

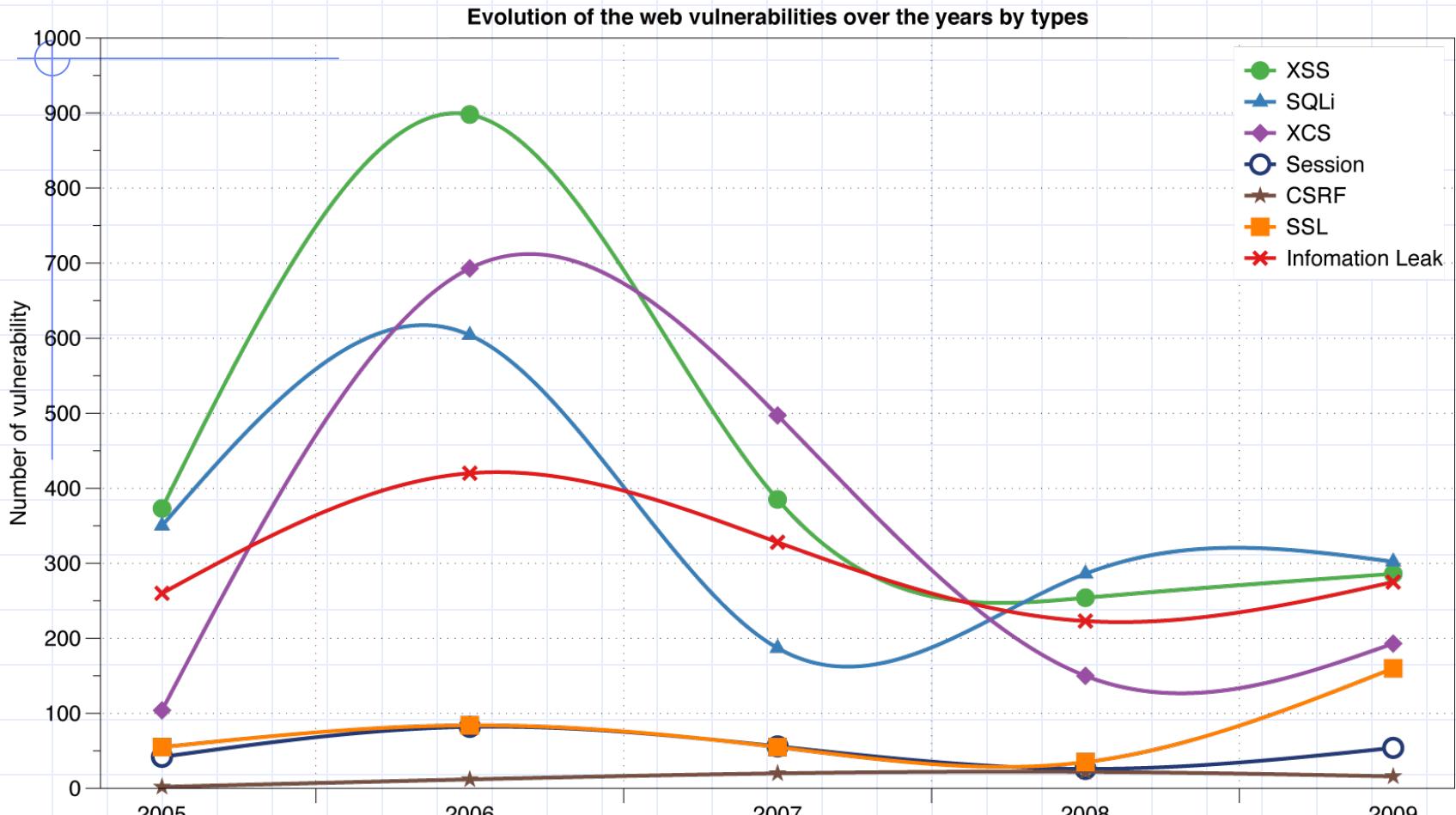
# Browser Security Model

John Mitchell

# Top Web Vulnerabilities 2017

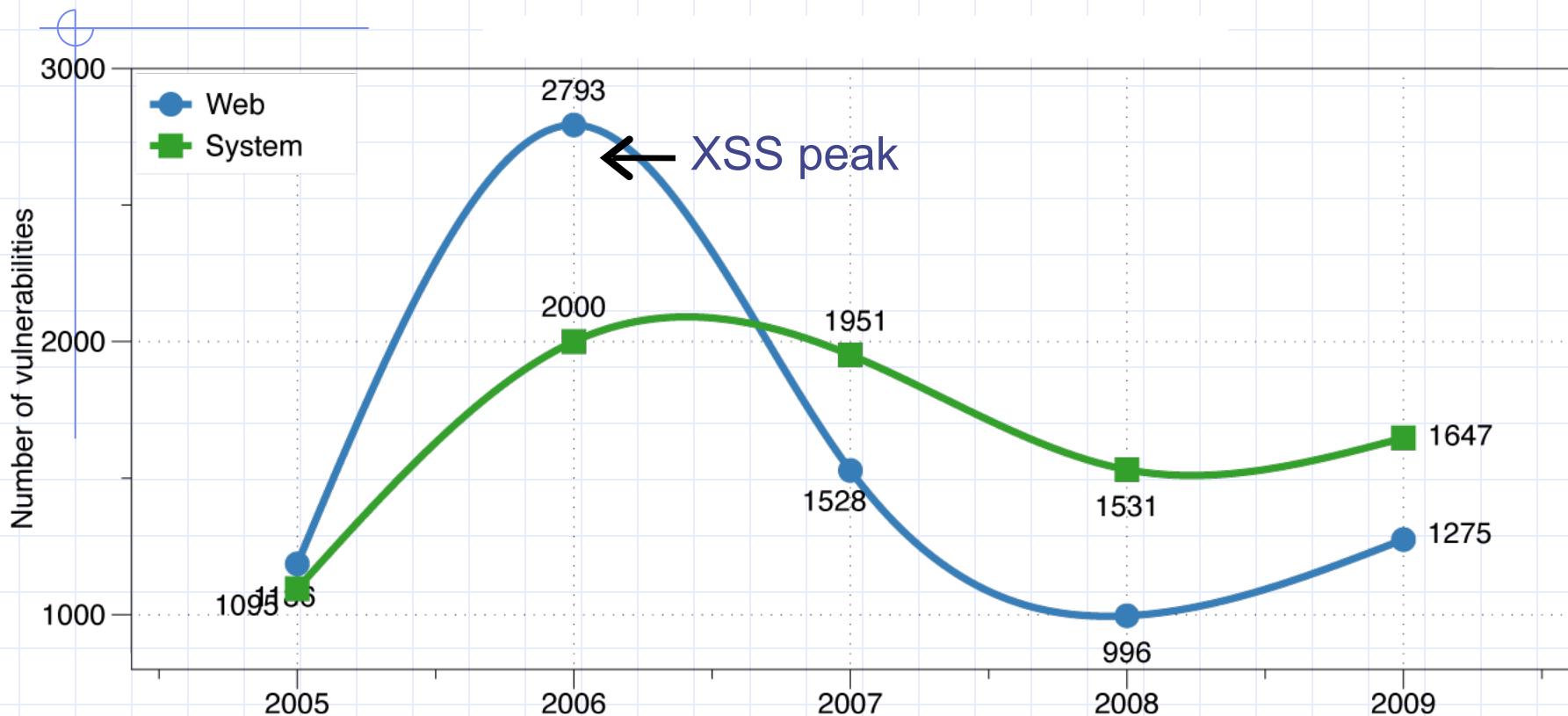
OWASP Top 10 – 2013 (Previous)	OWASP Top 10 – 2017 (New)
A1 – Injection	A1 – Injection
A2 – Broken Authentication and Session Management	A2 – Broken Authentication and Session Management
A3 – Cross-Site Scripting (XSS)	A3 – Cross-Site Scripting (XSS)
A4 – Insecure Direct Object References - Merged with 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)

# Historical Web Vulnerabilities "In the Wild"



Data from aggregator and validator of NVD-reported vulnerabilities

# Historical Web vs System vulnerabilities



◆ Decline in % web vulns since 2009

- 49% in 2010 -> 37% in 2011.
- Big decline in SQL Injection vulnerabilities

# Five lectures on Web security

## ◆ Browser security model

- The browser as an OS and execution platform
- Protocols, isolation, communication, ...

## ◆ Web application security

- Application pitfalls and defenses

## ◆ Session management and user authentication

- How users authenticate to web sites
- Browser-server mechanisms for managing state

## ◆ Content security policies

- Additional mechanisms for sandboxing and security

## ◆ HTTPS: goals and pitfalls (after Crypto lecture)

- Network issues and browser protocol handling

This 2.5-week section could fill an entire course

# Web programming poll

◆ Familiar with basic html?

◆ Developed a web application using:

- Apache?
- Python?
- JavaScript?
- JSON?

PHP?

SQL?

CSS?

Ruby?

◆ Know about:

- postMessage?    NaCl?    Webworkers?    CSP?
- WebView?

Resource: <http://www.w3schools.com/>

# Goals of web security

## ◆ Safely browse the web

- Visit a variety of web sites without incurring harm
  - ◆ Confidentiality: no stolen information
  - ◆ Integrity: Site A cannot compromise session at Site B

## ◆ Support secure web apps

- Apps provided over the web can have same security properties as stand-alone applications

