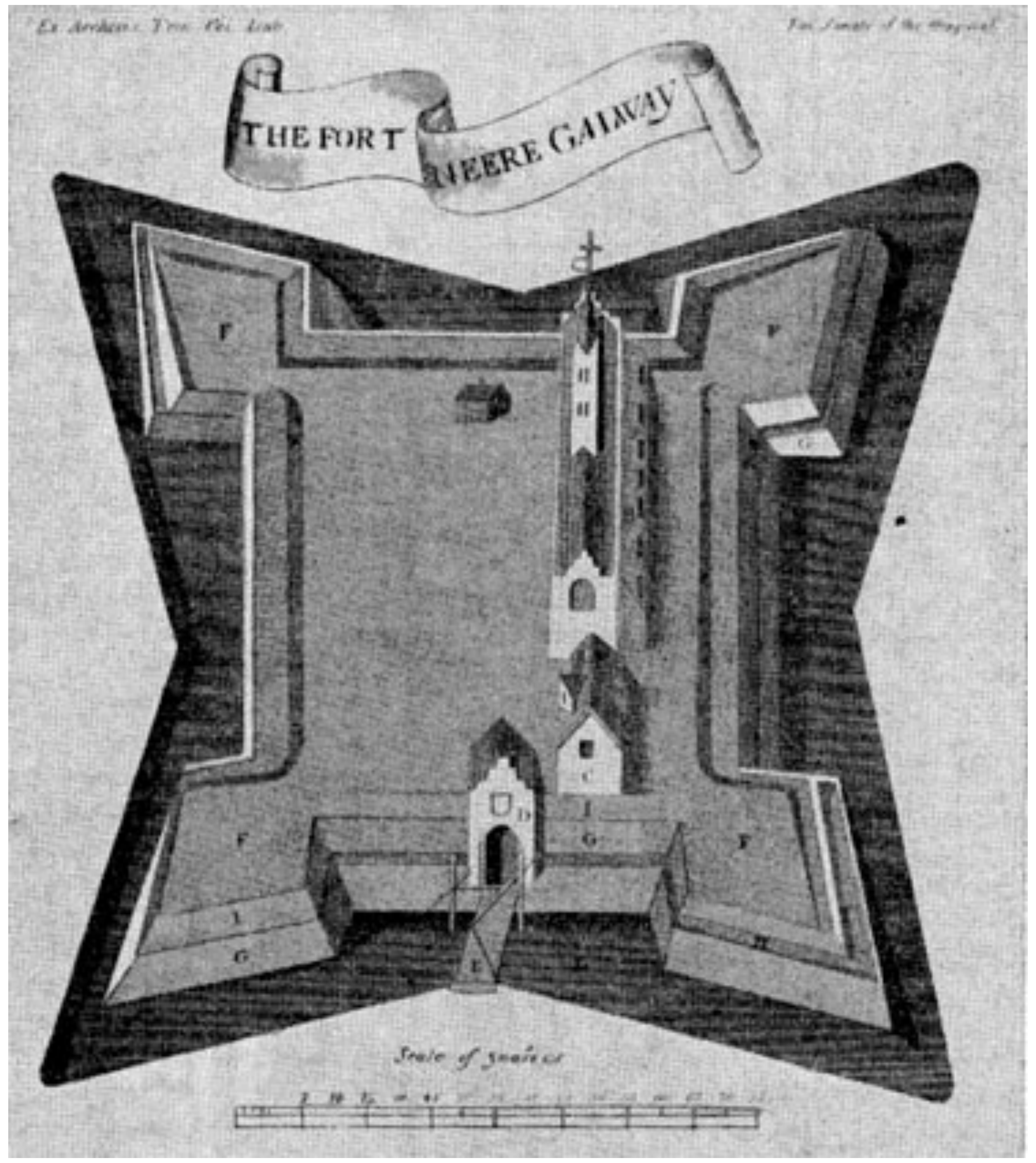
CSE543: Computer Security
Module: Web Security

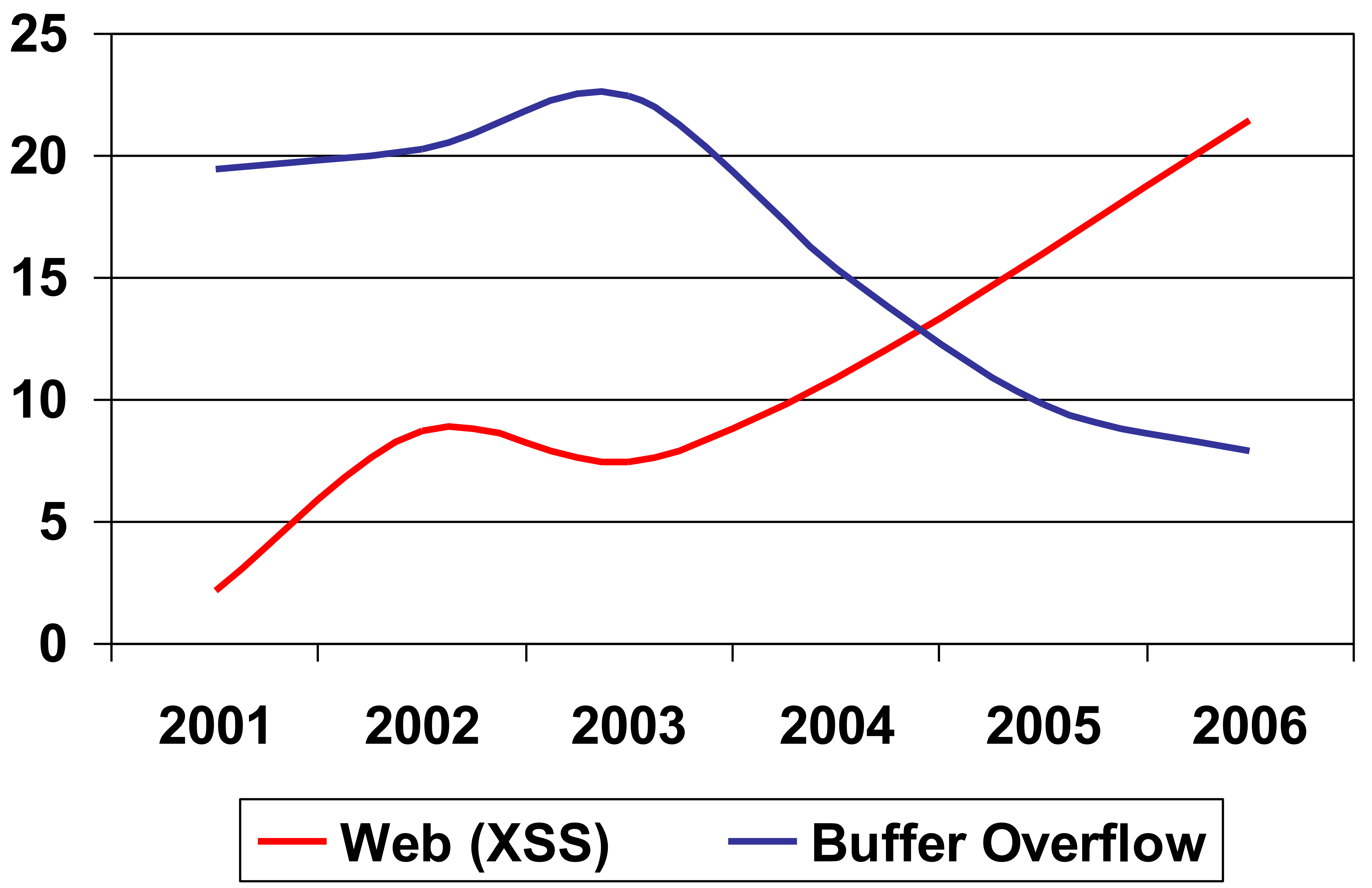
Prof. Syed Rafiul Hussain
Department of Computer Science and Engineering
Pennsylvania State University

