Web Security

CS642: Computer Security

Liberal borrowing from Mitchell, Boneh, Stanford CS 155

Web security part 1

Basic web security models

Browser security

Same-origin policy / Navigation policy

Cookies / Session handling

WWW

Tim Berners-Lee and Robert Cailliau 1990

HTTP, CERN httpd, gopher

1993 Mosiac web browser (UIUC, Marc Andreesen)

1994 W3C WWW Consortium --- generate standards

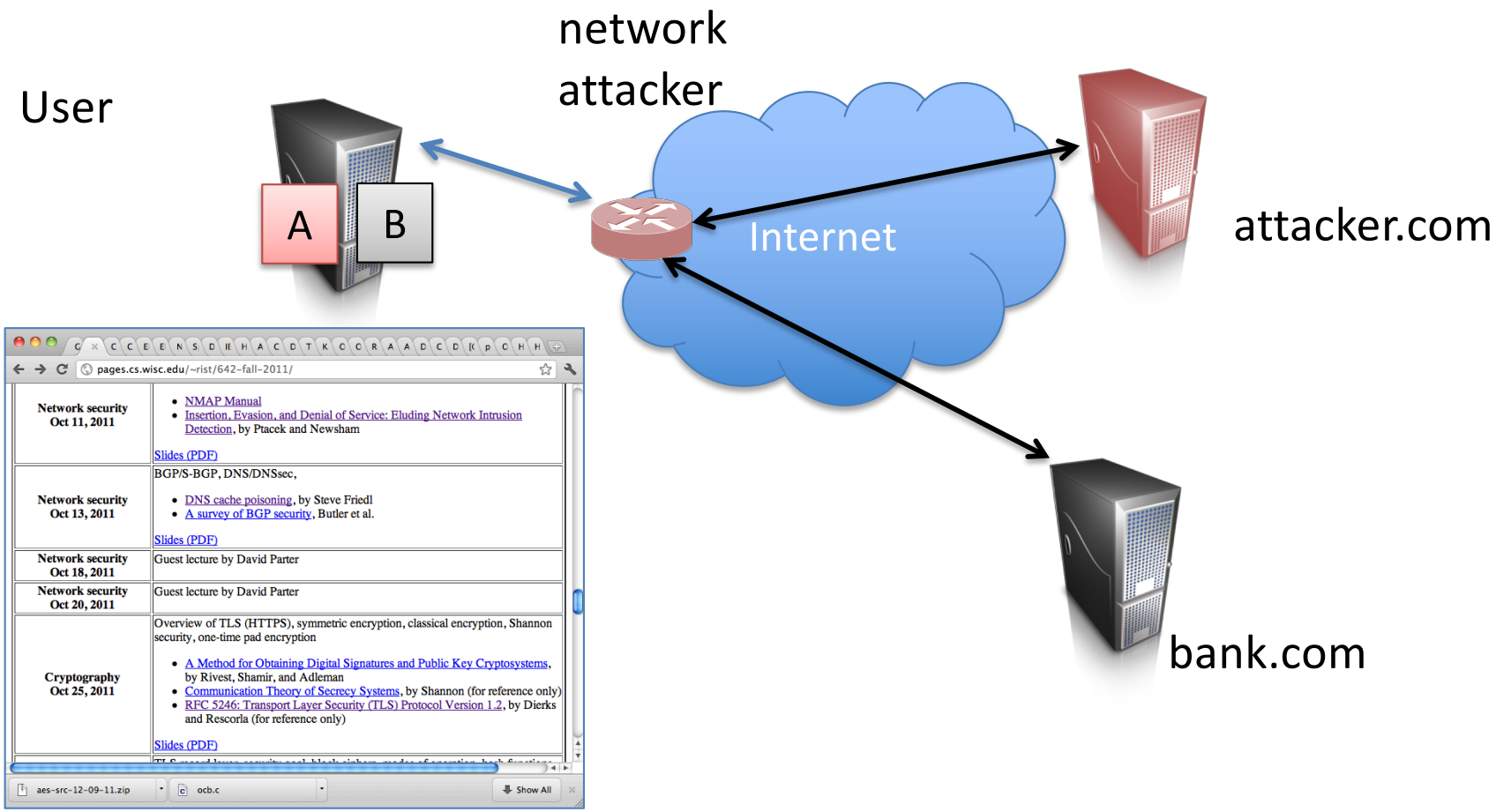
Gopher started charging licensing fees

(Univ of Minnesota)

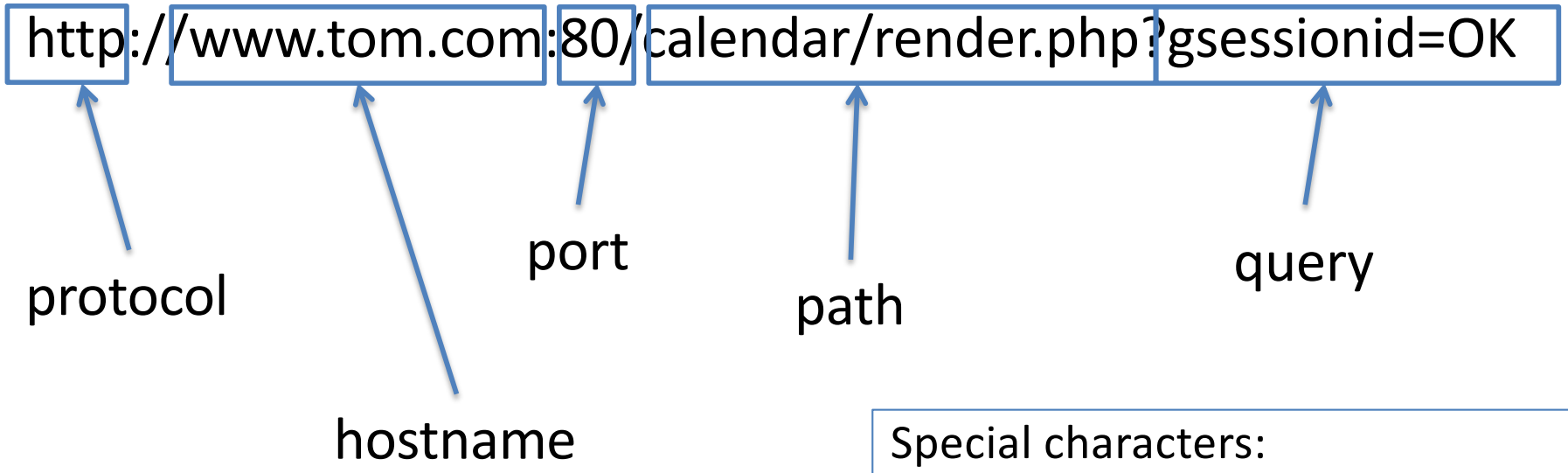
Nowadays: ecosystem of technologies

- HTTP / HTTPS
- AJAX
- PHP
- Javascript
- SQL
- Apache
- Ruby
- http://w3schools.com/

