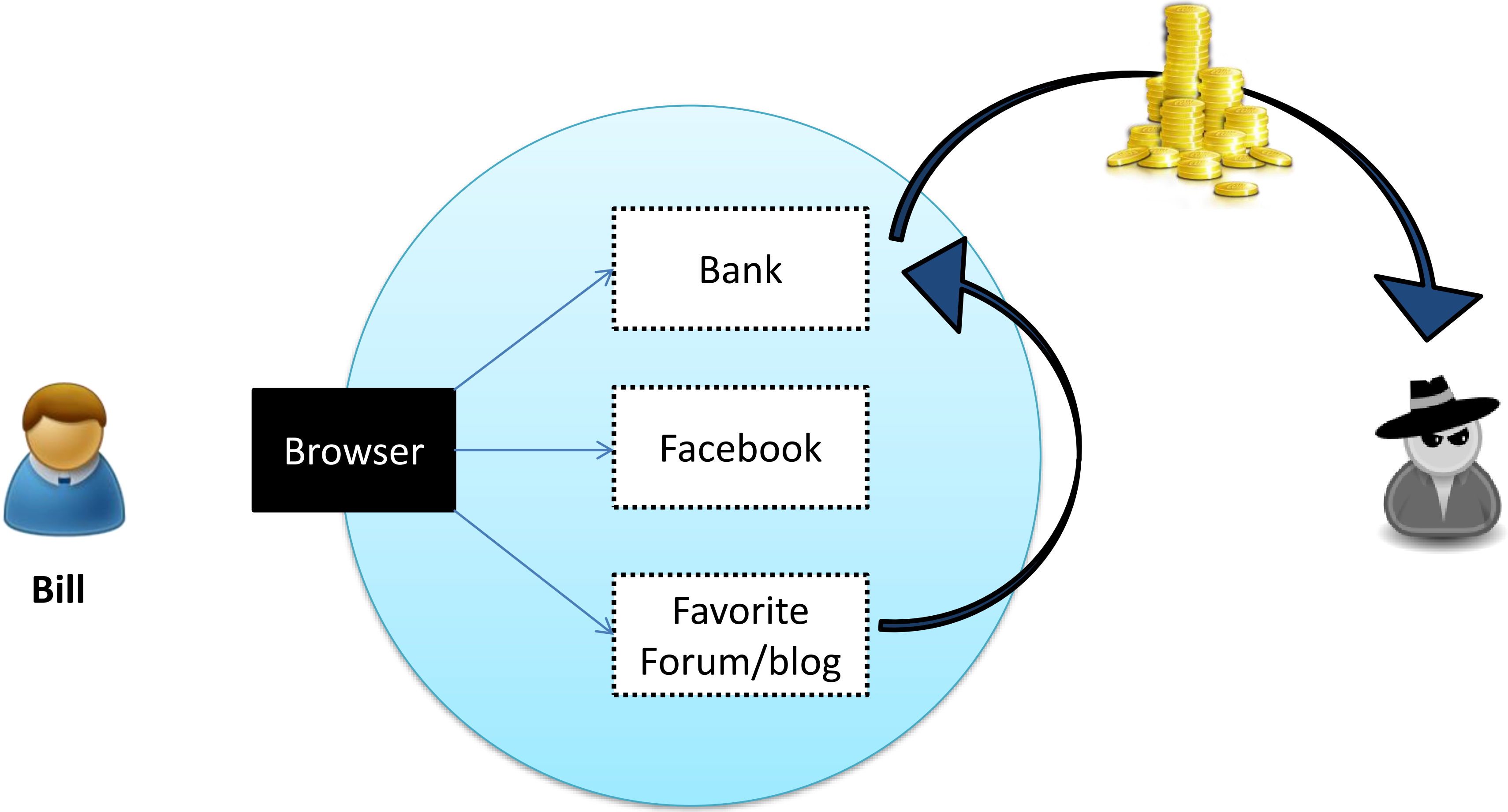
A8: Cross-site request forgery

- Cross-site request forgery (CSRF) – a scenario
  - Ali logs in at bank.com, and forgets to log out
  - Ali then visits a evil.com (or just webforums.com), with:

```
<img src="http://bank.com/
     transfer?amount=1000000&to_account=123456789">
```

  - Ali's browser wants to display the image, so sends a request to bank.com, without Ali's consent
  - if Ali is still logged in, then bank.com accepts the request and performs the action, transparently for Ali (!)
- There is no simple solution, but the following can help:
  - Expire early user sessions, encourage users to log out
  - Use secret hidden fields in forms
  - Use POST rather than GET, and check referrer value
  - Re-authenticate or CAPTCHA for extra-sensitive pages

CSRF Example



http://mybank.com/showaccount?id=bill

http://mybank.com/transfer?from=bill&amount=10000&for=someguy

<img src=http://mybank.com/transfer?from=bill&amount=10000&for=someguy />

9 – Using Known Vulnerable Components

Vulnerable Components Are Common

- Some vulnerable components (e.g., framework libraries) can be identified and exploited with automated tools

Widespread

- Virtually every application has these issues because most development teams don’t focus on ensuring their components/libraries are up to date