Network vs. Web Security



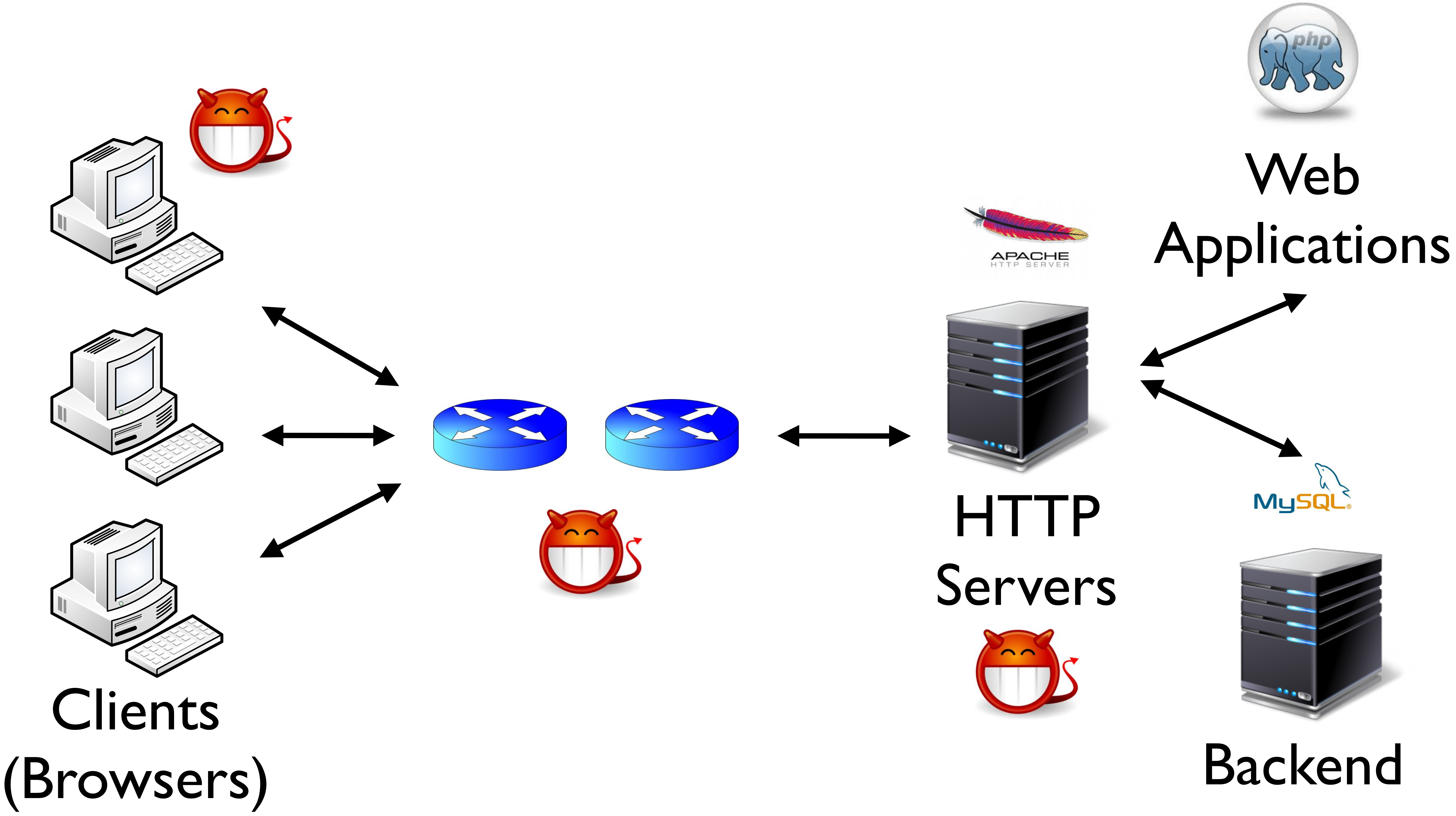
Web Vulnerabilities

- Web vulnerabilities surpassed OS vulnerabilities around 2005
  - The "new" buffer overflow



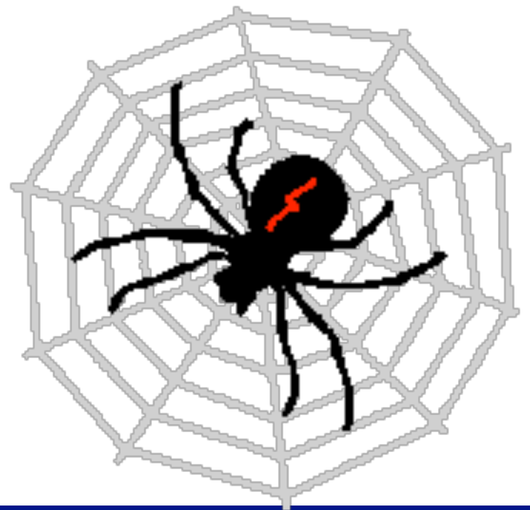
Components of the Web

- Multiple interacting components



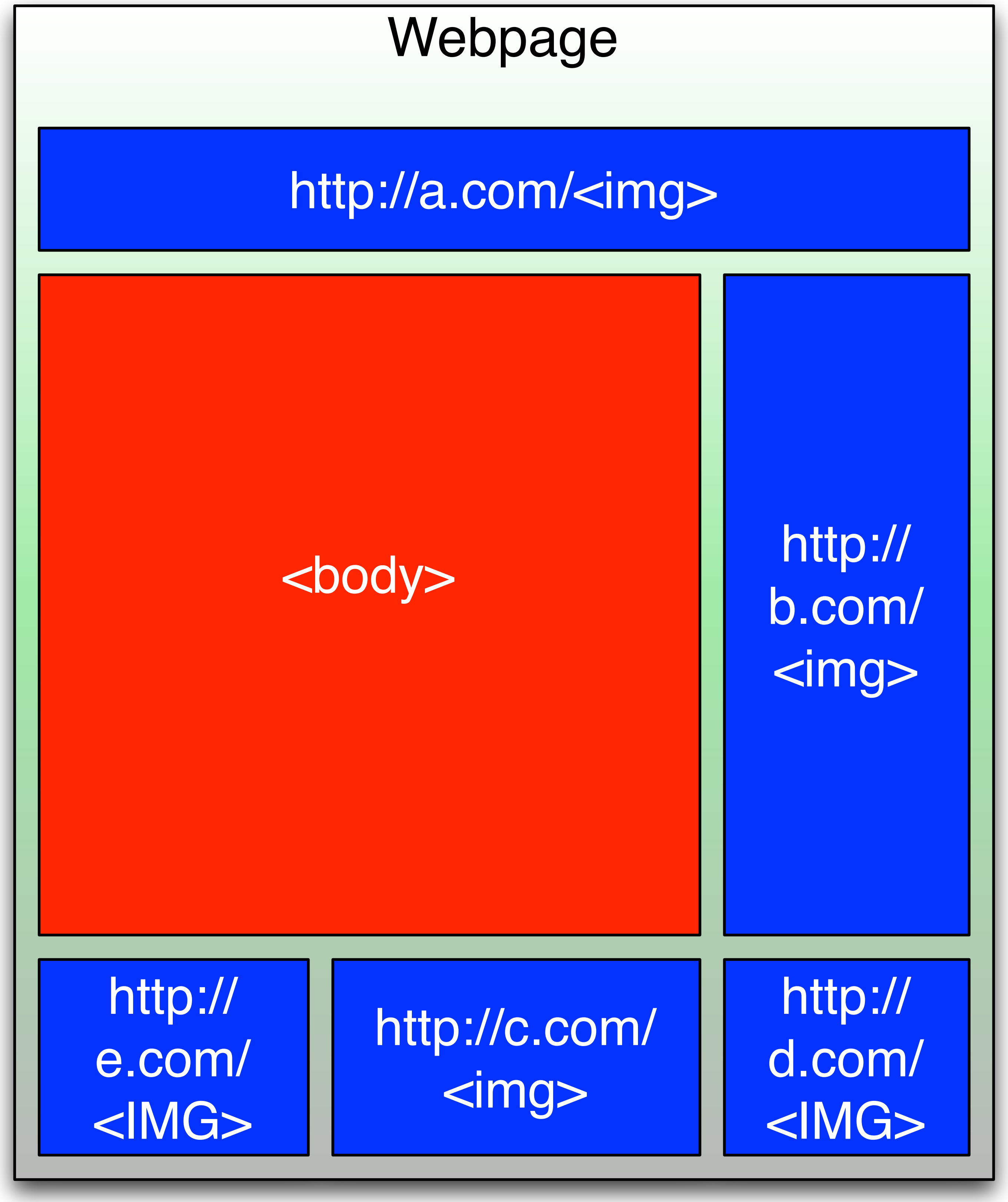
Web security: the high bits

- The largest distributed system in existence
- Multiple sources of threats, varied threat models
  - Users
  - Servers
  - Web Applications
  - Network infrastructure
  - We shall examine various threat models, attacks, and defenses
- Another way of seeing web security is
  - Securing the web infrastructure such that the integrity, confidentiality, and availability of content and user information is maintained



Early Web Systems

- Early web systems provided a click-render-click cycle of acquiring web content.
  - Web content consisted of static content with little user interaction.