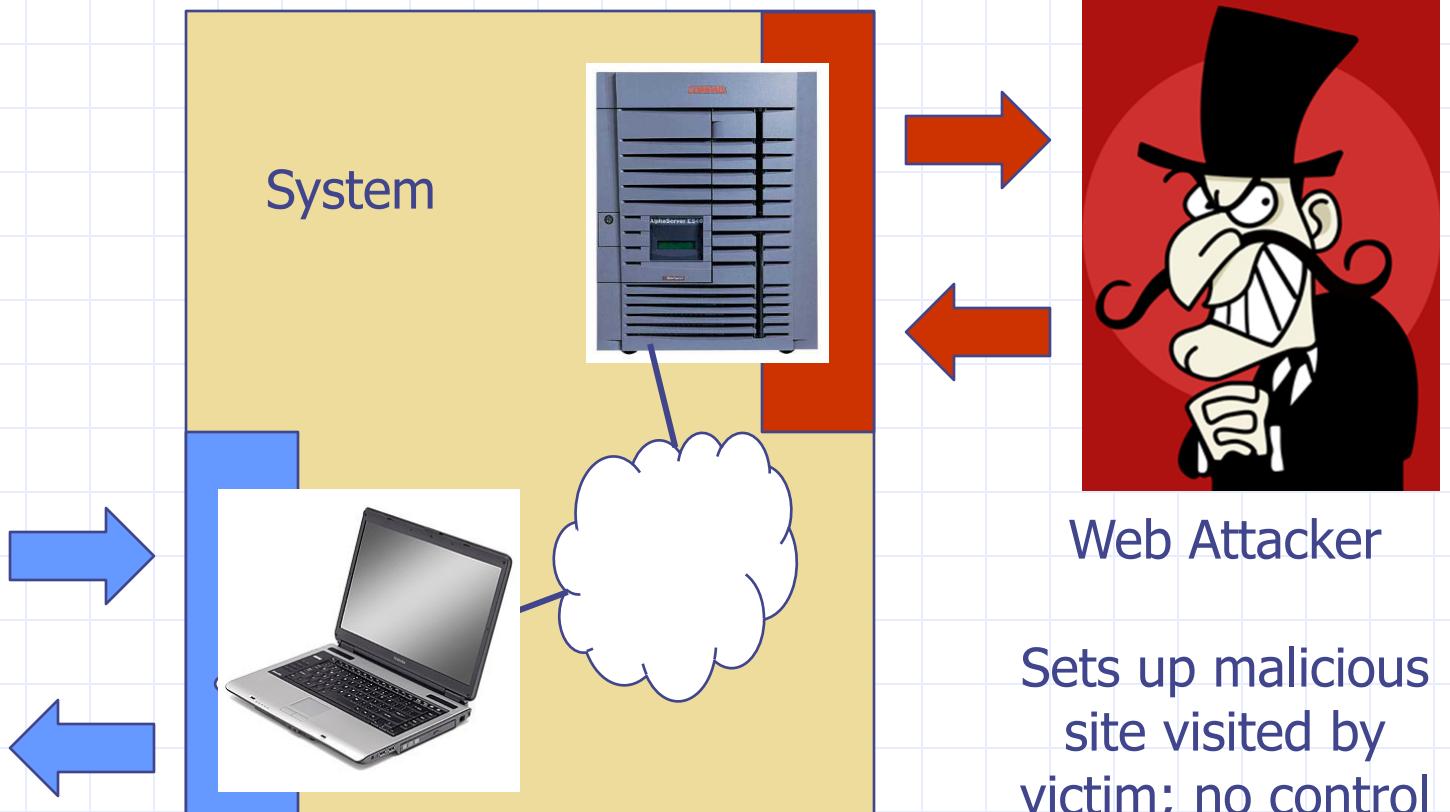
## ◆ Support secure mobile apps

- Web protocols and content standards are used as back end of many mobile apps

# Web security threat model



Alice



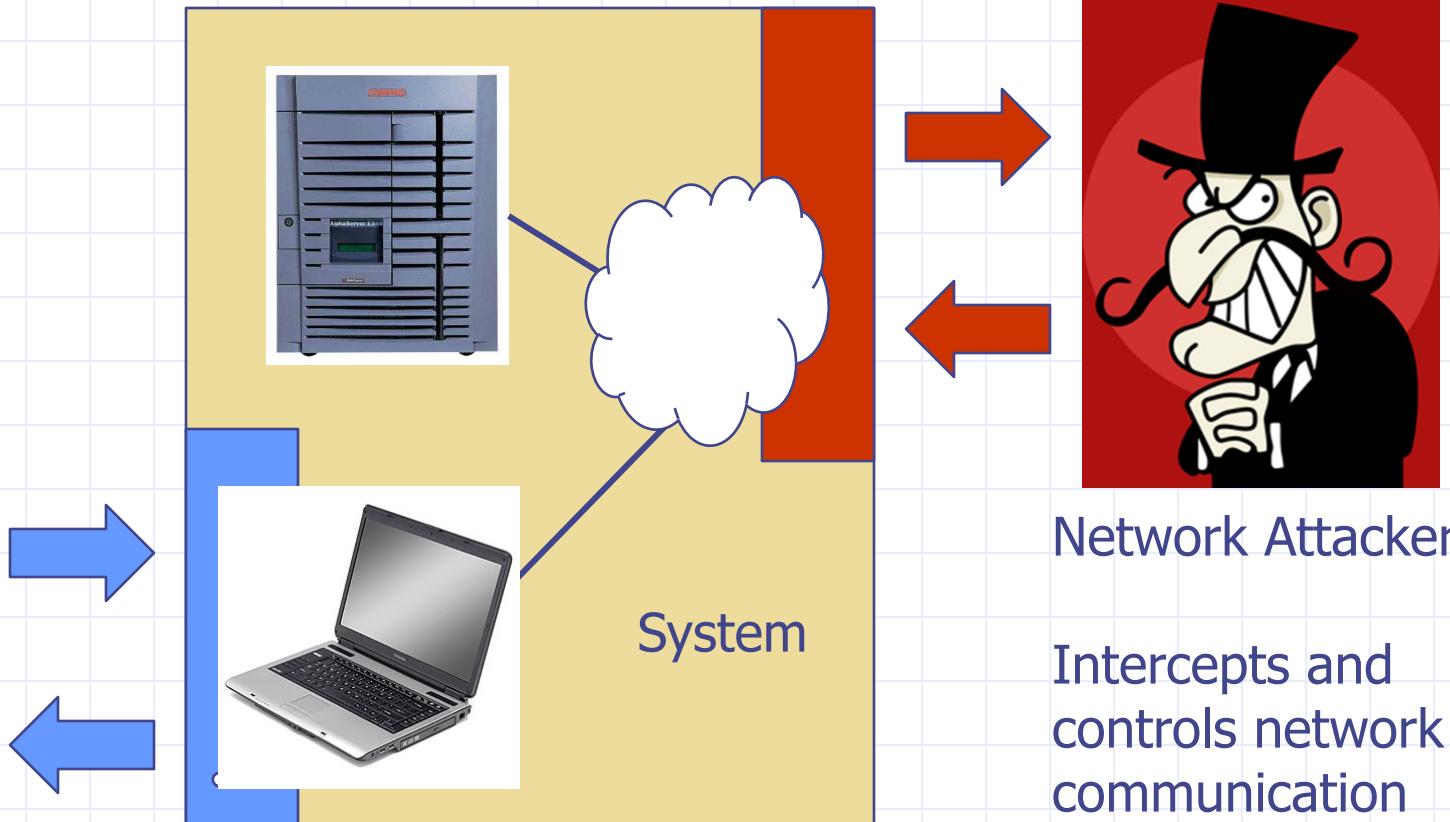
Web Attacker

Sets up malicious site visited by victim; no control of network

# Network security threat model



Alice

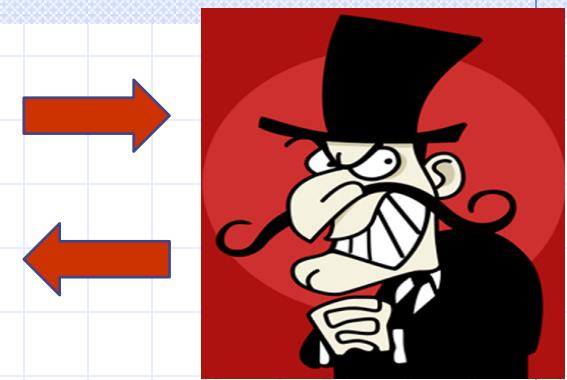
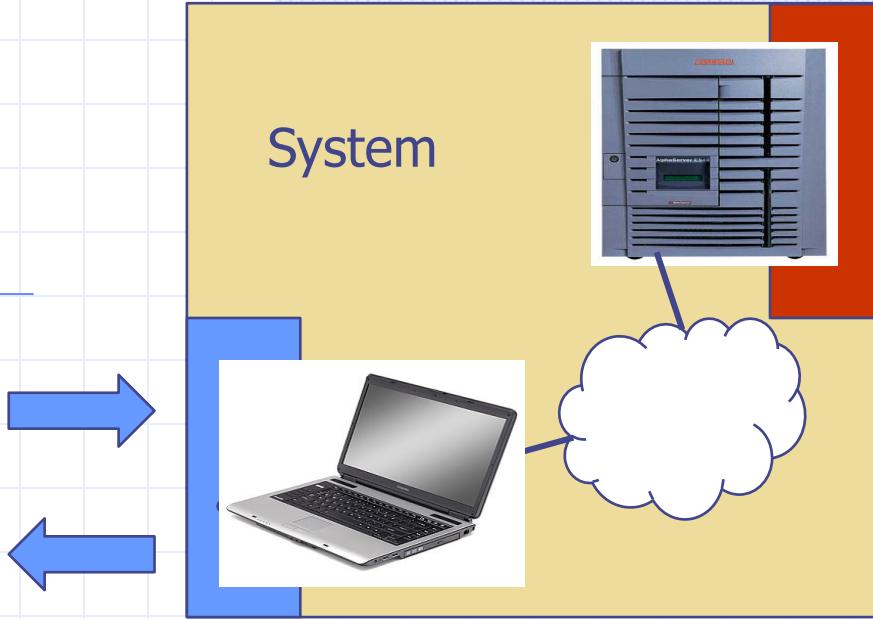


Network Attacker

Intercepts and  
controls network  
communication



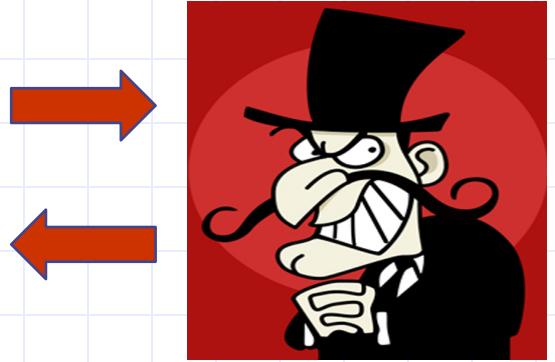
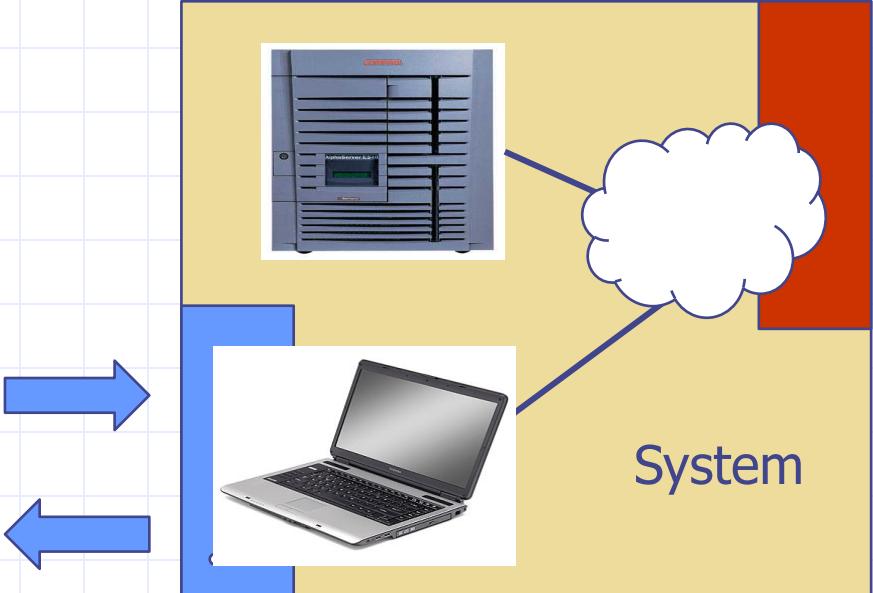
Alice



Web Attacker



Alice



Network Attacker

# Web Threat Models



## Web attacker

- Controls attacker.com
- Can obtain SSL/TLS certificate for attacker.com
- User visits attacker.com
  - ◆ Or: runs attacker's Facebook app, etc.



## Network attacker

- Passive: Wireless eavesdropper
- Active: Evil router, DNS poisoning



## Malware attacker

- Attacker escapes browser isolation mechanisms and run separately under control of OS

# Malware attacker



## Browsers may contain exploitable bugs

- Often enable remote code execution by web sites
- Google study: [the ghost in the browser 2007]
  - ◆ Found Trojans on 300,000 web pages (URLs)
  - ◆ Found adware on 18,000 web pages (URLs)

NOT OUR FOCUS IN THIS PART OF COURSE



## Even if browsers were bug-tree, still lots of vulnerabilities associated with the web

- All vulnerabilities on previous slide: XSS, SQLi, CSRF,

...