Threat model



Some basics of HTTP

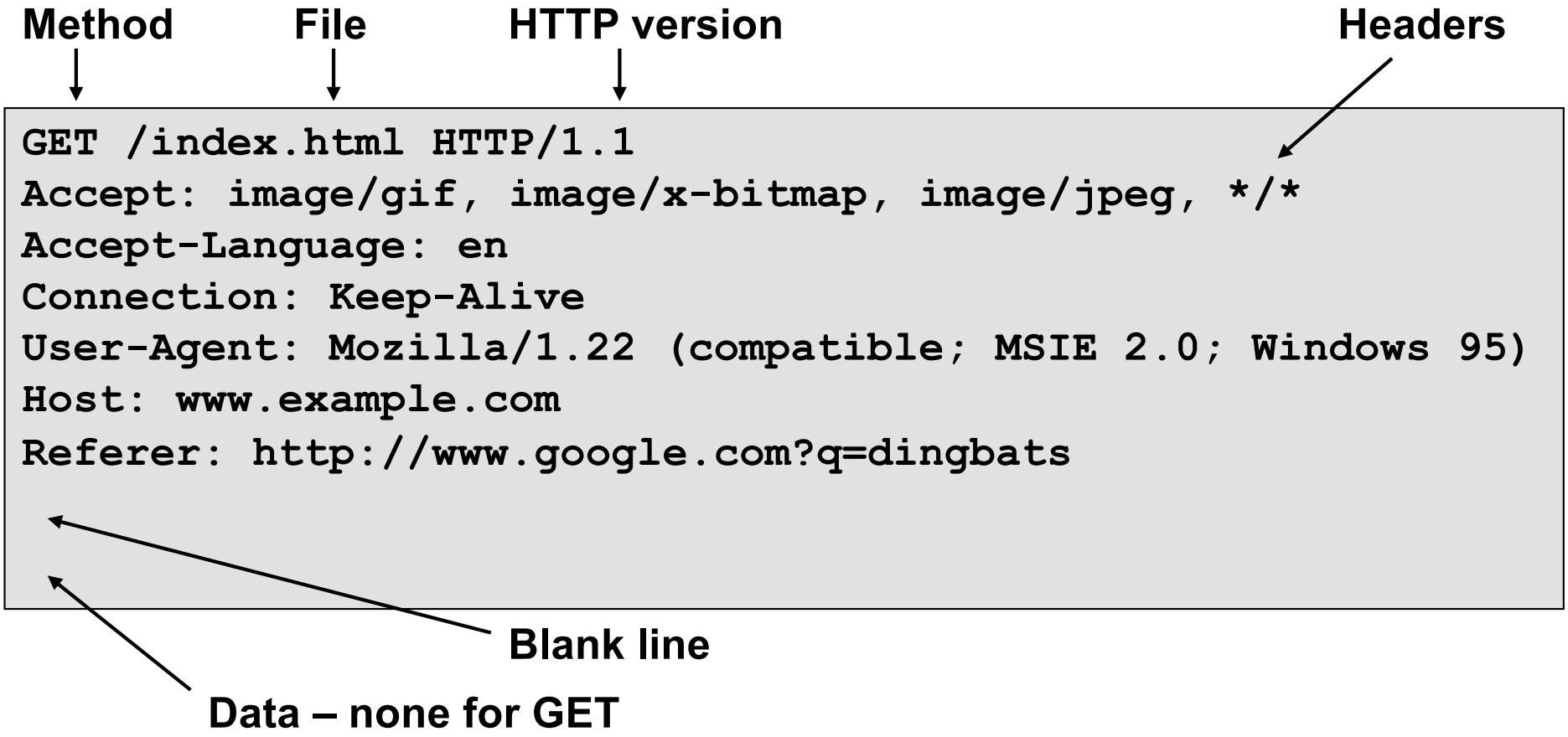


URL's only allow ASCII-US characters.
Encode other characters:

%0A = newline
%20 = space

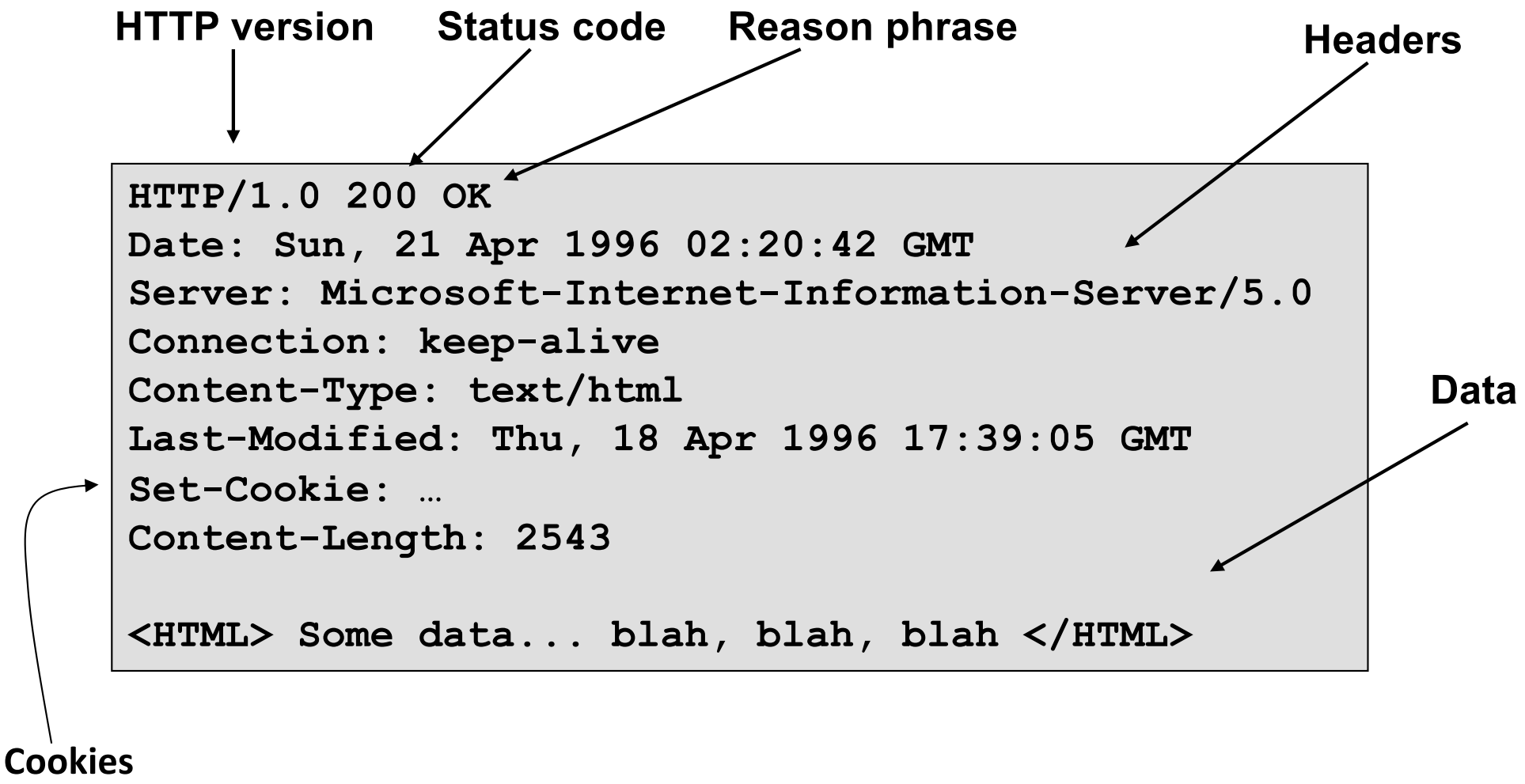
Special characters:
+ = space
? = separates URL from parameters
% = special characters
/ = divides directories, subdirectories
= bookmark
& = separator between parameters

