

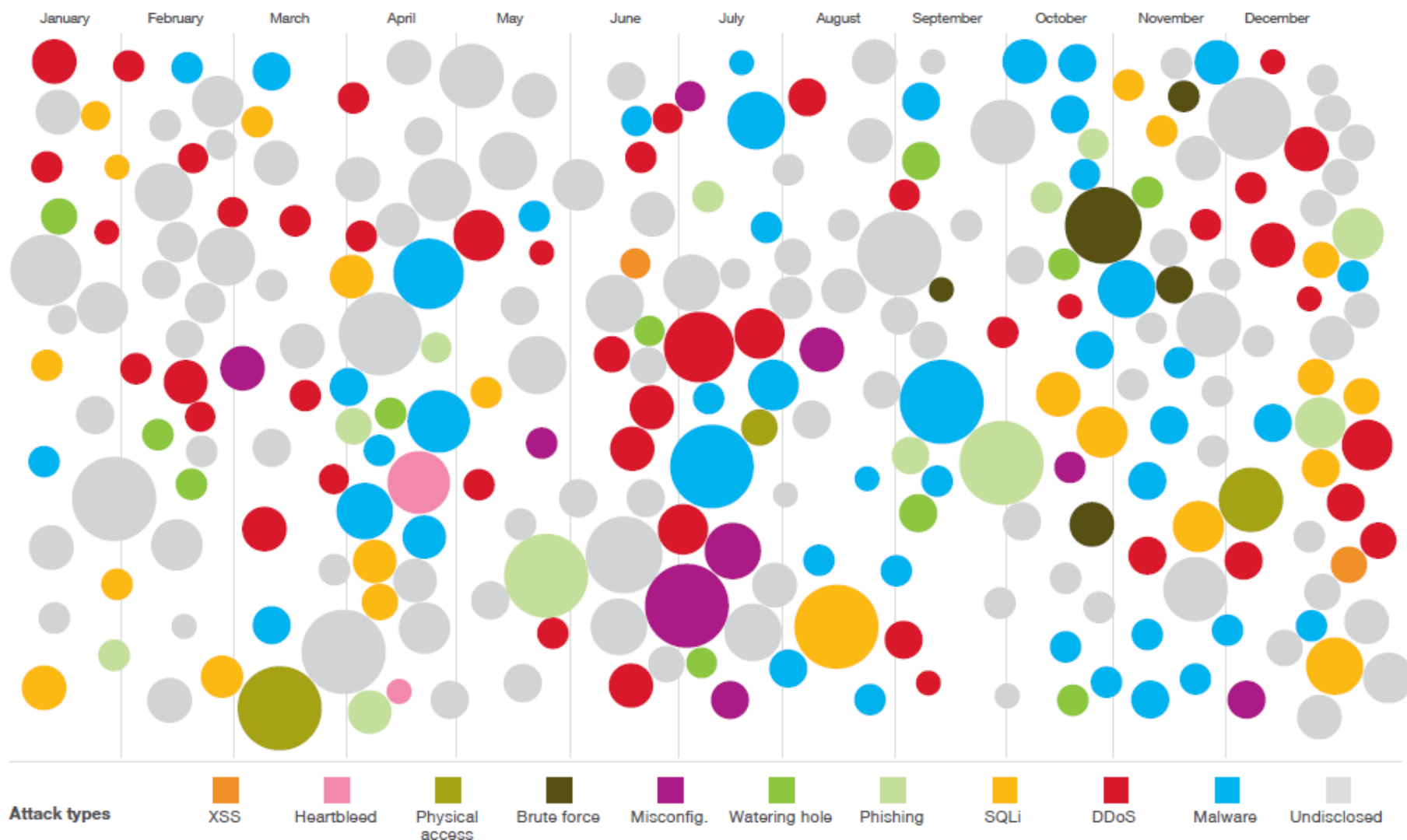
Module 4: Web Client Security

Acknowledgements

- Sandeep Shukla (IIT Kanpur)
- Dan Boneh (Stanford University)
- John C. Mitchell (Stanford University)
- Nicolai Zeldovich (MIT)
- Jungmin Park (Virginia Tech)
- Patrick Schaumont (Virginia Tech)
- C. Edward Chow
- Arun Hodigere
- Web Resources

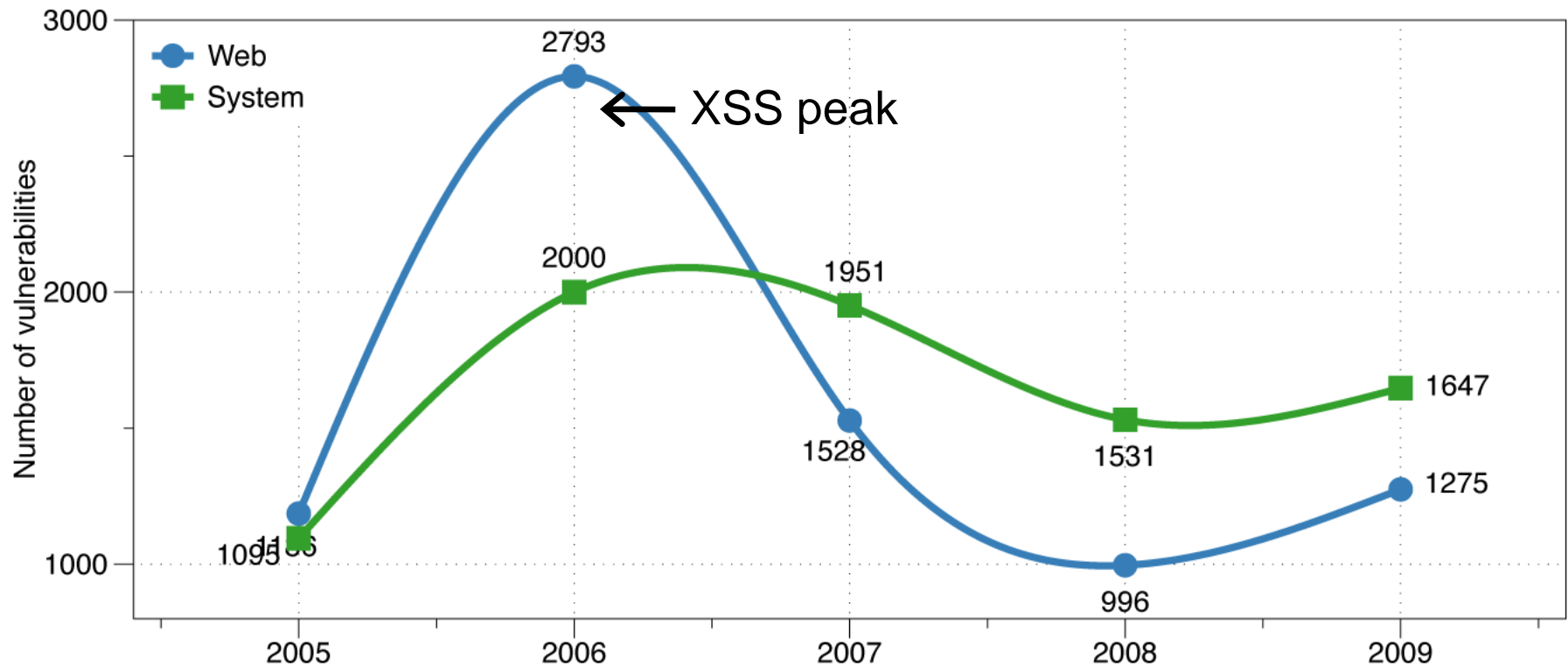
Sampling of 2014 security incidents by attack type, time and impact

conjecture of relative breach impact is based on publicly disclosed information regarding leaked records and financial losses



<https://securityintelligence.com/financial-institutions-top-malware-attacks-target/>

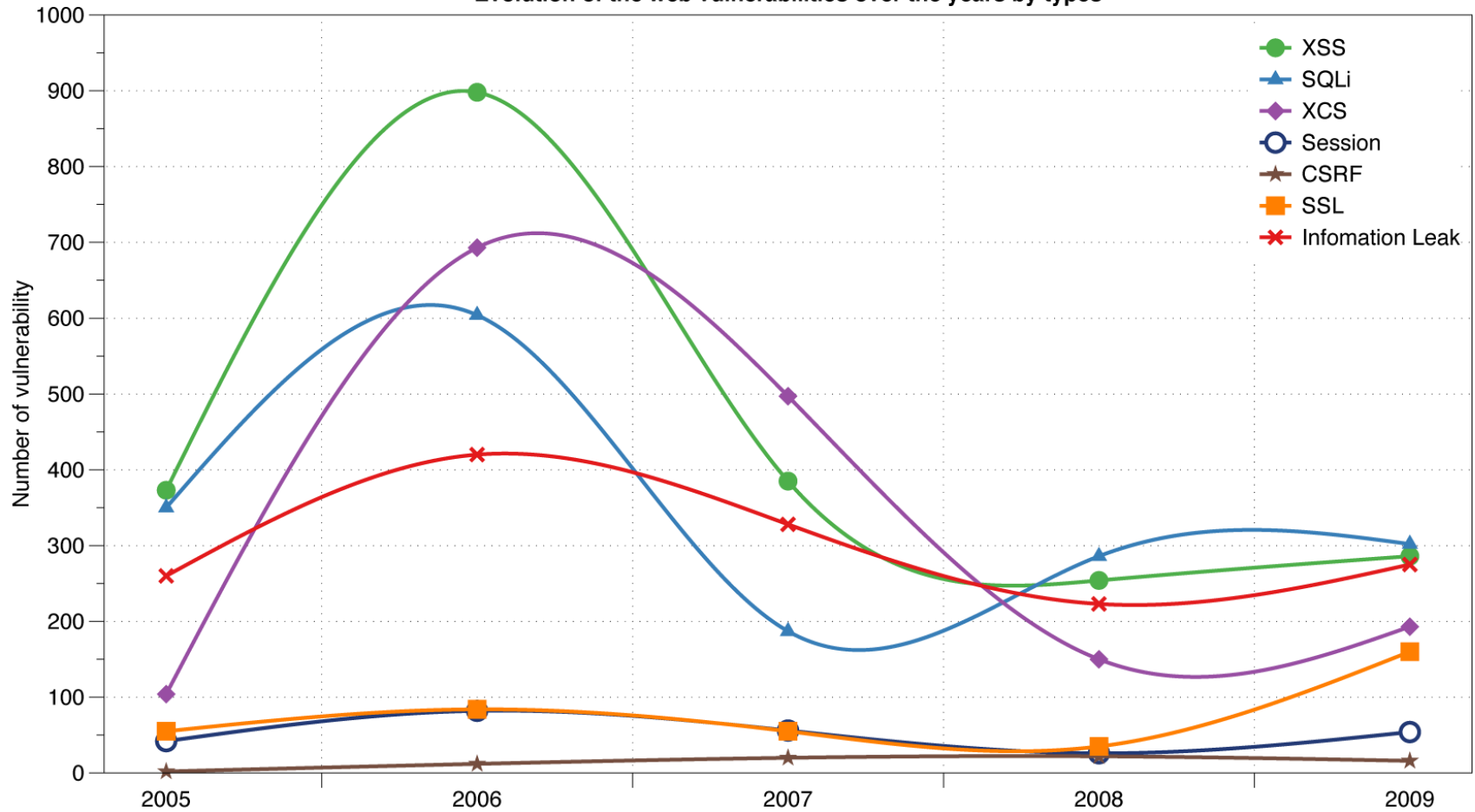
Web vs System vulnerabilities



- Decline in % web vulns since 2009
 - 49% in 2010 -> 37% in 2011.
 - Big decline in SQL Injection vulnerabilities

Reported Web Vulnerabilities "In the Wild"