# Outline

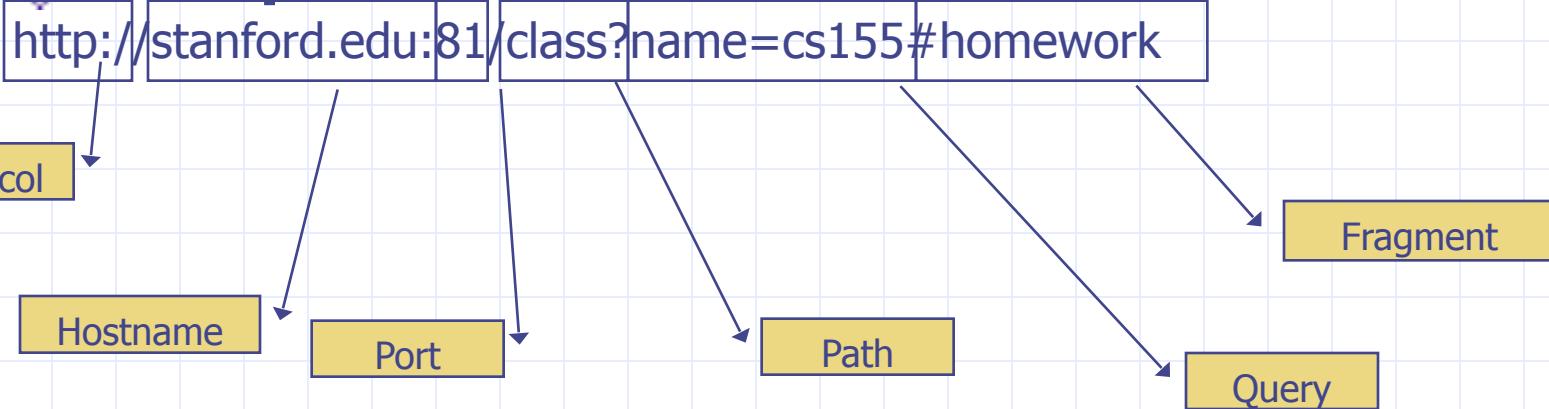
- ◊ Http
- ◊ Rendering content
- ◊ Isolation
- ◊ Communication
- ◊ Navigation
- ◊ Security User Interface
- ◊ Cookies
- ◊ Frames and frame busting

**HTTP**

# URLs

◆ Global identifiers of network-retrievable documents

◆ Example:



◆ Special characters are encoded as hex:

- `%0A` = newline
- `%20` or `+` = space, `%2B` = `+` (special exception)

# HTTP Request

Method      File      HTTP version      Headers

```
GET /index.html HTTP/1.1
Accept: image/gif, image/x-bitmap, image/jpeg, */*
Accept-Language: en
Connection: Keep-Alive
User-Agent: Mozilla/1.22 (compatible; MSIE 2.0; Windows 95)
Host: www.example.com
Referer: http://www.google.com?q=dingbats
```

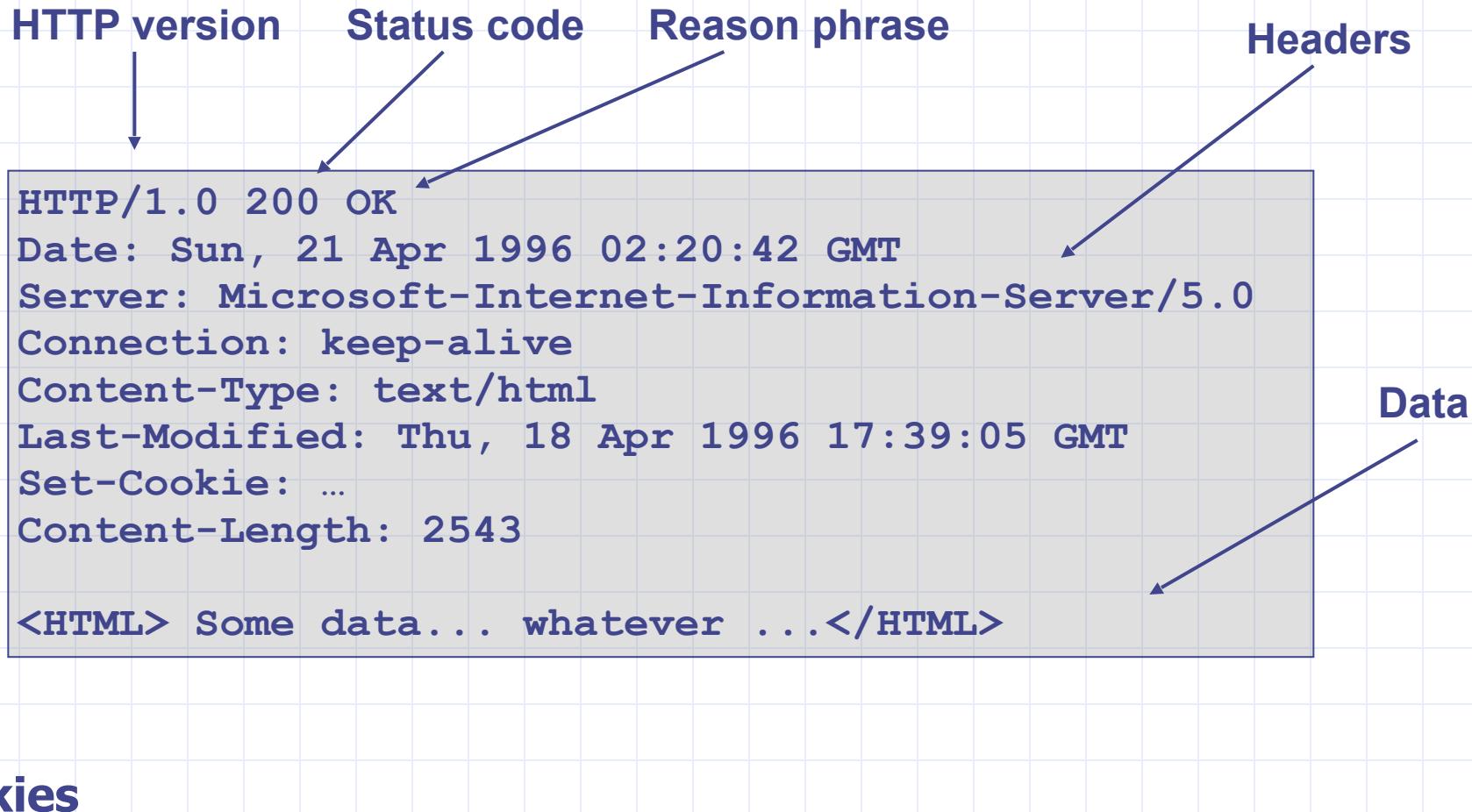
Blank line

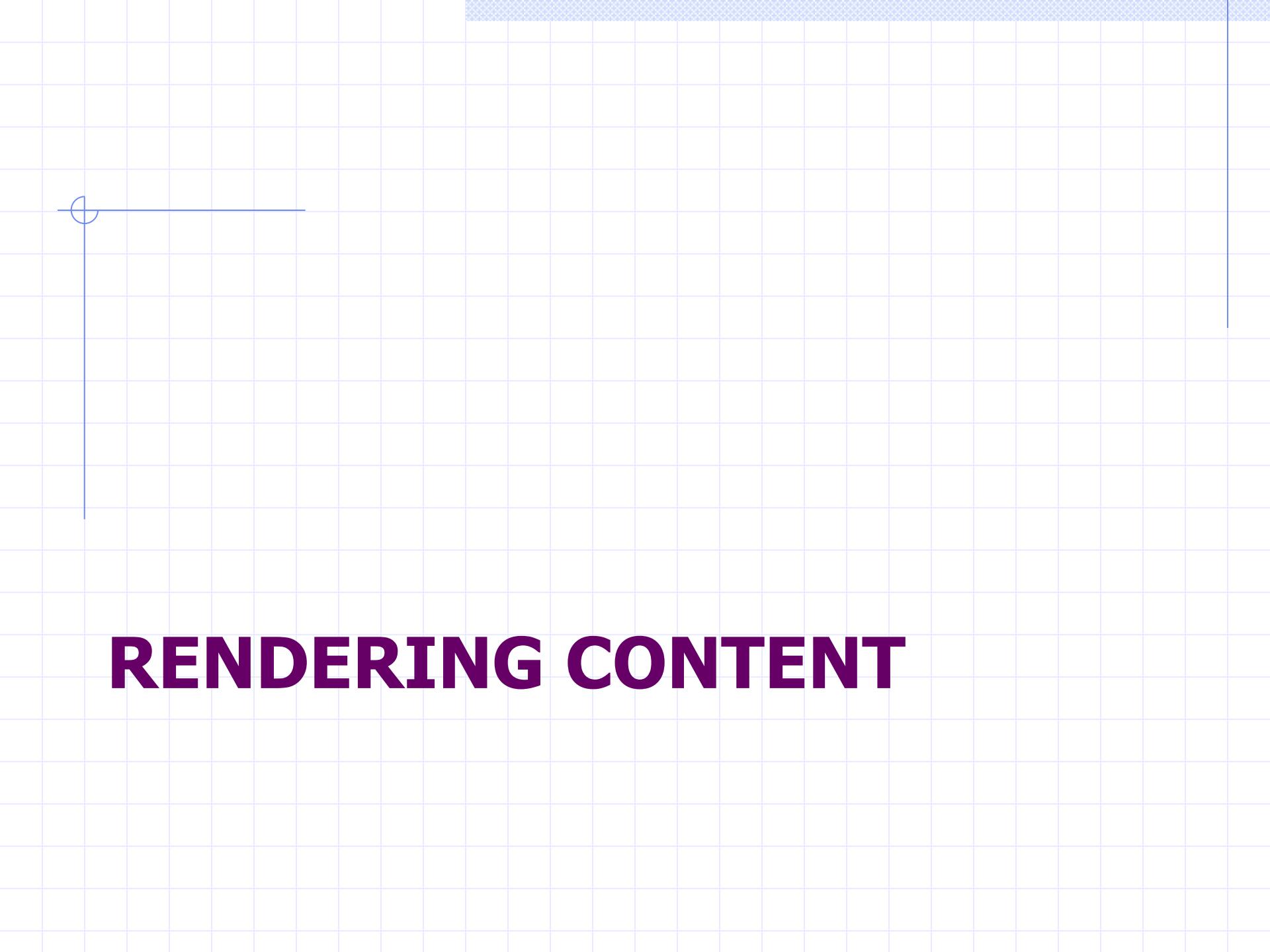
Data – none for GET

GET : no side effect

POST : possible side effect

# HTTP Response





# RENDERING CONTENT

# Rendering and events

## ◆ Basic browser execution model

- Each browser window or frame
  - ◆ Loads content
  - ◆ Renders it
    - Processes HTML and scripts to display page
    - May involve images, subframes, etc.
  - ◆ Responds to events

## ◆ Events can be

- User actions: OnClick, OnMouseover
- Rendering: OnLoad, OnBeforeUnload
- Timing: setTimeout(), clearTimeout()

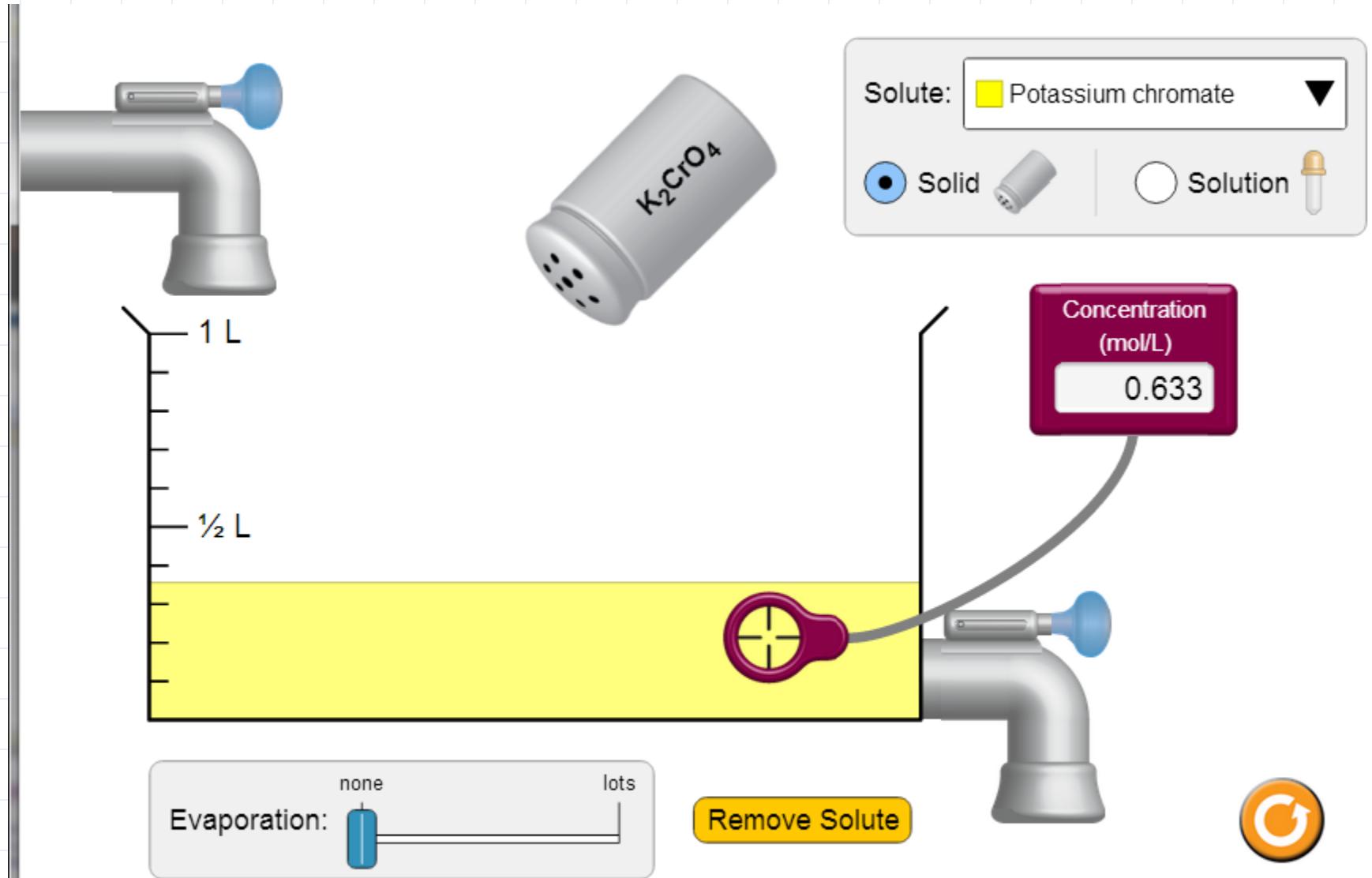
# Example

```
<!DOCTYPE html>
<html>
<body>

<h1>My First Web Page</h1>
<p>My first paragraph.</p>

<button onclick="document.write(5 + 6)">Try it</button>

</body>
</html>
```