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

10 – Unvalidated Redirects and Forwards

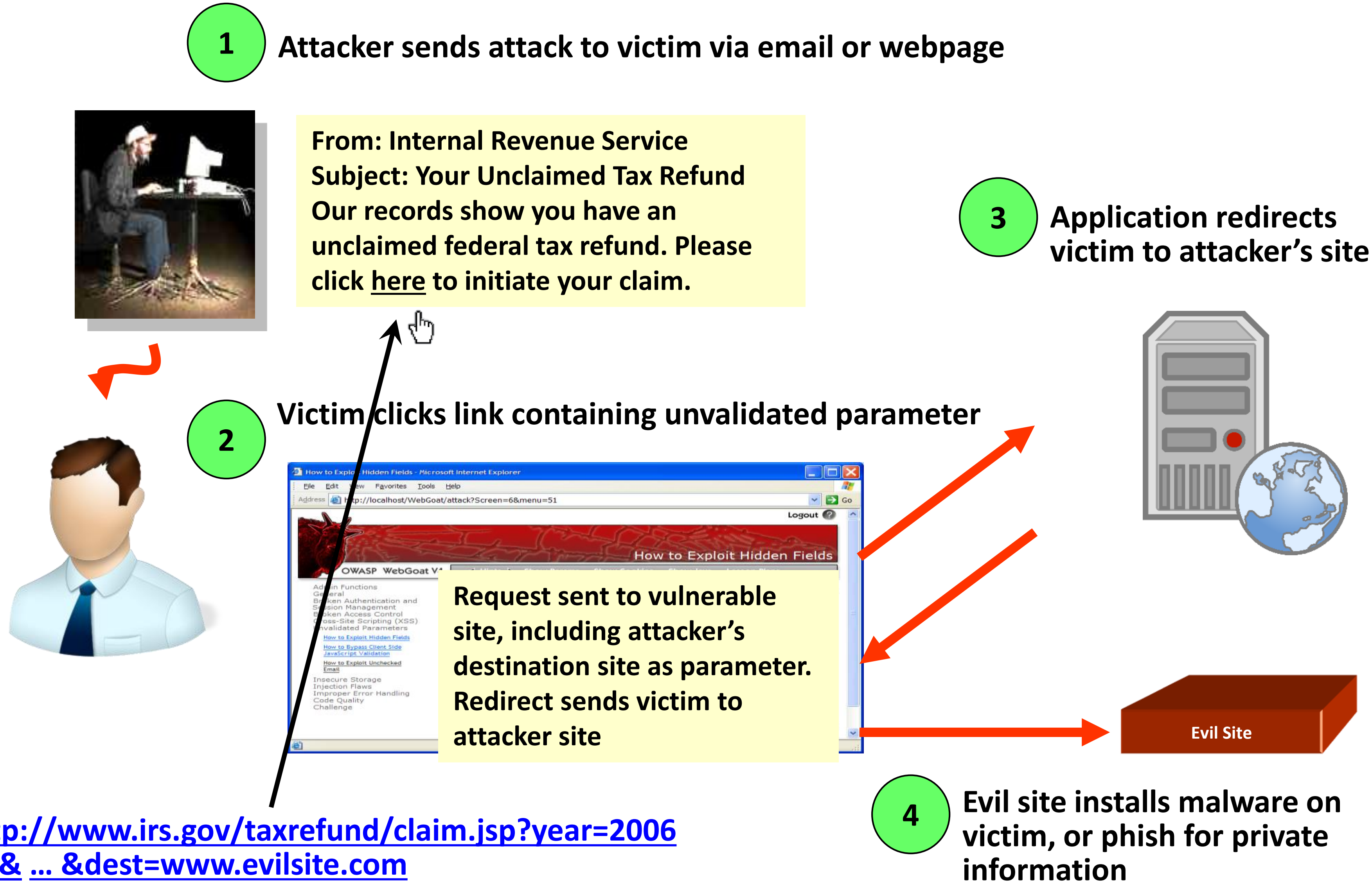
Web application redirects are very common

- Sometimes parameters define the destination URL
- If they aren't validated, attacker can send victim to a site of their choice
- 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

3 Application redirects victim to attacker's site

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

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

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

How to Exploit Hidden Fields - Microsoft Internet Explorer
Logout
How to Exploit Hidden Fields
OWASP WebGoat V4
Evil Site

Client-server – no trust

- Security on the client side doesn’t work (and cannot)
  - don’t rely on the client to perform security checks (validation etc.)
  - e.g. `<input type=”text” maxlength=”20”>` is not enough
  - authentication should be done on the server side, not by the client
- Don’t trust your client
  - HTTP response header fields like referrer, cookies etc.
  - HTTP query string values (from hidden fields or explicit links)
  - e.g. `<input type=”hidden” name=”price” value=”299”>` in an online shop can (and will!) be abused
- Do all security-related checks on the server
- Don’t expect your clients to send you SQL queries, shell commands etc. to execute – it’s not your code anymore
- Put limits on the number of connections, set timeouts

Summary

- Understand threats and typical attacks
- Validate, validate, validate (!)
- Do not trust the client
- Read and follow recommendations for your platform
- Use web scanning tools https://www.owasp.org/index.php/Category:Vulnerability_Scanning_Tools
- Harden the Web server and programming platform configuration