Offense-Based Cybersecurity: Exploitation of Web Vulnerabilities

CS 459/559: Science of Cyber Security
8th Lecture

Instructor:

Guanhua Yan

Agenda

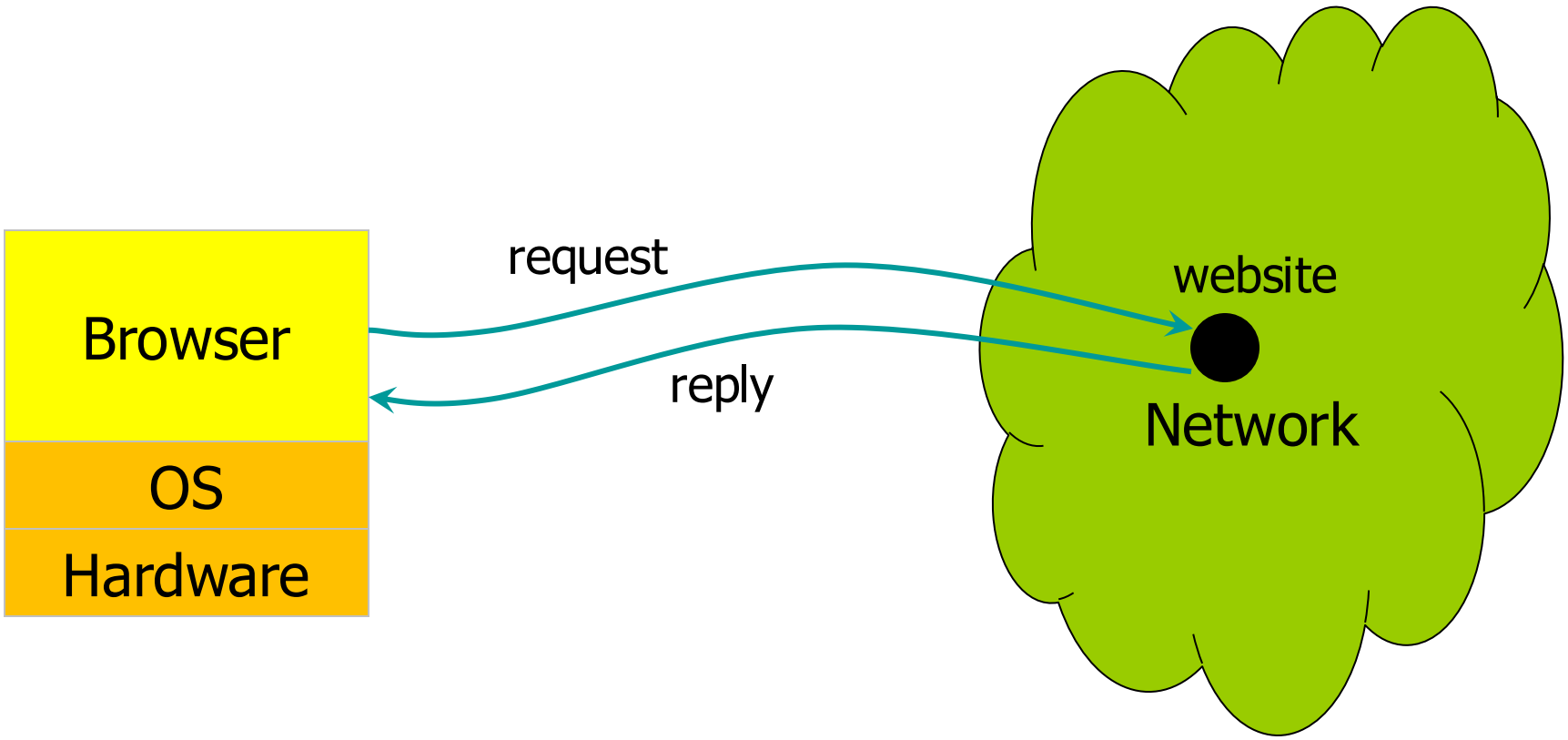
- Quiz 1: September 29
- Project 1 (offense): October 10
- Project 2 (defense): December 5
- Presentations: 11/17, 11/19, 11/24, 12/1, 12/3
- Final report: December 15

What we will learn in this lecture

- Primer on web applications
- XSS: cross-site scripting
- SQL injection
- XSRF: cross-site request forgery

Primer on Web Applications

Browser and Network

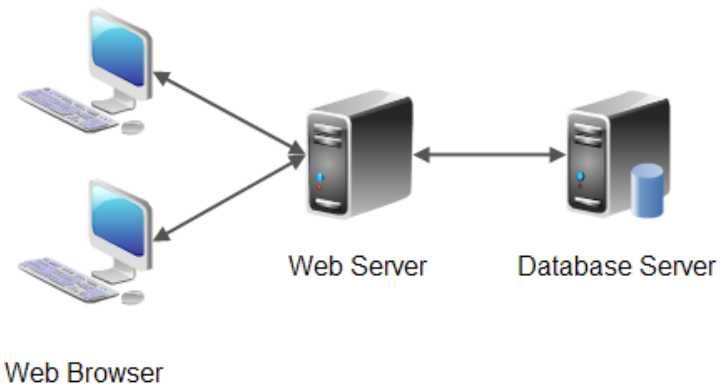


Web Applications

- **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
    - 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



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

GET vs. POST

What is going on when a page loads?

GET	**POST**
Requests data from a specific resource.	Submits data to be processed by a specific resource.
Data is submitted as part of the URL	Data is submitted in the request body
Less secure but faster	More secure but slower
Can be cached by browser	Not Cached by Browser
Length Limited by URL size	MaxLength determined by server

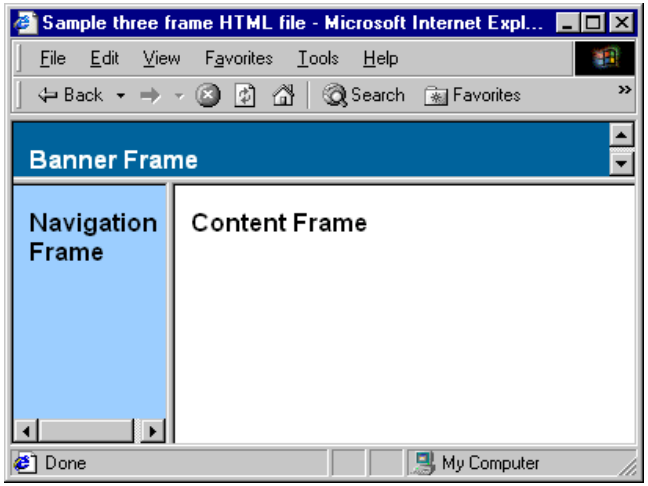
Web Attacker

- Controls malicious website (attacker.com)
  - Can even obtain SSL/TLS certificate for this 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!