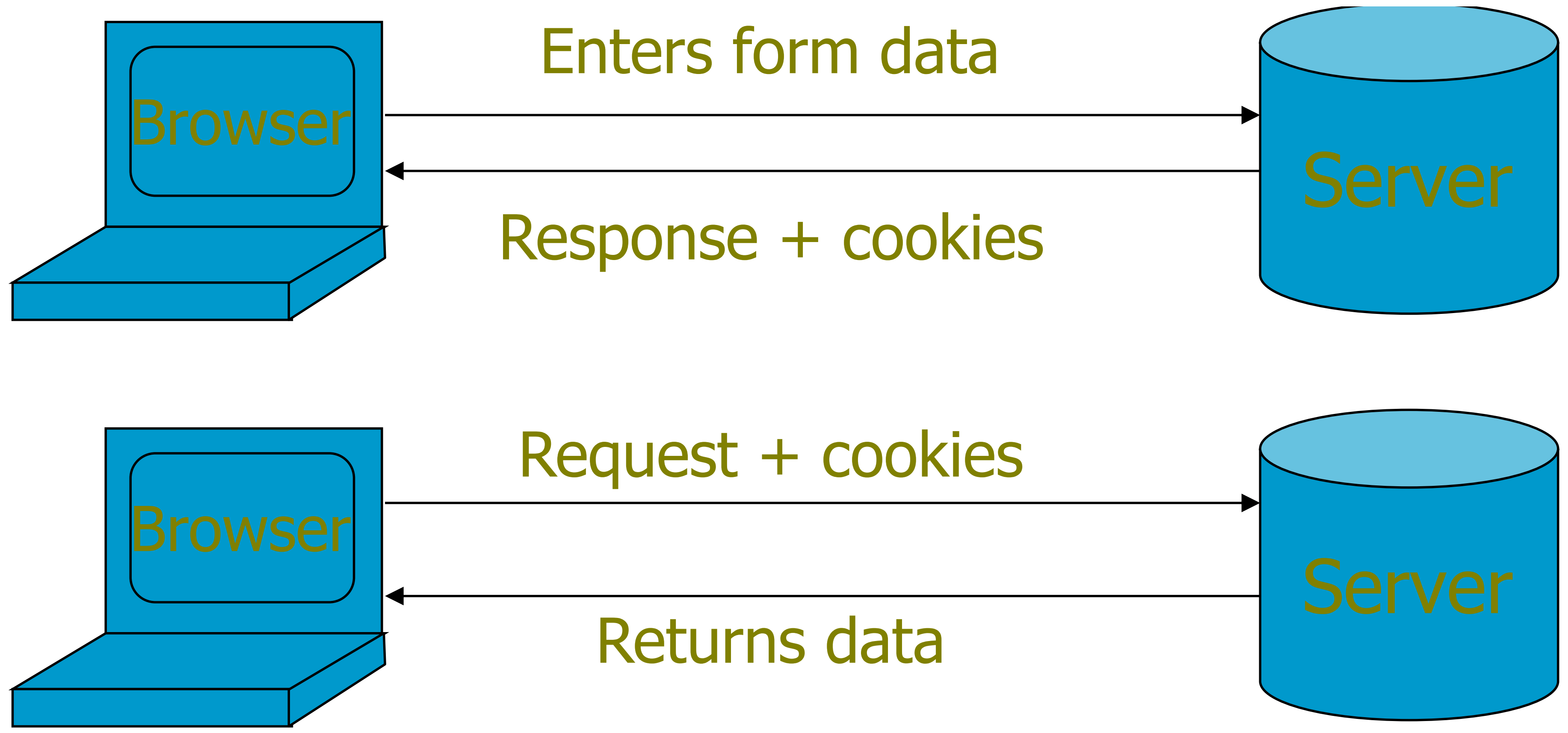


HTTP: Hyper Text Transfer Protocol

- Browser sends HTTP requests to the server
  - Methods: GET, POST, HEAD, …
  - GET: to retrieve a resource (html, image, script, css,…)
  - POST: to submit a form (login, register, …)
  - HEAD (a HEAD request could its Content-Length header to check the filesize without actually downloading the file)
- Server replies with a HTTP response
- Stateless request/response protocol
  - Each request is independent of previous requests
  - Statelessness has a significant impact on design and implementation of applications

Adding State to the Web:Cookies

- Cookies were designed to offload server state to browsers
  - Not initially part of web tools (Netscape)
  - Allows users to have cohesive experience
  - E.g., flow from page to page,
- Someone made a design choice
  - Use cookies to *authenticate* and *authorize* users
  - E.g. Amazon.com shopping cart, WSJ.com
- Q: What is the threat model?



Cookies
A cookie is a name/value pair created by a website to store information on your computer

Cookies

- An example cookie from my browser

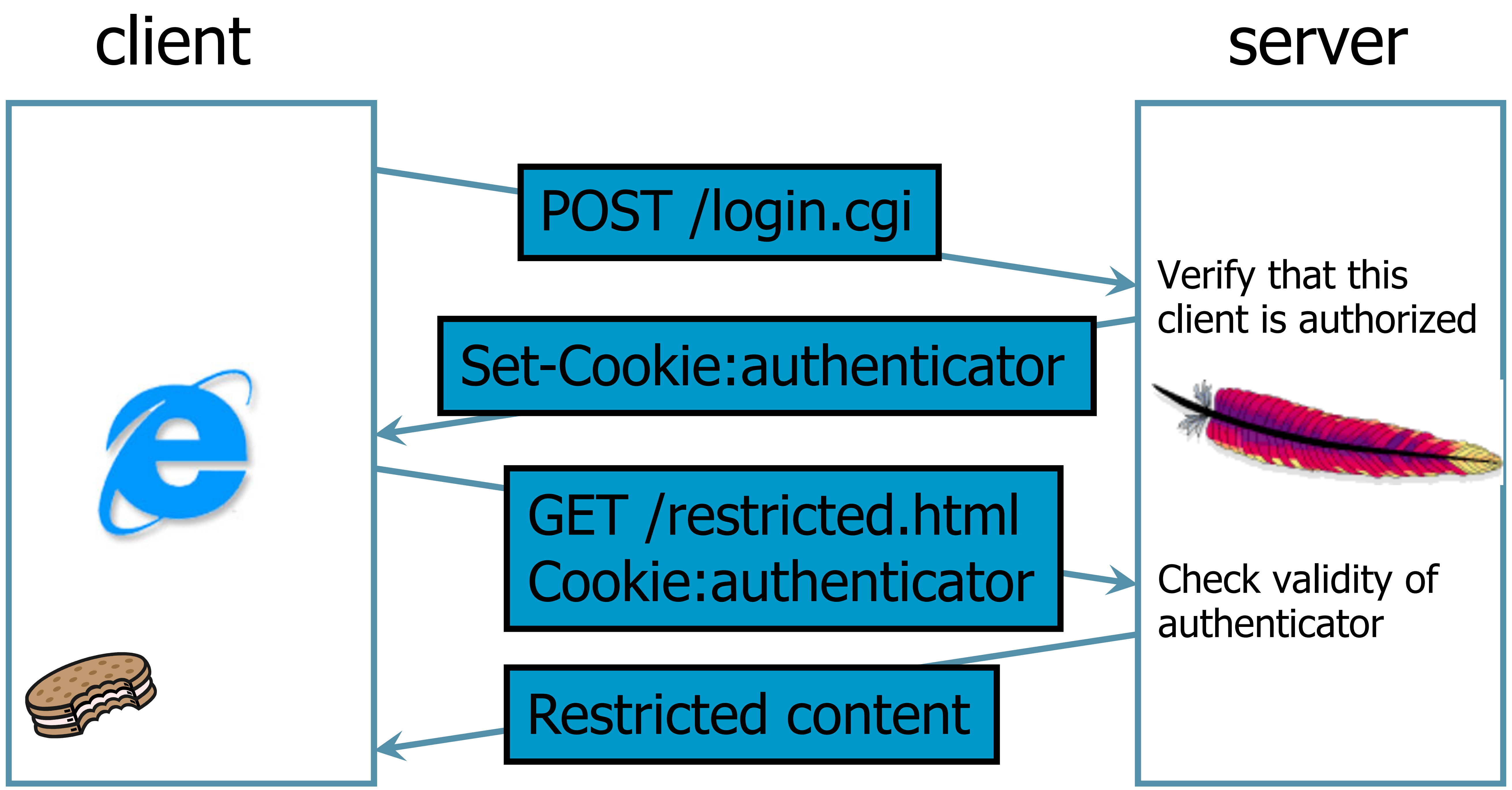
Name	session-token
Content	"s7yZiOvFm4YymG…."
Domain	.amazon.com
Path	/
Send For	Any type of connection
Expires	Monday, September 08, 2031 7:19:41 PM

- Stored by the browser and used by the web applications
  - used for authenticating, tracking, and maintaining specific information about users
  - e.g., site preferences, contents of shopping carts
  - data may be sensitive
  - may be used to gather information about specific users
- Cookie ownership: Once a cookie is saved on your computer, only the website that created the cookie can read it

Web Authentication via Cookies

- HTTP is stateless
  - How does the server recognize a user who has signed in?
- Servers can use cookies to store state on client
  - After client successfully authenticates, server computes an authenticator and gives it to browser in a cookie
    - Client cannot forge authenticator on his own (session id)
  - With each request, browser presents the cookie
  - Server verifies the authenticator

A Typical Session with Cookies



Authenticators must be unforgeable and tamper-proof
(malicious clients shouldn’t be able to modify an existing authenticator)
How to design it?

Cookie Issues …

- New design choice means
  - Cookies must be protected
    - Against forgery (integrity)
    - Against disclosure (confidentiality)
- Cookies not robust against web designer mistakes, committed attackers
  - Were never intended to be
  - Need the same scrutiny as any other tech.

Many security problems arise out of a technology built for one thing incorrectly applied to something else.

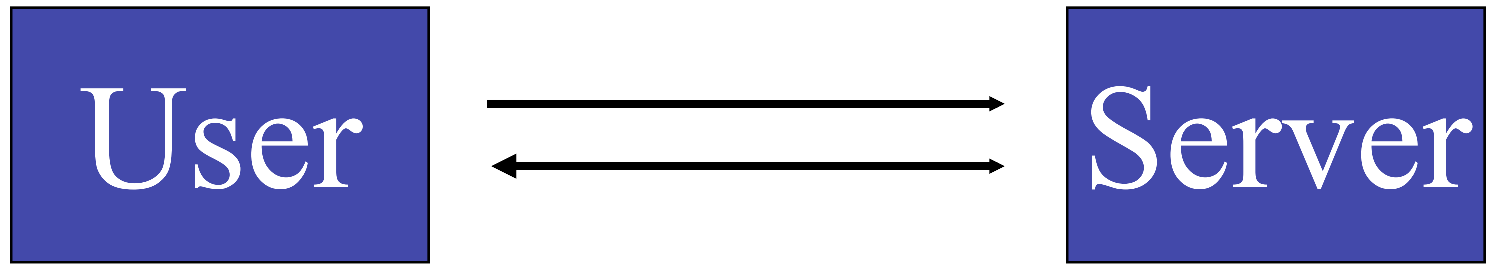
Cookie Design 1: mygorilla.com

- Requirement: authenticate users on site

myschool.com

- Design:

1. set cookie containing hashed username
2. check cookie for hashed username



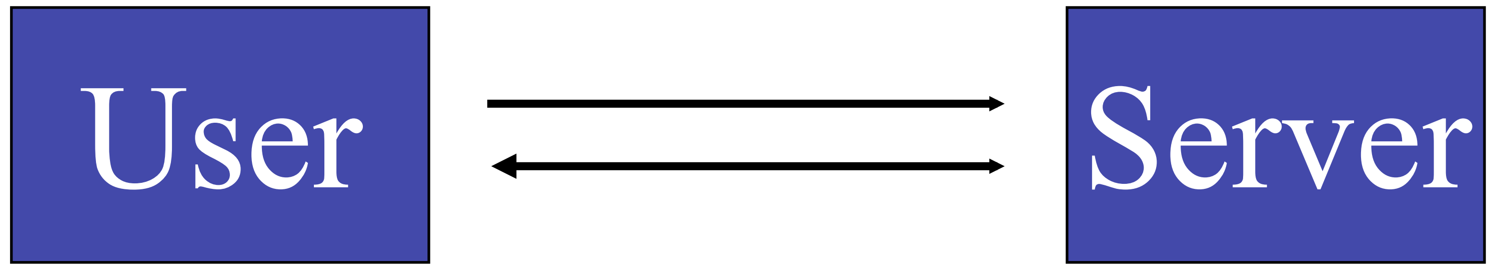
- Q: Is there anything wrong with this design?