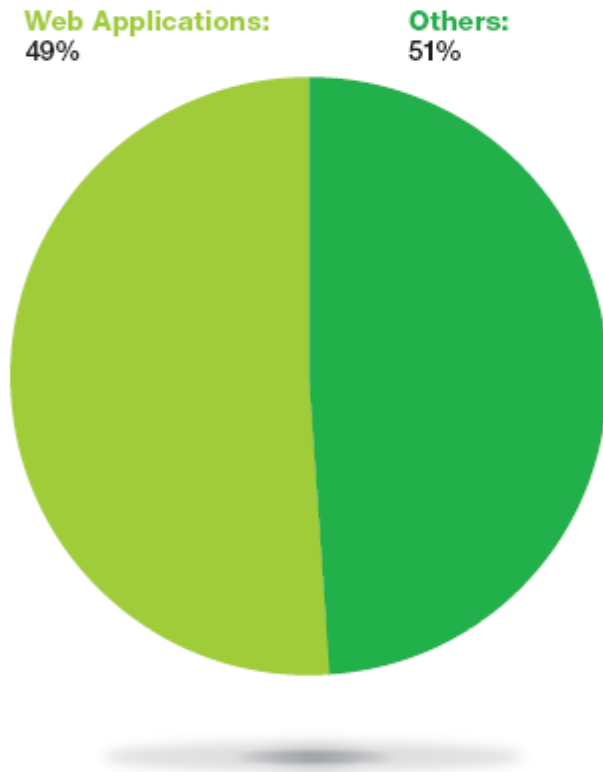
Evolution of the web vulnerabilities over the years by types



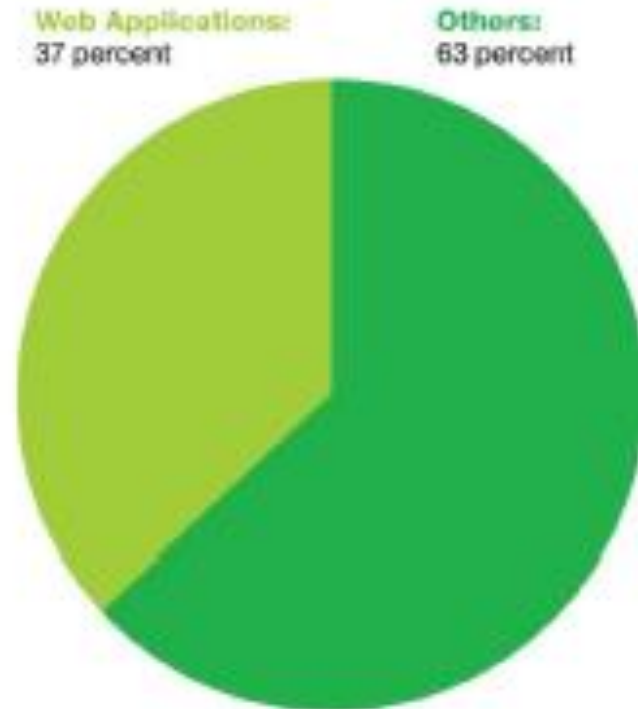
Data from aggregator and validator of NVD-reported vulnerabilities

Web application vulnerabilities

Web Application Vulnerabilities
as a Percentage of All Disclosures in 2010



Web Application Vulnerabilities
as a Percentage of All Disclosures in 2011 H1



Web security

- Browser security model
 - The browser as an OS and execution platform
 - Protocols, isolation, communication, ...
- Web application security
 - Application pitfalls and defenses
- Content security policies
 - Additional mechanisms for sandboxing and security
- Authentication and session management
 - How users authenticate to web sites
 - Browser-server mechanisms for managing state
- HTTPS: goals and pitfalls
 - Network issues and browser protocol handling

Web programming poll

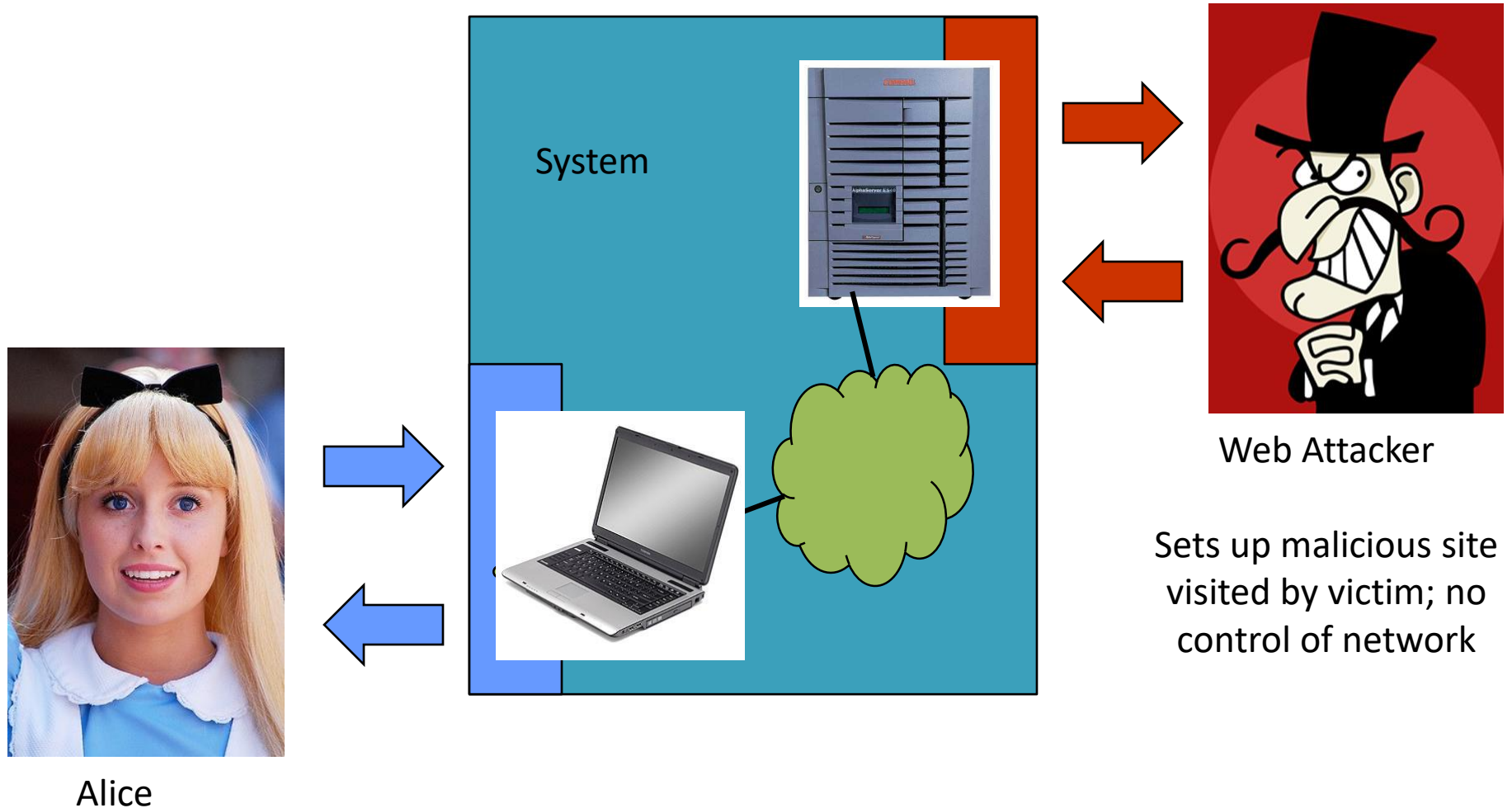
- Familiar with basic html?
- Developed a web application using:
 - Apache? PHP? Ruby?
 - Python? SQL?
 - JavaScript? CSS?
 - JSON?
- Know about:
 - postMessage? NaCL? Webworkers? CSP?
 - WebView?

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

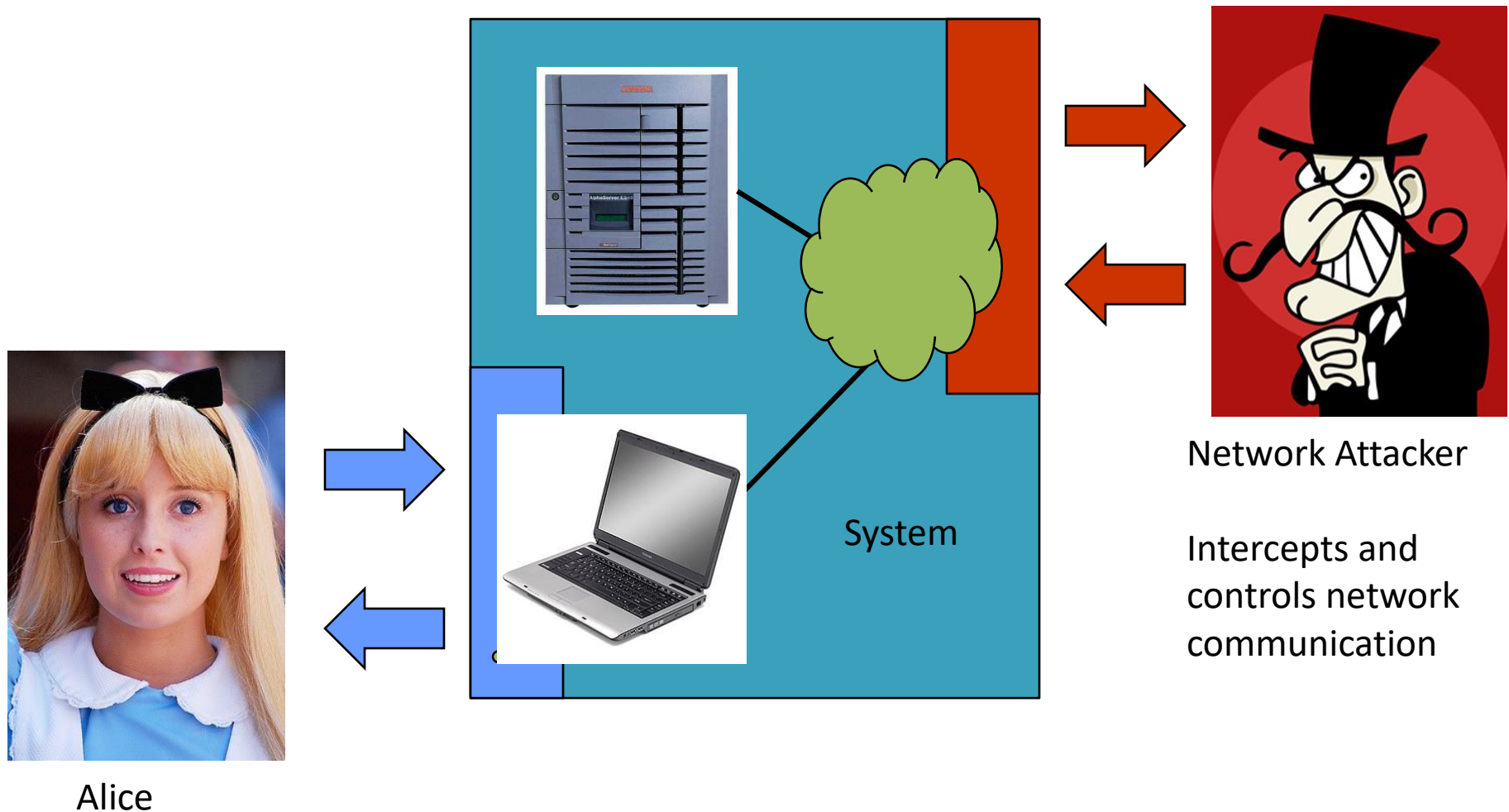
Goals of web security

- Safely browse the web
 - Users should be able to visit a variety of web sites, without incurring harm:
 - No stolen information
 - **Site A** cannot compromise session at **Site B**
- Support secure web applications
 - Applications delivered over the web should be able to achieve the same security properties as stand-alone applications