# Document Object Model (DOM)

◆ Object-oriented interface used to read and write docs

- web page in HTML is structured data
- DOM provides representation of this data structure

◆ Examples

- **Properties:** document.alinkColor, document.URL, document.forms[ ], document.links[ ], document.anchors[ ]
- **Methods:** document.write(document.referrer)

◆ Includes Browser Object Model (BOM)

- window, document, frames[], history, location, navigator (type and version of browser)

# Changing HTML using Script, DOM

## Some possibilities

- createElement(elementName)
- createTextNode(text)
- appendChild(newChild)
- removeChild(node)

HTML

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

## Example: Add a new list item:

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

# Example

```
<!DOCTYPE html>
<html>
<body>

<h1>My First Web Page</h1>
<p>My First Paragraph</p>

<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML = 5 + 6;
</script>

</body>
</html>
```

# HTML Image Tags

```
<html>  
...  
<p> ... </p>  
...  
  
...  
</html>
```

Displays this nice picture →  
Security issues?



# Image tag security issues

## ◆ Communicate with other sites

- 

## ◆ Hide resulting image

- 

## ◆ Spoof other sites

- Add logos that fool a user

Important Point: A web page can send information to any site

Q: what threat model are we talking about here?

# JavaScript onError

## Basic function

- Triggered when error occurs loading a document or an image

## Example

```

```

- Runs onError handler if image does not exist and cannot load

# JavaScript timing

## Sample code

```
<html><body><img id="test" style="display: none">
<script>
  var test = document.getElementById('test');
  var start = new Date();
  test.onerror = function() {
    var end = new Date();
    alert("Total time: " + (end - start));
  }
  test.src = "http://www.example.com/page.html";
</script>
</body></html>
```

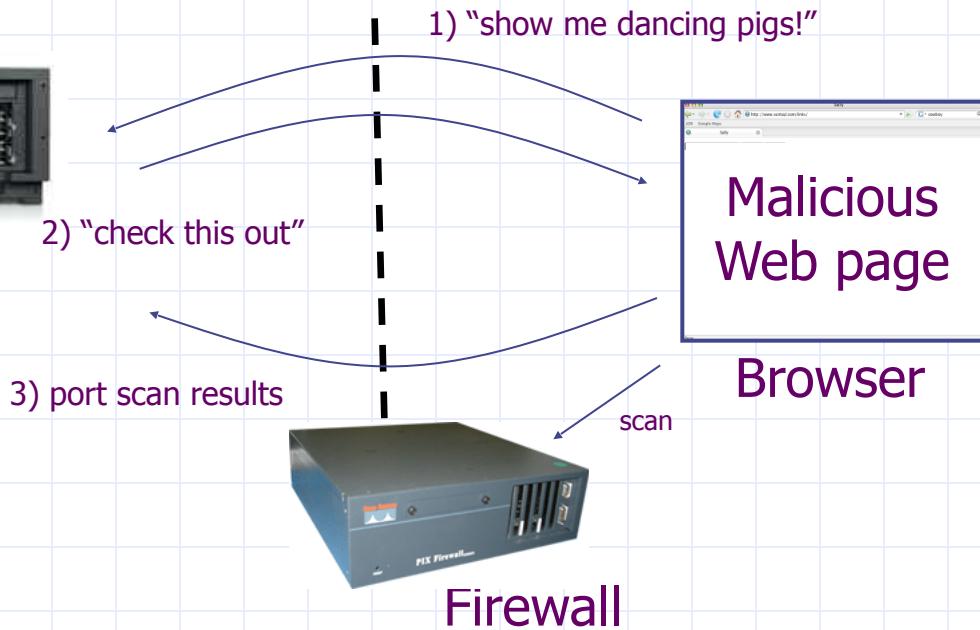
and notifies JavaScript via the onerror handler.

# Port scanning behind firewall

## JavaScript can:

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

Server



# Remote scripting

◆ Goal: communicate between client-side app running in browser and server-side app, without reloading

## ◆ Methods

- Java Applet/ActiveX control/Flash
  - ◆ Can make HTTP requests and interact with client-side JavaScript code, but some aspects may be browser specific
- XML-RPC
  - ◆ open, standards-based technology that requires XML-RPC libraries on server and in your client-side code.
- Simple HTTP via a hidden IFRAME
  - ◆ IFRAME with a script on your web server is by far the easiest of the three remote scripting options

Important Point: A page can maintain bi-directional communication with browser (until user closes/quits)