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

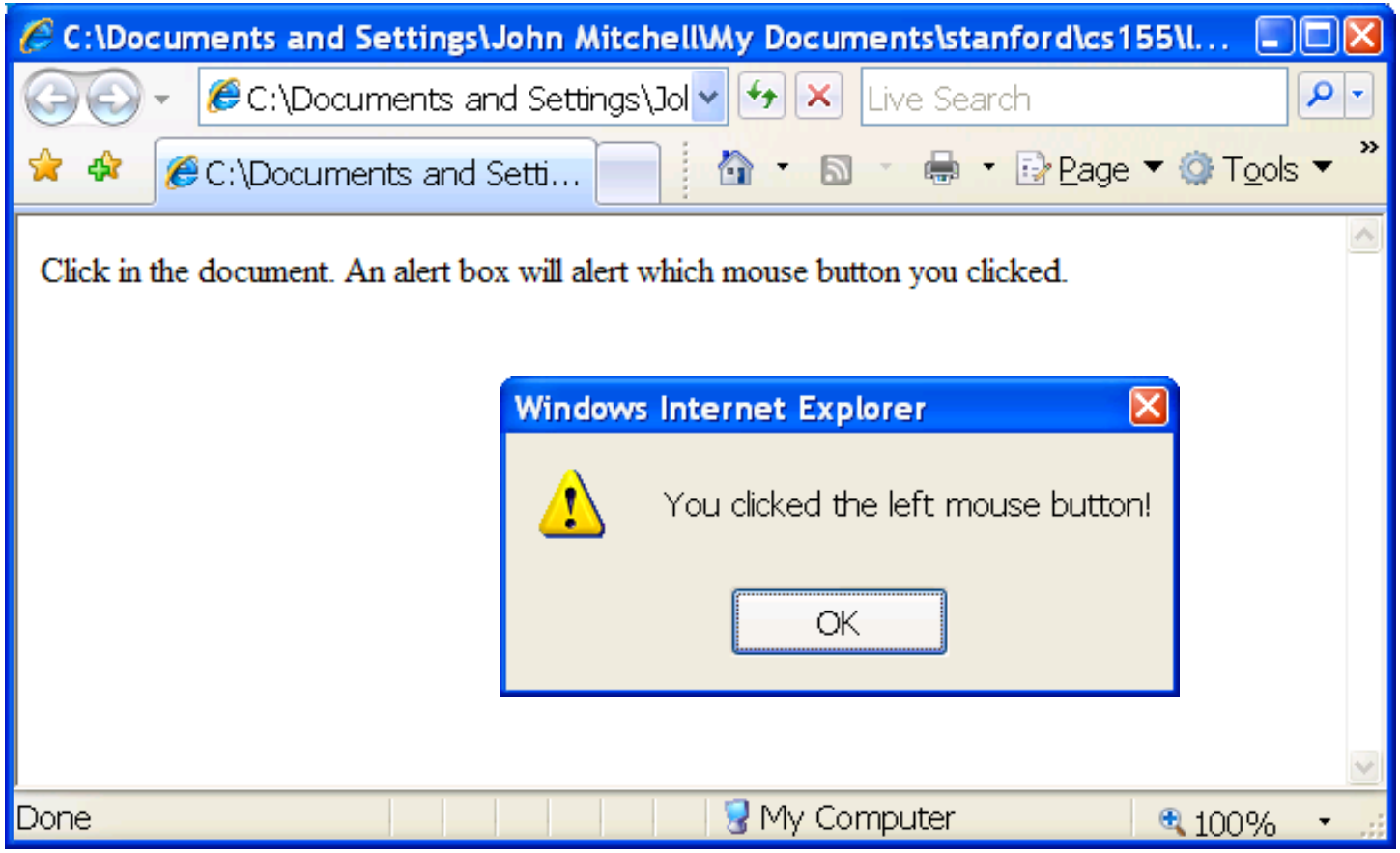
Event-Driven Script Execution

```
<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
- **Attacker’s goal is to execute code on user’s machine**
- “The world’s most misunderstood programing language”

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**
  - For absolute URIs, the origin is

    triple {protocol, host, port}

  - Two resources are considered of the same origin if and only if **all these values are exactly the same**.

  - SOP: can only **read** properties of documents and windows from the same host, protocol, and port

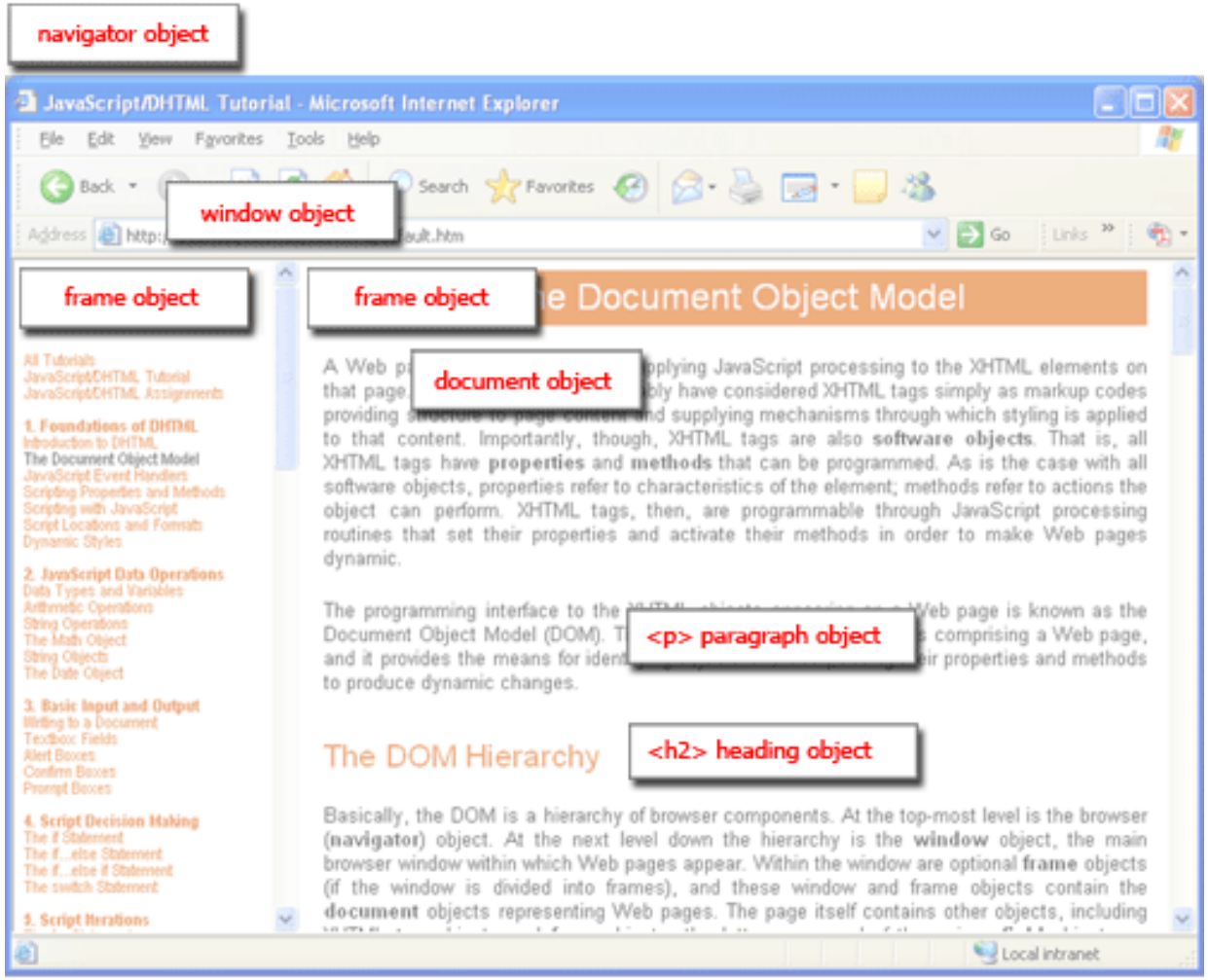
URL: http://www.example.com/dir/page.html

Compared URL	Outcome	Reason
http://www.example.com/dir/page2.html	Success	Same protocol, host and port