HTTP Request

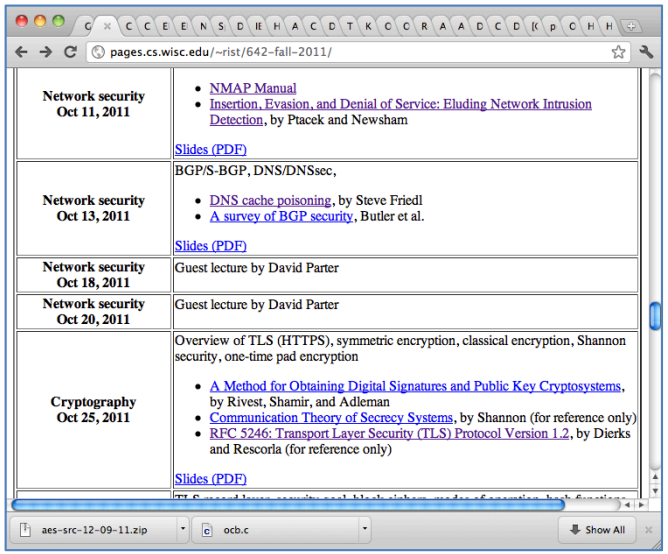


GET : no side effect POST : possible side effect

HTTP Response



Browser execution



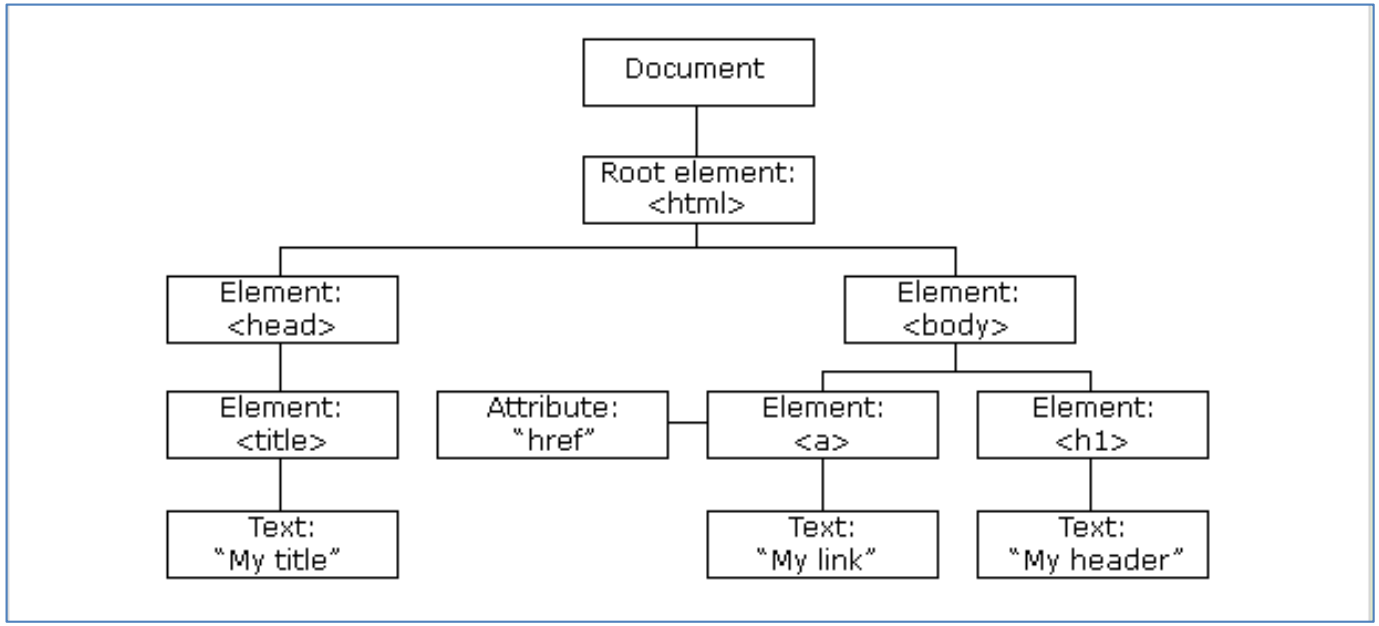
- Each window (or tab):
  - Retrieve/load content
  - Render it
    - Process the HTML
    - Might run scripts, fetch more content, etc.
  - Respond to events
    - User actions: OnClick, OnMouseover
    - Rendering: OnLoad, OnBeforeUnload
    - Timing: setTimeout(), clearTimeout()

Document object model (DOM)

Object-oriented way to organize objects in a web page

Properties: document.alinkColor, document.URL, document.forms[ ], document.links[ ], document.anchors[ ]

Methods: document.write(document.referrer)

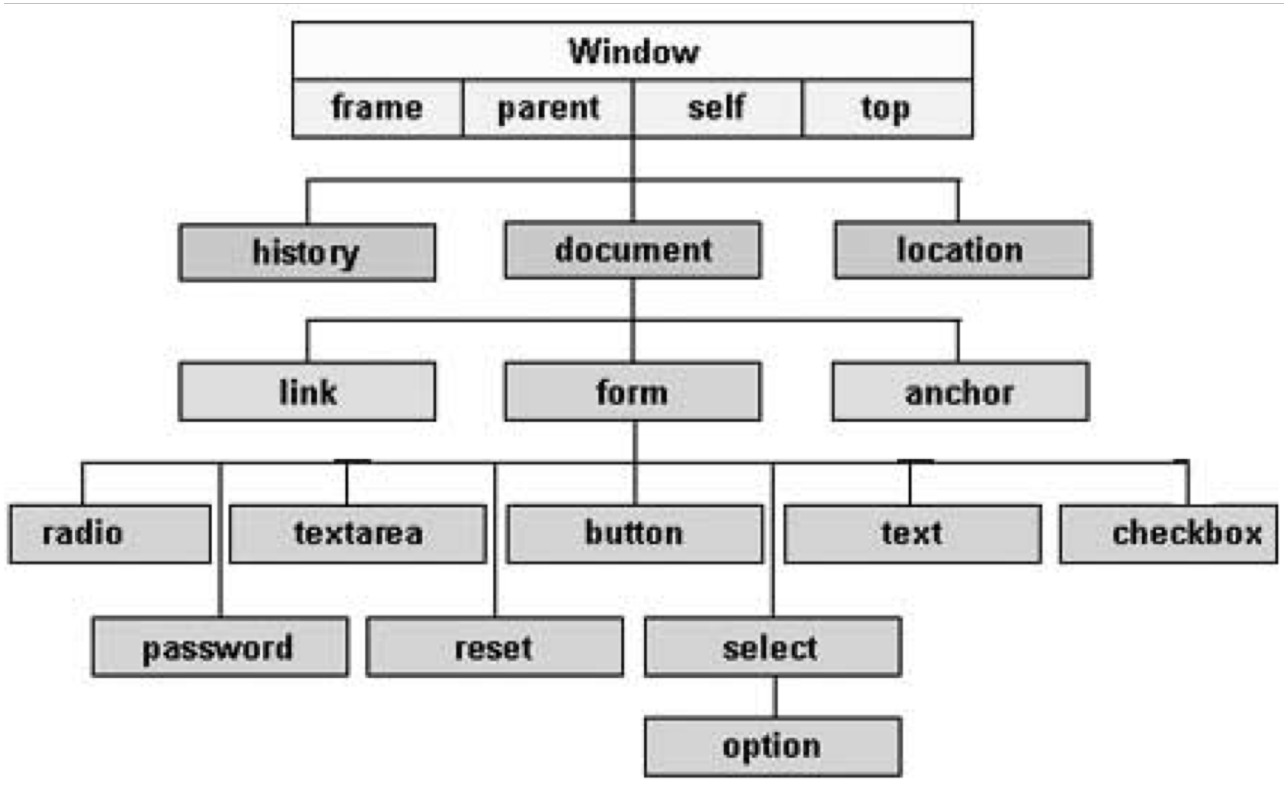


https://www.w3schools.com/js/js_htmldom.asp

Browser object model (BOM)

Corresponding model for larger browser window,

document, frames[], history, location,
navigator (type and version of browser)



From boostlog.io

Seemingly innocuous features?