Web security threat model

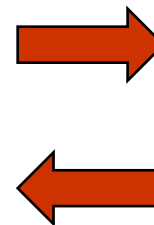
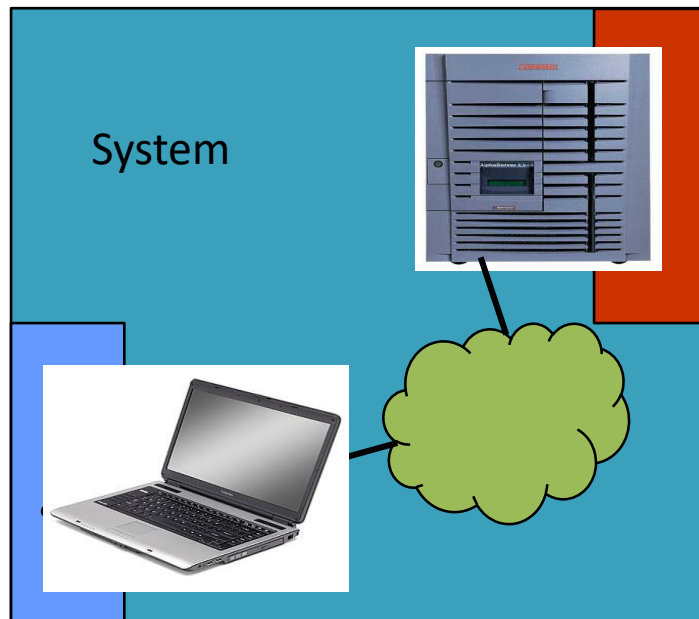
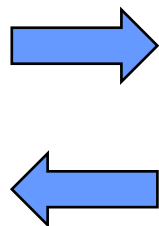


Network security threat model





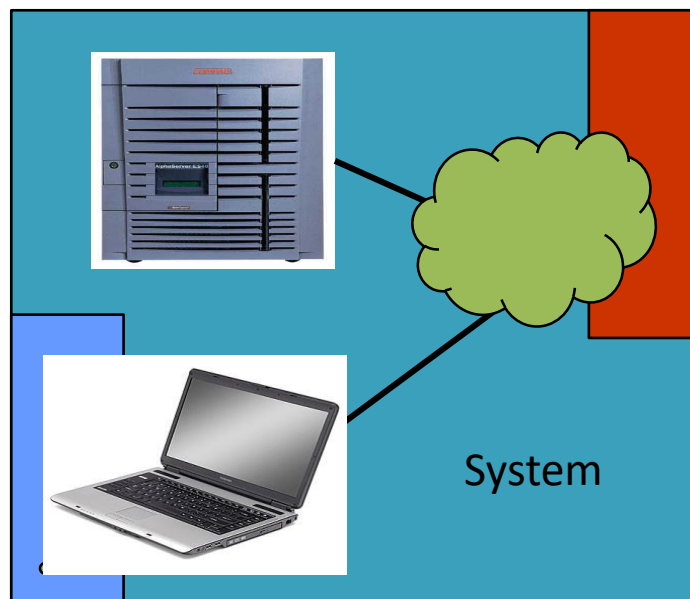
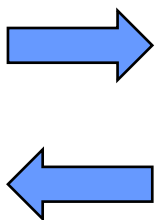
Alice



Web Attacker



Alice



Network Attacker

Web Threat Models

- Web attacker
 - Control 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-free, still lots of vulnerabilities on the web
 - *All* of the vulnerabilities on previous graph: XSS, SQLi, CSRF, ...

Outline

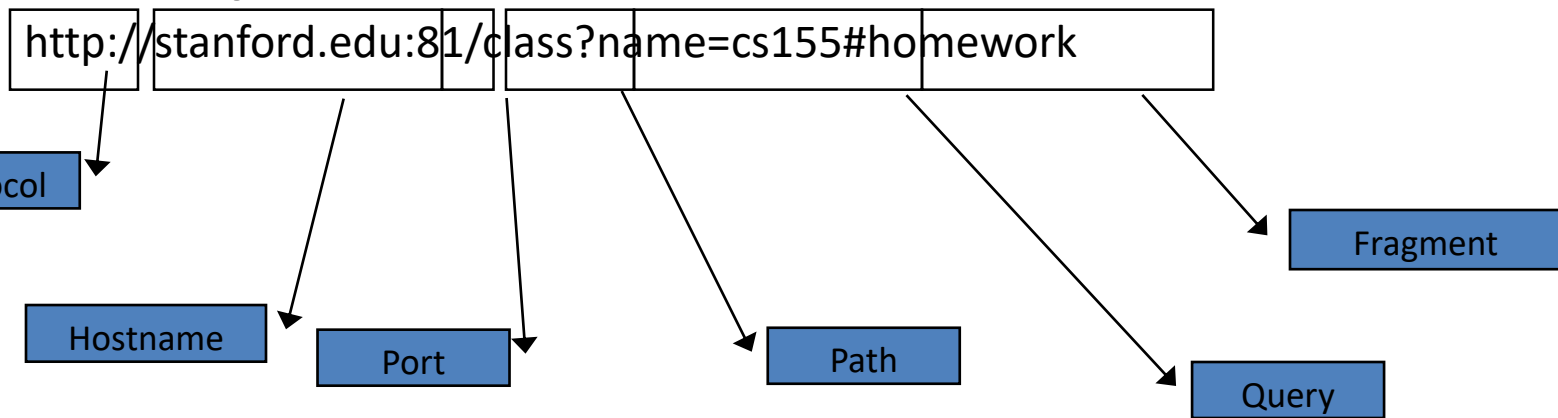
- Http
- Rendering content
- Isolation
- Communication
- Navigation
- Security User Interface
- Cookies
- Frames and frame busting

HTTP

URLs

- Global identifiers of network-retrievable documents

- **Example:**



- Special characters are encoded as hex:
 - `%0A` = newline
 - `%20` or `+` = space, `%2B` = `+` (special exception)

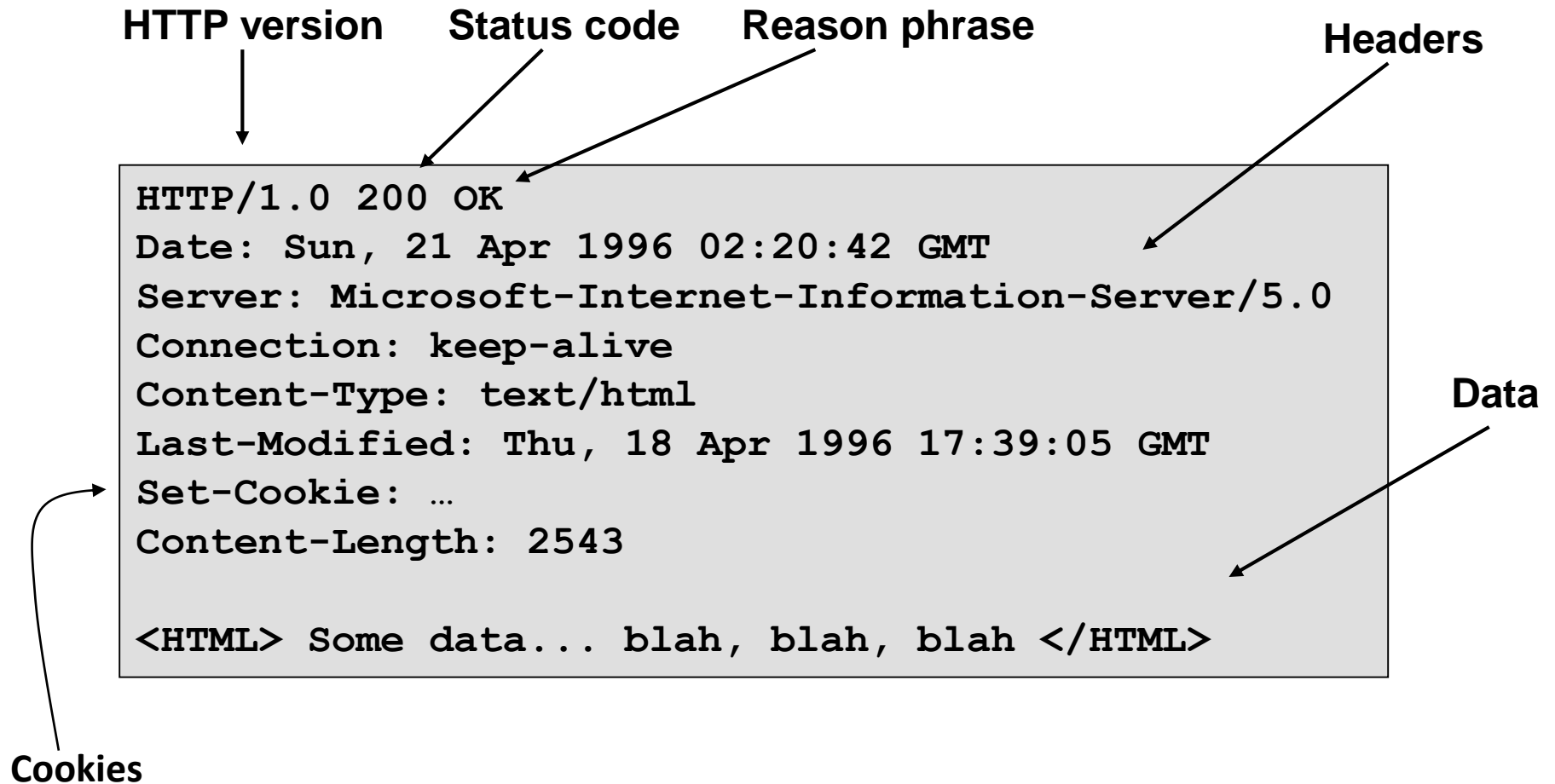
HTTP Request



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 hierarchy
- 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)

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>

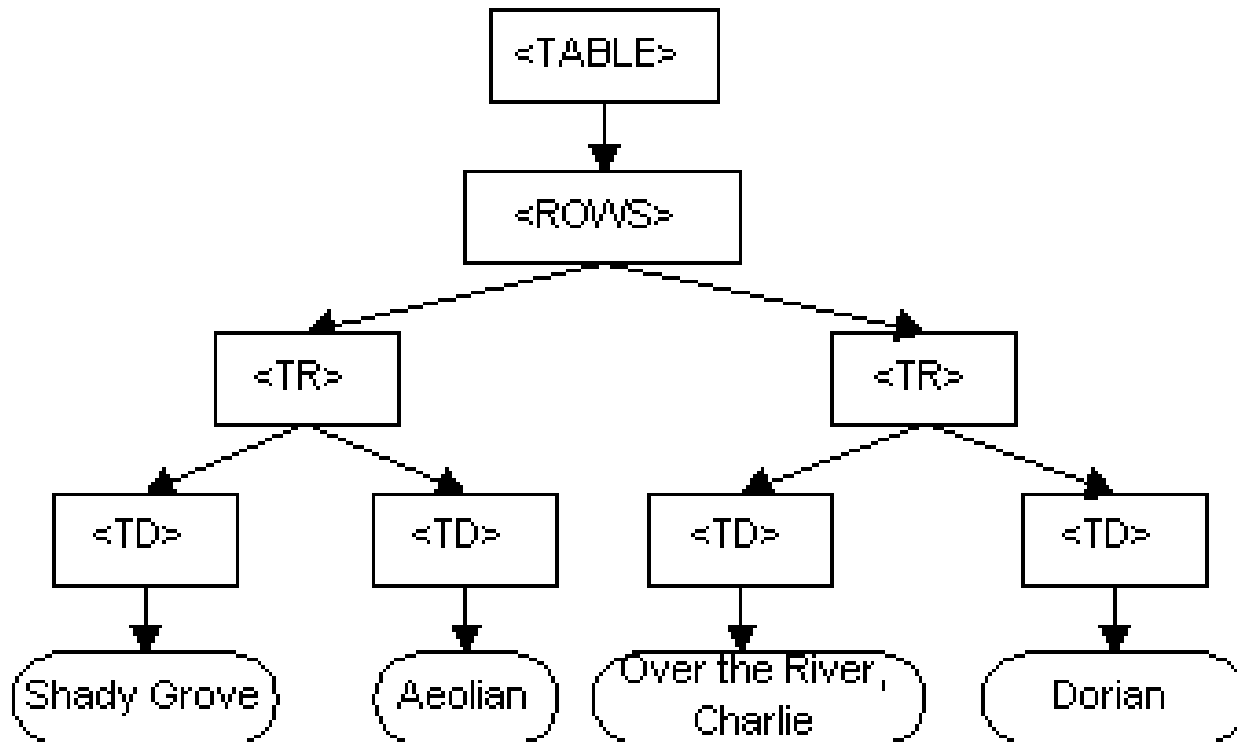
</body>
</html>
```


Document Object Model

- The Document Object Model is a programming API for documents.
- The object model itself closely resembles the structure of the documents it models.
- Consider this table, taken from an HTML document:

```
<TABLE>
  <ROWS>
    <TR>
      <TD>Shady Grove</TD>
      <TD>Aeolian</TD>
    </TR>
    <TR>
      <TD>Over the River, Charlie</TD>
      <TD>Dorian</TD>
    </TR>
  </ROWS>
</TABLE>
```

DOM for the HTML Table



Changing HTML using Script, DOM

HTML

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

```
<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)
```

HTML Image Tags

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

Displays this nice picture →
Security issues?



- Remember, the `img` tag includes a whole *separate file* into your web page. You're only supposed to use this tag for images. Like this:
 - ``
- But suppose some evil teenager in Alaska posts a comment with this little gem:
 - ``

If you visit this website --

```
<html>
<title>Picture of a Cloud</title>
<body>
<p>Here's a picture of a cloud</p>
<image
  onload="alert('image loaded')"
  onerror="alert('load failed.')"
  src="http://spramod.cse.iitk.ac.in:8000/images/cloud.png"
  height="400" width="400">
</body>
</html>
```

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

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

<https://iconewsblog.wordpress.com/2015/09/16/does-your-website-have-a-leak/>

JavaScript timing

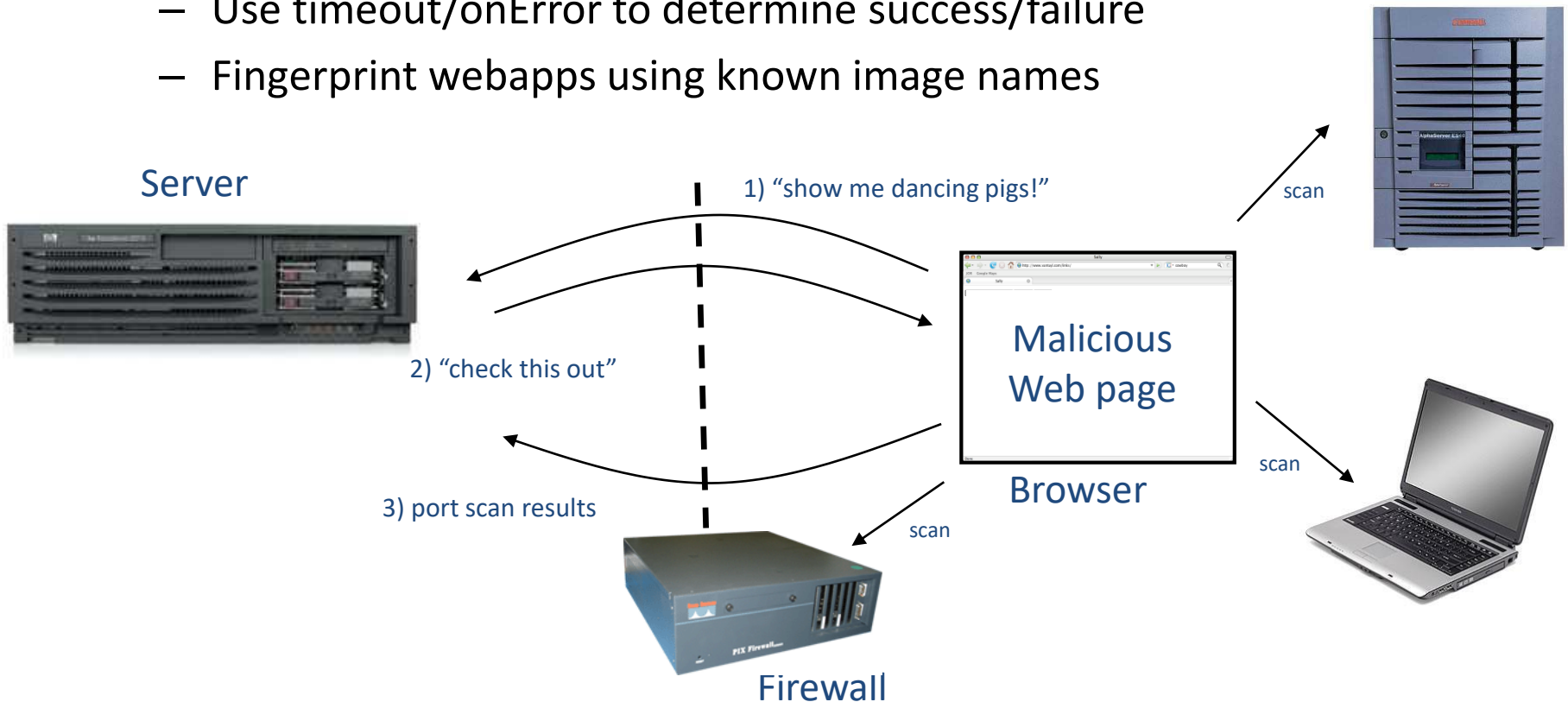
- Sample code

```
<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>
```

- When response header indicates that page is not an image, the browser stops 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



Remote scripting

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

- Goal
 - Exchange data between a client-side app running in a browser and server-side app, without reloading page
- Methods
 - Java Applet/ActiveX control/Flash
 - Can make HTTP requests and interact with client-side JavaScript code, but requires LiveConnect (not available on all browsers)
 - XML-RPC, JSON-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 (or database of static HTML files) is by far the easiest of the three remote scripting options

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

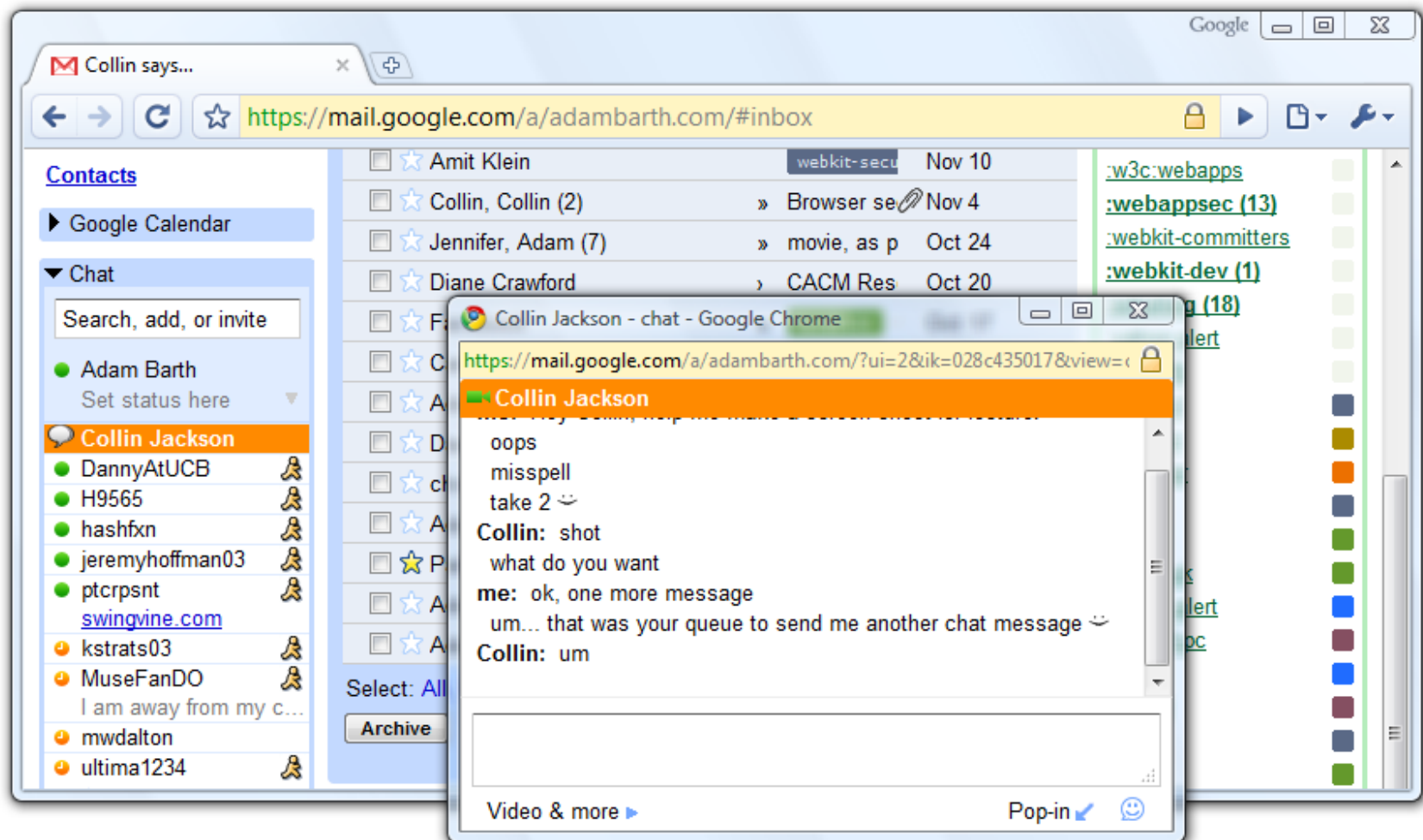
- Window may contain frames from different sources
 - Frame: rigid division as part of frameset
 - iFrame: floating inline frame
- iFrame example

```
<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

http://www.w3schools.com/tags/tryit.asp?filename=tryhtml_iframe

Windows Interact



Analogy

Operating system

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

Web browser

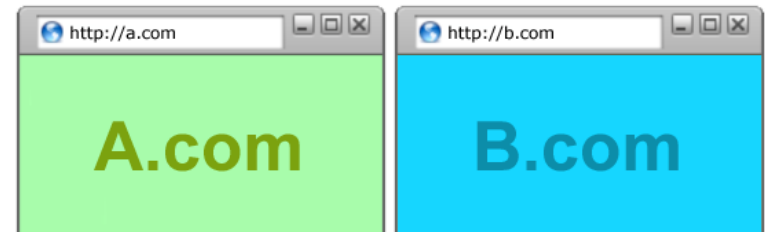
- Primitives
 - Document object model
 - Frames
 - Cookies / localStorage
- Principals: “Origins”
 - Mandatory access control
- Vulnerabilities
 - Cross-site scripting
 - Cross-site request forgery
 - Cache history attacks
 - ...

Policy Goals

- Safe to visit an evil web site



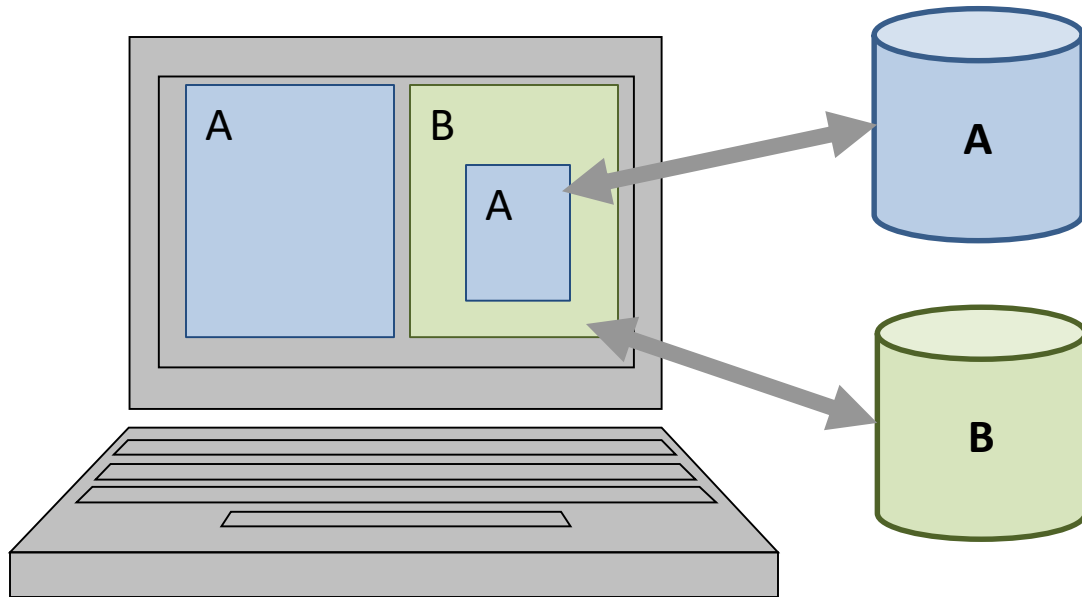
- Safe to visit two pages at the same time
 - Address bar distinguishes them



- 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
 - canScript(A,B)
 - Can Frame A execute a script that manipulates arbitrary/nontrivial DOM elements of Frame B?
 - canNavigate(A,B)
 - Can Frame A change the origin of content for Frame B?
- Frame-principal relationships
 - readCookie(A,S), writeCookie(A,S)
 - Can Frame A read/write cookies from site S?