See: <http://developer.apple.com/internet/webcontent/iframe.html>

# Simple remote scripting example

client.html: "RPC" by passing arguments to server.html in query string

```
<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: another page on same server, could be server.php, etc

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

RPC can be done silently in JavaScript, passing and receiving arguments



**ISOLATION**

# Frame and iFrame

## ◆ Window may contain frames from different sources

- Frame: rigid division as part of frameset
- iFrame: floating inline frame

## ◆ iFrame example

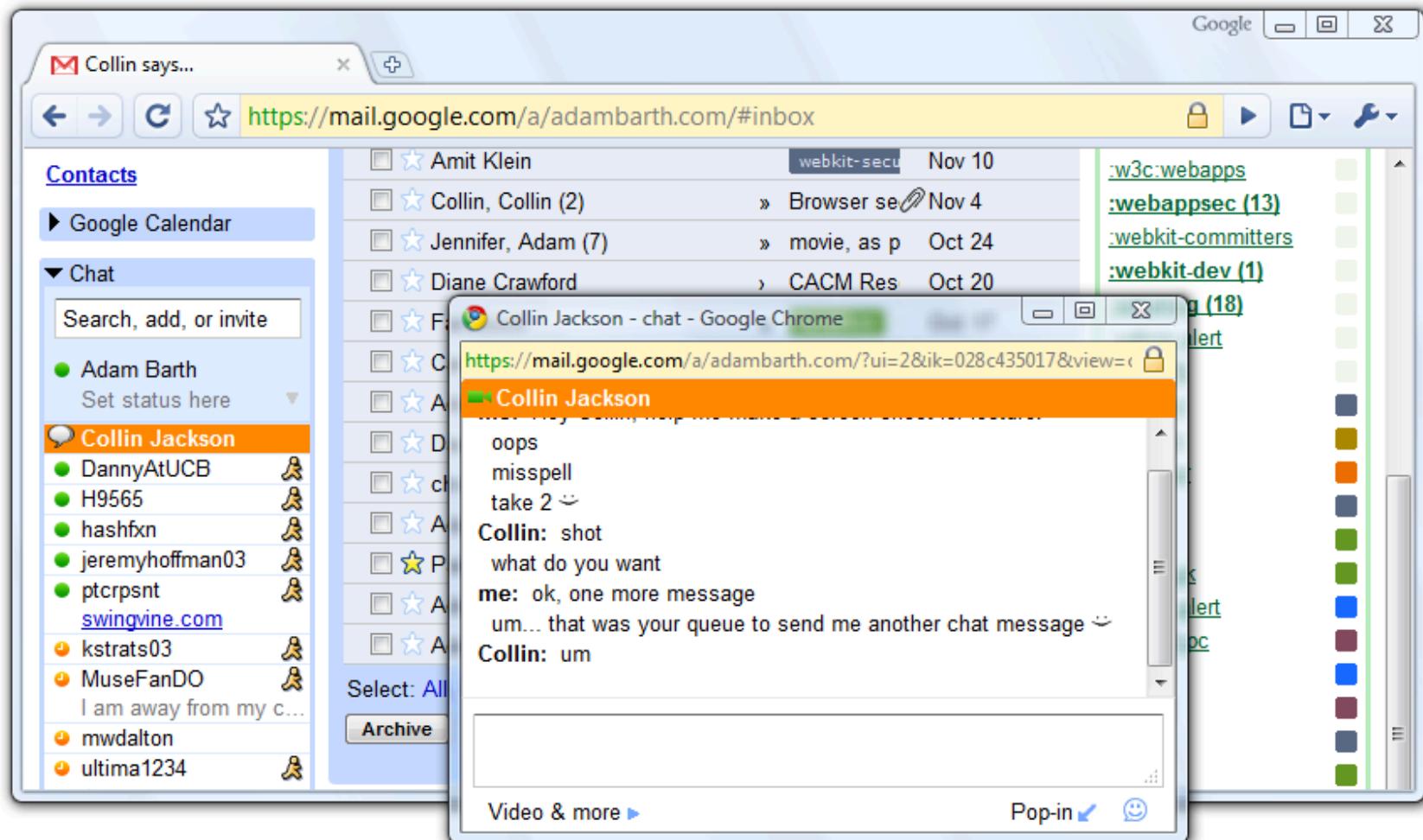
```
<iframe src="hello.html" width=450 height=100>
```

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

# Windows and frames interact



# Analogy

## Operating system

### ◆ Primitives

- System calls
- Processes
- Disk

### ◆ Principals: Users

- Discretionary access control

### ◆ Vulnerabilities

- Buffer overflow
- Root exploit

## Web browser

### ◆ Primitives

- Document object model
- Frames
- Cookies / localStorage

### ◆ Principals: "Origins"

- Mandatory access control

### ◆ Vulnerabilities

- Cross-site scripting
- Cross-site request forgery
- Cache history attacks
- ...

# Policy Goals

◆ Safe to visit an evil web site



◆ Safe to visit two pages at the same time

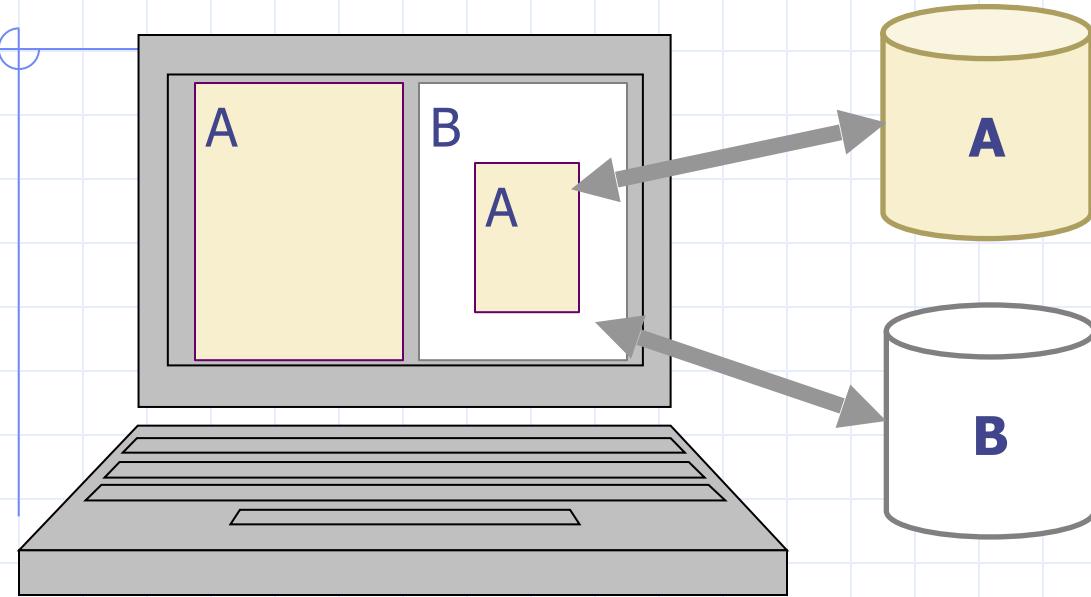
- Address bar distinguishes them



◆ Allow safe delegation



# Browser security mechanism



- ◆ Each frame of a page has an origin
  - Origin = protocol://host:port
- ◆ Frame can access its own origin
  - Network access, Read/write DOM, Storage (cookies)
- ◆ Frame cannot access data associated with a different origin

# Components of browser security policy

## ◆ Frame-Frame relationships

- $\text{canScript}(A, B)$ 
  - ◆ Can Frame A execute a script that manipulates arbitrary/nontrivial DOM elements of Frame B?
- $\text{canNavigate}(A, B)$ 
  - ◆ Can Frame A change the origin of content for Frame B?

## ◆ Frame-principal relationships

- $\text{readCookie}(A, S), \text{writeCookie}(A, S)$ 
  - ◆ Can Frame A read/write cookies from site S?

See <https://code.google.com/p/browsersec/wiki/Part1>  
<https://code.google.com/p/browsersec/wiki/Part2>

# Library import excluded from SOP

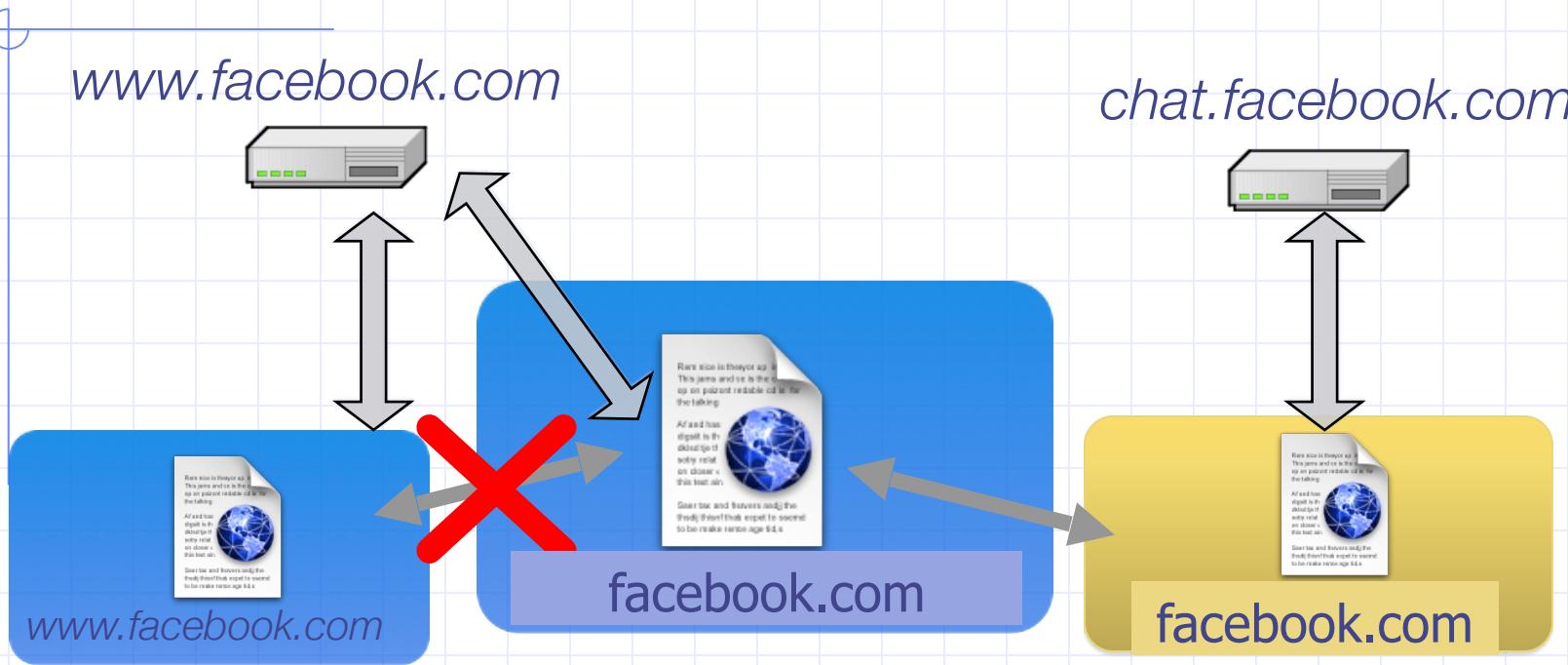
```
<script src="https://seal.verisign.com/getseal?  
host_name=a.com"></script>
```



- Script has access to entire document source server.
- Can script other pages in this origin, load more scripts
- Other forms of importing



# Domain Relaxation

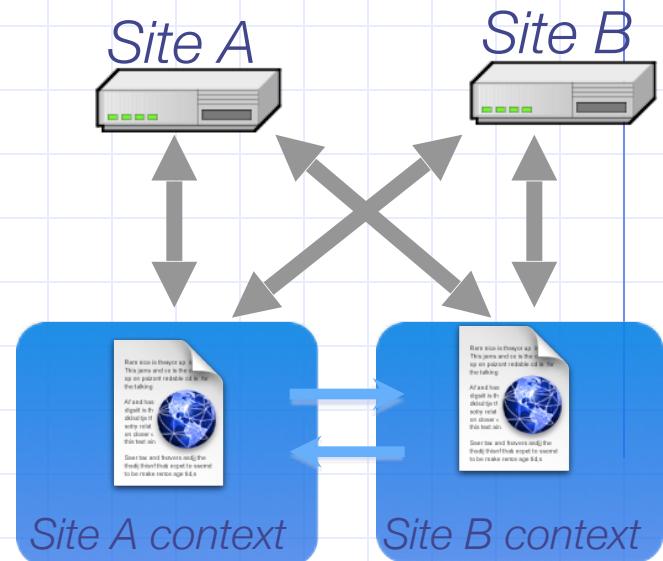


- ❖ Origin: scheme, host, (port), hasSetDomain
- ❖ Try `document.domain = document.domain`

# Additional mechanisms

## Cross-origin network requests

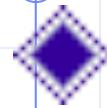
- Access-Control-Allow-Origin: <list of domains>
- Access-Control-Allow-Origin: \*
- Cross-origin client side communication
  - Client-side messaging via navigation (old browsers)
  - postMessage (modern browsers)





# **COMMUNICATION**

# window.postMessage

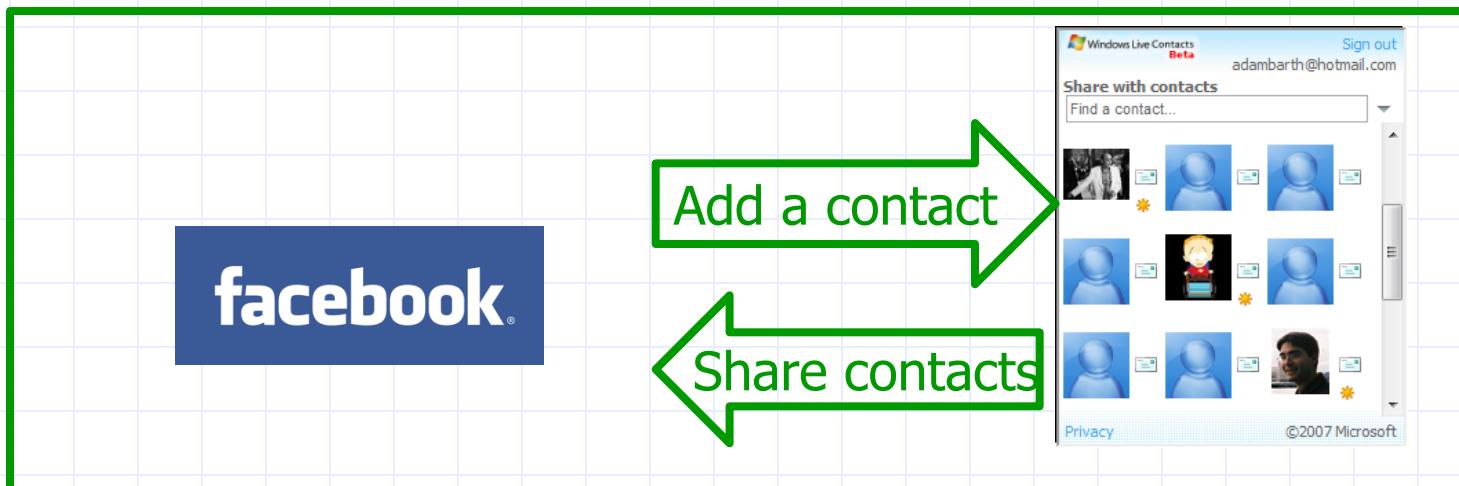


API for inter-frame communication

- Supported in standard browsers



- A network-like channel between frames



# postMessage syntax

```
frames[0].postMessage("Attack at dawn!",  
                      "http://b.com/");
```

```
window.addEventListener("message", function (e) {  
  if (e.origin == "http://a.com") {  
    ... e.data ... }  
}, false);
```



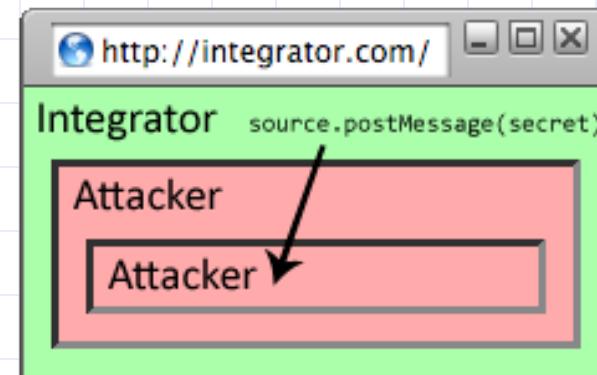
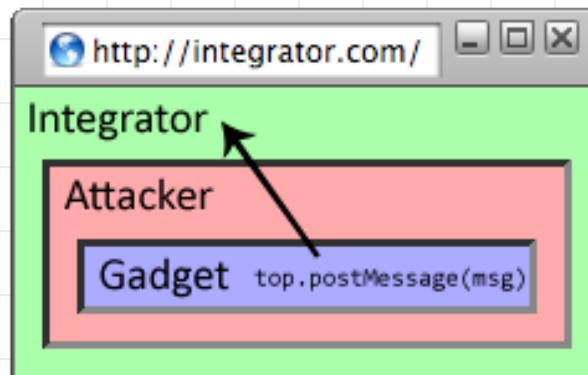
# Why include “targetOrigin”?

## ◆ What goes wrong?