Cookie Design 2: mygorilla.com

- Requirement: authenticate users on site

myschool.com

- Design:
  1. set cookie containing encrypted username
  2. check cookie for encrypted username



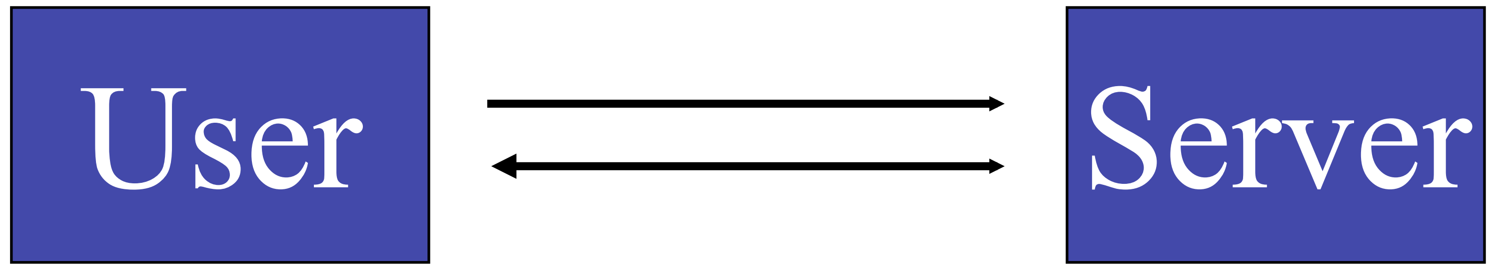
- Q: Is there anything wrong with this design?

Cookie Design 2: mygorilla.com

- Requirement: authenticate users on site

myschool.com

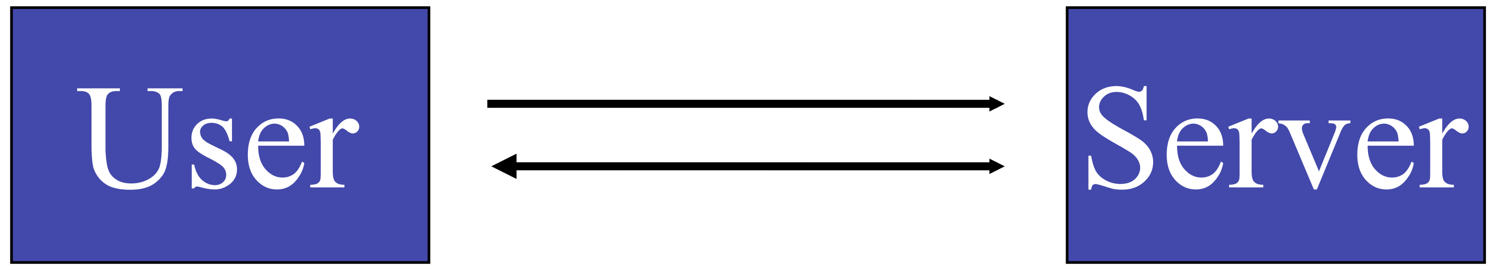
- Design:
  1. set cookie containing encrypted + HMAC'd username
  2. check cookie for encrypted + HMAC'd username



- Q: Is there anything wrong with this design?

Exercise: Cookie Design

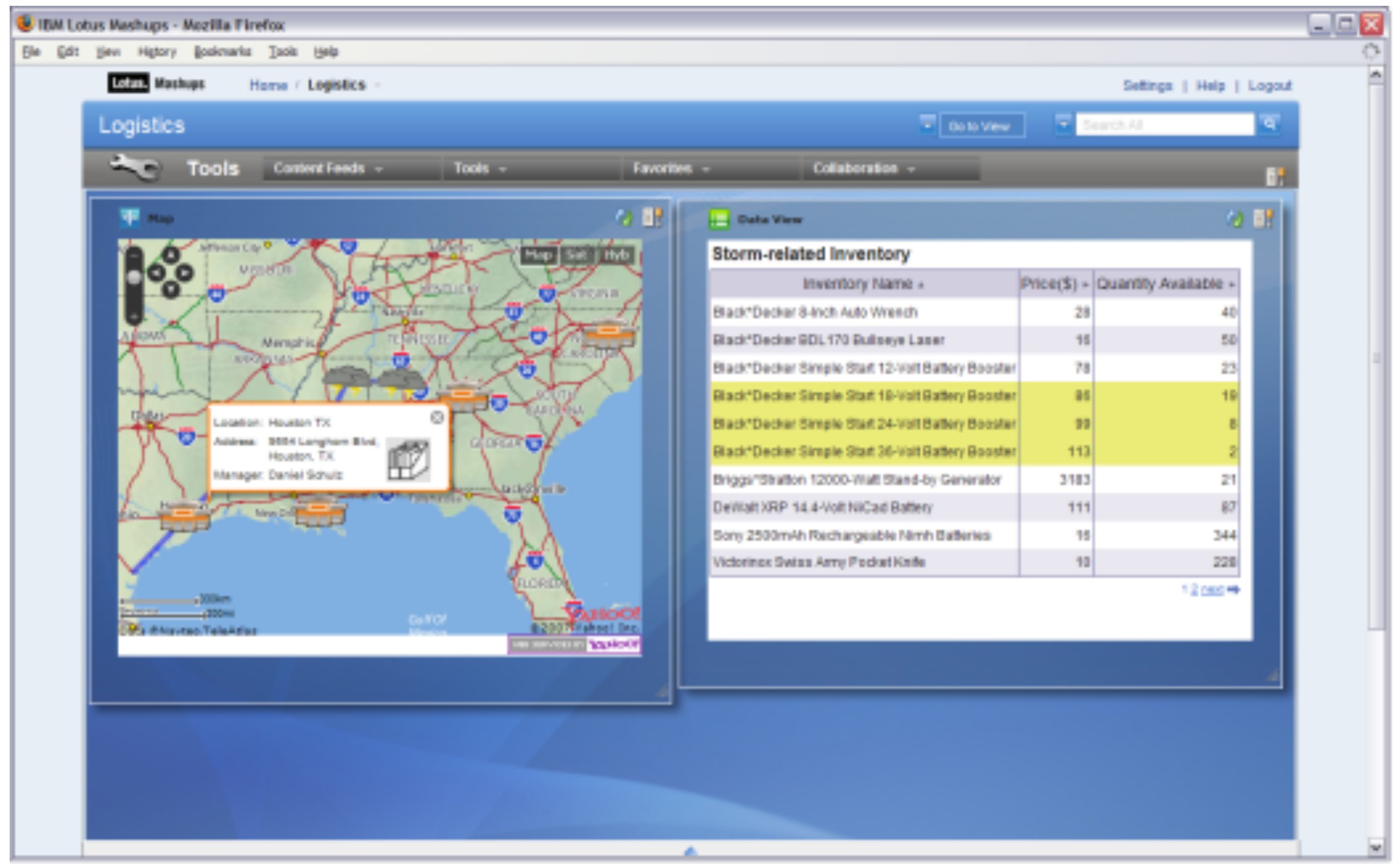
- Design a secure cookie for myschool.com that meets the following requirements
- Requirements
  - Users must be authenticated (assume digest completed)
  - Time limited (to 24 hours)
  - Unforgeable (only server can create)
  - Privacy-protected (username not exposed)
  - Location safe (cannot be replayed by another host)



$$E\{k_s, \text{"}host\_ip : timestamp : username\text{"}\} + HMAC\{k_s, \text{"}\ldots\text{"}\}$$

Content from Multiple Sites

- Browser stores cookies from multiple websites
  - Tabs, mashups, ...
- Q. What is the threat model?
- More generally, browser stores *content* from multiple websites
  - HTML pages
  - Cookies
  - Flash
  - Java applets
  - JavaScript



- How do we isolate content from multiple sites?

Client Side Scripting

- Web pages (HTML) can embed dynamic contents (code) that can be executed on the browser
- JavaScript
  - embedded in web pages and executed inside browser
- Java applets
  - small pieces of Java bytecodes executed in browsers