http://www.example.com/dir2/other.html	Success	Same protocol, host and port
http://username:password@**www.example.com**/dir2/other.html	Success	Same protocol, host and port
http://www.example.com:**81**/dir/other.html	Failure	Same protocol and host but different port
https://www.example.com/dir/other.html	Failure	Different protocol
http://**en.example.com**/dir/other.html	Failure	Different host
http://**example.com**/dir/other.html	Failure	Different host (exact match required)
http://**v2.www.example.com**/dir/other.html	Failure	Different host (exact match required)
http://www.example.com:**80**/dir/other.html	Depends	Port explicit. Depends on implementation in browser.

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

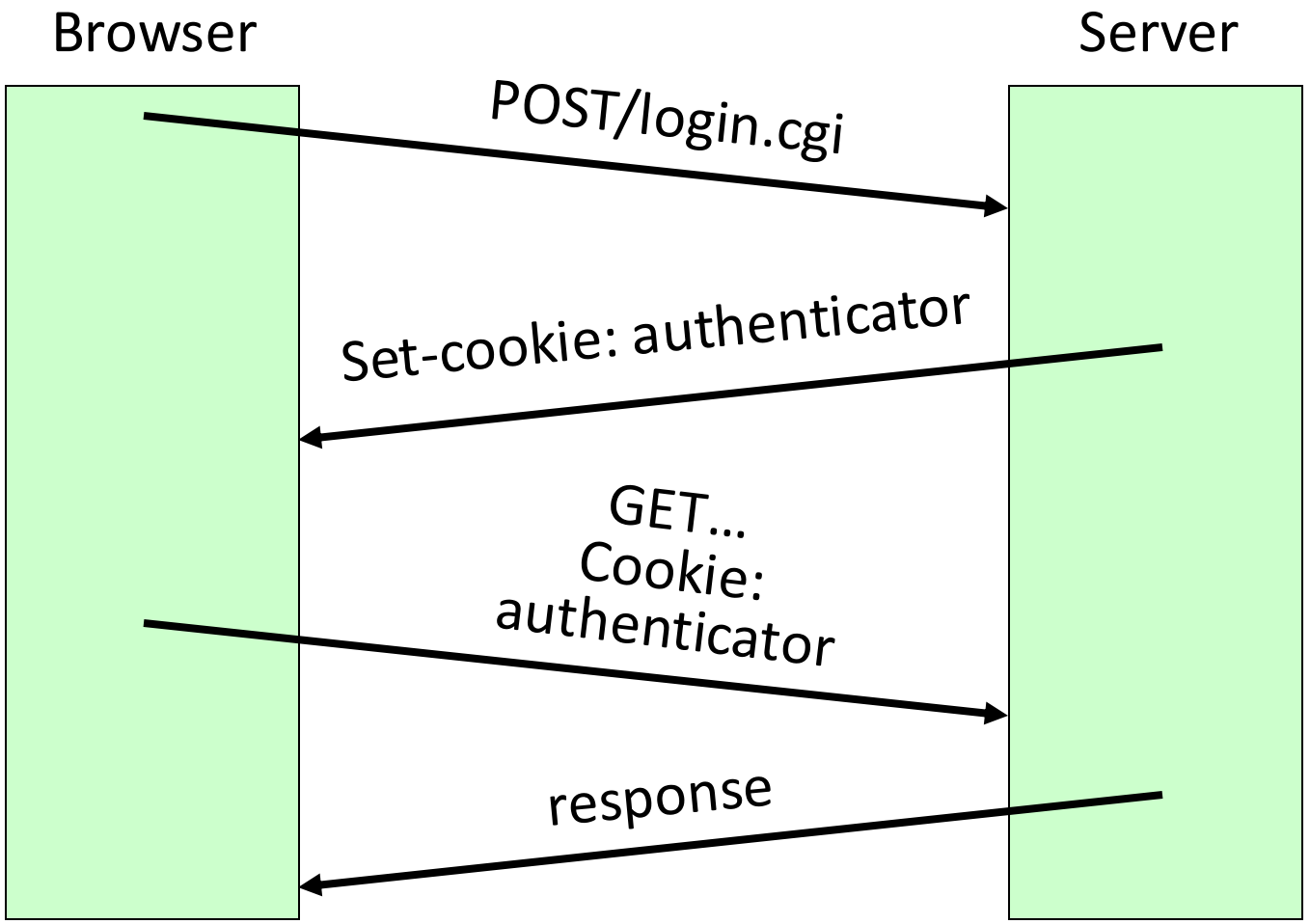


Three common web-based attacks

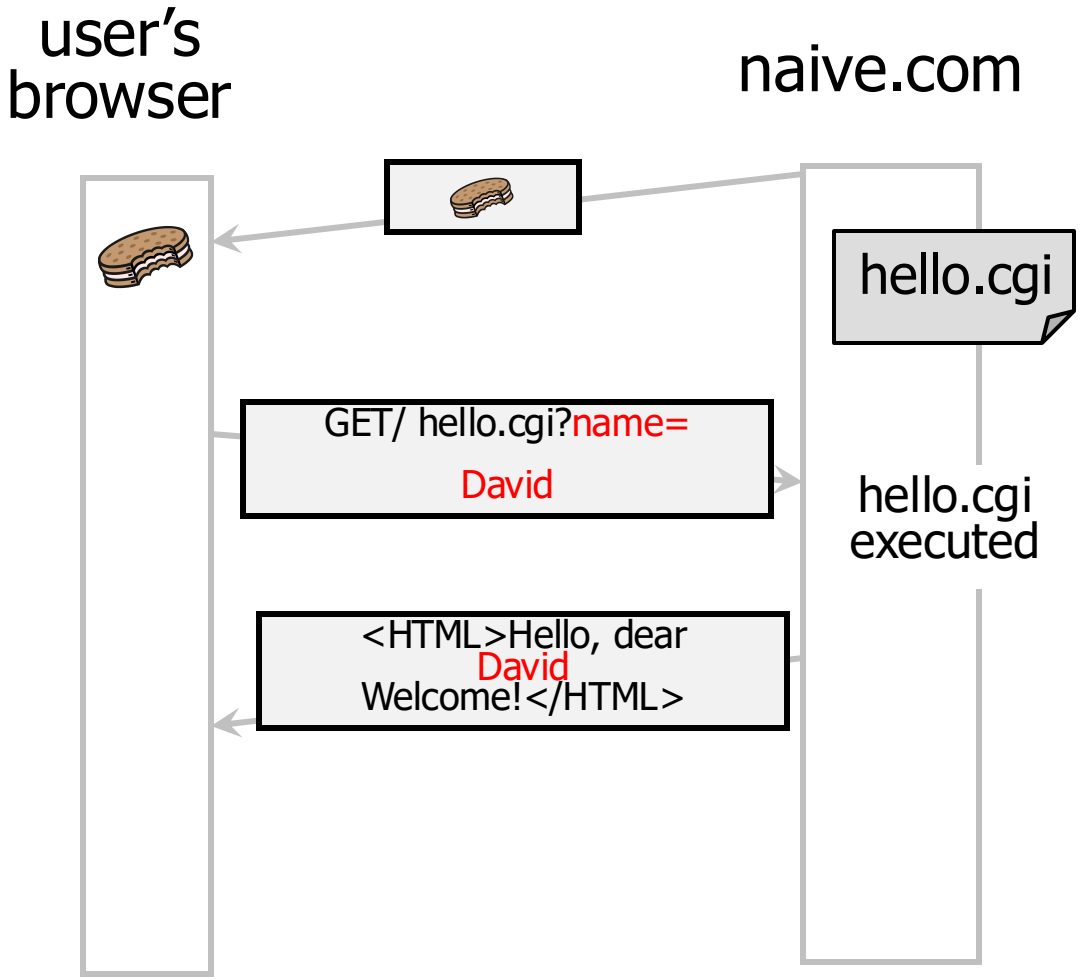
- XSS: cross-site scripting
- SQL injection