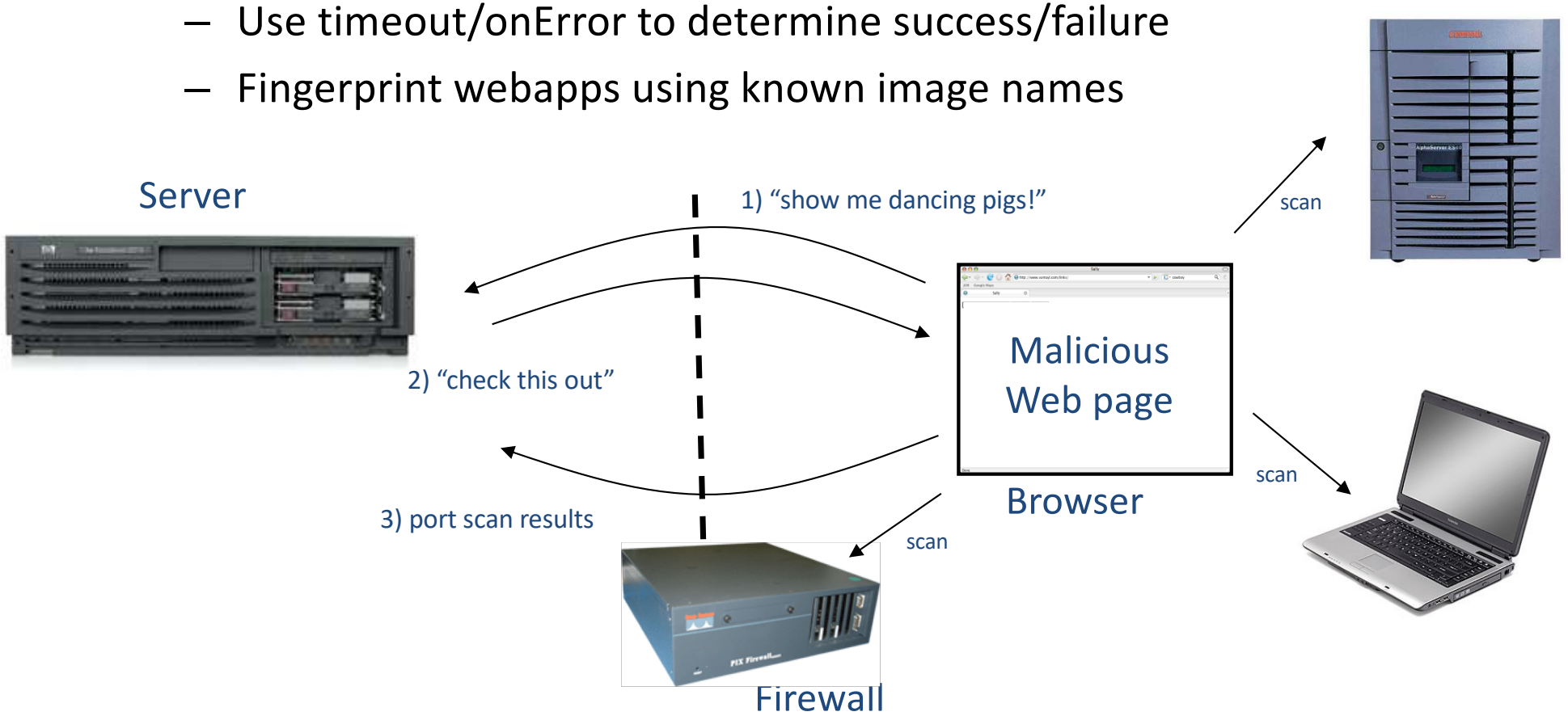
- <img src="bucky.jpg" height="50pt" width="50pt">
- Displays an image
- What can attacker do?

Javascript timing

```
<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>
```

Behind-firewall webapp scanning

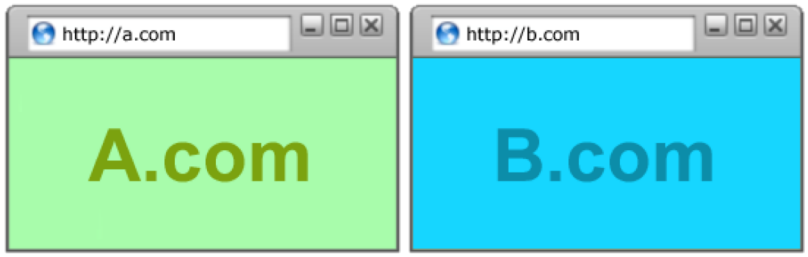
- JavaScript can:
  - Request images from internal IP addresses
    - Example:
  - Use timeout/onError to determine success/failure
  - Fingerprint webapps using known image names



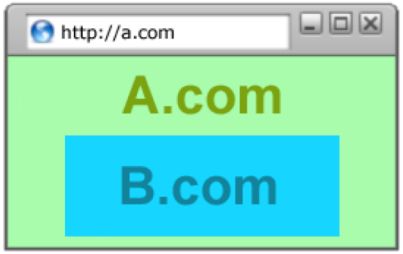
Browser security model

Should be safe to visit an attacker website

Should be safe to visit sites simultaneously



Should be safe to delegate content



Browser isolation

Browser is running untrusted inputs (attacker webpage)

Like all big, complex software, browser has security vulnerabilities

Browsers include “Rich Internet Applications” (RIAs) that increase attack surface:

e.g., Adobe Flash

Malicious website exploits browser, from there system

Web pages are not single-origin

IFrames: **<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]

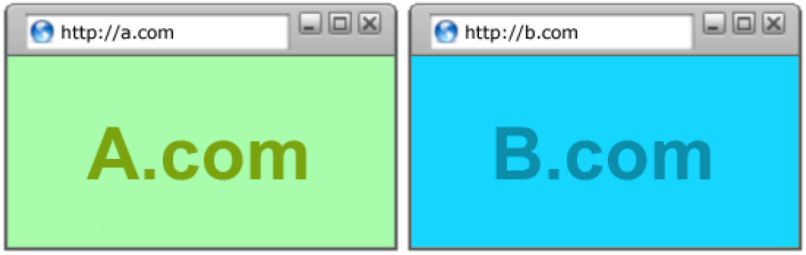
```
<script>
        var so = new SWFObject('//site.com/flash.swf', …);
        so.addParam('allowscriptaccess', 'always');
        so.write('flashdiv');
</script>
```

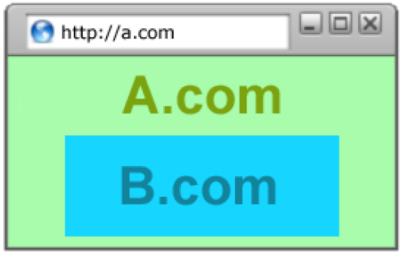
multi-origin pages

- iframes: `<iframe src="//site.com/frame.html"/>`
- scripts: `<script src="//site.com/script.js"/>`
- CSS: `<link rel="stylesheet" type="text/css" href="//site.com/theme.css"/>`
- Images
- Videos
- Content delivery network (CDN)
- Authentication (OAUTH, others)
- Payment
- Social sharing buttons
- Tracking beacons
- Analytics
- Ads
- Ads
- Ads
- Even more ads

Isolation challenges

- What are the subjects?
  - To what things do we grant access?
- What are the objects?
  - What are we controlling protection over?
  - What are the operations we control?
- What are the policies?
  - What do we want to allow / disallow?