HTML and Scripting

```
<html>
    …
  <P>
<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

Client-side scripting can access (read/wrtie) the following resources

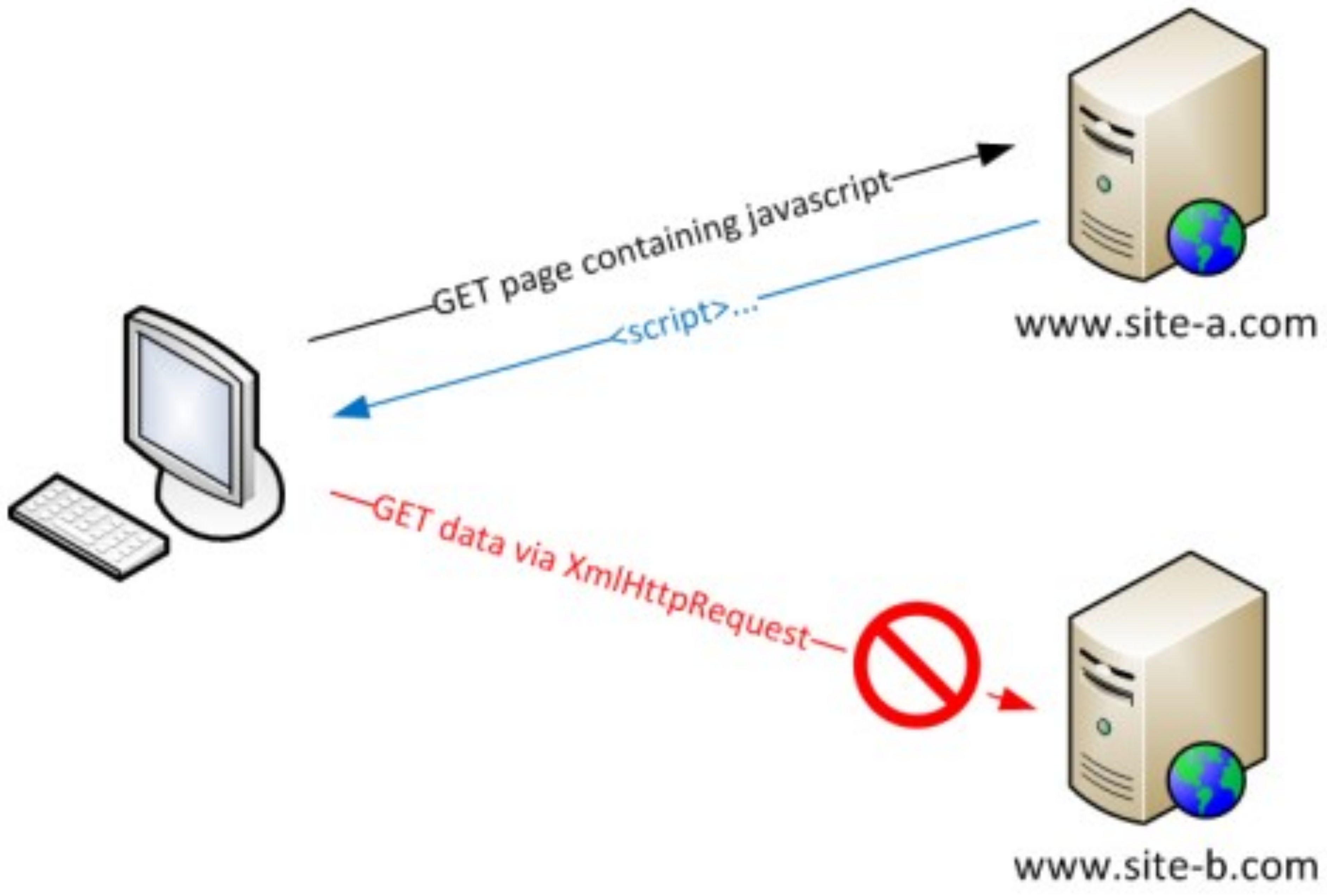
- Local files on the client-side host
- Webpage resources maintained by the browser: Cookies, Domain Object Model (DOM) objects
  - steal private information
  - control what users see
  - impersonate the user

Browser as an OS

- Web users visit multiple websites simultaneously
- A browser serves web pages (which may contain programs) from different web domains
  - i.e., a browser runs programs provided by mutually untrusted entities
  - Running code one does not know/trust is dangerous
  - A browser also maintains resources created/updated by web domains
- Browser must confine (sandbox) these scripts so that they cannot access arbitrary local resources
- Browser must have a security policy to manage/protect browser-maintained resources and to provide separation among mutually untrusted scripts

Same-Origin Policy

- A set of policies for isolating content (scripts and resources) across different sites (*origins*)
  - *E.g.,* evil.org *scripts cannot access* bank.com *resources.*
- What is an origin?
  - site1.com vs site2.com?
    - Different hosts are different origins
  - http://site.com vs https://site.com?
    - Different protocols are different origins
  - http://site.com:80 vs http://site.com:8080?
    - Different ports are different origins
  - http://site1.com vs http://a.site1.com?
    - Establishes a hierarchy of origins

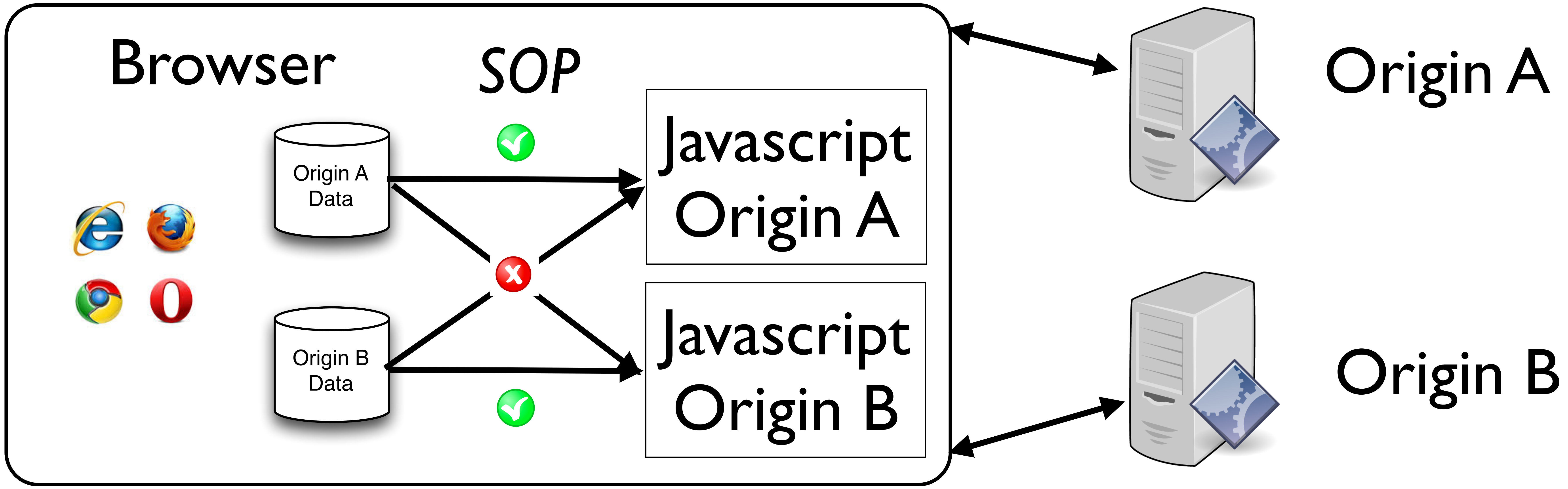


SOP: What it Controls?

- Same-origin policy applies to the following accesses:
  - manipulating browser windows
  - URLs requested via the XmlHttpRequest
    - XmlHttpRequest is an API that can be used by web browser scripting languages to transfer XML and other text data to and from a web server using HTTP, by establishing an independent and asynchronous communication channel.
    - used by AJAX
  - manipulating frames (including inline frames)
  - manipulating documents (included using the object tag)
  - manipulating cookies

Same-Origin Policy

- *Principle*: Any active code from an origin can read only information stored in the browser that is from the same origin
  - Active code: Javascript, VBScript,…
  - Information: cookies, HTML responses, ...



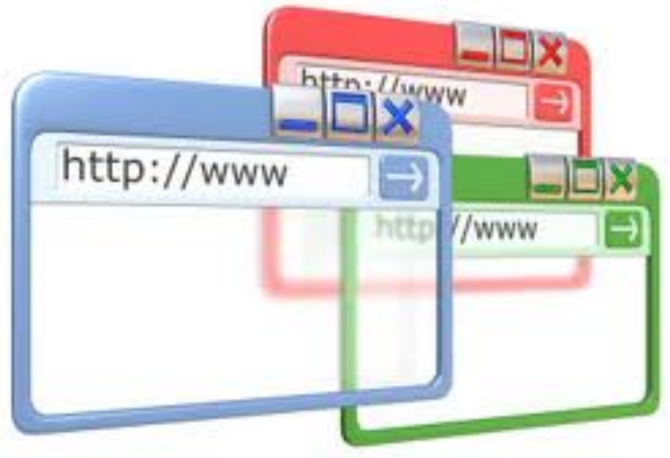
Document Domain

- Scripts from two origins in the same domain may wish to interact
  - www.example.com and program.example.com
- Any web page may set *document.domain* to a
  - “right-hand, fully-qualified fragment of its current host name” (example.com, but not ample.com)
- Then, all scripts in that domain may share access
  - All or nothing
- NOTE: Applies “null” for port, so does not actually share with normal example.com:80

SOP Weaknesses

PennState

- Complete and partial bypasses exist
  - Browser bugs
  - Limitations if site hosts unrelated pages
    - Example: Web server often hosts sites for unrelated parties
    - http://www.example.com/account/
    - http://www.example.com/otheraccount/
    - Same-origin policy allows script on one page to access document properties from another
  - Functionality often requires SOP bypass!
    - Many advertisement companies hire people to find and exploit SOP browser bugs for cross-domain communication
    - E.g., JSON with padding (JSONP)
- Cross-site scripting
  - Execute scripts from one origin in the context of another



Cross Site Scripting (XSS)

- Recall the basics
  - scripts embedded in web pages run in browsers
  - scripts can access cookies
    - get private information
  - and manipulate DOM objects
    - controls what users see
  - scripts controlled by the same-origin policy
- Why would XSS occur
  - Web applications often take user inputs and use them as part of webpage

Cross-Site Scripting

- Assume the following is posted to a message board on your favorite website which will be displayed to everyone:

```
Hello message board.