- XSRF: cross-site request forgery

XSS: Cross-Site Scripting

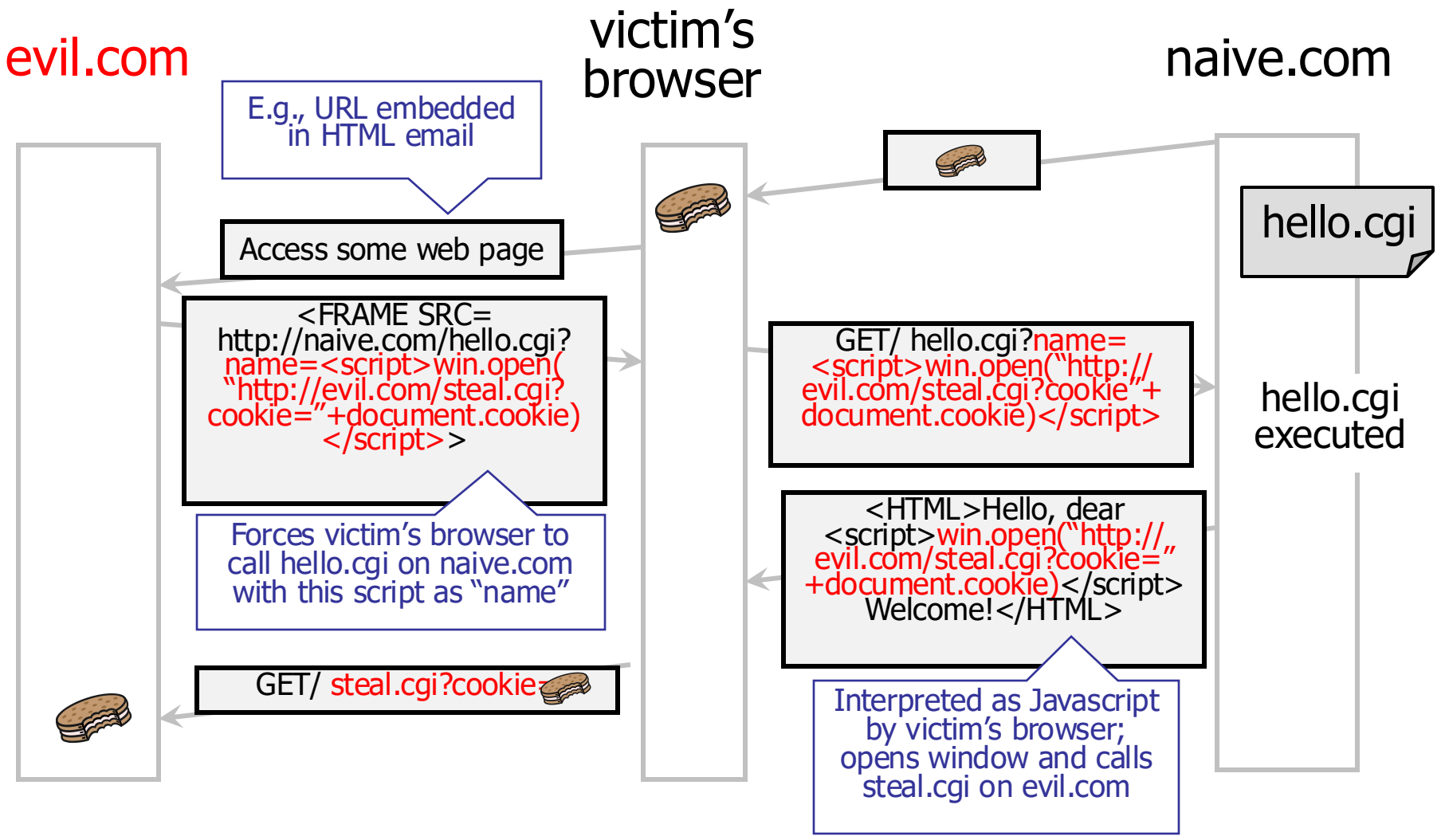
Cookie-Based Authentication



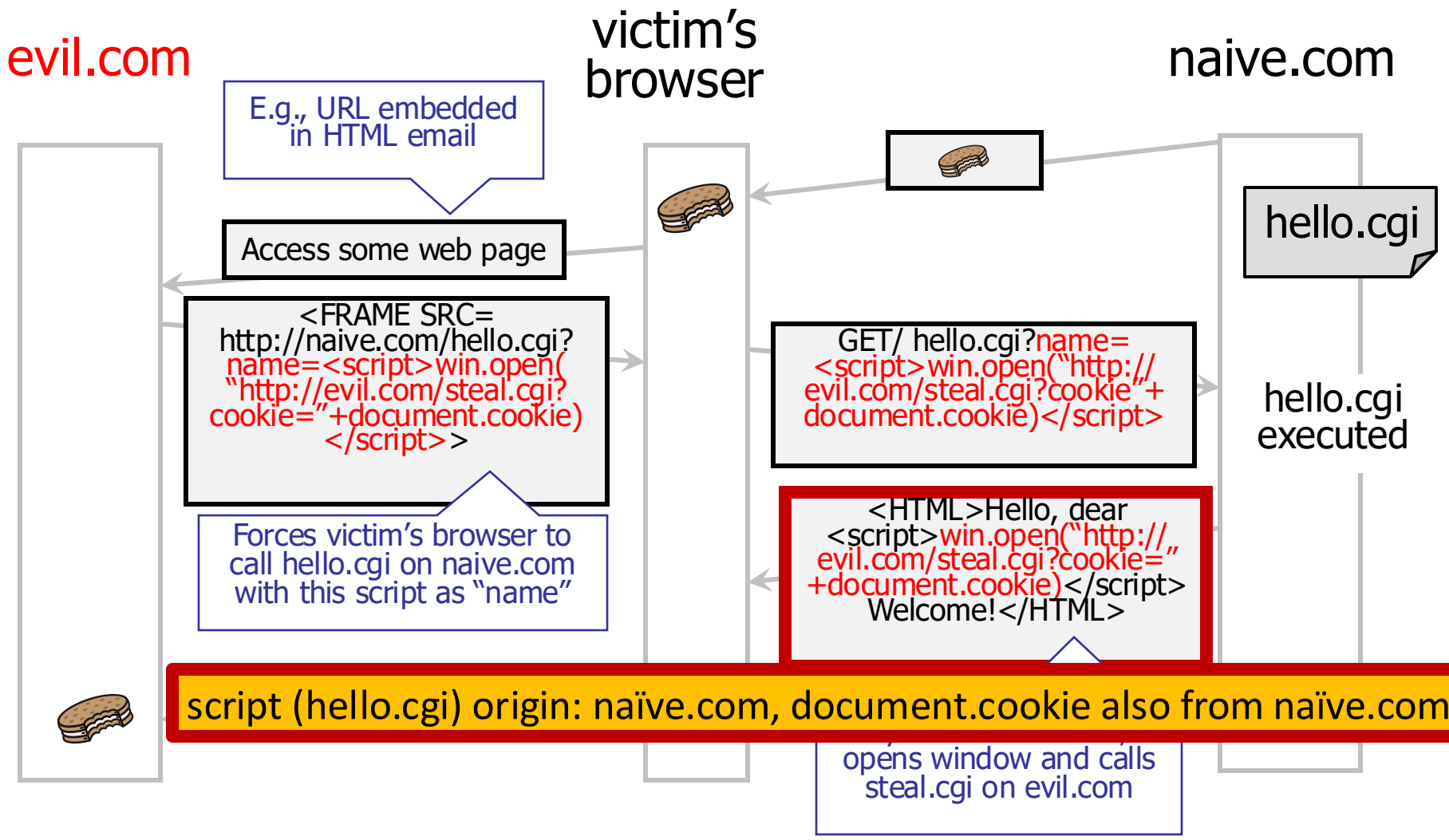
Normal use


user's browser
naive.com
hello.cgi
GET/ hello.cgi?name=
David
hello.cgi executed
<HTML>Hello, dear
David
Welcome!</HTML>

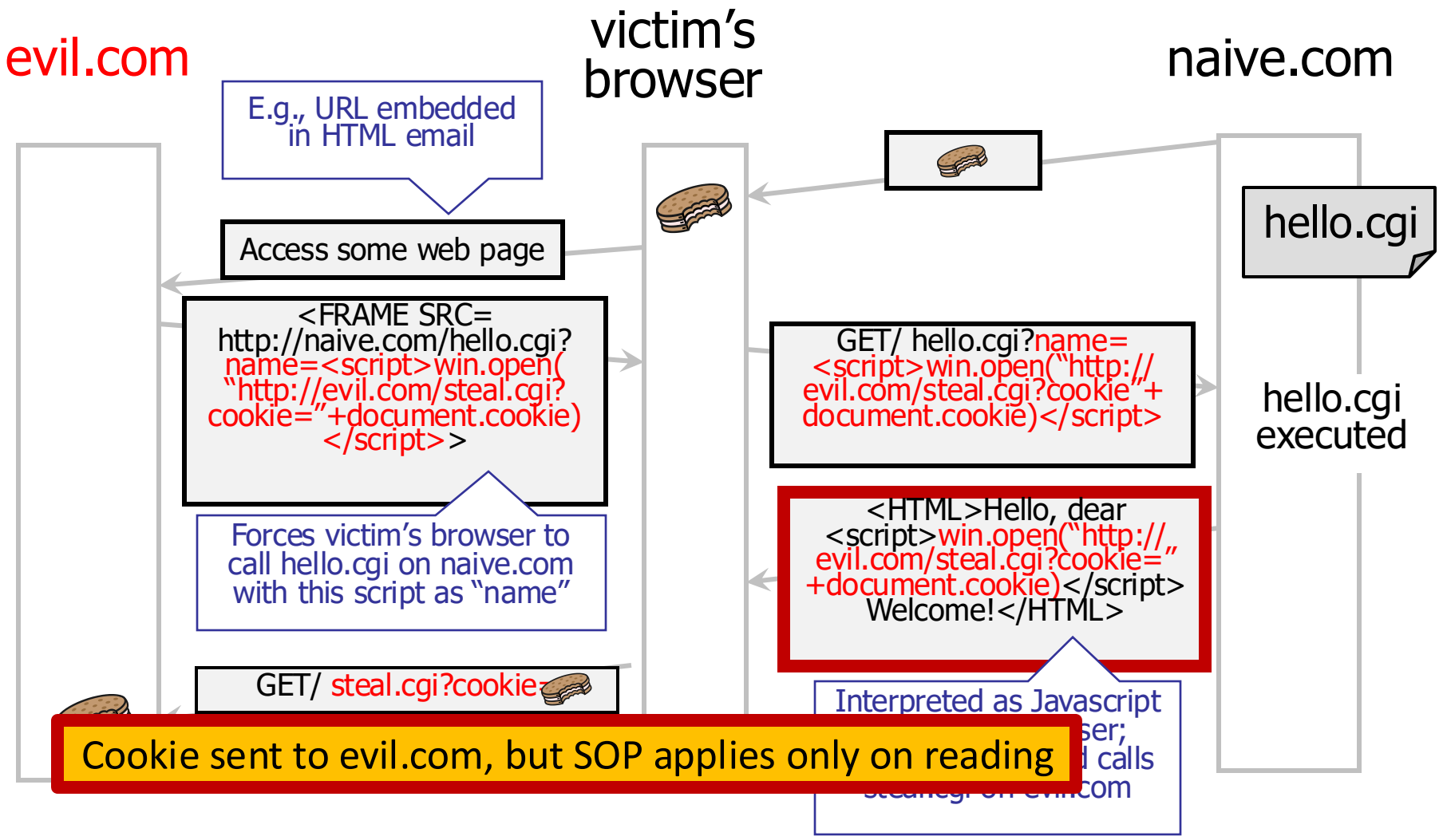
XSS: Cross-Site Scripting



Violation of SOP (Same Origin Policy)?



Violation of SOP (Same Origin Policy)?



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

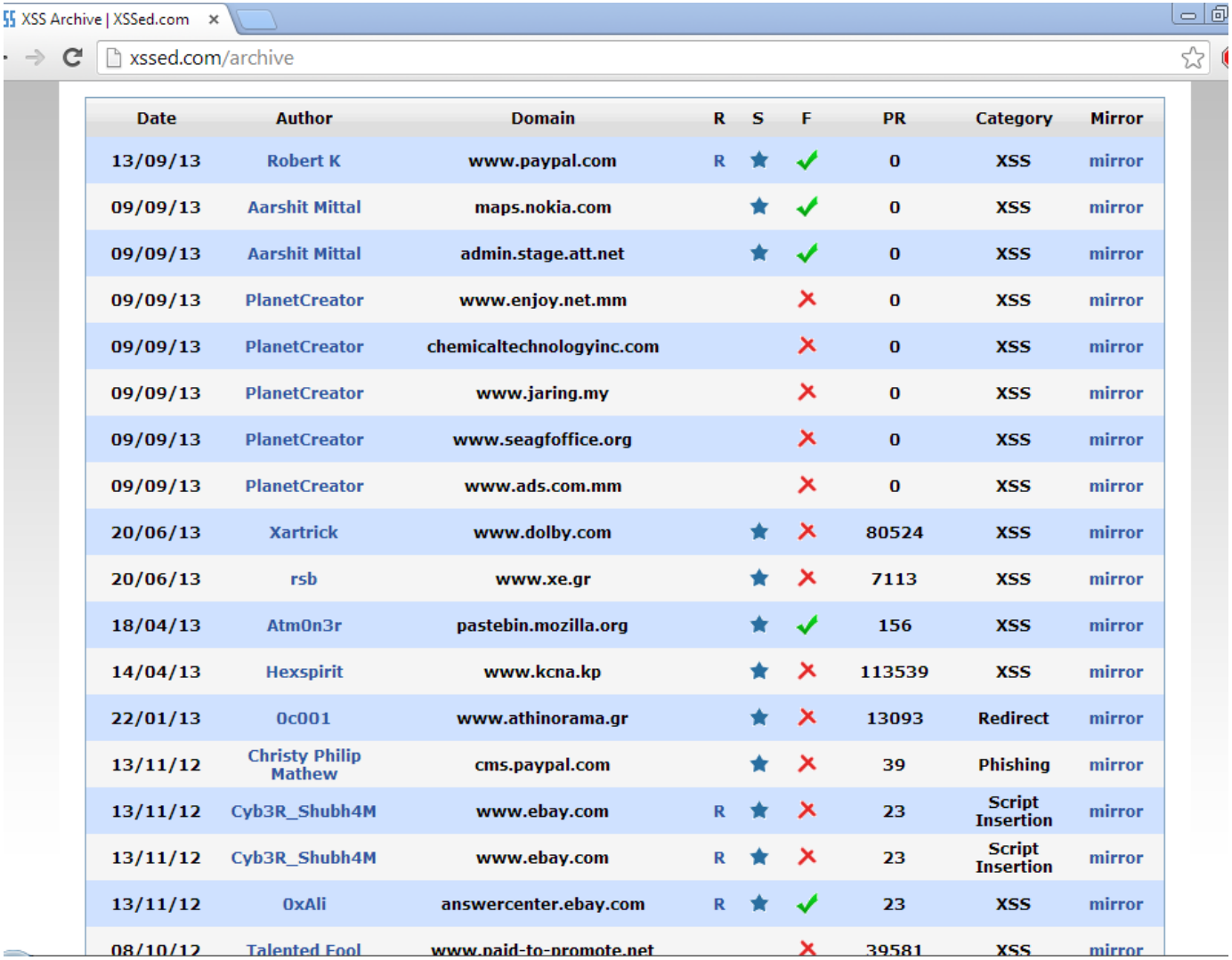
- So what if evil.com gets cookie for naive.com?
  - Cookie can include session authenticator for naive.com
    - Or other data intended only for naive.com
  - Violates the “intent” of the same-origin policy

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**

XSS in the Wild

http://xssed.com/archive