```
frames[0].postMessage("Attack at dawn!");
```

## ◆ Messages sent to frames, not principals

- When would this happen?



# NAVIGATION

# A Guninski Attack

Welcome to AdSense - Windows Internet Explorer

https://www.google.com/adsense/login/en\_US/

Welcome to AdSense

Google

English (US) Help Center

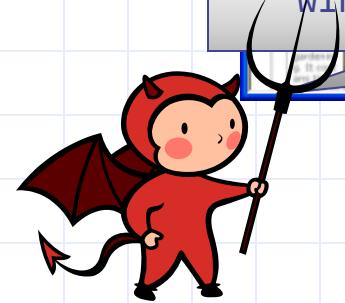
Sign up now! awglogin

Existing AdSense users:

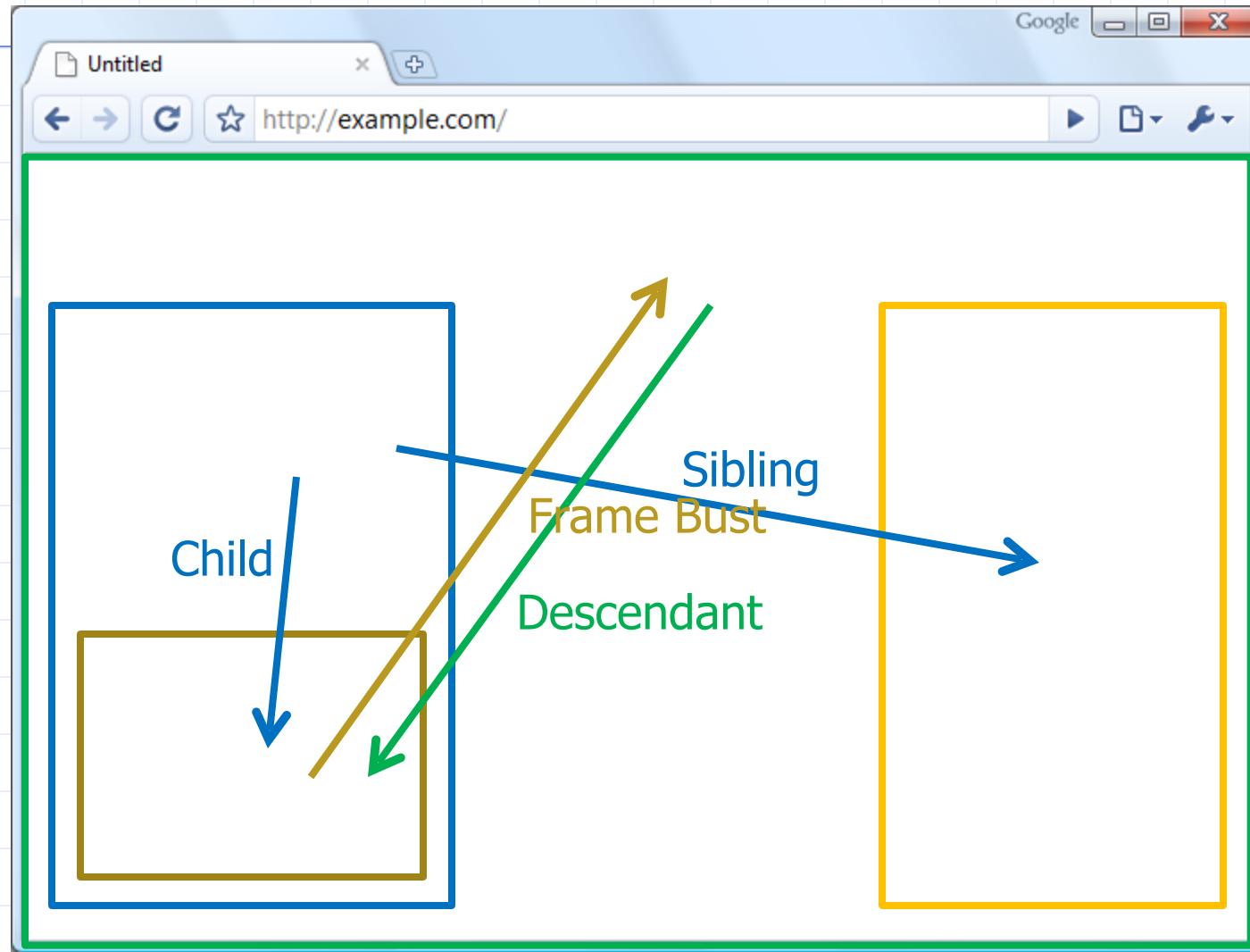
Sign in to Google AdSense with your Google Email

window.open("https://attacker.com/", "awglogin");

I cannot access my account



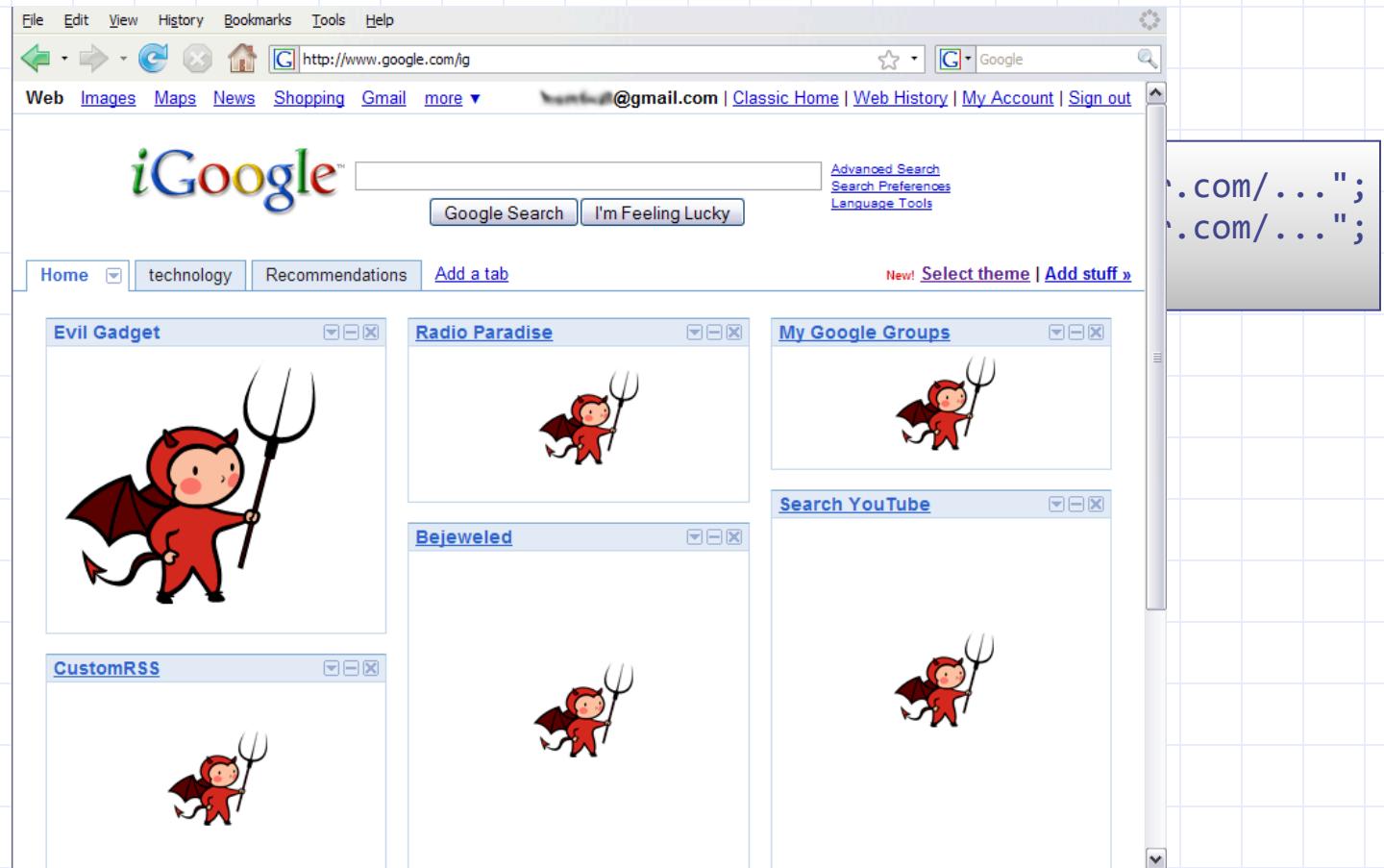
# What should the policy be?



# Legacy Browser Behavior

Browser	Policy
 IE 6 (default)	Permissive
 IE 6 (option)	Child
 IE7 (no Flash)	Descendant
 IE7 (with Flash)	Permissive
 Firefox 2	Window
 Safari 3	Permissive
 Opera 9	Window
 HTML 5	Child
	

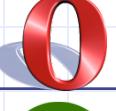
# Window Policy Anomaly



# Legacy Browser Behavior

Browser	Policy
 IE 6 (default)	Permissive
 IE 6 (option)	Child
 IE7 (no Flash)	Descendant
 IE7 (with Flash)	Permissive
 Firefox 2	Window
 Safari 3	Permissive
 Opera 9	Window
 HTML 5	Child
	

# Adoption of Descendant Policy

Browser	Policy
 IE7 (no Flash)	Descendant
 IE7 (with Flash)	Descendant
 Firefox 3	Descendant
 Safari 3	Descendant
 Opera 9	(many policies)
 HTML 5	Descendant



When is it safe to type my password?

# **SECURITY USER INTERFACE**

# Safe to type your password?

Bank of the West | - Mozilla Firefox

File Edit View History Bookmarks Tools Help

Bank of the West (US) https://www.bankofthewest.com/JOW/home

BANK OF THE WEST

PERSON SMALL BUSINESS COMMERCIAL

Products & Services

- Checking
- Savings & CDs
- Credit Cards
- Loans
- Wealth Management & Trust
- Insurance

Achieve Your Goals

- Buy a home
- Buy a new car
- Save for college
- Maximize home equity
- Consolidate debt
- Try our financial calculators

Bank Online

- Apply for an account online
- Learn about online banking
- Enroll in eTimeBanker

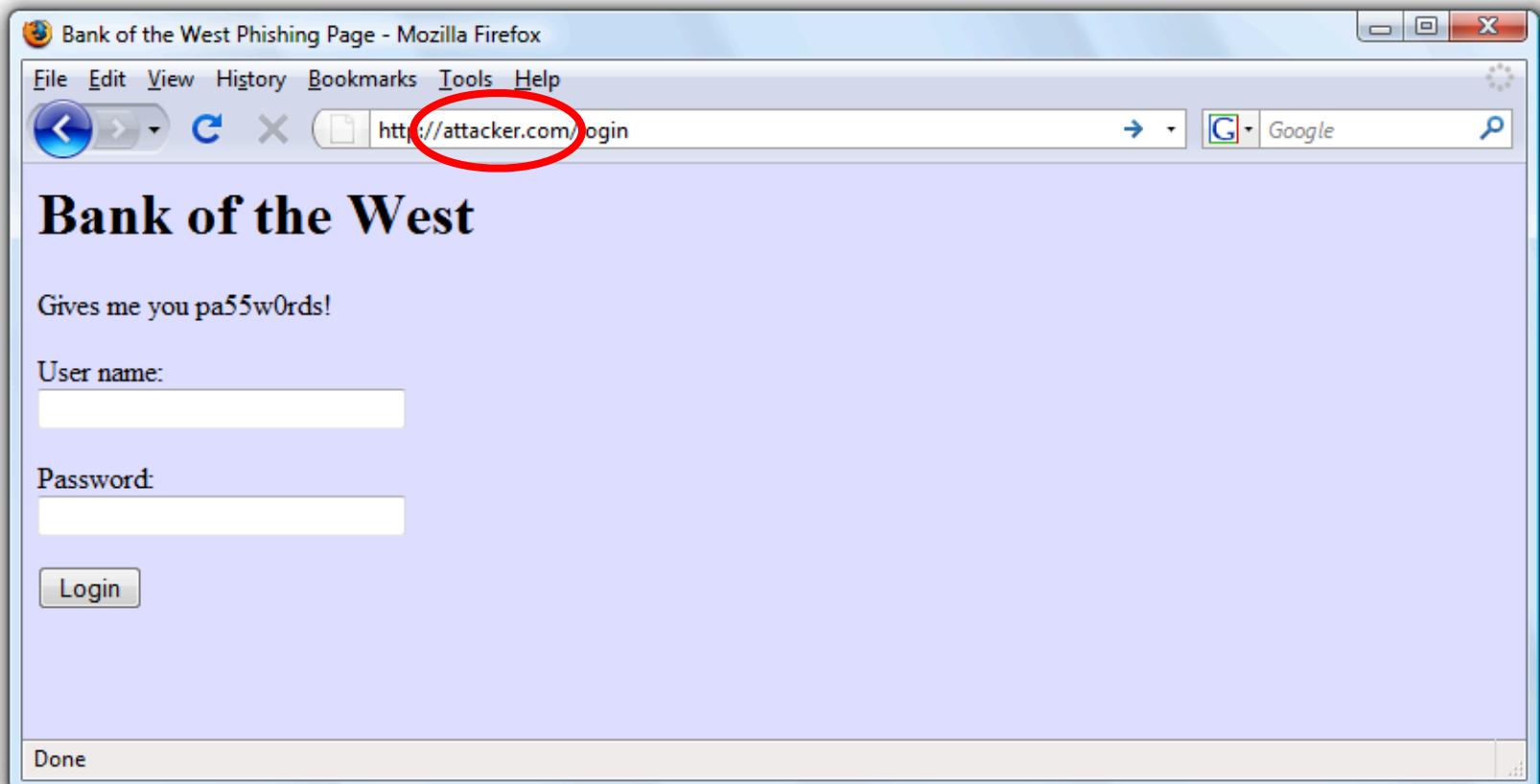
eTimeBanker

Where do I enter my password?  
Alternate Login

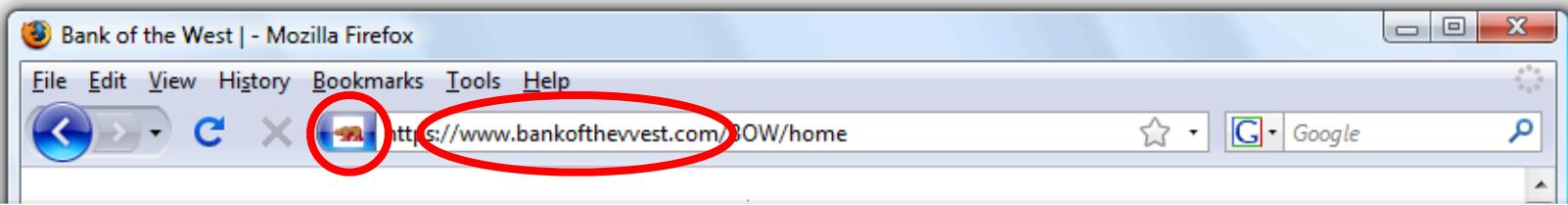
Done

www.bankofthewest.com

# Safe to type your password?



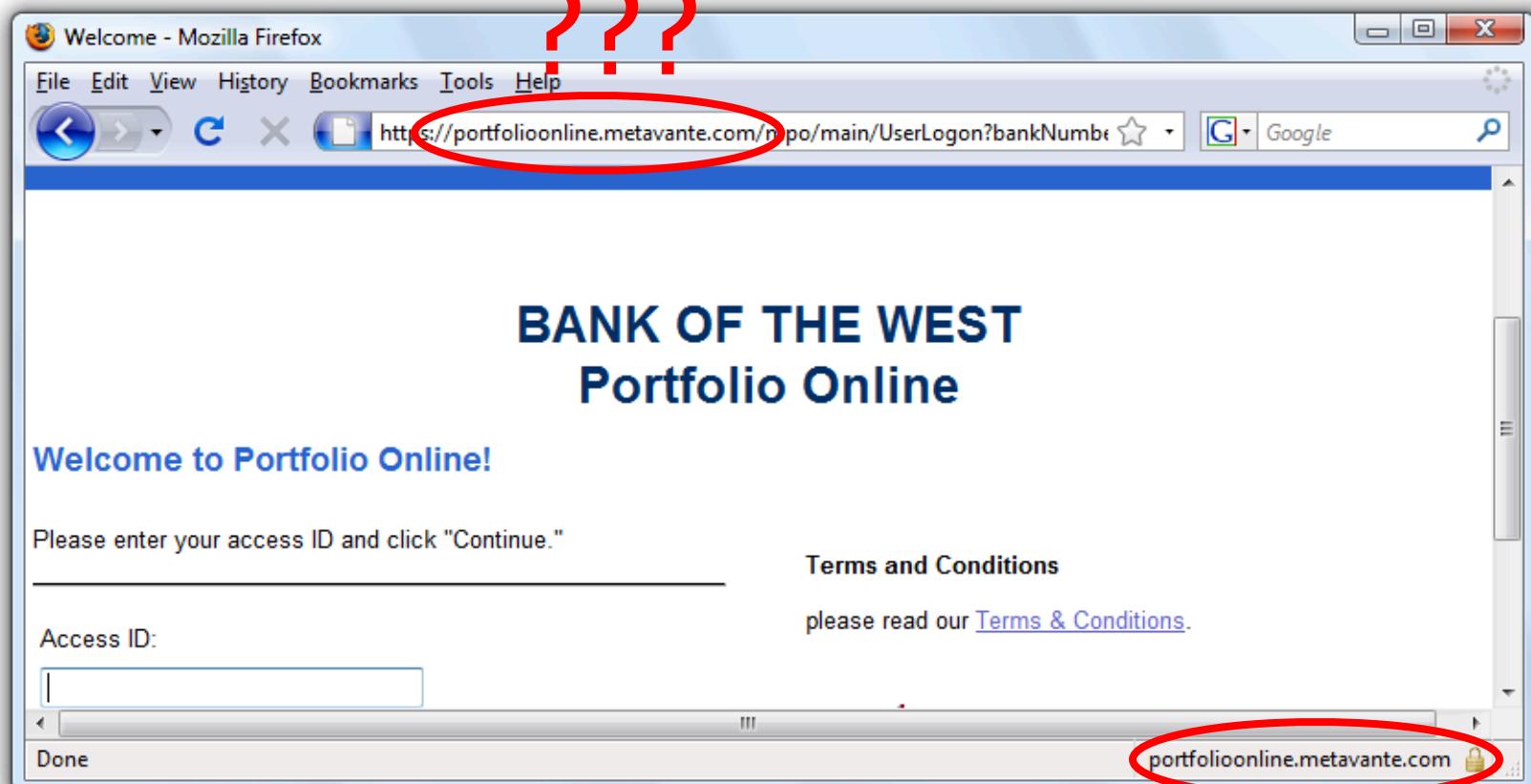
# Safe to type your password?



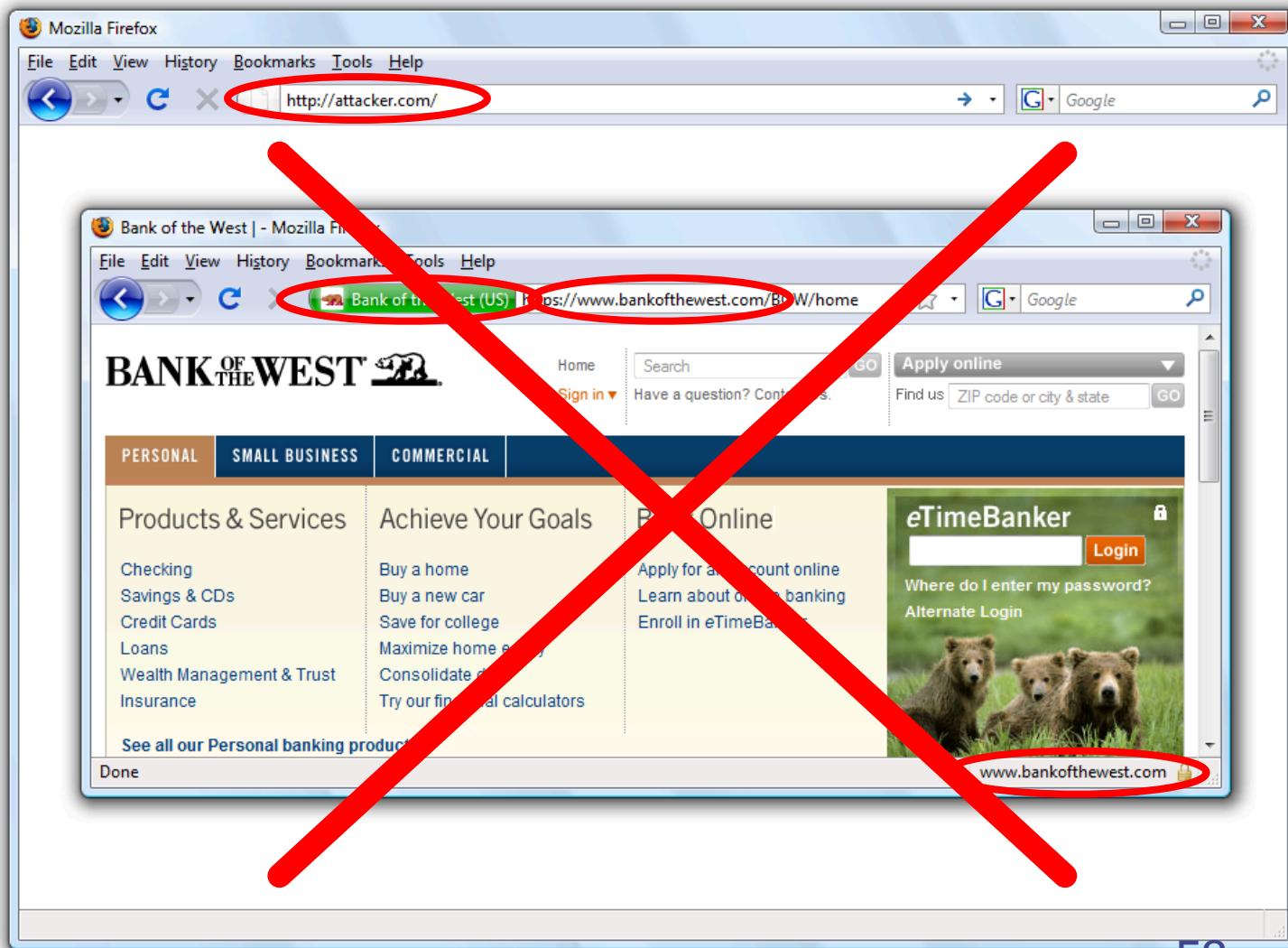
t h e n w e s t c o m

t h e n w e s t c o m

# Safe to type your password?



# Safe to type your password?



# Mixed Content: HTTP and HTTPS

## ◆ Problem

- Page loads over HTTPS, but has HTTP content
- Network attacker can control page

## ◆ IE: displays mixed-content dialog to user

- Flash files over HTTP loaded with no warning (!)
- Note: Flash can script the embedding page

## ◆ Firefox: red slash over lock icon (no dialog)

- Flash files over HTTP do not trigger the slash

## ◆ Safari: does not detect mixed content

Dan will talk about this later...

# Mixed content and network attacks



## Old sites: after login all content over HTTPS

- Developer error: Somewhere on bank site write  