Browser handles multiple sites, must maintain separate security contexts for each

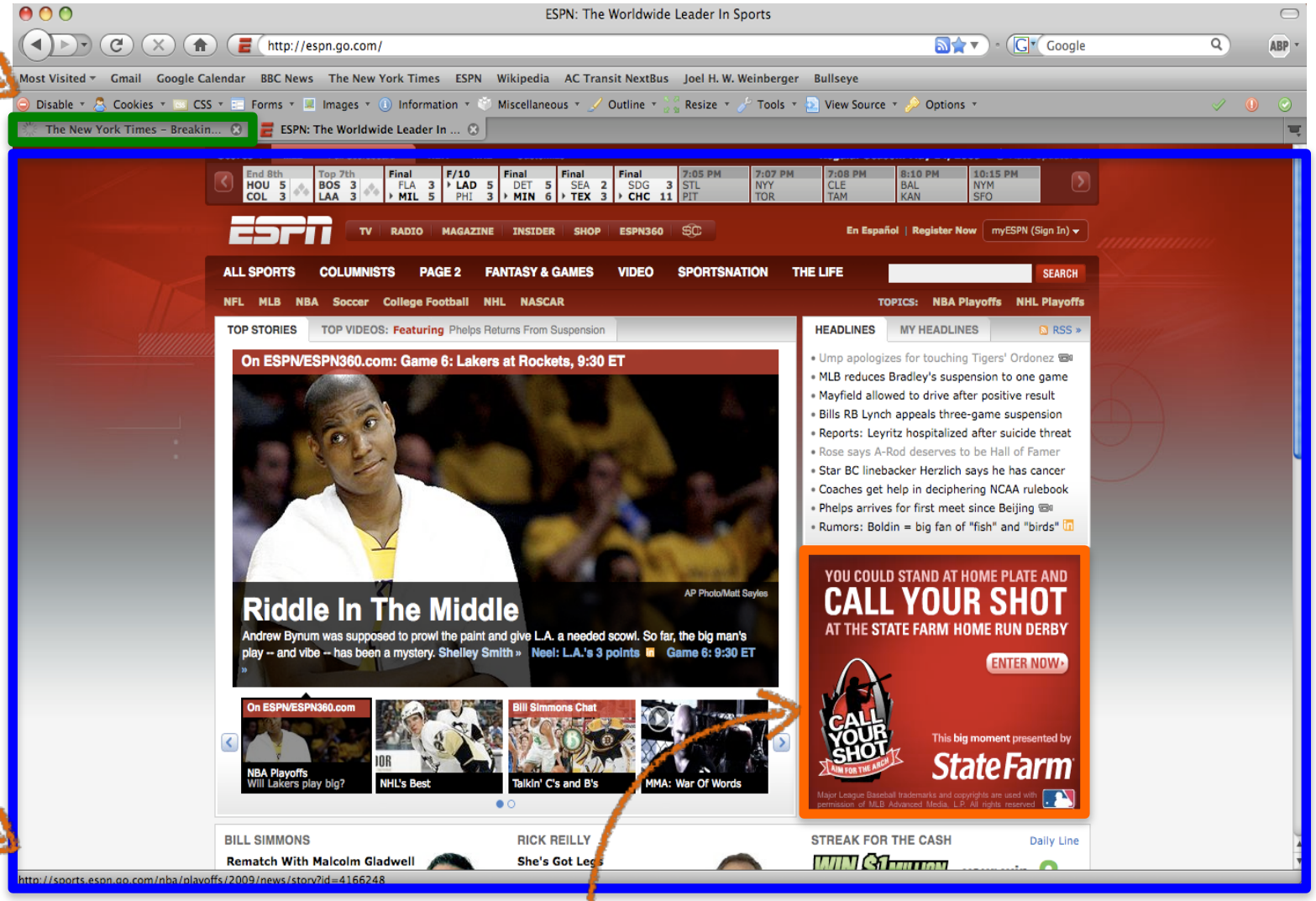
Operating system

- Primitives
  - System calls
  - Processes
  - Disks
- Principals: Users
  - Discretionary access controls
- Vulnerabilities
  - Buffer overflows
  - root exploit
  - …

Browsers

- Primitives
  - Document object model
  - Frames
  - Cookies / local storage
- Principals: Origins
  - Mandatory access controls
- Vulnerabilities
  - Cross-site scripting (XSS)
  - Cross-site request forgery (CSRF)
  - Cache history attacks
  - …

[slide credit: V. Shmatikov, CS380]