Date	Author	Domain	R	S	F	PR	Category	Mirror
13/09/13	Robert K	www.paypal.com	R	★	✓	0	XSS	mirror
09/09/13	Aarshit Mittal	maps.nokia.com		★	✓	0	XSS	mirror
09/09/13	Aarshit Mittal	admin.stage.att.net		★	✓	0	XSS	mirror
09/09/13	PlanetCreator	www.enjoy.net.mm			✗	0	XSS	mirror
09/09/13	PlanetCreator	chemicaltechnologyinc.com			✗	0	XSS	mirror
09/09/13	PlanetCreator	www.jaring.my			✗	0	XSS	mirror
09/09/13	PlanetCreator	www.seagfoffice.org			✗	0	XSS	mirror
09/09/13	PlanetCreator	www.ads.com.mm			✗	0	XSS	mirror
20/06/13	Xartrick	www.dolby.com		★	✗	80524	XSS	mirror
20/06/13	rsb	www.xe.gr		★	✗	7113	XSS	mirror
18/04/13	Atm0n3r	pastebin.mozilla.org		★	✓	156	XSS	mirror
14/04/13	Hexspirit	www.kcna.kp		★	✗	113539	XSS	mirror
22/01/13	0c001	www.athinorama.gr		★	✗	13093	Redirect	mirror
13/11/12	Christy Philip Mathew	cms.paypal.com		★	✗	39	Phishing	mirror