<SCRIPT>malicious code</SCRIPT>
This is the end of my message.
```

- Now a reasonable ASP (or some other dynamic content generator) uses the input to create a webpage (e.g., blogger nonsense).
- Anyone who view the post on the webpage can have local authentication cookies stolen.
- Now a malicious script is running
  - Applet, ActiveX control, JavaScript…

Cross-Site Scripting

PennState

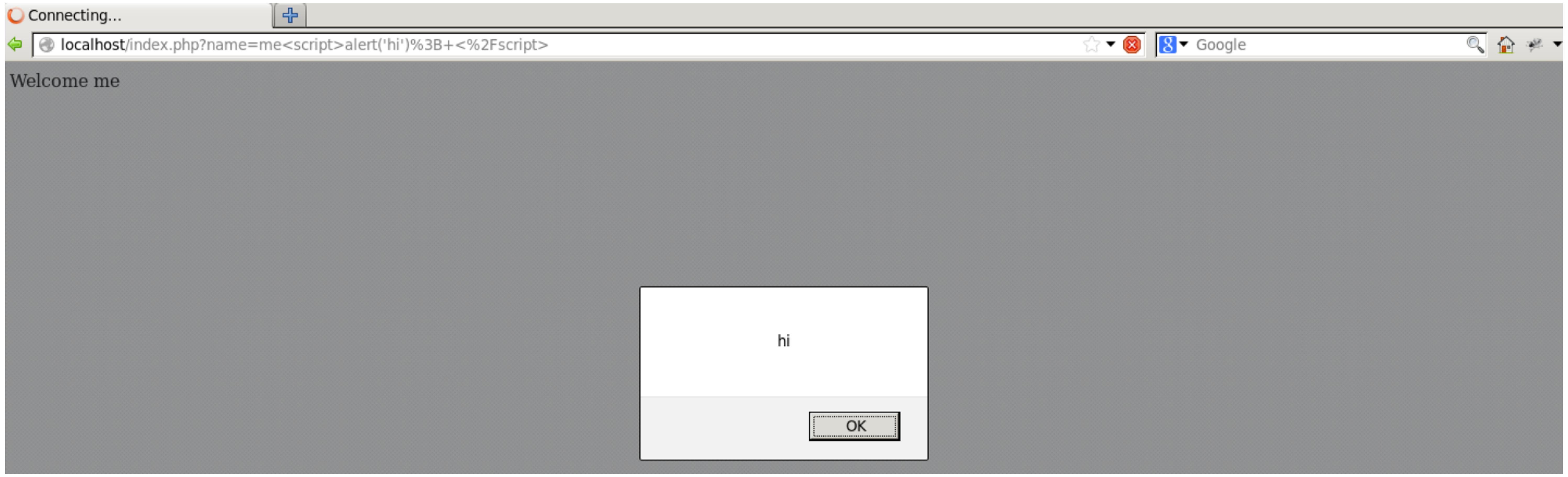
- Script from attacker is executed in the victim origin's context
  - Enabled by inadequate filtering on server-side
- Effects of Cross-Site Scripting
  - Can manipulate any DOM component on victim.com
  - Control links on page
  - Control form fields (e.g. password field) on this page and linked pages.
  - Can infect other users: MySpace.com worm
- Three types
  - Reflected
  - Stored
  - DOM Injection

Reflected XSS

```php
<?php
$name = $_GET['name'];
echo "Welcome $name<br>";
?>

<form method="get" action="index.php">
    Name: <input type="text" name="name" /><br />
    <input type="submit" value="submit" />
</form>
```

```
index.php?name=guest<script>alert('hi')</script>
```



MySpace.com (Samy worm)

PennState

- Users can post HTML on their pages
  - MySpace.com ensures HTML contains no

```
<script>, <body>, onclick, <a href=javascript://>
```

  - However, attacker find out that a way to include Javascript within CSS tags:

```
<div style="background:url('javascript:alert(1)')">
```

  - And can hide "javascript" as "java\nscript"
- With careful javascript hacking:
  - Samy's worm: infects anyone who visits an infected MySpace page … and adds Samy as a friend.
  - Samy had millions of friends within 24 hours.
- More info: http://namb.la/popular/tech.html

Web Systems Evolve ...

- The web has evolved from a *document retrieval* and rendering to sophisticated *distributed application platform* providing:
  - dynamic content
  - user-driven content
  - interactive interfaces
  - multi-site content
  - ....

- With new interfaces comes new vulnerabilities ...

Cross-site Request Forgery

PennState

- An XSS attack exploits the trust the browser has in the server to filter input properly
- A CSRF attack exploits the trust the server has in a browser
  - Authorized user submits unintended request
    - Attacker Maria notices weak bank URL
      ```
      GET http://bank.com/transfer.do?acct=BOB&amount=100 HTTP/1.1
      ```
    - Crafts a malicious URL
      ```
      http://bank.com/transfer.do?acct=MARIA&amount=100000
      ```
    - Exploits social engineering to get Bob to click the URL
      ```
      <a href="http://bank.com/transfer.do?acct=MARIA&amount=100000">View my Pictures!</a>
      ```
    - Can make attacks not obvious
      ```
      <img src="http://bank.com/transfer.do?acct=MARIA&amount=100000" width="1" height="1" border="0">
      ```
  - Defense: Referer header
    - Bank does not accept request unless referred to (linked from) the bank's own webpage
    - Disadvantage: privacy issues

CSRF Explained

- More Example:
  - User logs in to bank.com. Forgets to sign off.
  - Session cookie remains in browser state
- Then user visits another site containing:

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

  - Browser sends user auth cookie with request
  - Transaction will be fulfilled
- Problem: The browser is a confused deputy; it is serving both the websites and the user and gets confused who initiated a request
- https://www.youtube.com/watch?v=5joX1skQtVE&feature=emb_logo

HTTP Response Splitting

- Again, due to insufficient server-side filtering
  - Cookies can be set to arbitrary values to split HTTP response

```
String author = request.getParameter(AUTHOR_PARAM);
...
Cookie cookie = new Cookie("author", author);
cookie.setMaxAge(cookieExpiration);
response.addCookie(cookie);
```

```
HTTP/1.1 200 OK
...
Set-Cookie: author=Jane Smith
...
```

```
HTTP/1.1 200 OK
...
Set-Cookie: author=Wiley Hacker

HTTP/1.1 200 OK
...
```

  - Can be used for page hijacking through proxy server

Session Hijacking

- Virtual sessions are implemented in many ways
  - session ID in cookies, URLs
  - If I can *guess*, *infer*, or *steal* the session ID, game over
  - Login page using HTTPS, but subsequent communication is not! Cookies sent in cleartext
  - If your bank encodes the session ID in the url, then a malicious attacker can simply keep trying session IDs until gets a good one.
  - ... note that if the user was logged in, then the attacker has full control over that account. `http://www.mybank.com/loggedin?sessionid=11`
  - Countermeasure: HTTPS, secure cookie design

Privacy

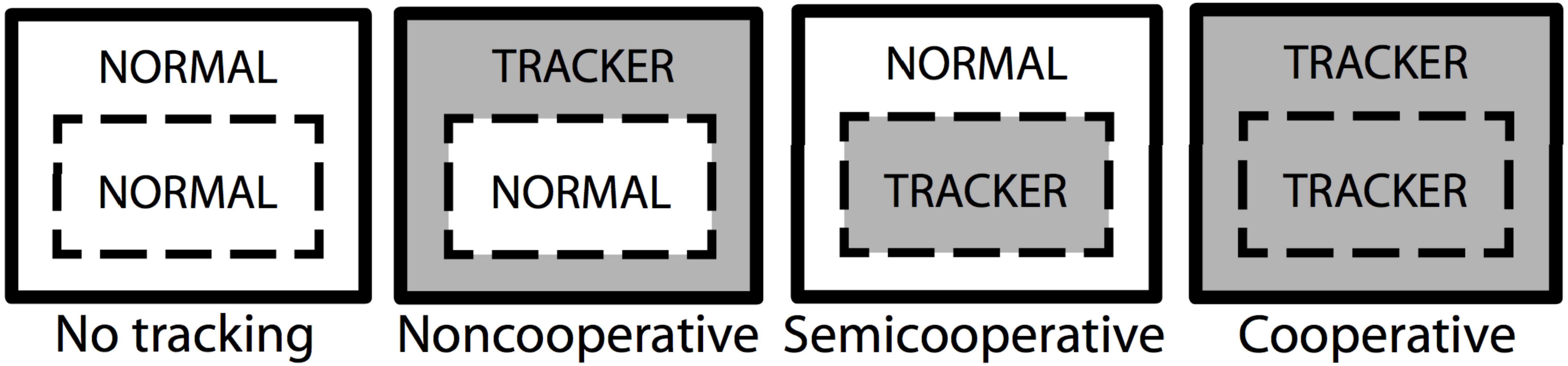
- Have you ever …
  - Searched for a product on some website
  - …Advertisement for the same product shows up on another website?
  - Reason: Tracking! Profile users for targeted advertisement
- Study by WSJ found (2012)
  - 75% of top 1000 sites feature social networking plugins
    - Can match users' identities with web-browsing activities
- abine and UC Berkeley found
  - Online tracking is 25% of browser traffic
    - 20.28% google analytics
    - 18.84% facebook

http://www.abine.com/

Privacy

- Tracking is done in following configurations

Protecting Browser State from Web Privacy Attacks : Jackson et al.



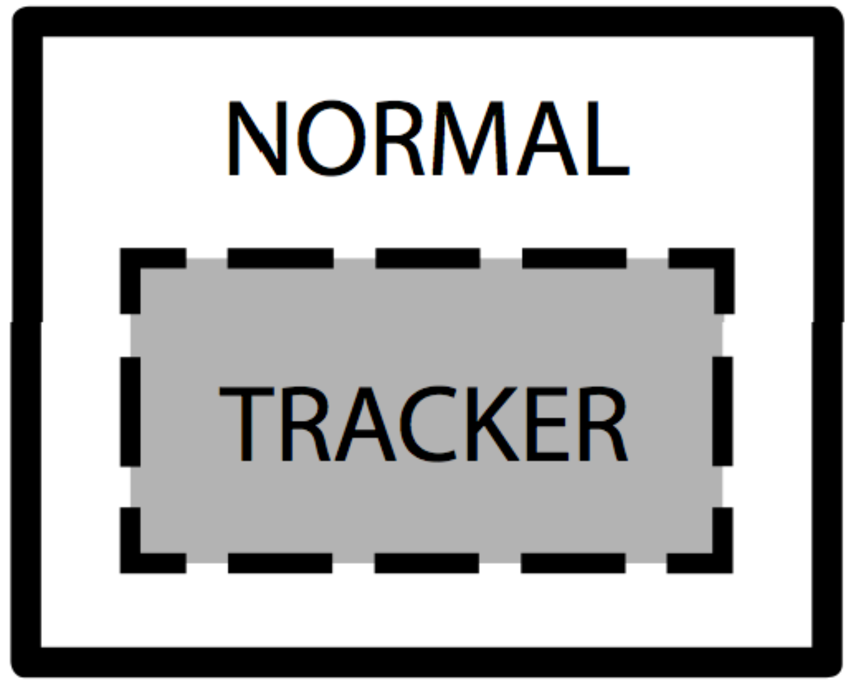
- "Tracker" code is from
  - Social networking sites
  - Analytics
  - Advertisement agencies
  - ...

Privacy

- Objective of tracking code is to maintain state of users across multiple sites
  - Build profile of sites visited
- Semi-cooperative tracking done by
  - Javascript
    - e.g., Cached redirect URLs
  - Web bugs
    - 1x1 images
    - Ever wondered why email clients have "Display images"?
  - IFrames
  - Cookies
    - Traditional, flash, HTML5 LocalStorage, …
- Tasks: (1) get your tracking code running; (2) store state; (3) send to server

Third-Party Cookies

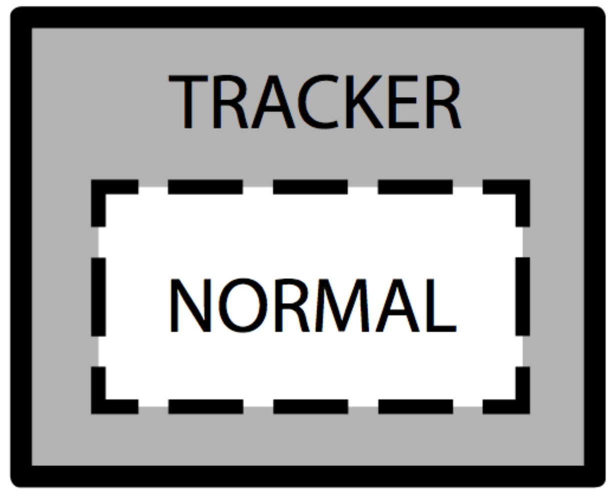
- A third-party cookie is a cookie from a website different from the website being viewed
- Browsers can block third-party cookies
  - Different browsers have different variations
    - Some completely block
    - “Do Not Track” - except Chrome
- Limitation
  - Other ways exist to store state (more)
    - Canvas fingerprinting
    - Evercookies
    - “Cookie syncing”
- OpenWPM - https://github.com/citp/OpenWPM



Semicooperative

Unintended Tracking

- “Data” access not all governed by same-origin policy
  - Specified: HTML DOM, cookies
  - What about
    - Web caches?
      - Tracking notes time to fetch URL
      - If URL in cache, served faster
    - Visited links?
      - Mostly fixed in current browsers