JavaScript context 1

JavaScript context 2

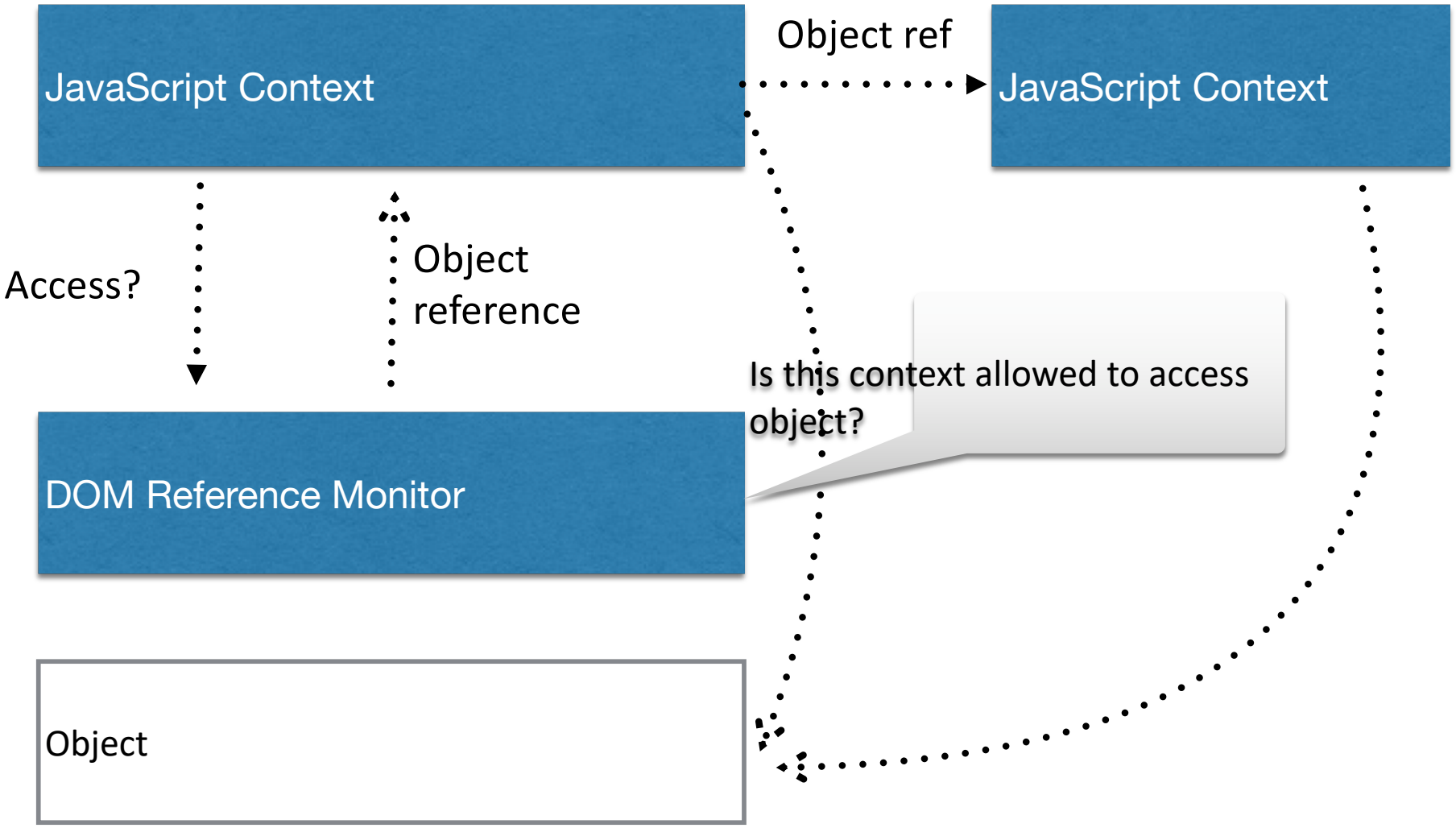
JavaScript context 3

Policy for scripts

- Scripts want to access resources
  - Contact web servers
  - Access cookies
  - Read/write DOM
- How do we know if this is allowed?
  - When requests are to the same company?
  - When requests are to the same web site?
  - When requests are to the same web page?

[slide credit: V. Shmatikov, CS380]

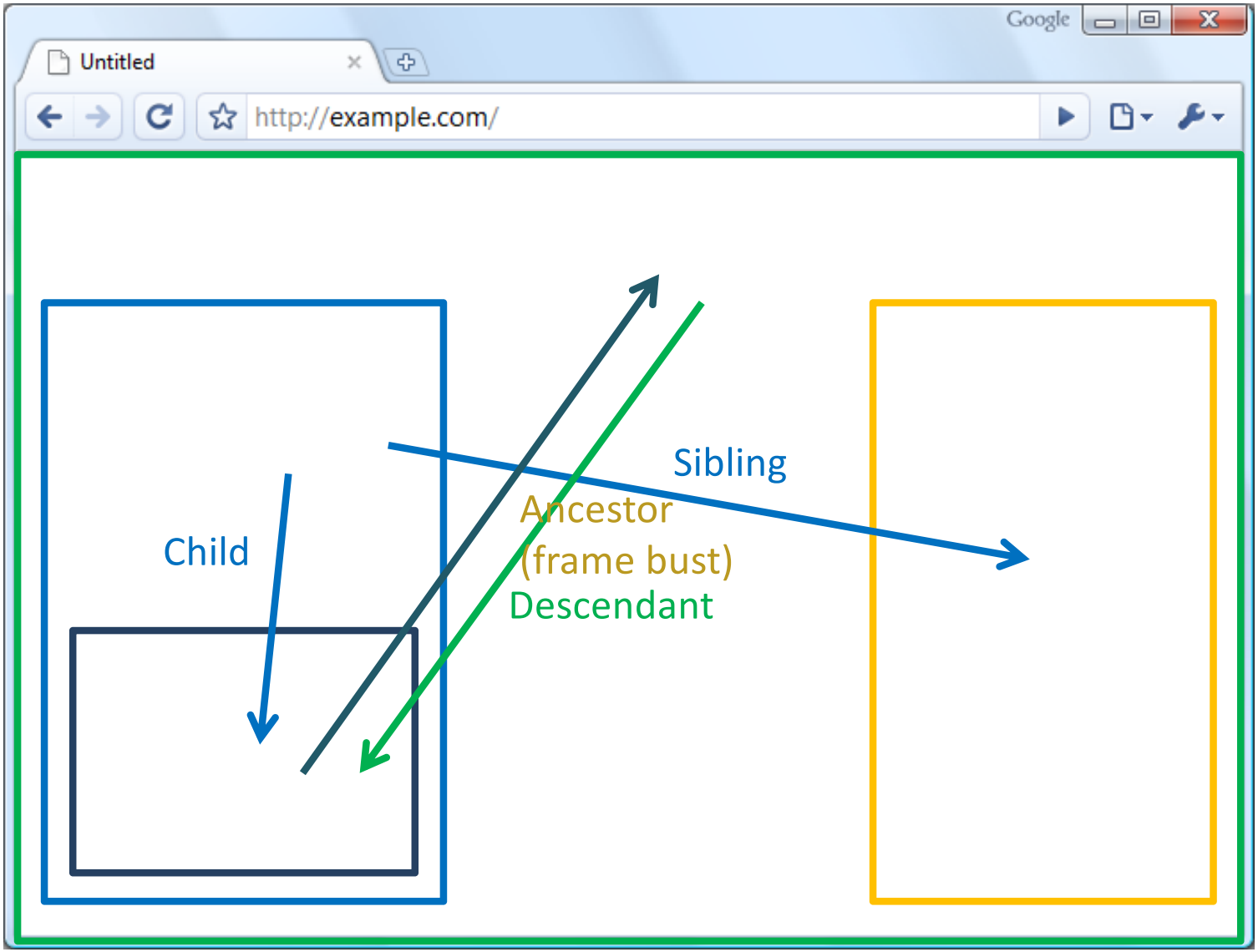
DOM access control



Same-origin policy for scripts

- Each frame of page(s) has an origin
  - protocol://host:port
  - Origin is (protocol,host,port)
- Script in a frame can access its own origin
  - Network access, Read/write DOM, storage (cookies)
  - Content from other frames from same origin
- Frame cannot access data associated with another origin

Frame relationships



frame policies

- `canScript(A,B)`
  - Can frame A execute a script that manipulates arbitrary DOM elements in frame B?
- `canNavigate(A,B)`
  - Can frame A change the origin of content for frame B?
  - `frameB.src = "http://newurl.com/page5.html"`