13/11/12	Cyb3R_Shubh4M	www.ebay.com	R	★	✗	23	Script Insertion	mirror
13/11/12	Cyb3R_Shubh4M	www.ebay.com	R	★	✗	23	Script Insertion	mirror
13/11/12	0xAli	answercenter.ebay.com	R	★	✓	23	XSS	mirror
08/10/12	Talented Fool	www.paid-to-promote.net			✗	39581	XSS	mirror

SQL injection

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='Alice'
- Add data to the table

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

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

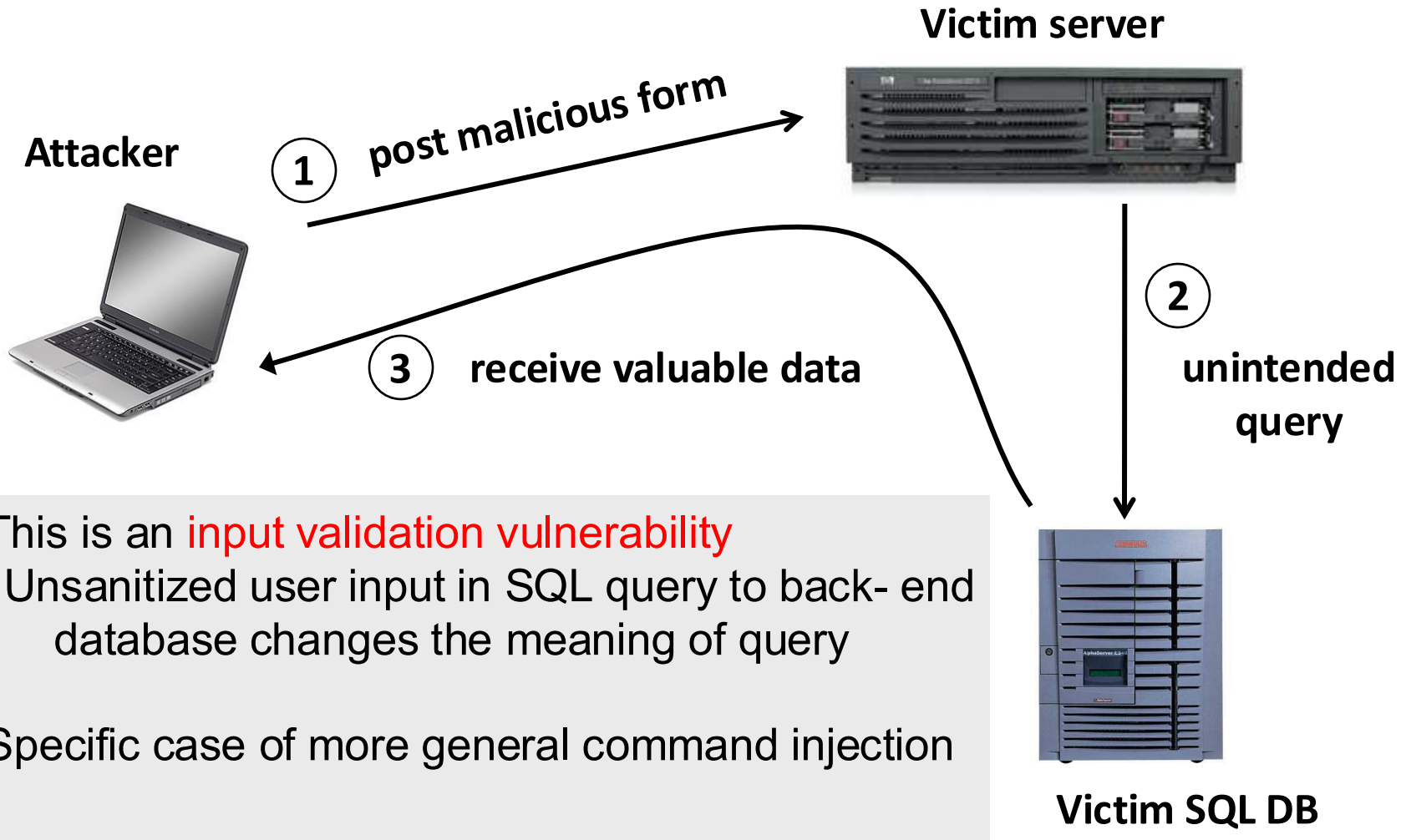
Sample PHP Code

- Sample PHP

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

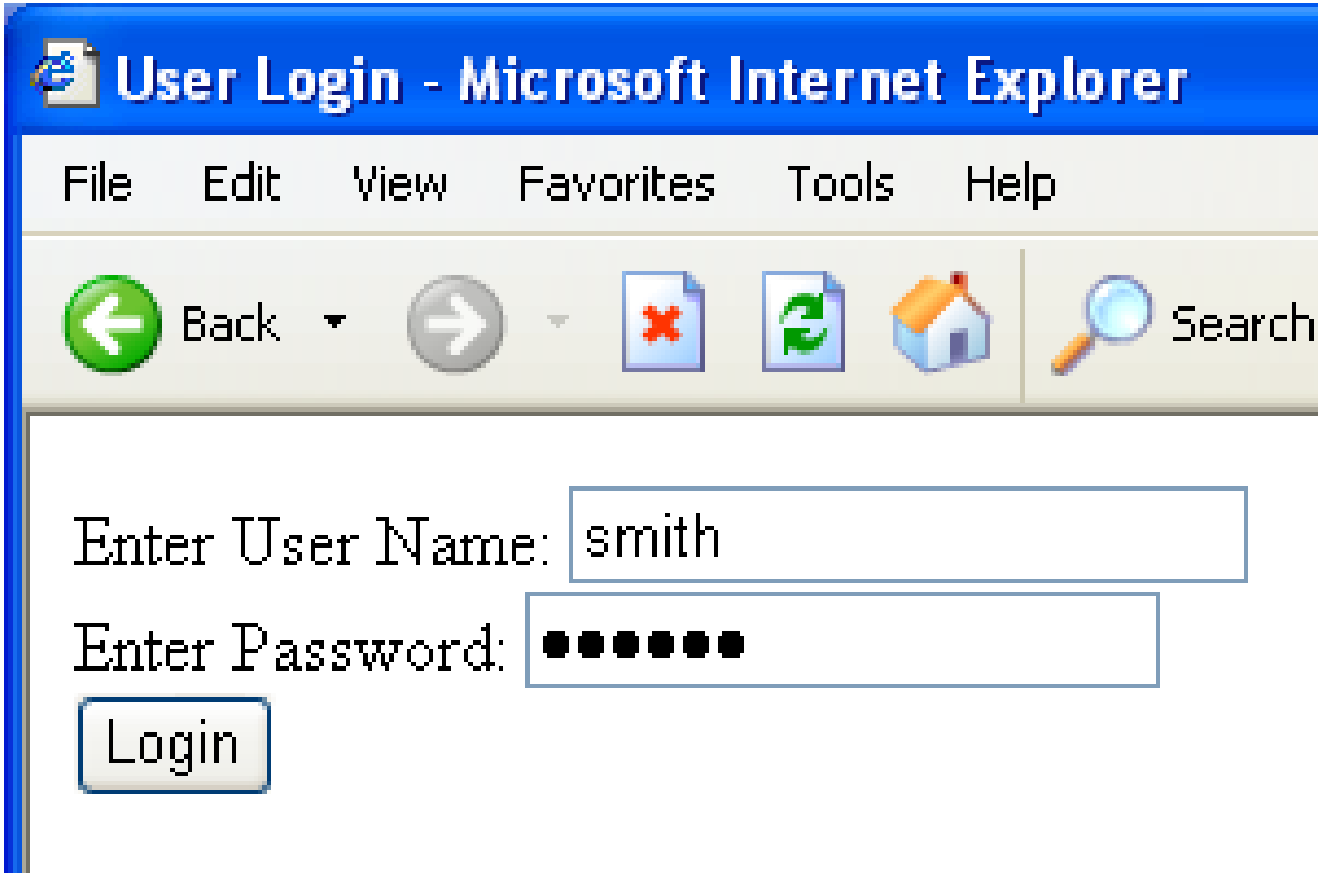
- What if ‘user’ is a malicious string that changes the meaning of the query?

SQL Injection: Basic Idea

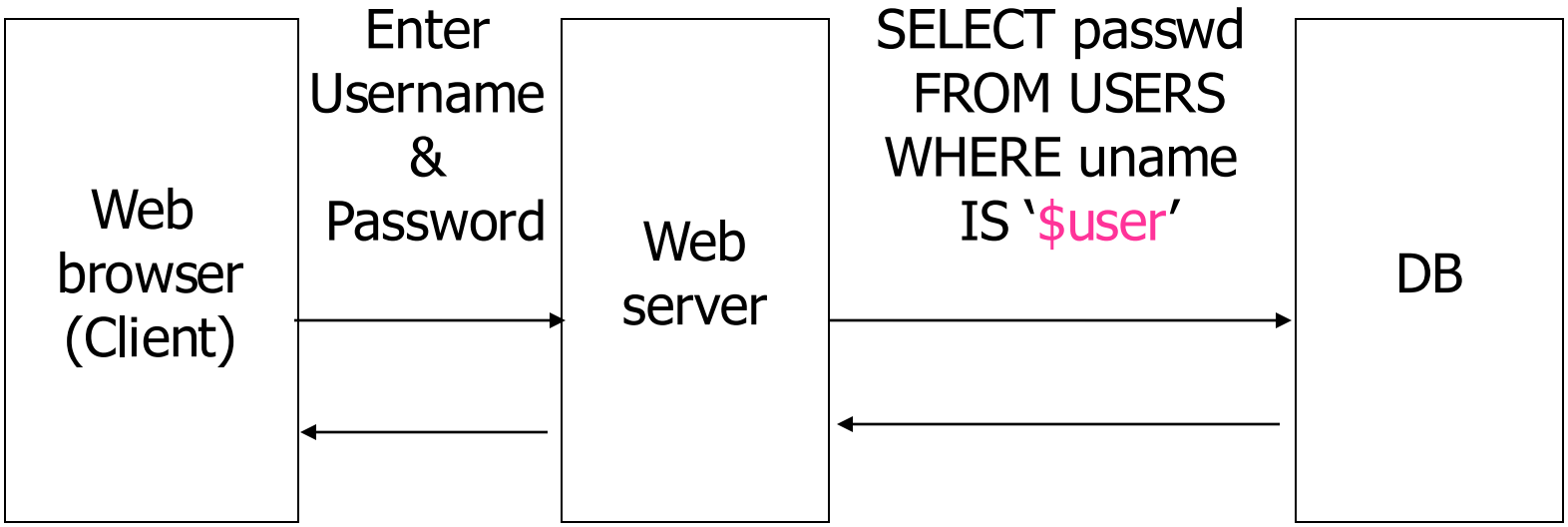


- This is an input validation vulnerability