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

Few Words about Browser Side Java Scripts

- JavaScript
 - object-based imperative language tightly integrated with HTML
 - supported by all contemporary web browsers
- Browser-side JavaScript is invoked from within HTML documents in four primary ways:
 - Standalone <SCRIPT> tags that enclose code blocks,
 - Event handlers tied to HTML tags (e.g. onmouseover="..."),
 - Stylesheet expression(...) blocks that permit JavaScript syntax in some browsers,
 - Special URL schemes specified as targets for certain resources or actions (javascript:...) - in HTML and in stylesheets.

Javascript (2)

- Regardless of source (`<SCRIPT SRC="...">`), remote scripts execute in context of document they are attached to.
 - Once called, JavaScript has full access to the current DOM, and limited access to DOMs of other windows;
- May also further invoke new JavaScript by calling `eval()`, configuring timers (`setTimeout(...)` and `setInterval(...)`), or producing JavaScript-invoking HTML.
- JavaScript may also configure self to launch when objects are interacted with by third-party JS code, by configuring watches, setters, or getters, or cross contexts etc.

Some Security Relevant Features of Javascript

- **Dynamic, runtime code interpretation with no strict code caching rules.**
 - Any code snippets located in-line with HTML tags would be interpreted and executed,
 - JavaScript itself has a possibility to either directly evaluate strings as JavaScript code (`eval(...)`), or
 - to produce new HTML that in turn may contain more JavaScript (`.innerHTML` and `.outerHTML` properties, `document.write()`, event handlers).

Cascading Style Sheets (CSS)

- Set of tools to alter the visual appearance of any portion of the document
- A stylesheet outlines visual rendering rules for various types of document elements (e.g., lists, tables, links, or quotations) using a separate block of data

Three distinct ways to place CSS directives in HTML documents

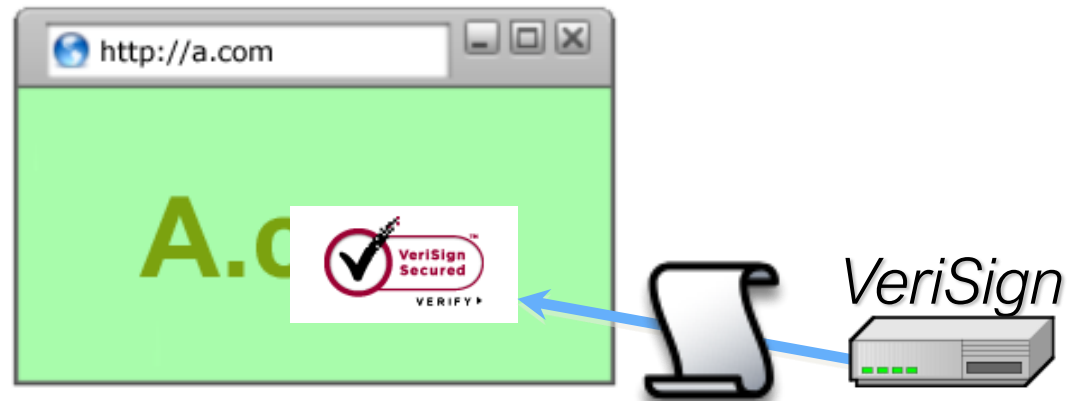
- `STYLE="..."` parameter attached to HTML tags of any type
 - attributes specified this way apply to this and nested tags only
- Block of CSS code with `<STYLE>...</STYLE>`
 - may change the default appearance of any tag
 - or define named rule sets that may be explicitly applied to specific tags with a `CLASS="..."` parameter,
- Inclusion of a remote style sheet with a `LINK REL="stylesheet" HREF="...">`, with the same global effect as a `<STYLE>` block.

Security consequences of attacker controlled style sheets

- **Some CSS implementations permit JavaScript** code to be embedded in stylesheets.
 - Using the `expression(...)` directive, which gives the ability to evaluate arbitrary JavaScript statements and use their value as a CSS parameter;
 - Using `url('javascript:...')` directive on properties that support it;
 - By invoking browser-specific features such as the `-moz-binding` mechanism of Firefox.
- **The ability to reuse trusted classes.**
 - If user-controlled `CLASS="..."` attributes are permitted in HTML syntax, the attacker may have luck "borrowing" a class used to render elements of the trusted UI and impersonate them.

Library import excluded from SOP

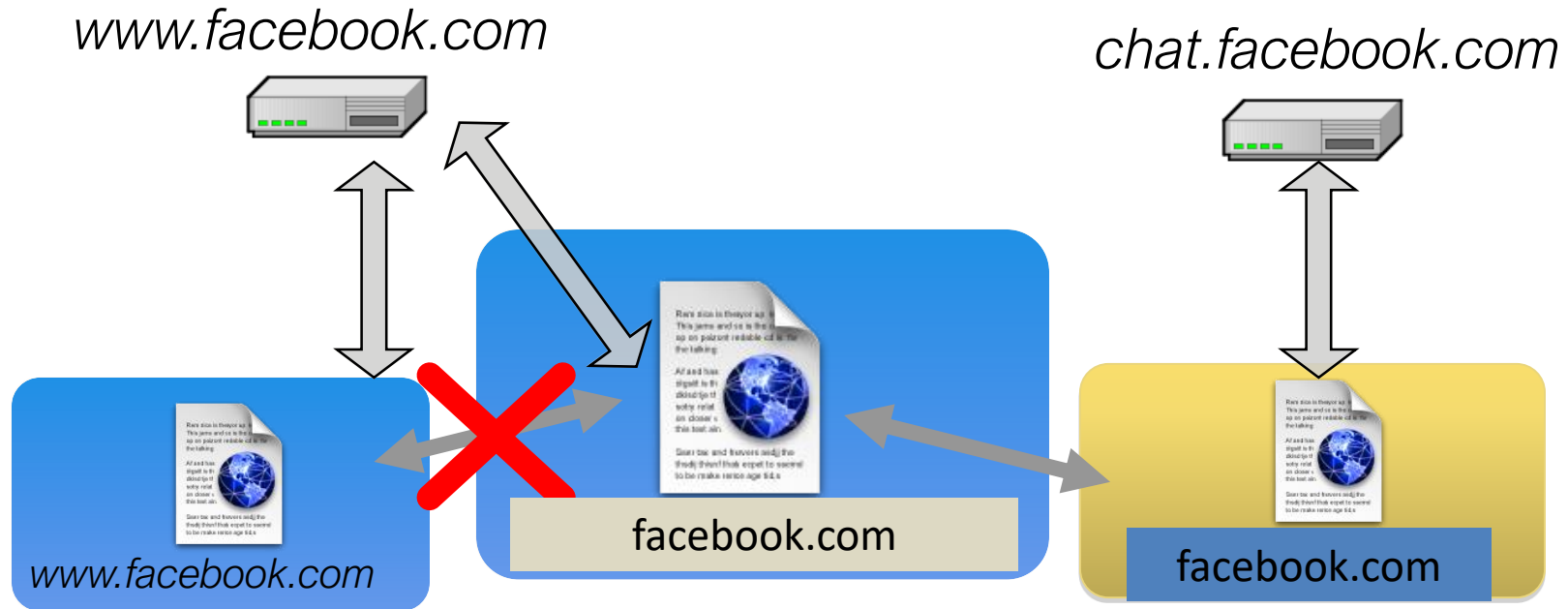
```
<script  
  src=https://seal.verisign.com/getseal?host_name=a.c  
om></script>
```



- Script has privileges of imported page, NOT source server.
- Can script other pages in this origin, load more scripts
- Other forms of importing