```
<script src=http://www.site.com/script.js> </script>
```
- Active network attacker can now hijack any session



## Better way to include content:

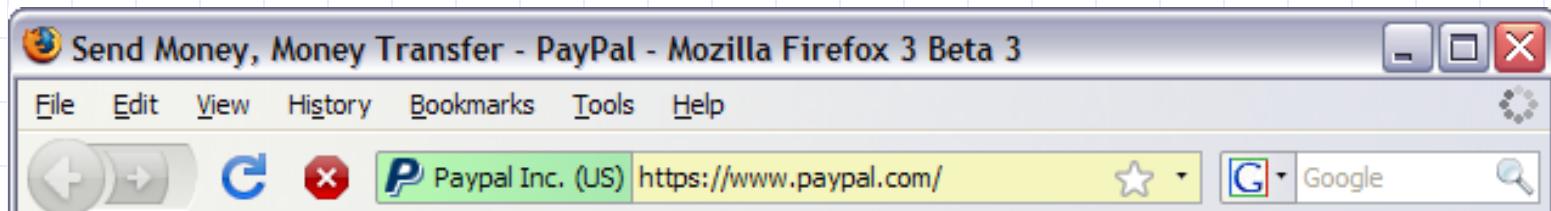
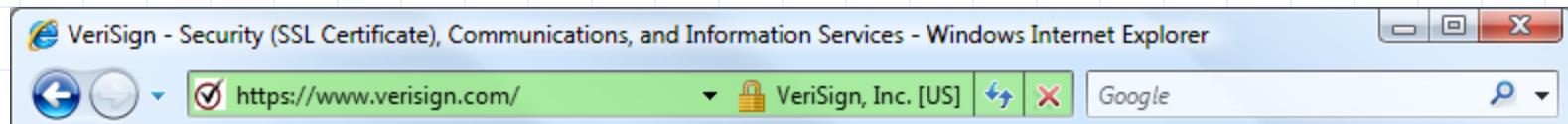
```
<script src=//www.site.com/script.js> </script>
```

- served over the same protocol as embedding page

# Lock Icon 2.0

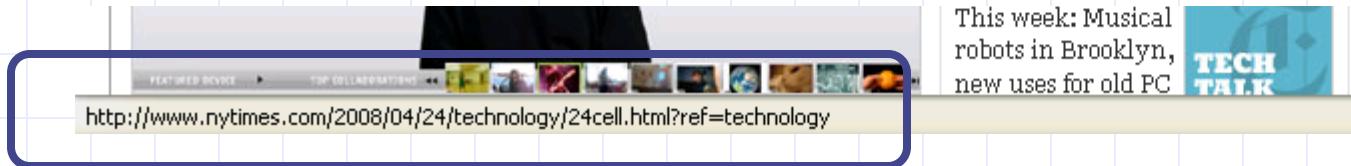


## Extended validation (EV) certs



- Prominent security indicator for EV certificates
- note: EV site loading content from non-EV site does not trigger mixed content warning

# Finally: the status Bar



◆ Trivially spoofable

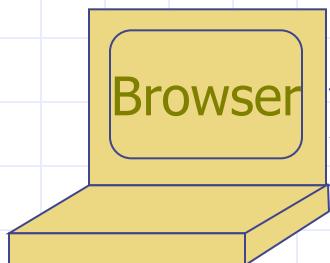
```
<a href="http://www.paypal.com/"  
    onclick="this.href = 'http://www.evil.com/';">  
PayPal</a>
```

# **COOKIES: CLIENT STATE**

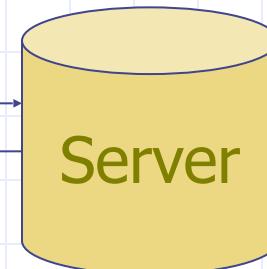
# Cookies



Used to store state on user's machine



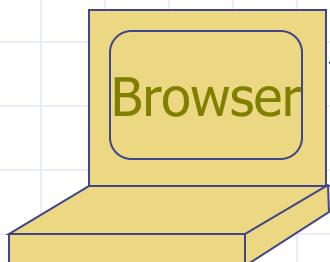
POST ...



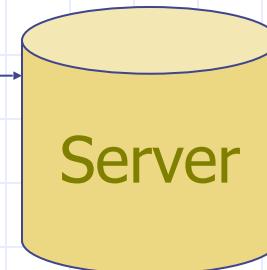
HTTP Header:

Set-cookie: NAME=VALUE ;  
domain = (who can read) ;  
expires = (when expires) ;  
secure = (only over SSL)

If expires=NULL:  
this session only



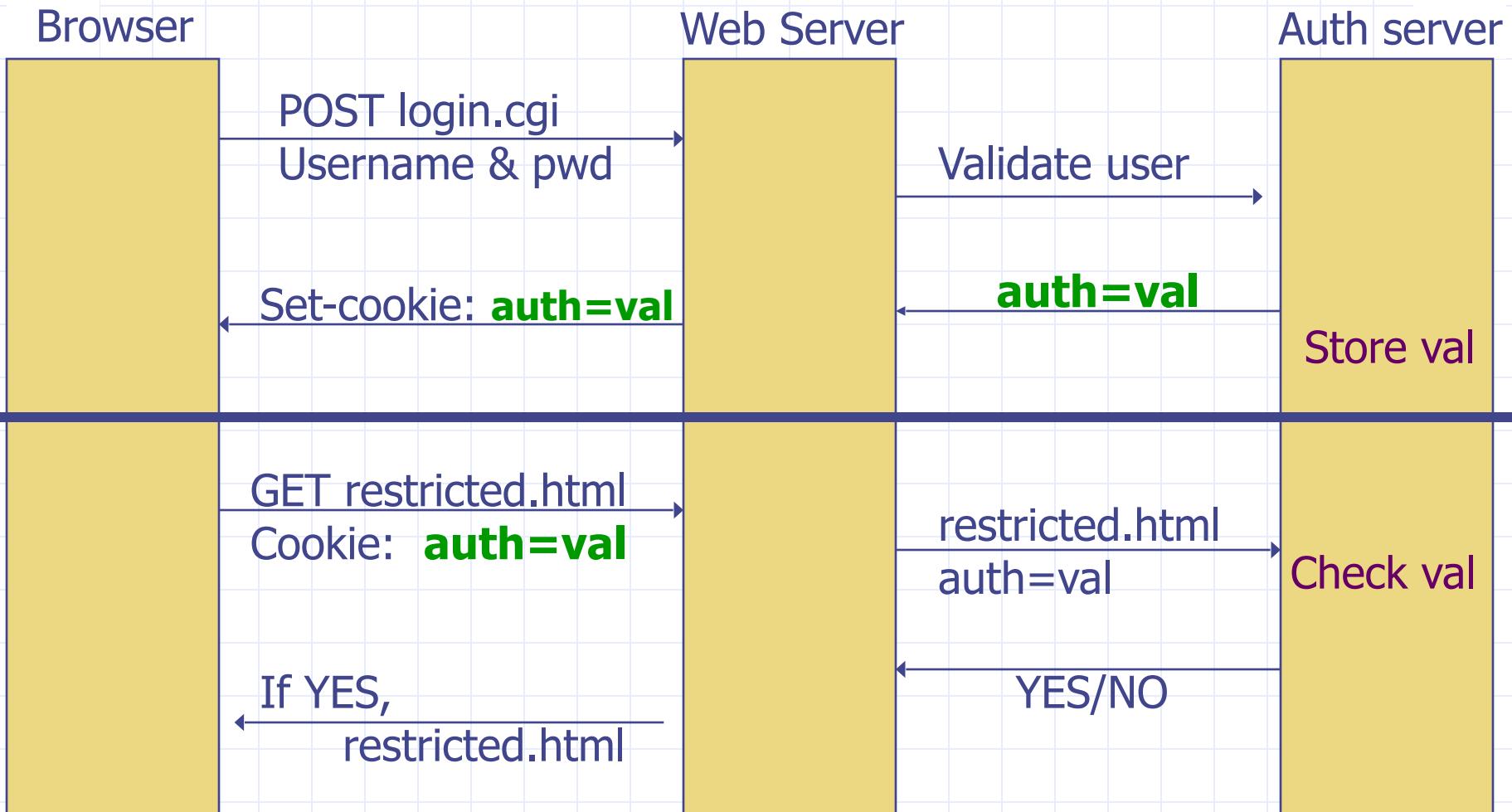
POST ...



Cookie: NAME = VALUE

HTTP is stateless protocol; cookies add state

# Cookie authentication



# Cookie Security Policy

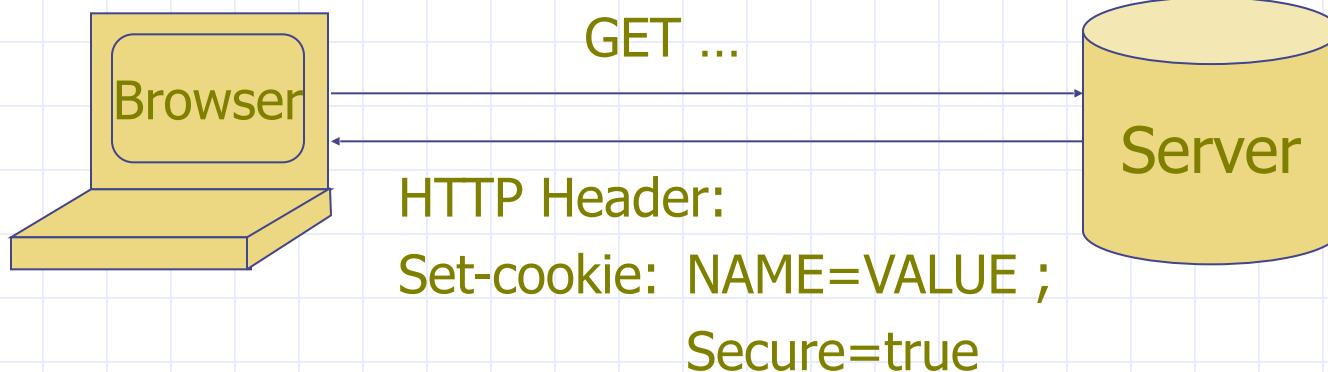
## ◆ Uses:

- User authentication
- Personalization
- User tracking: e.g. Doubleclick (3<sup>rd</sup> party cookies)

## ◆ Origin is the tuple <**domain, path**>

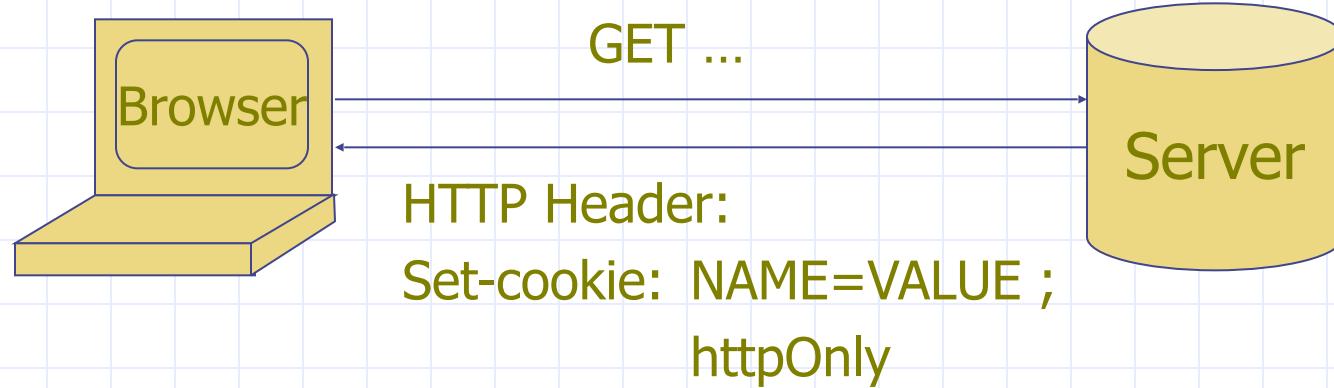
- Can set cookies valid across a domain suffix

# Secure Cookies



- Provides confidentiality against network attacker
  - Browser will only send cookie back over HTTPS
- ... but no integrity
  - Can rewrite secure cookies over HTTP
    - ⇒ network attacker can rewrite secure cookies
    - ⇒ can log user into attacker's account

# httpOnly Cookies



- Cookie sent over HTTP(s), but not accessible to scripts
  - cannot be read via `document.cookie`
  - Helps prevent cookie theft via XSS
- ... but does not stop most other risks of XSS bugs



# **FRAMES AND FRAME BUSTING**

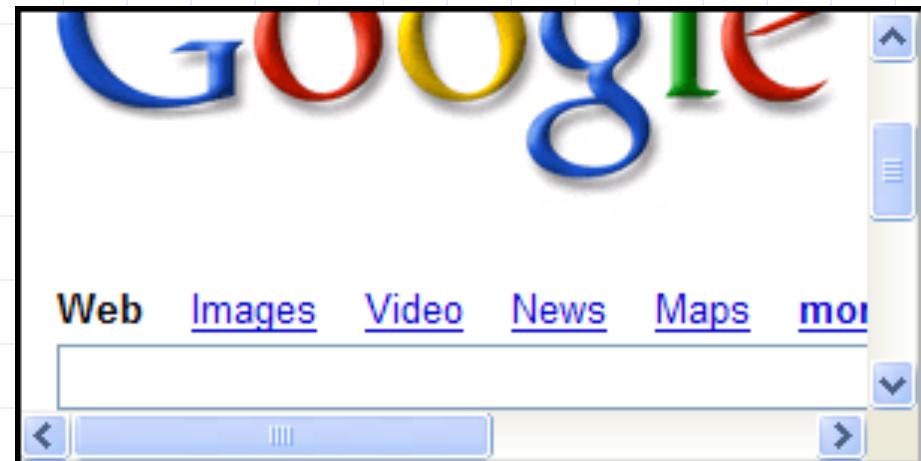
# Frames

- ◆ Embed HTML documents in other documents