```
a          { color: blue; }
a:visited { color: red;  }

if (document.getElementById('jones').currentStyle.color=='red')
  document.writeln('<p>Hello! I see you\'ve been to Jones.');
  document.writeln('Don\'t buy from Jones - their widgets');
  document.writeln('are made from recycled babies.<\/p>');
```

- Take-away: Difficult to prevent tracking if *any* browser state is stored
- To mitigate tracking
  - Reset browser regularly, store no state, visit random sites!

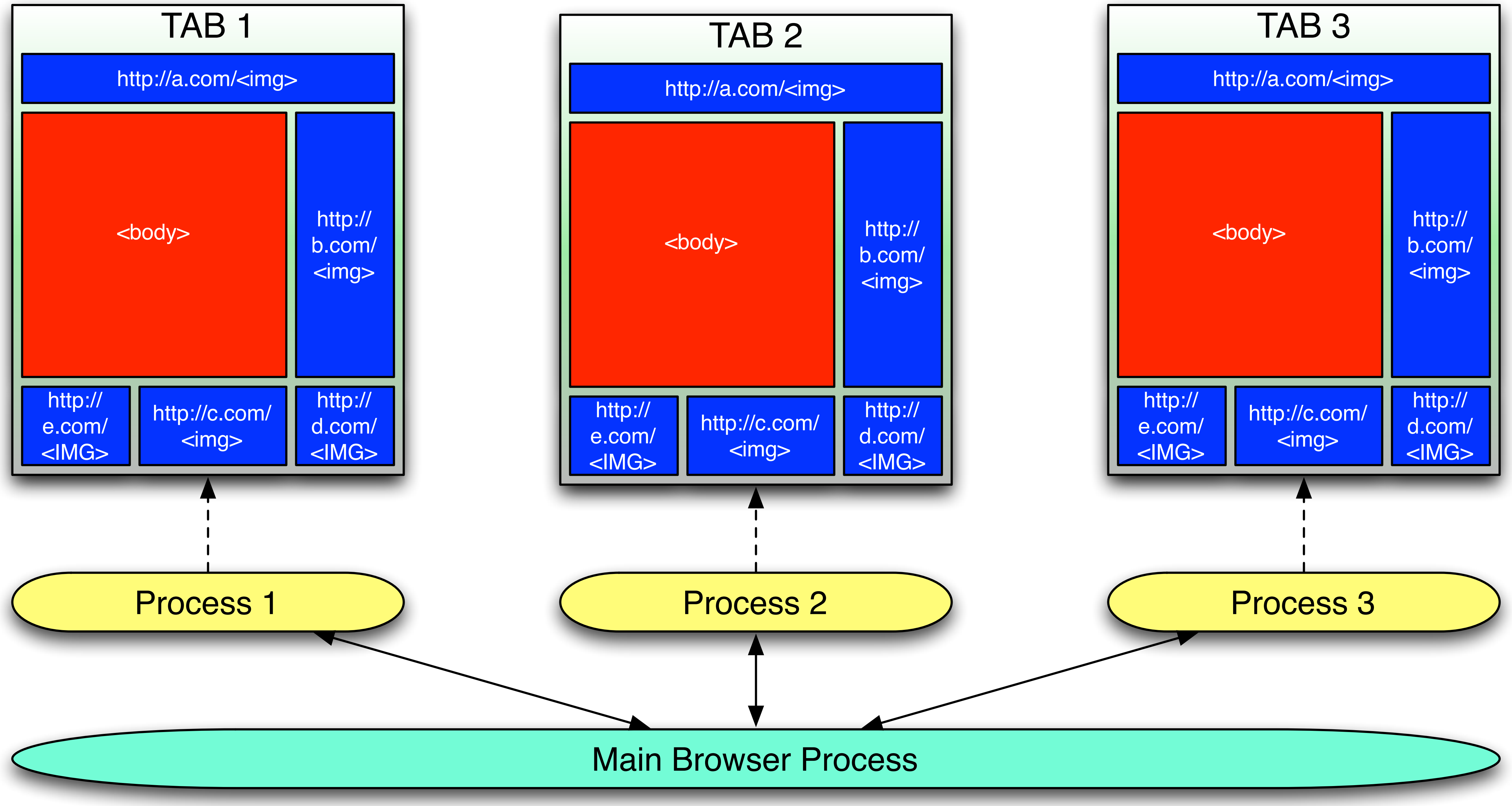
Browsers

- Browsers are the new operating systems
- Huge, complex systems that support
  - Many document types, structures, e.g., HTML, XML, ...
  - Complex rendering, e.g., CSS, CSS 2.0
  - Many "program/scripting" languages, e.g., JavaScript
  - Dynamic content, e.g., AJAX
  - Native code execution, e.g., ActiveX

- Virtualized computers in a single program ...

Browser Security

- We don't have the ability to control this much complexity, so we have to try other things …
  - Restricting functionality, e.g., NoScript
  - Process Isolation, e.g., OP, Chrome
    - Read: http://www.google.com/googlebooks/chrome/



OP Browser

- What did they do to build a more secure browser?
- (1) Decompose the browser into multiple processes
  - Called "Privilege Separation"
- What are the permissions of a set of processes forked from the same parent?

OP Browser

- What did they do to build a more secure browser?
- (1) Decompose the browser into multiple processes
  - Called "Privilege Separation"
- What are the permissions of a set of processes forked from the same parent? Same as parent
- (2) Need different policy for each process
  - Multiple subjects in the access control policy
- What browser processes are trusted to manage the permissions?

OP Browser

- What did they do to build a more secure browser?
- (1) Decompose the browser into multiple processes
  - Called "Privilege Separation"
- What are the permissions of a set of processes forked from the same parent? Same as parent
- (2) Need different policy for each process
  - Multiple subjects in the access control policy
- What browser processes are trusted to manage the permissions? None
- (3) Need mandatory access control
  - Subjects cannot escape confined "protection domain"

OP Browser

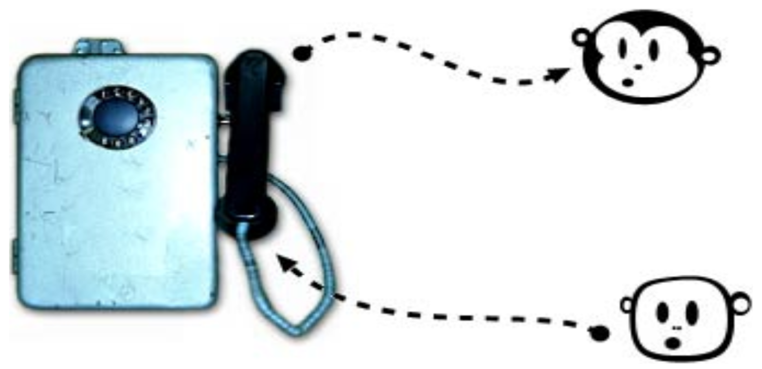
- How do you determine what parts of the browser should be a "subject" and identify the permissions to be assigned to that subject?
- One subject (client)
  - Code that requires the same permissions to run
  - E.g., a particular web page
- Another subject (server)
  - Code that manages the same permissions
  - E.g., UI, network, and storage subsystems
- How do we determine the permission assignments?

OP Browser

- How do you determine what parts of the browser should be a "subject" and identify the permissions to be assigned to that subject?
- One subject (client)
  - Code that requires the same permissions to run
  - E.g., a particular web page
- Another subject (server)
  - Code that manages the same permissions
  - E.g., UI, network, and storage subsystems
- How do we determine the permission assignments?