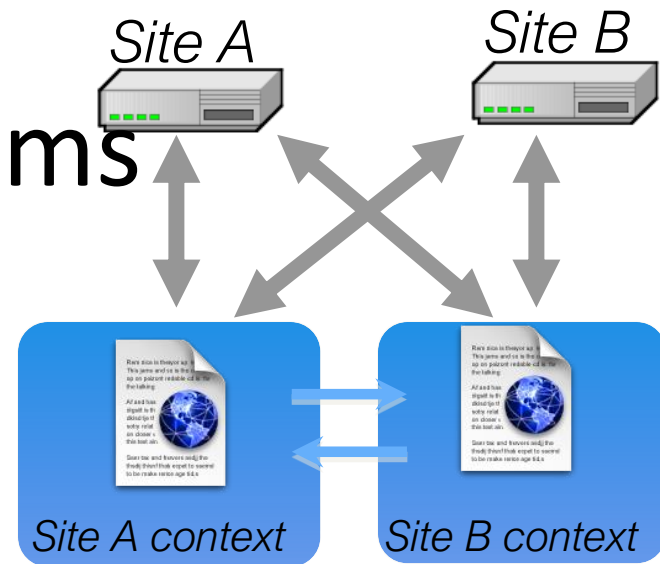


Domain Relaxation



- 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

postMessage

COMMUNICATION

window.postMessage

- API for inter-frame communication
 - Supported in standard browsers

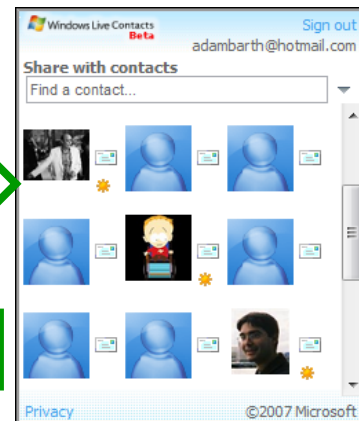


- A network-like channel between frames



Add a contact

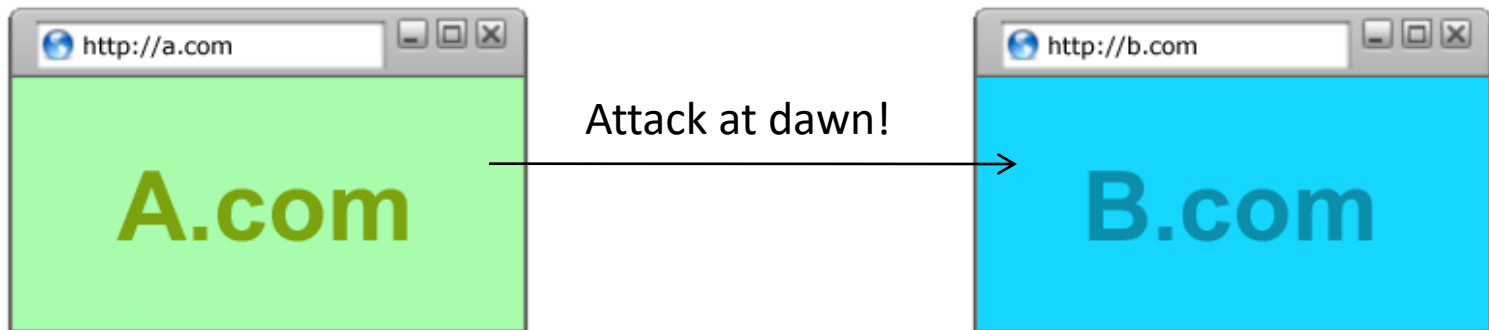
Share contacts



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);
```

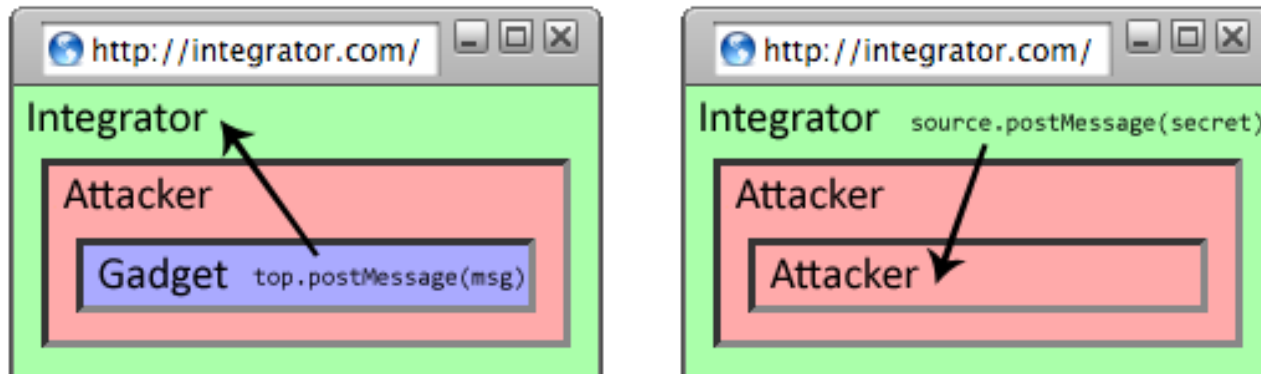


Why include “targetOrigin”?

- What goes wrong?

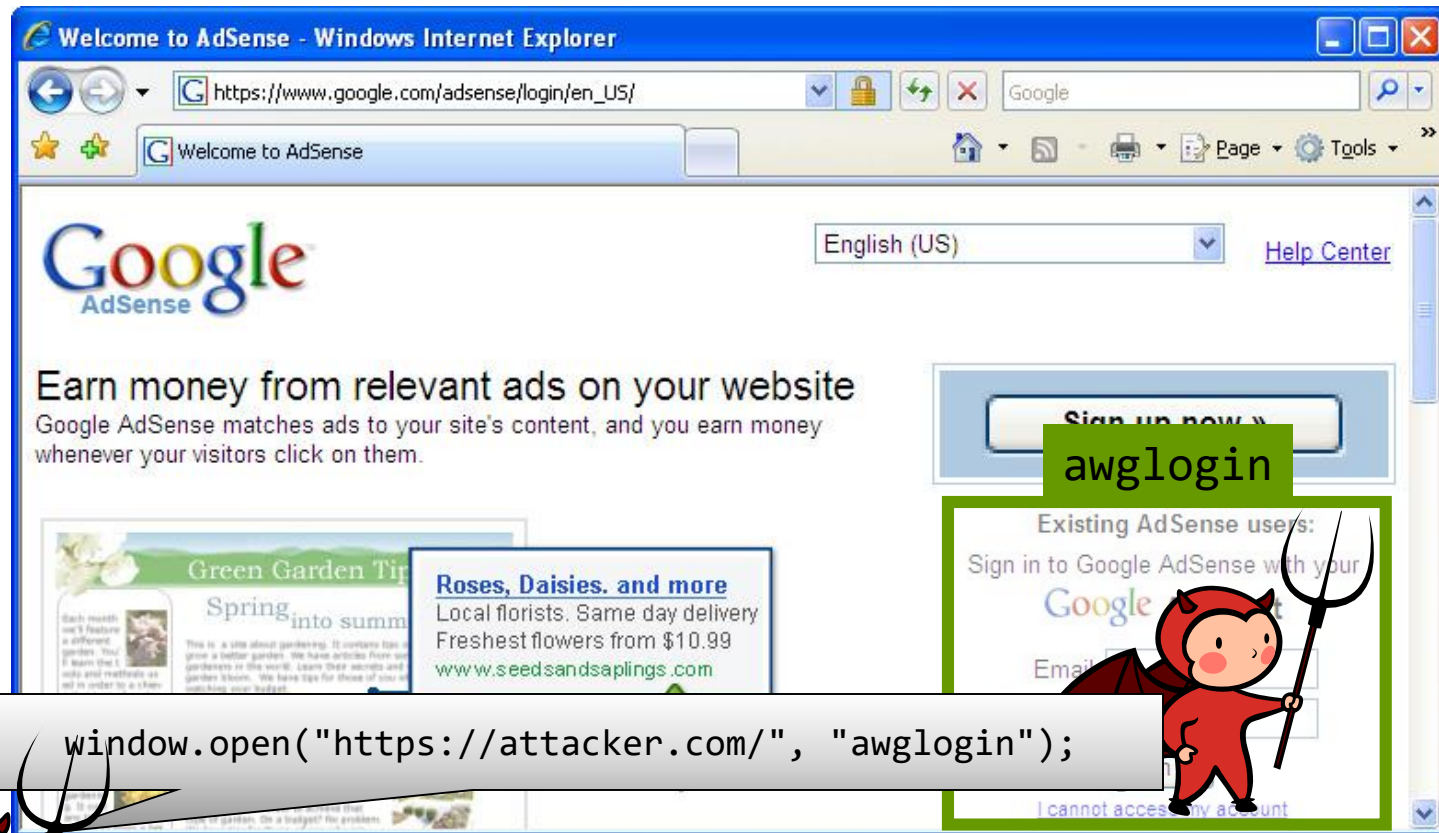
```
frames[0].postMessage("Attack at dawn!");
```

- Messages sent to *frames*, not principals
 - When would this happen?



NAVIGATION

A Guninski Attack



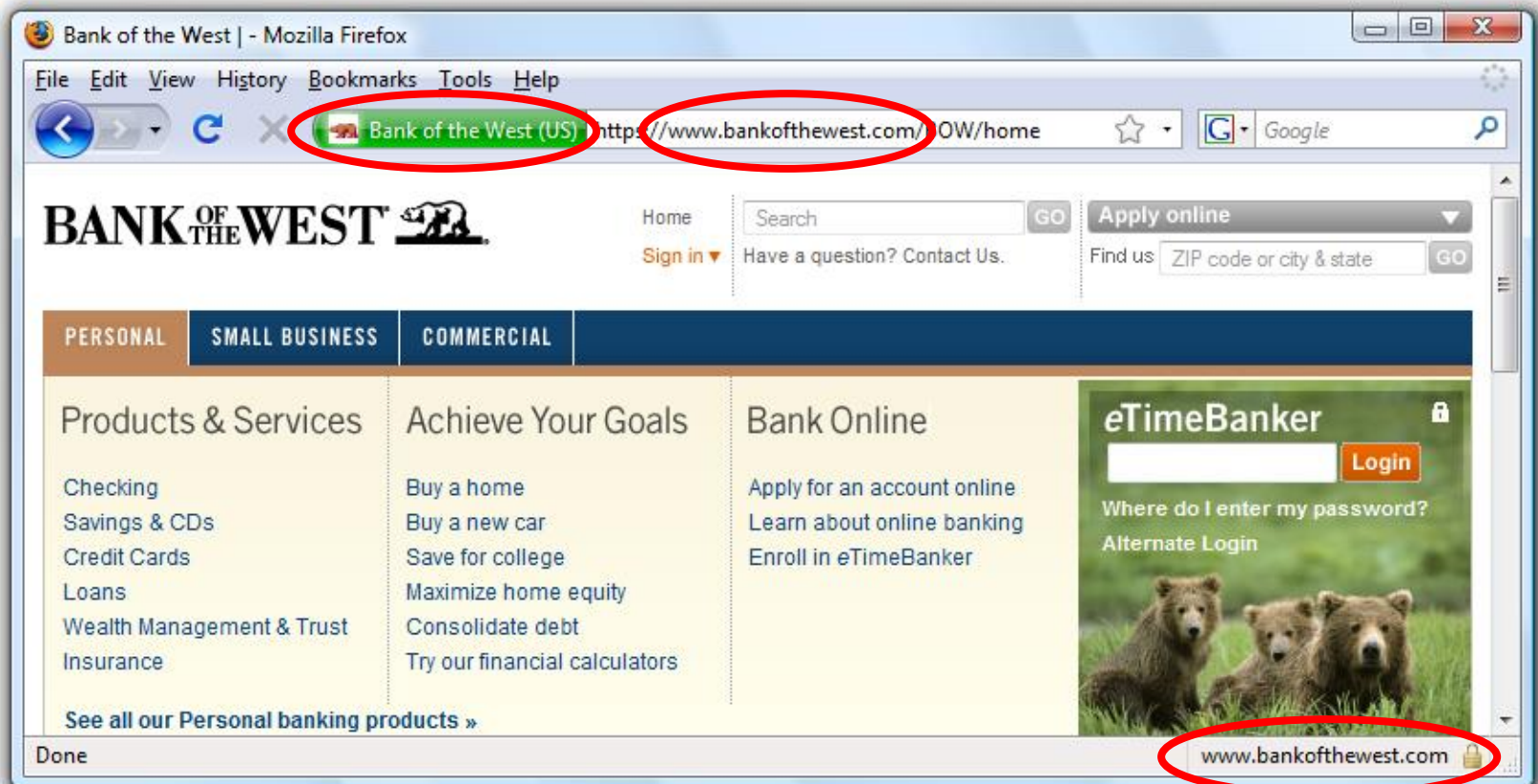
Preventing Guninski Attacks