```
<iframe name="myframe"  
src="http://www.google.com/">
```

This text is ignored by most browsers.

```
</iframe>
```



# Frame Busting

- ◆ Goal: prevent web page from loading in a frame
  - example: opening login page in a frame will display correct passmark image

- ◆ Frame busting:

```
if (top != self)
    top.location.href = location.href
```



# Better Frame Busting

- ◆ Problem: **Javascript OnUnload event**

```
<body onUnload="javascript: cause_an_abort;">
```

- ◆ Try this instead:

```
if  (top != self)
    top.location.href = location.href
else { ... code of page here ...}
```

# Even better (after ~2010)



## Set X-Frame-Options HTTP response header

- Tell browser not to render a page in a <frame> or <iframe>
- Ensuring that content is not embedded into other sites.
- Use options "DENY", "SAMEORIGIN", or "ALLOW-FROM uri"

Browser	DENY/SAMEORIGIN Support Introduced	ALLOW-FROM Support Introduced
Chrome	<a href="#">4.1.249.1042</a>	<a href="#">Supports CSP frame-ancestors instead</a>
Firefox (Gecko)	<a href="#">3.6.9 (1.9.2.9)</a>	<a href="#">18.0</a>
Internet Explorer	<a href="#">8.0</a>	<a href="#">9.0</a>
Opera	<a href="#">10.50</a>	
Safari	<a href="#">4.0</a>	<a href="#">Won't support - Supports CSP frame-ancestors ins</a>

# Summary

- ◆ Http
- ◆ Rendering content
- ◆ Isolation
- ◆ Communication
- ◆ Navigation
- ◆ Security User Interface
- ◆ Cookies
- ◆ Frames and frame busting