  Unsanitized user input in SQL query to back- end database changes the meaning of query
- Specific case of more general command injection

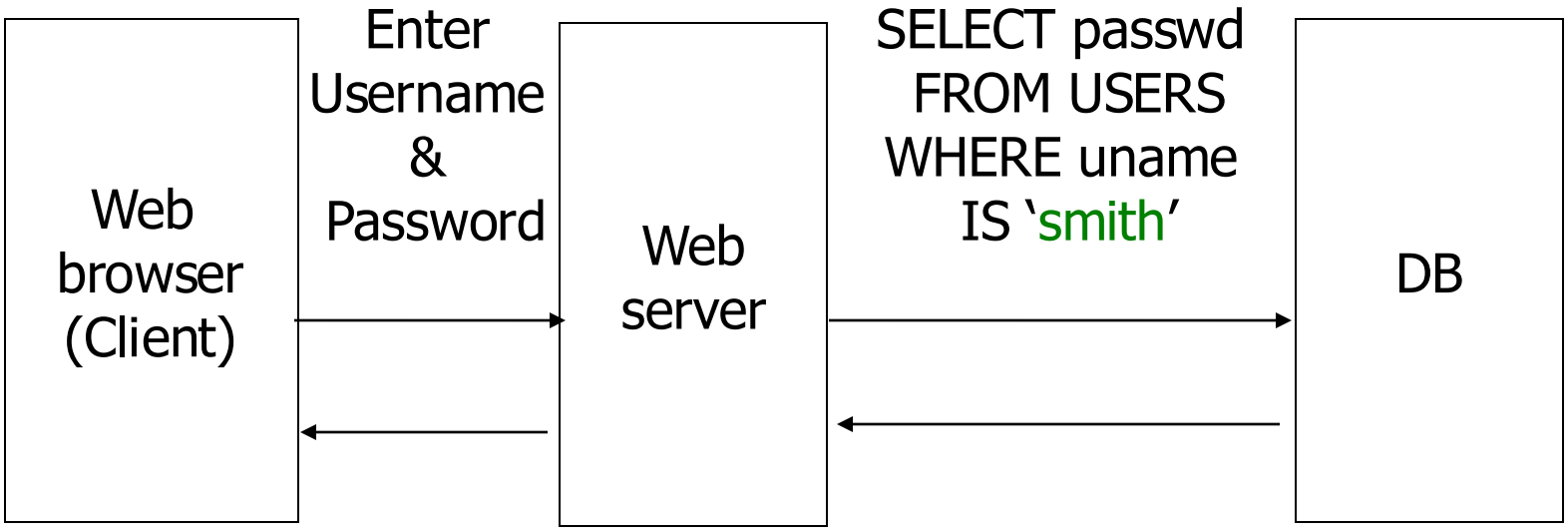
Typical Login Prompt



User Input Becomes Part of Query

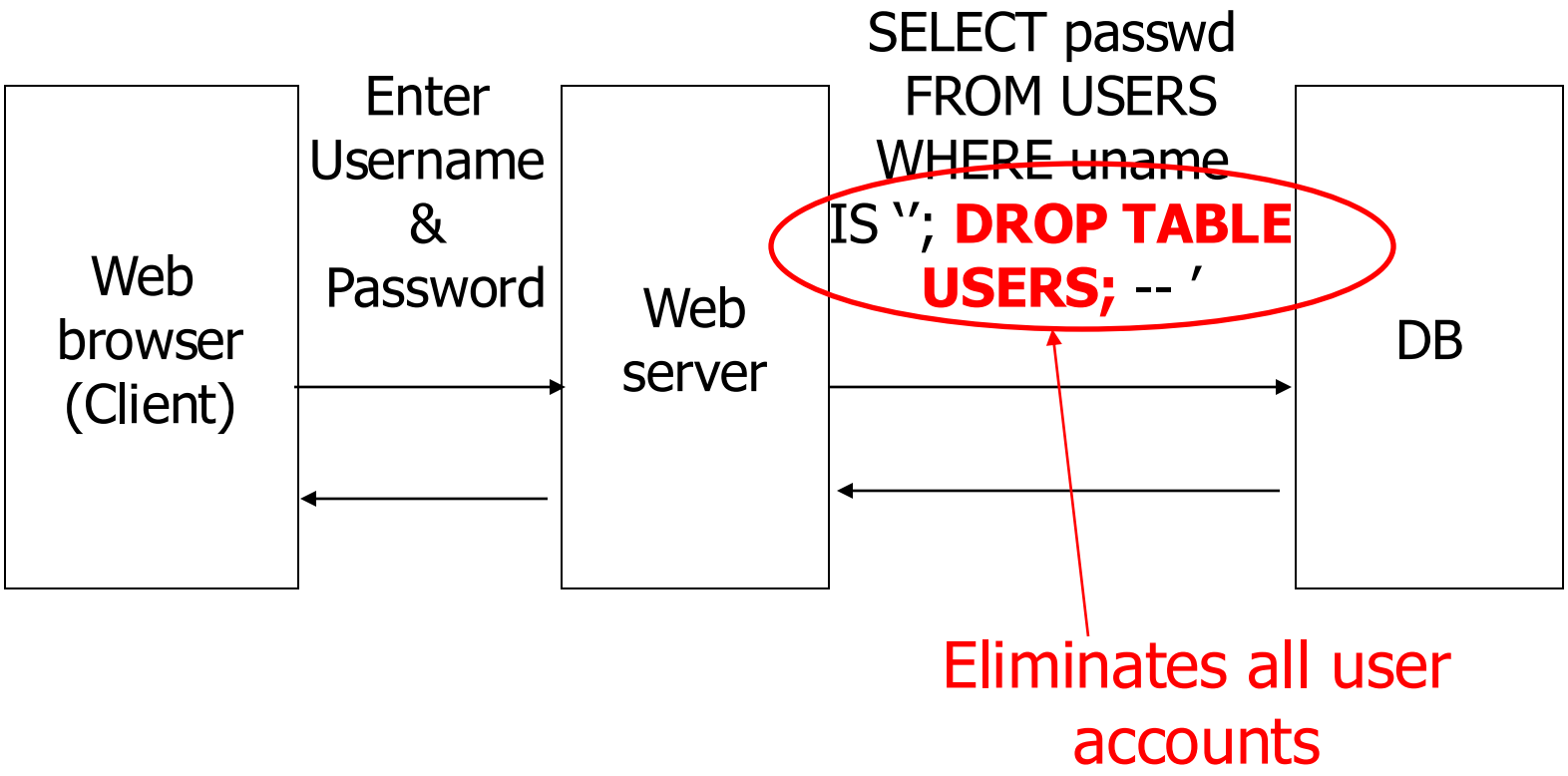


Normal Login



Malicious User Input

SQL Injection Attack



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!

SQL Injection in the Real World

CardSystems 40M credit card accounts [Jun 2005]

134M credit card accounts [Mar 2008]

450,000 passwords [Jul 2012]

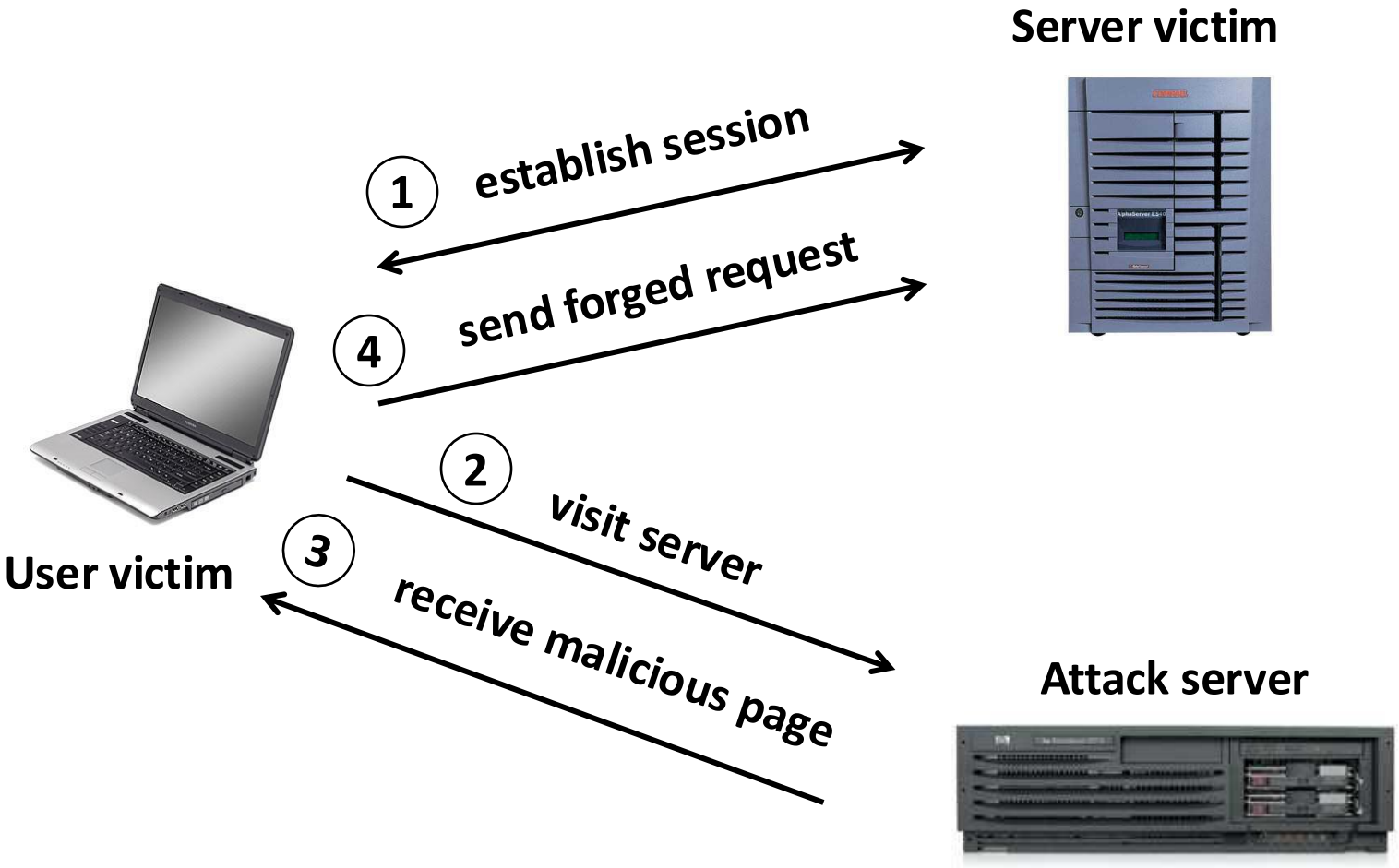
CyberVor booty 1.2 billion accounts [Reported in 2014] from 420,000 websites

XSRF: cross-site request forgery

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?

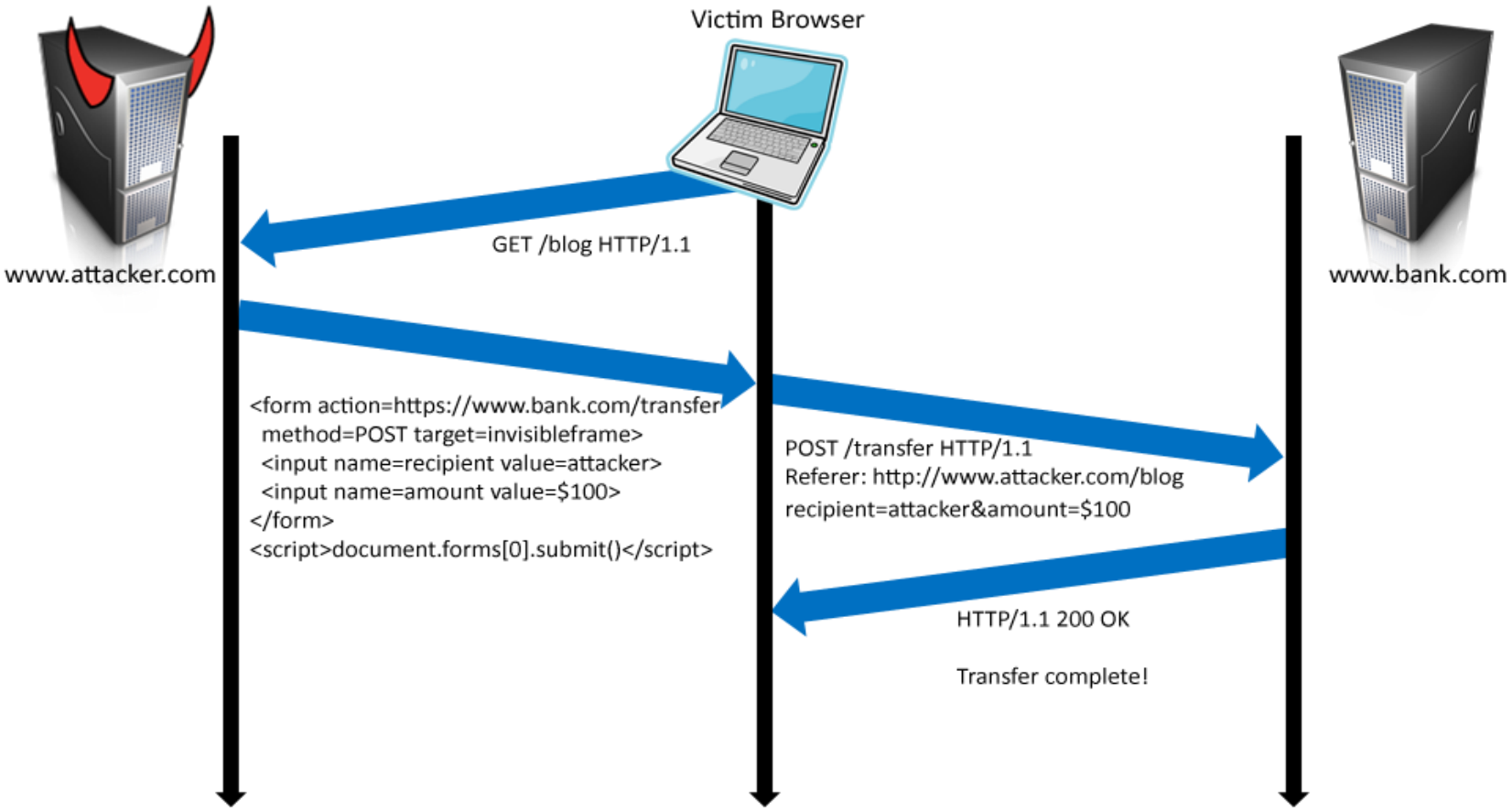
Cookie Authentication: Not Enough!

- 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