- SOP controls frame navigation nowadays
- There used to be descendant policy, but it is no longer used

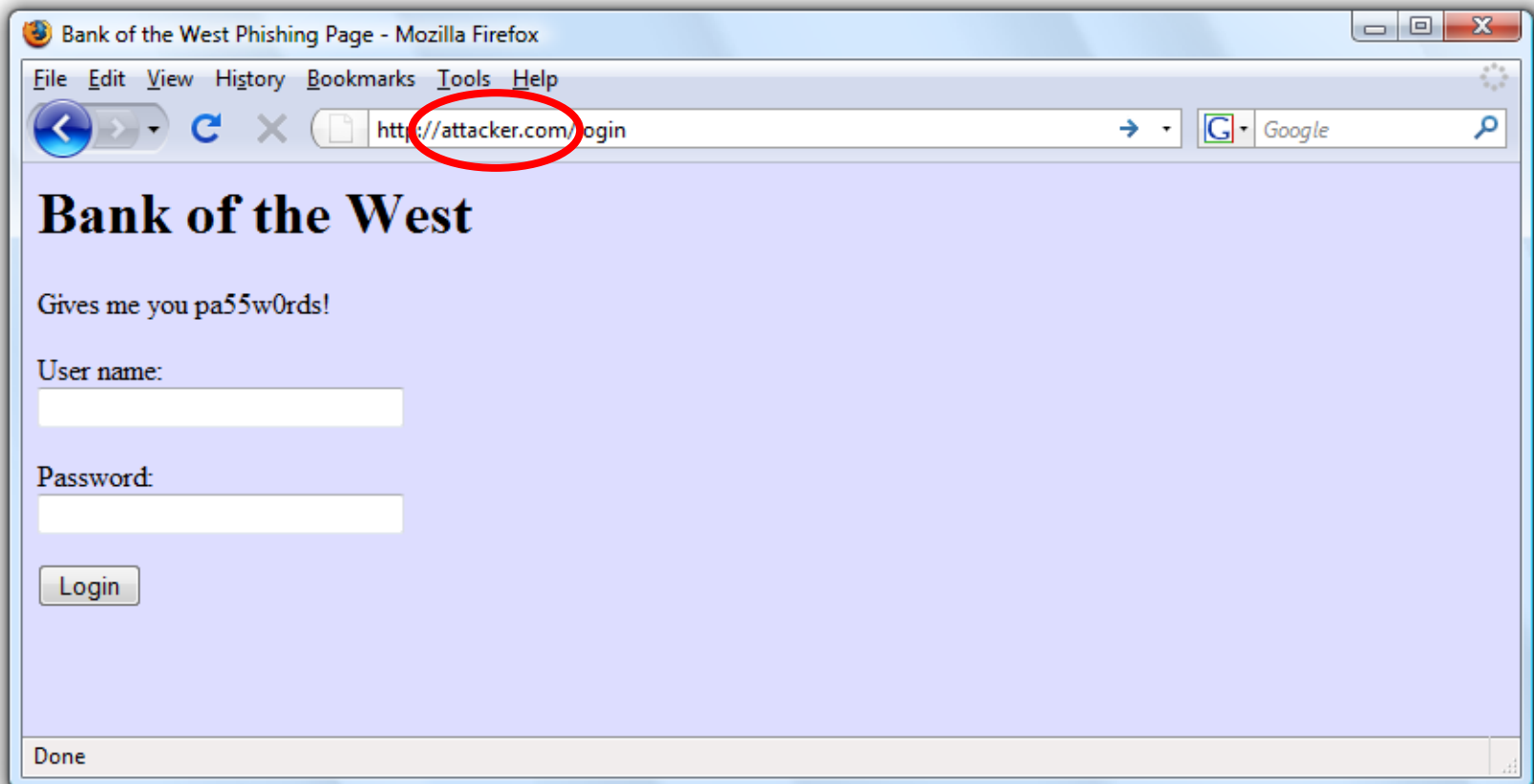
When is it safe to type my password?

SECURITY USER INTERFACE

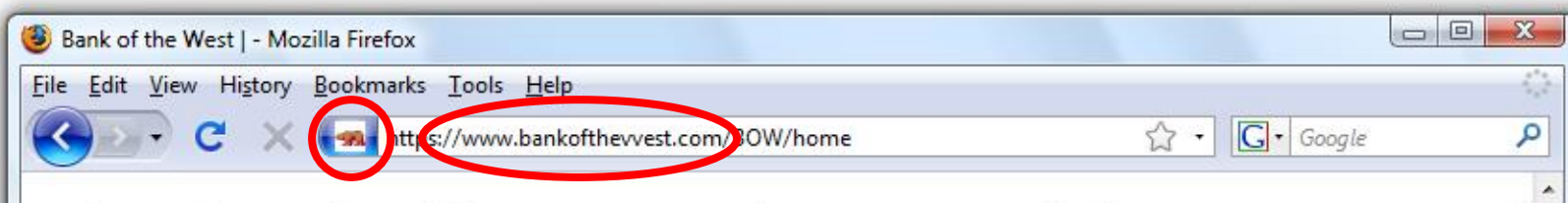
Safe to type your password?



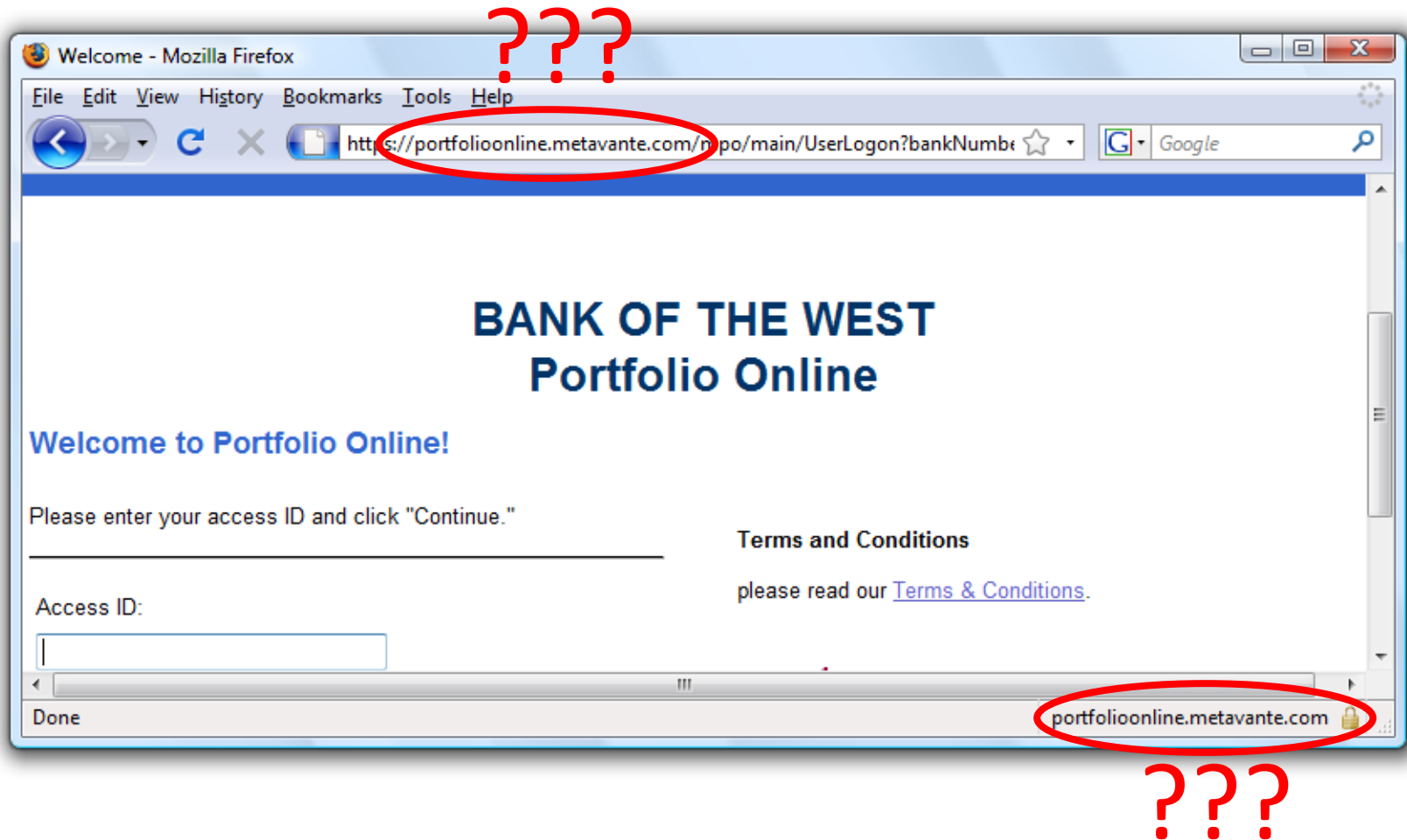
Safe to type your password?



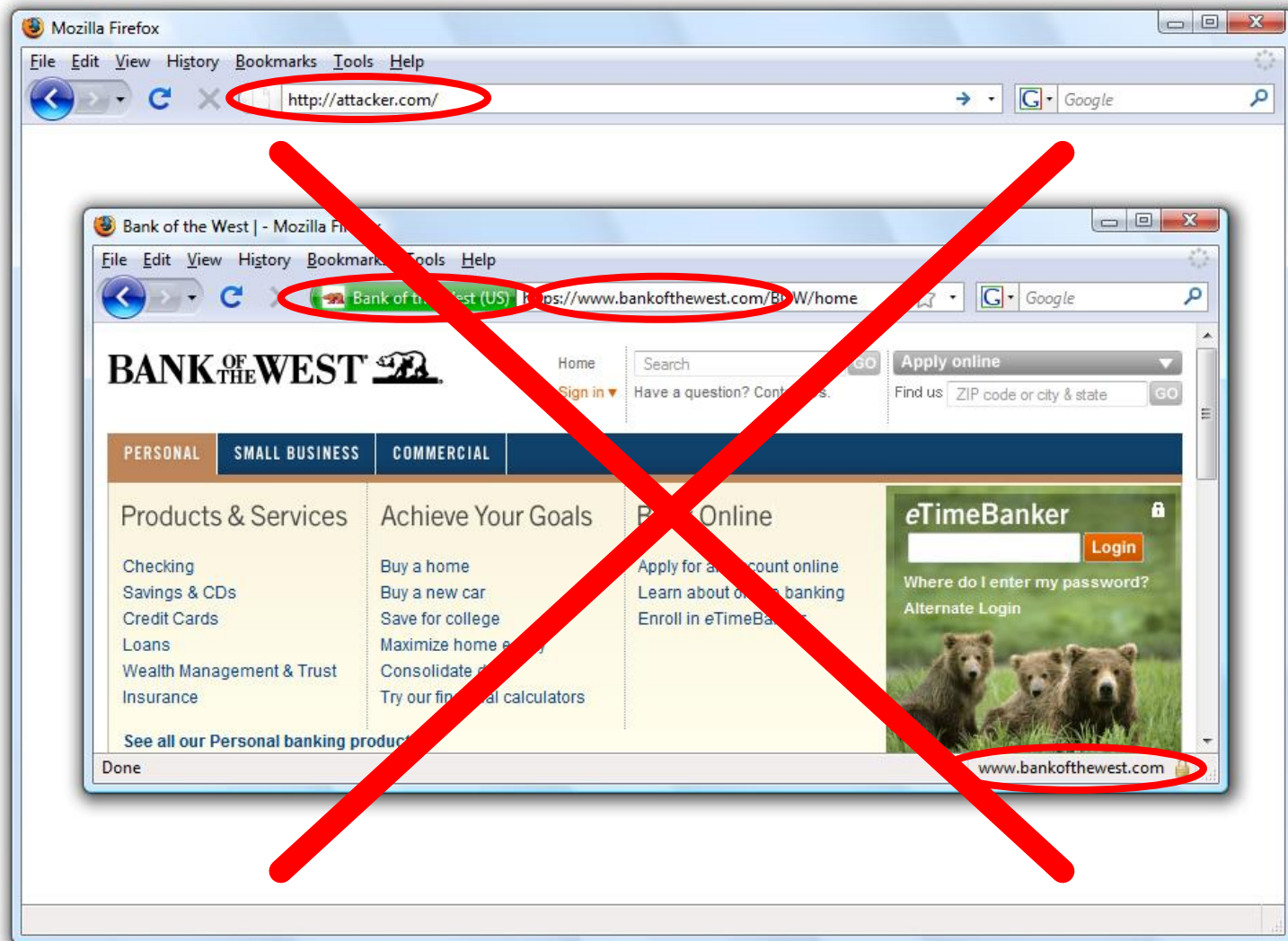
Safe to type your password?



Safe to type your password?

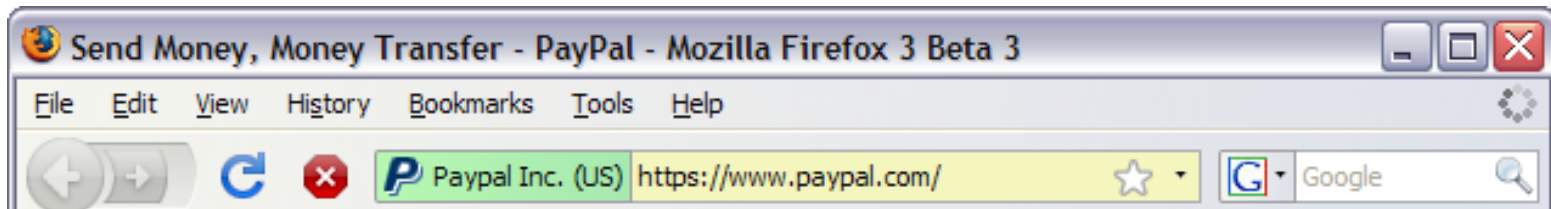
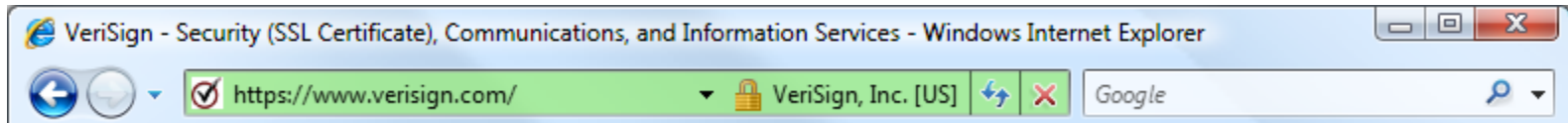


Safe to type your password?



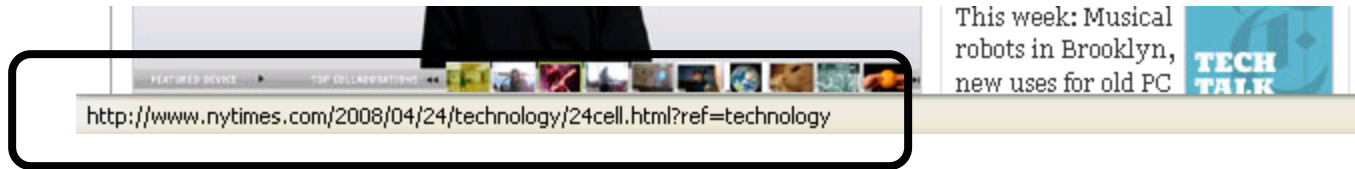
Lock Icon 2.0

- 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



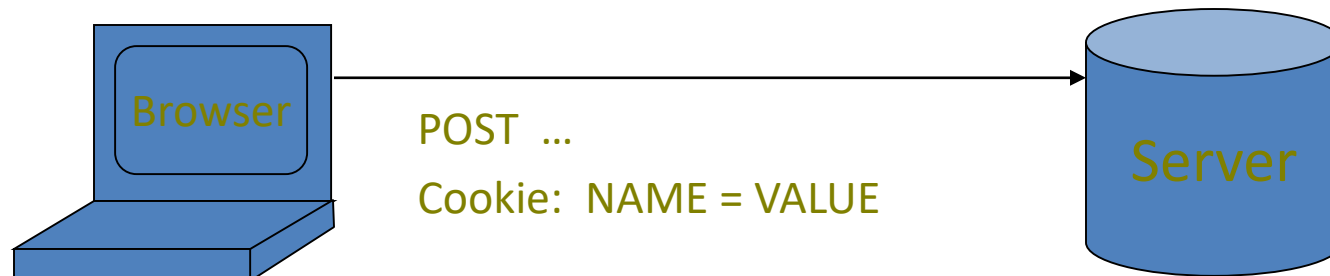
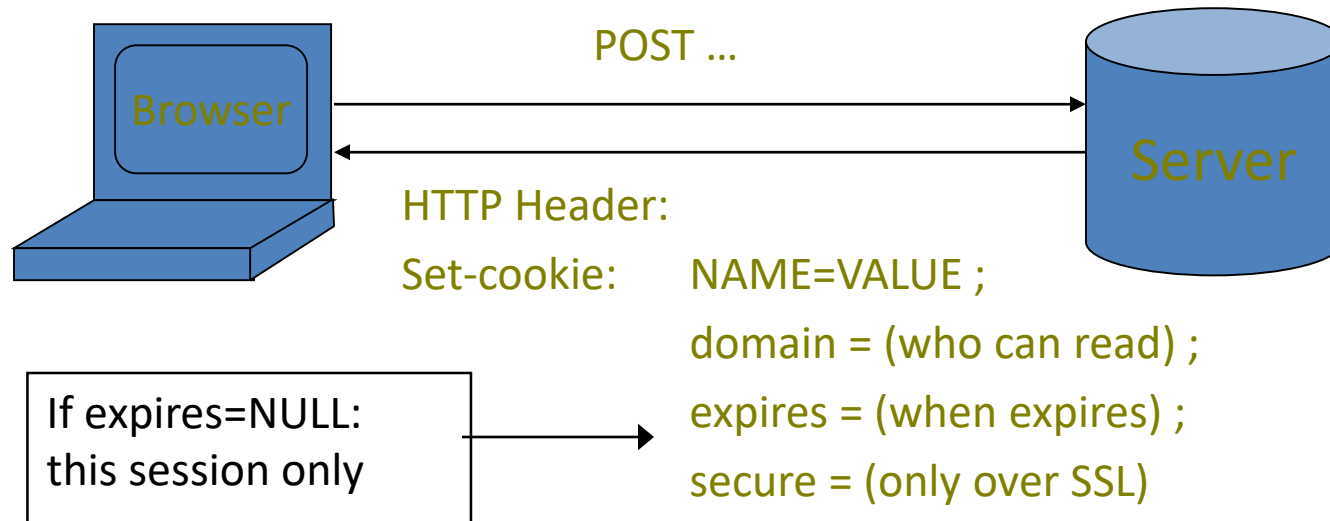
- Trivially spoofable

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