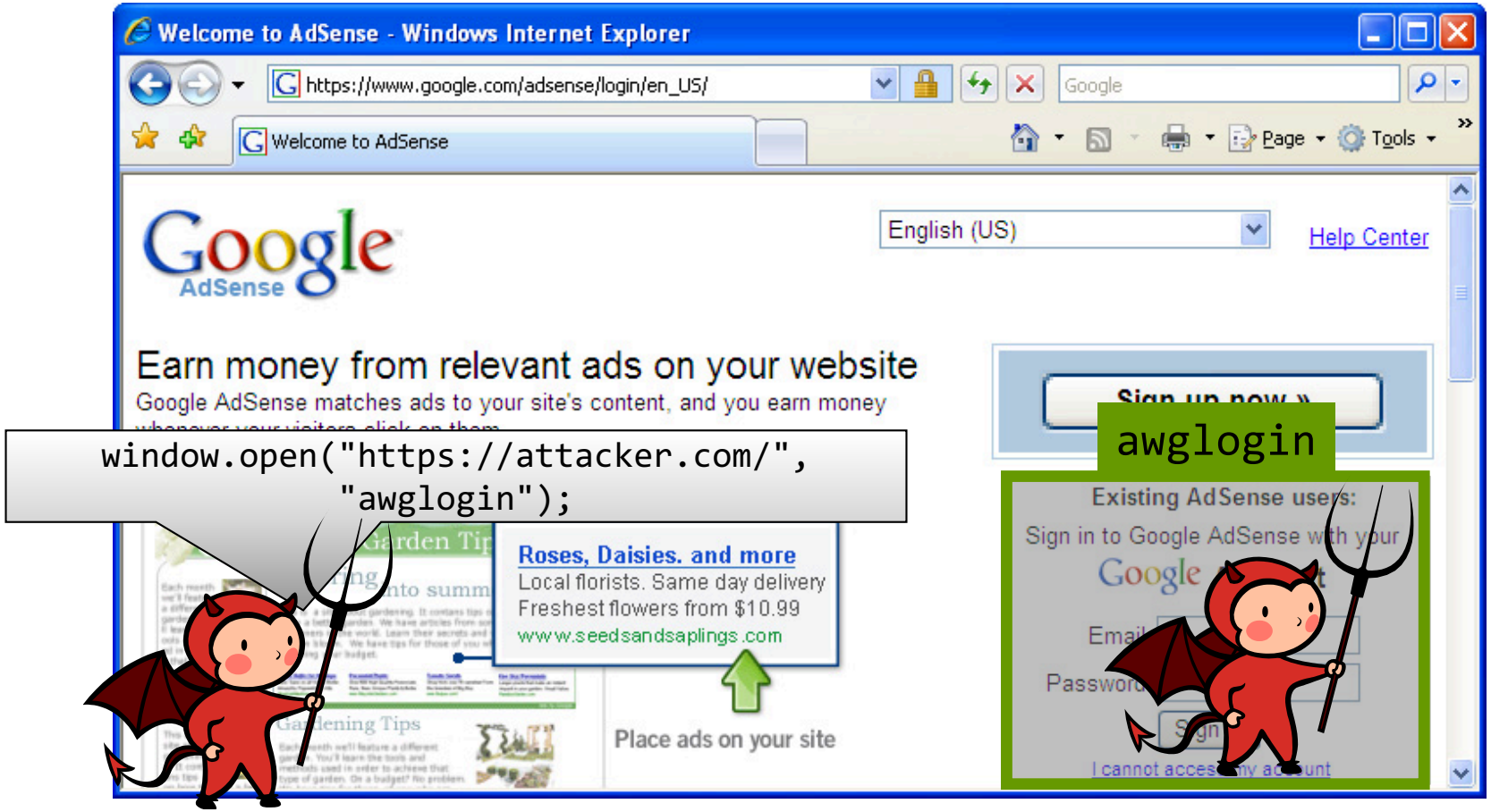
Frame policies

- Permissive
  - any frame can navigate any other frame
- Child
  - only can navigate if you are parent
- Descendent
  - only can navigate if you are ancestor

Which do you think should be used?

Problems with permissive

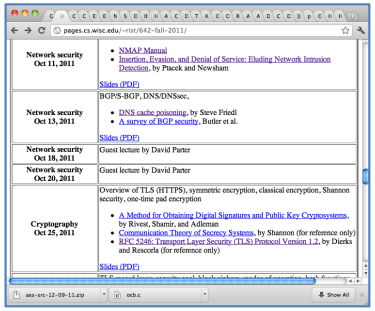
frames['right'].window.location="evil.com/login.html";



General Approach

- A frame can navigate another frame that it owns the pixels for
  - If you delegate some pixels in your frame to another frame, you can make that other frame navigate places
- Why?
  - You could draw anything you want in those pixels anyway

Cookies: Setting/Deleting



GET …

HTTP Header:

Set-cookie: NAME=VALUE ;

domain = (when to send) ;
path = (when to send)

scope

secure = (only send over SSL);

expires = (when expires) ;

HttpOnly

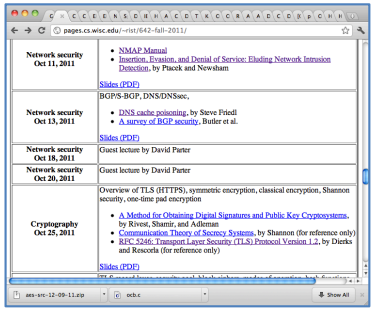
if expires=NULL:
this session only

- Delete cookie by setting “expires” to date in past
- Default scope is domain and path of setting URL
- Client can also set cookies (Javascript)

Cookie scope rules (domain and path)

- Say we are at www.wisc.edu
  - Any non-TLD suffix can be scope:
    - allowed: www.wisc.edu or wisc.edu
    - disallowed: www2.wisc.edu or ucsd.edu
- Path can be set to anything

Cookies: reading by server



GET /url-domain/url-path

Cookie: name=value

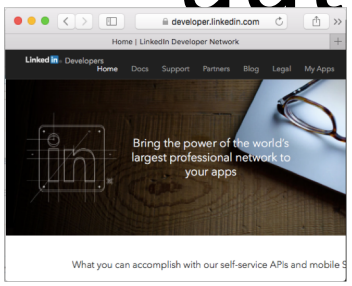
- Browser sends all cookies such that
  - domain scope is suffix of url-domain
  - path is prefix of url-path
  - protocol is HTTPS if cookie marked “secure”

Cookie security issues?

- Cookies have no integrity
  - HTTPS cookies can be overwritten by HTTP cookie (network injection)
  - Malicious clients can modify cookies
    - Shopping cart vulnerabilities
- Scoping rules can be abused
  - blog.example.com can read/set cookies for example.com
- Privacy
  - Cookies can be used to track you around the Internet
- HTTP cookies sent in clear
  - Session hijacking

authentication cookies

website.com



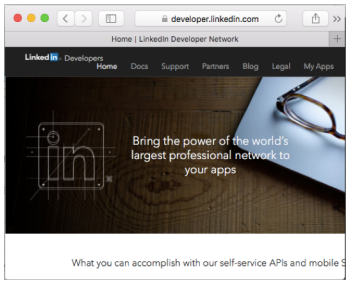
Browser

```
POST /login.html HTTP/1.1
username=user&passwd=pass
```

```
db.users.insert({
  username:"user",
  auth-cookie:981mnd89...,
  login:true,
})
```

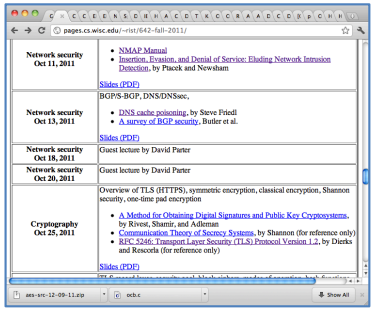
```
HTTP/1.1 200 OK
Set-Cookie:auth=981mndg897asdfd
```