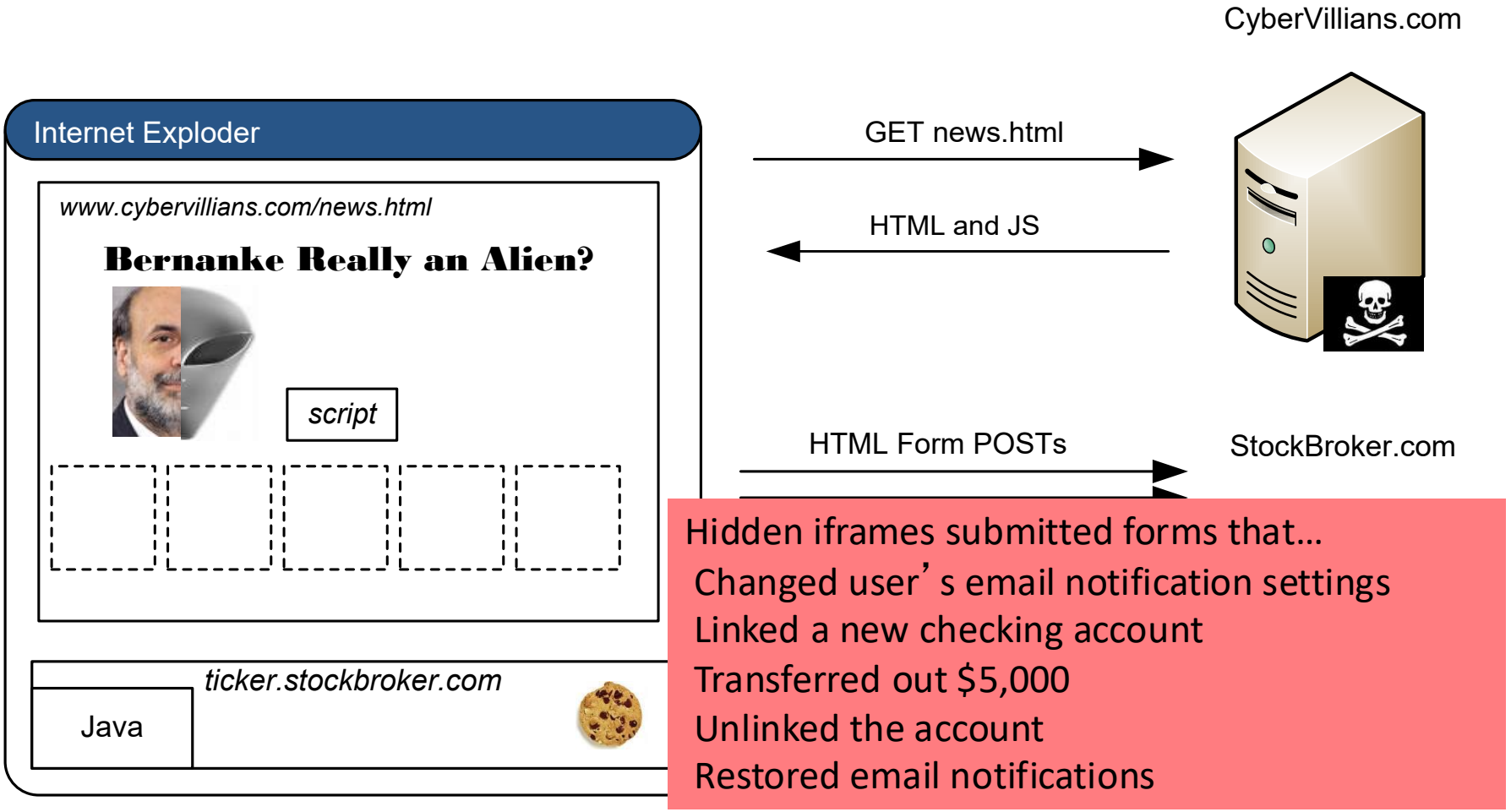
XSRF in More Detail



XSRF True Story (1)

- User has a Java stock ticker from his broker's website running in his browser
  - Ticker has a cookie to access user's account on the site
- A comment on a public message board on finance.yahoo.com points to "leaked news"
  - TinyURL redirects to cybervillians.com/news.html
- User spends a minute reading a story, gets bored, leaves the news site
- Gets his monthly statement from the broker - $5,000 transferred out of his account!

XSRF True Story (2)



Hidden iframes submitted forms that…
- Changed user's email notification settings
- Linked a new checking account
- Transferred out $5,000
- Unlinked the account
- Restored email notifications

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

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)

End of Lecture 8