COOKIES: CLIENT STATE

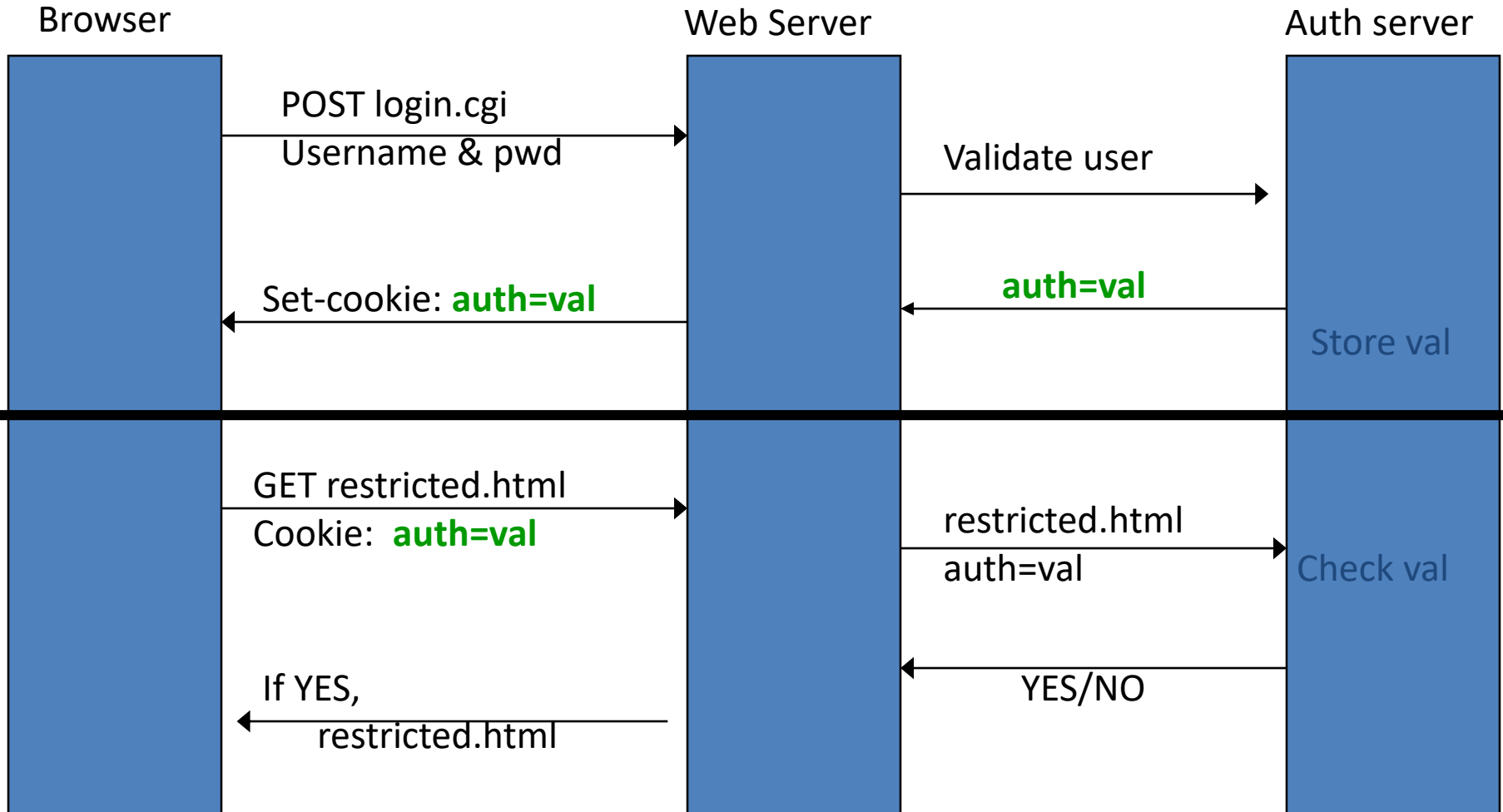
Cookies

- Used to store state on user's machine



HTTP is stateless protocol; cookies add state

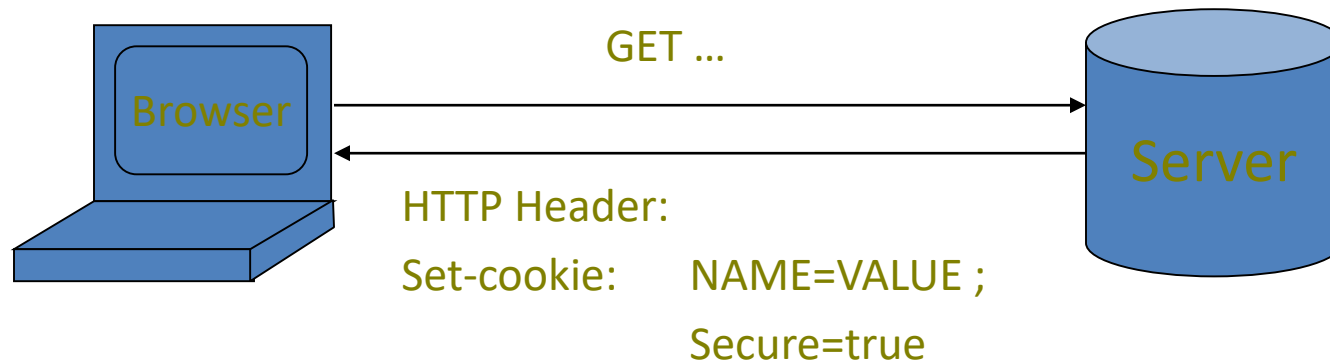
Cookie authentication



Cookie Security Policy

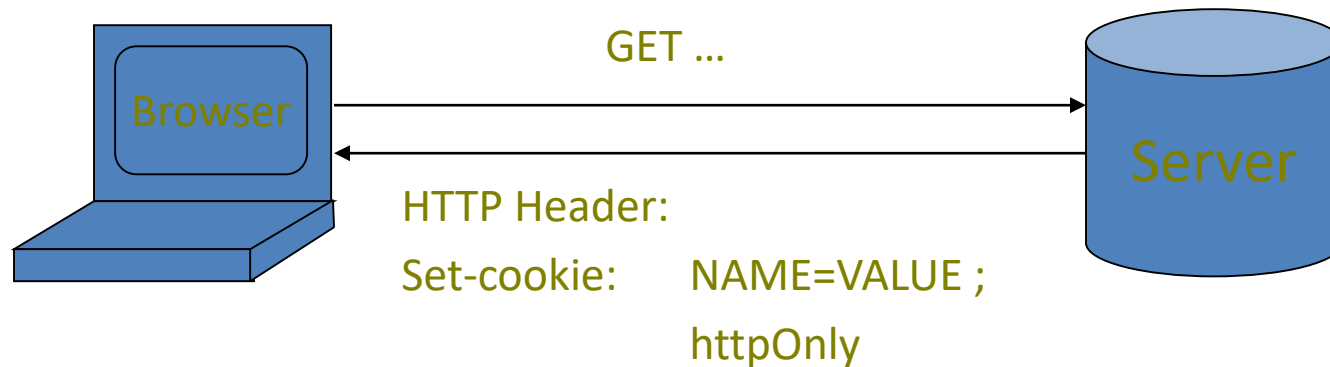
- Uses:
 - User authentication
 - Personalization
 - User tracking: e.g. Doubleclick (3rd party cookies)
- Browser will store:
 - At most 20 cookies/site, 3 KB / cookie
- Origin is the tuple **<domain, path>**
 - Can set cookies valid across a domain suffix

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

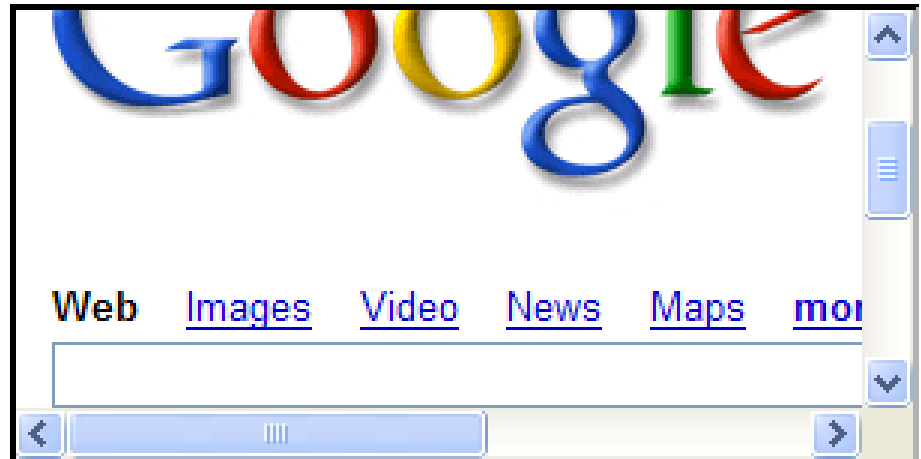
Frames

- 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 HTTP response header instead:

```
X-Frame-Options: deny
```

Summary

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