. . . . .




```
GET /index.html HTTP/1.1
Cookie: auth=981mndg897asdfd
```

Session handling and login



GET /index.html

Set-Cookie: AnonSessID=134fds1431

Protocol is HTTPS. Elsewhere just HTTP

POST /login.html?name=bob&pw=12345

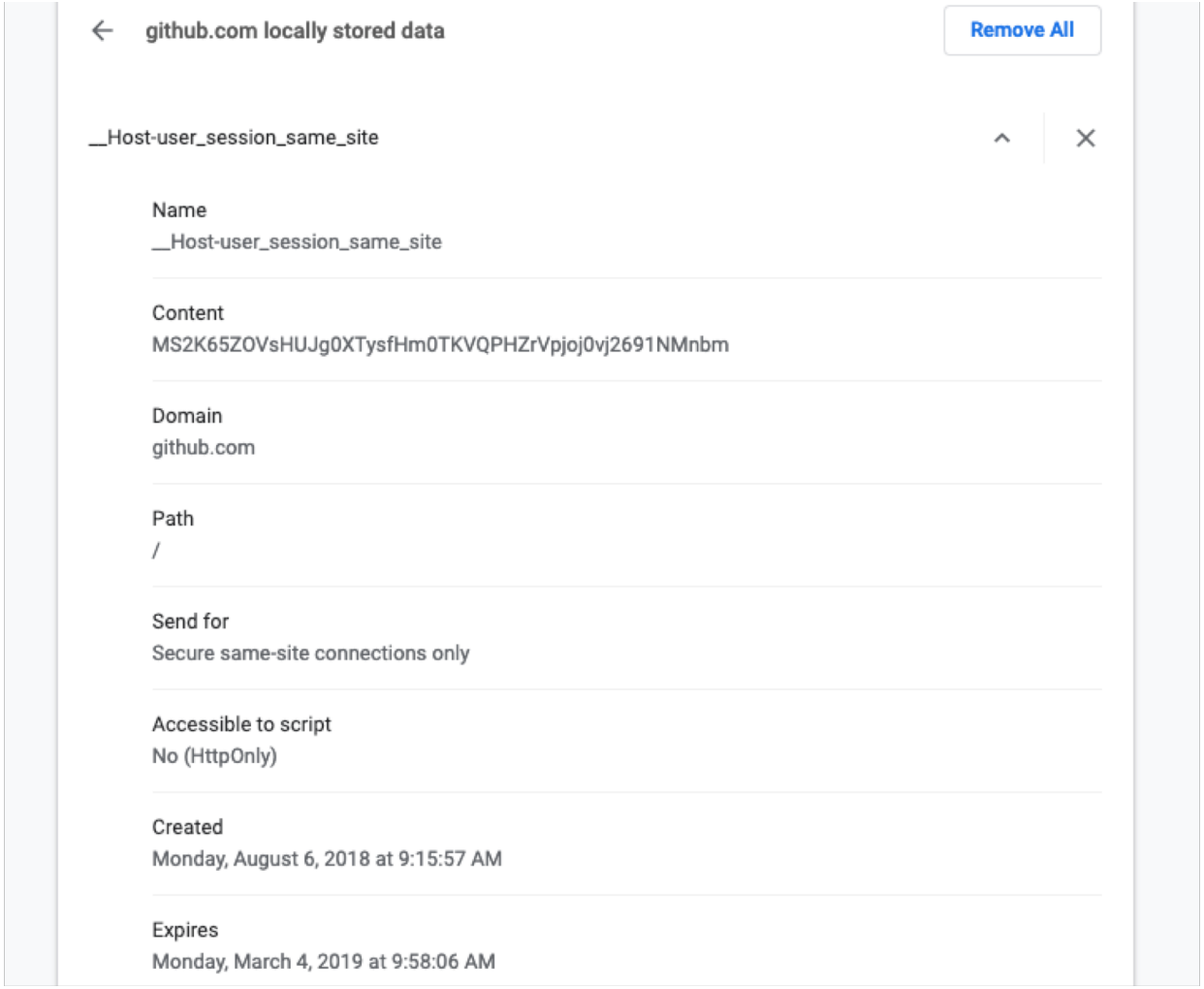
Cookie: AnonSessID=134fds1431

Set-Cookie: SessID=83431Adf

GET /account.html

Cookie: SessID=83431Adf

Cookie example

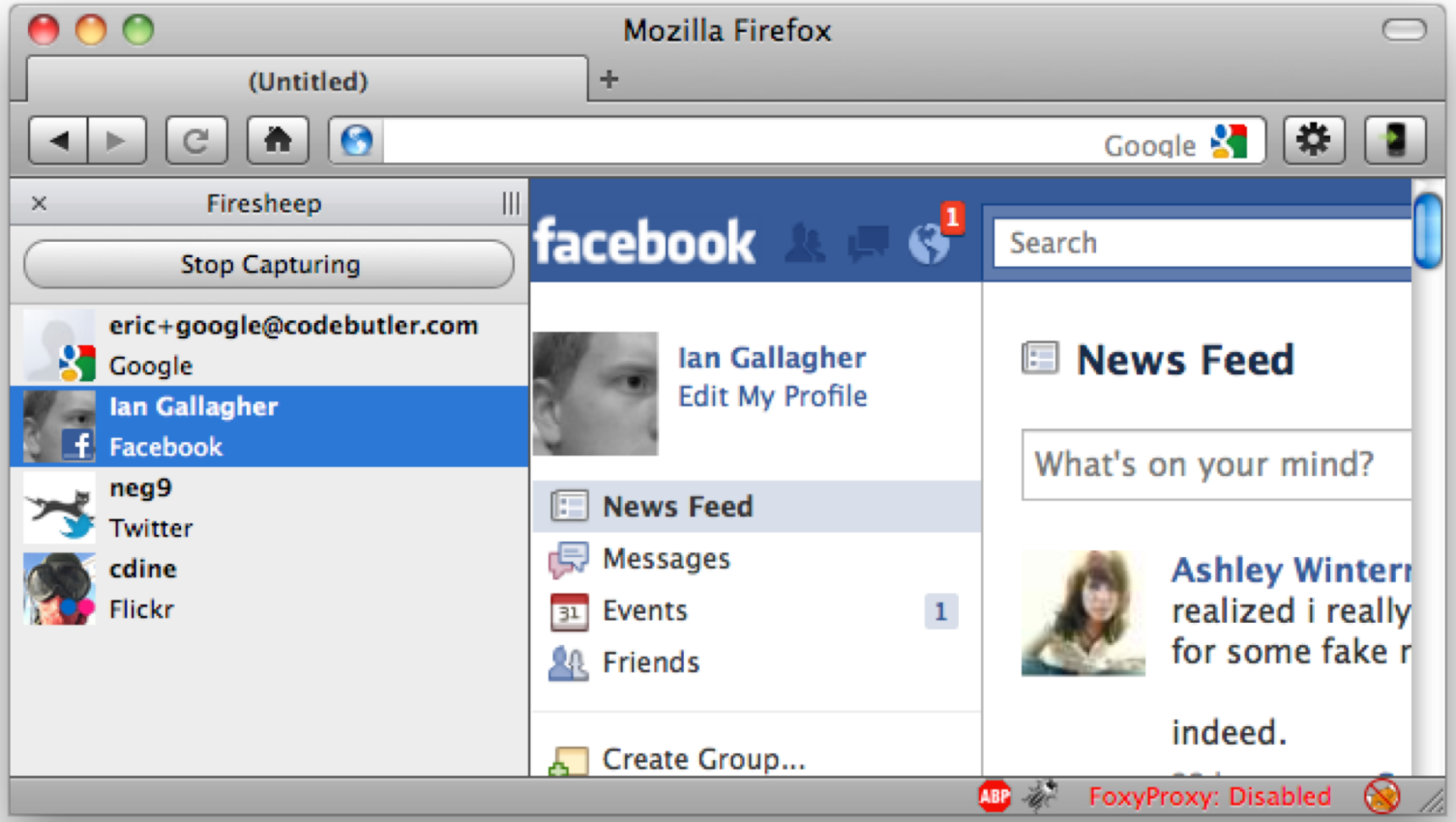


Session Hijacking

Eavesdrop on network

Listen for unencrypted session cookies

Make requests with other's cookies



From http://codebutler.com/firesheep

Towards preventing hijacking

- Use encryption when setting session cookies
- SessID = Enc(K,info) where :
  - K is server-side secret key
  - Enc is Encrypt-then-MAC encryption scheme
  - info contains:
    - user id
    - expiration time
    - other data
- Server should record if user logs out
- Does this prevent Firesheep hijacking?
  - No
  - include in data machine-specific information
  - turn on HTTPS always