  - Least privilege
  - Information flow

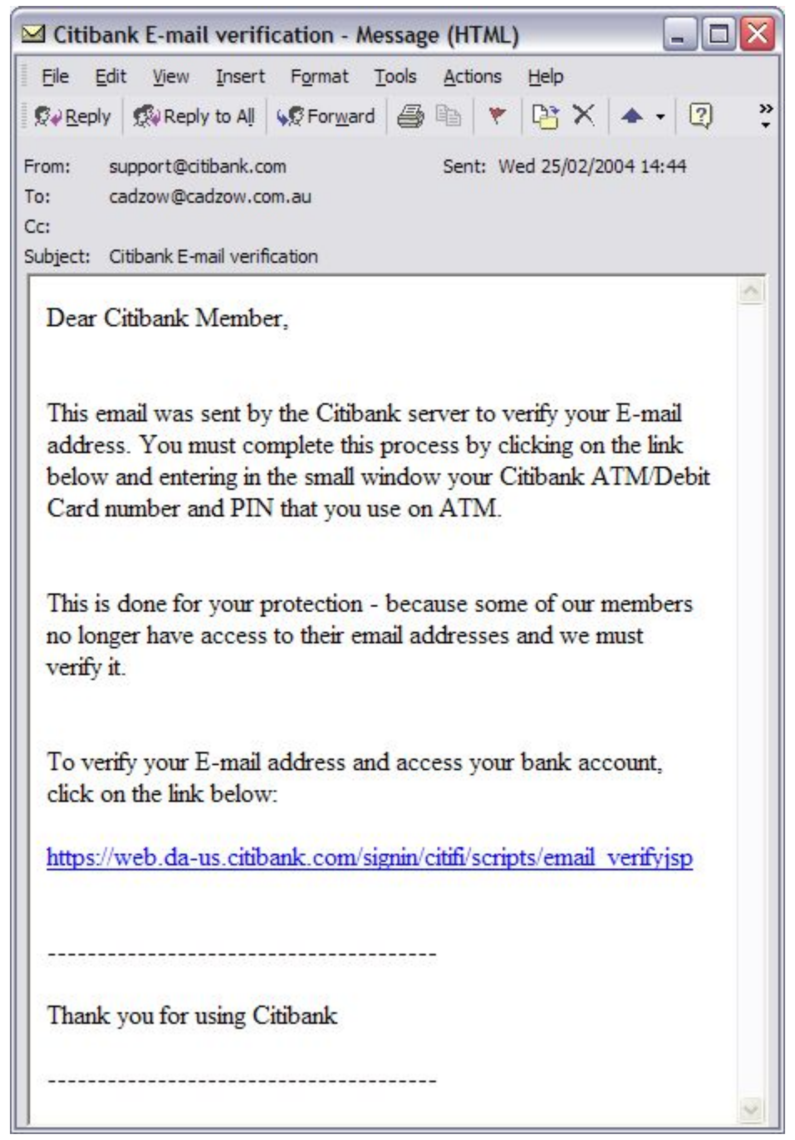
Applications/Plugins

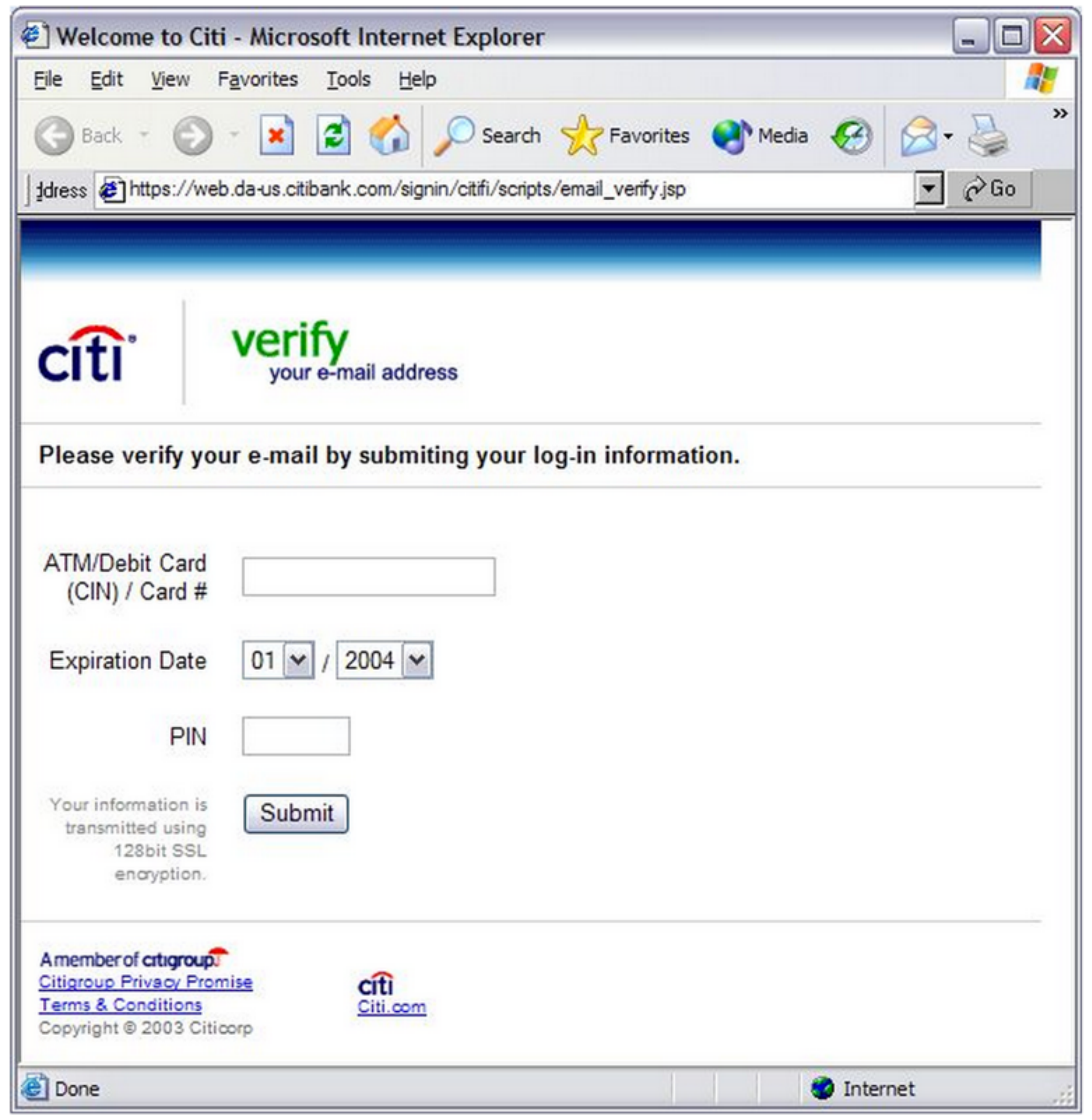
- A *plugin* is a simply a program used by a browser to process content
  - MIME type maps content to plugin
  - Like any old application (e.g., RealAudio)
  - Newer browsers have autoinstall features
- Plugins are sandboxed, but have been circumvented in various ways
  - Interesting design point - Google Chrome allows "native" plugins but still preserves (some) security!
    - Native Client sandbox for running compiled C/C++ code
- Moral: beware of plugins



Social Engineering

- Attacks another weak point -- users!
- Phishing
  - Lure users using bait (fishing) to steal valuable information
  - Common technique: mimic original site and use similar URL
    - www.aol.com vs www.ao1.com
    - Combine with other techniques e.g., turn off address bar





Drive by downloads

- Using a deceptive means to get someone to install something on their own (spyware/adware)

  - Often appears as an error message on the browser
  - Sometimes, user does not click anything at all!
  - Concern: *extortion-ware* -- pay us $ to unencrypt your data
    - Used to demand $ for uninstall of annoying software
  - “biggest cybersecurity threat” - Kaspersky
- Answer: Back up stuff externally that you really want!

SQL Injection

- An injection that exploits the fact that many inputs to web applications are
  - under control of the user
  - used directly in SQL queries against back-end databases
- Bad form inserts escaped code into the input ...

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

- This vulnerability became one of the most widely exploited and costly in web history.
  - Industry reported as many as 16% of websites were vulnerable to SQL injection in 2007
  - This may be inflated, but has been an ongoing problem.

SQL Injection

- An injection that exploits the fact that many inputs to web applications are
  - under control of the user
  - used directly in SQL queries against back-end databases
- Bad form inserts escaped code into the input ...

```
SELECT email, login, last_name
  FROM user_table
 WHERE email = 'x'; DROP TABLE members; --';
```

- This vulnerability became one of the most widely exploited and costly in web history.
  - Industry reported as many as 16% of websites were vulnerable to SQL injection in 2007
  - This may be inflated, but has been an ongoing problem.

Preventing Web System Attacks

- Largely just applications
  - In as much as application are secure
  - Command shells, interpreters, are dangerous
- Broad Approaches
  - Validate input (also called *input sanitization*)
  - Limit program functionality
    - Don't leave open ended-functionality
  - Execute with limited privileges
  - Input tracking, e.g., *taint tracking*
  - Source code analysis, e.g., c-cured

Conclusion

- Web security has to consider threat models involving several parties
    - Web browsers
    - Web servers
    - Web applications
    - Users
    - Third-party sites
    - Other users
- Security is so difficult in the web because it was largely *retrofitted*
- zzz