

2550 Intro to cybersecurity

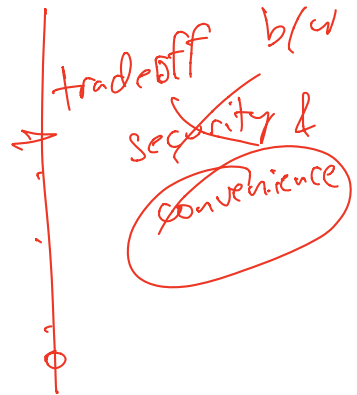
L23: Web Exploits

~
- 11:30
faculty
candidate
talk.

- P6 released
this
weekend
CTF

Today's plan

- Web Exploits
- HTTP works
- Browser model
- Security attacks against
 - cookie attacks
 - cross site request forgery
 - cross site scripting



HyperText Transfer Protocol

0.9 Tim Berners Lee 1991

1.0 1996

1.1 1999 <http://tools.ietf.org/html/rfc2616>

*message
exchange
format*



HyperText Transfer Protocol

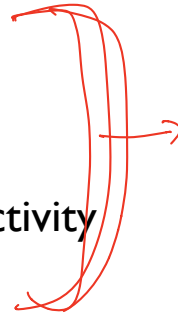
0.9 Tim Berners Lee 1991

1.1 1996

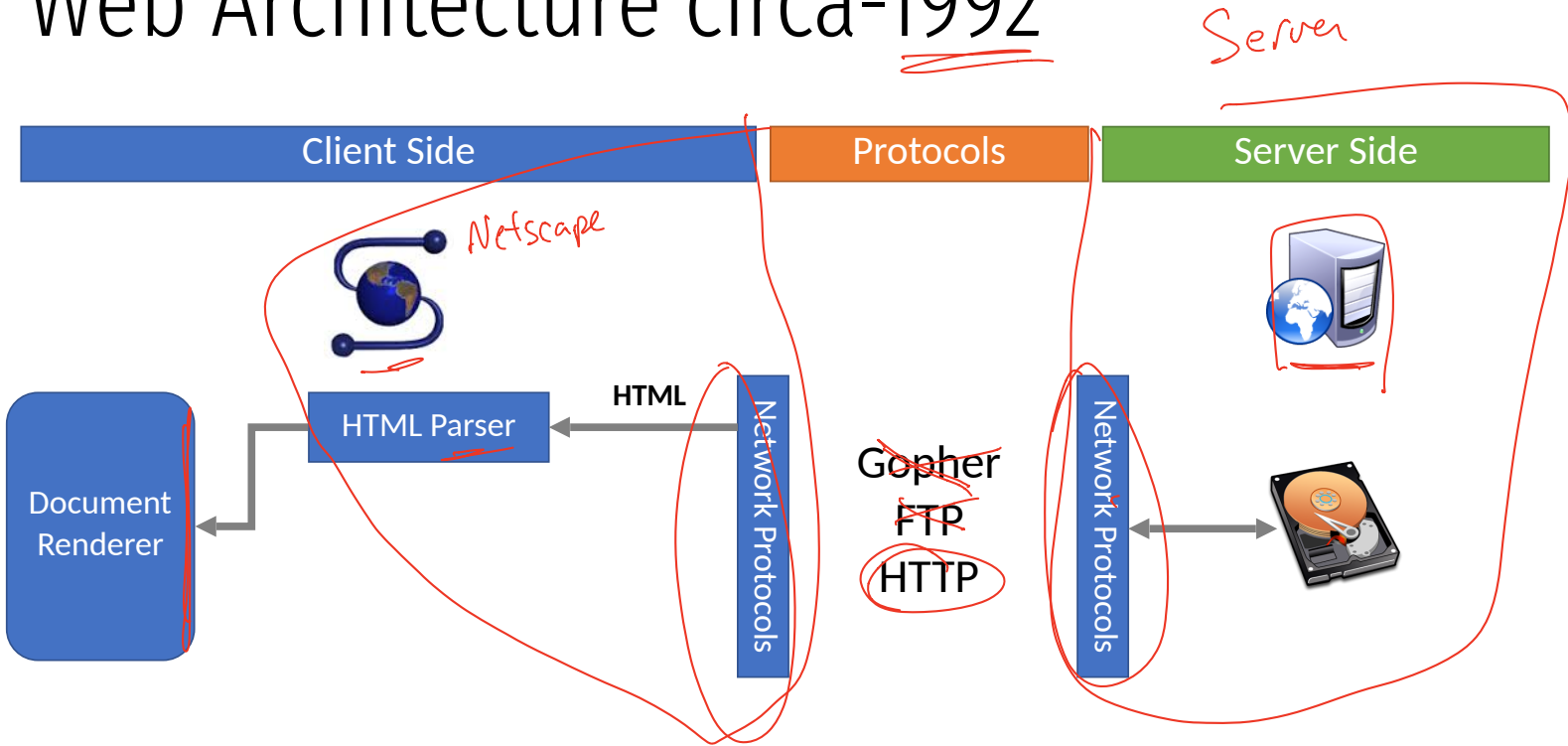
1.1 1999 <http://tools.ietf.org/html/rfc2616>

Stateless

Each request is independent of all other activity



Web Architecture circa-1992



Request/Response

http

curl -i nytimes.com

```
* Trying 151.101.193.164 ...
* TCP_NODELAY set
* Connected to nytimes.com (151.101.193.164) port 80 (#0)
> GET / HTTP/1.1
> Host: nytimes.com
> User-Agent: curl/7.64.1
> Accept: */*
>
```

Request

Response

```
< HTTP/1.1 301 Moved Permanently
< Server: Varnish
< Retry-After: 0
< Content-Length: 0
< Location: https://www.nytimes.com/
< Accept-Ranges: bytes
< Date: Fri, 03 Apr 2020 08:25:31 GMT
< X-Served-By: cache-bos4641-BOS
< X-Cache: HIT
< X-Cache-Hits: 0
< Set-Cookie: nyt-gdpr=0; Expires=Fri, 03 Apr 2020 14:25:31 GMT; Path=/; Domain=.nytimes.com
< x-gdpr: 0
< X-Frame-Options: DENY
< Connection: close
< X-API-Version: F-0
```

Request

GET / HTTP/1.1

Host: yahoo.com

Connection: keep-alive

User-Agent: Mozilla/5.0 (iPad; CPU OS 5_0 like Mac OS X) AppleWebKit/534.46 (KHTML, like Gecko) Version/5.1
Mobile/9A334 Safari/7534.48.3

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8

Accept-Encoding: gzip,deflate,sdch

Accept-Language: en-US,en;q=0.8

Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.3

Cookie: YLS=v=....

Response

HTTP/1.1 302 Found

Date: Tue, 18 Sep 2012 17:47:21 GMT

P3P: policyref="<http://info.yahoo.com/w3c/p3p.xml>", CP="CAO DSP COR CUR ADM DEV TAI PSA PSD IVAi IVDi CO
TELo OTPi OUR DELi SAMi OTRi UNRi PUBi IND PHY ONL UNI PUR FIN COM NAV INT DEM CNT STA POL HEA
PRE LOC GOV"

Cache-Control: private

X-Frame-Options: SAMEORIGIN

Set-Cookie: IU=deleted; expires=Mon, 19-Sep-2011 17:47:20 GMT; path=/; domain=.yahoo.com

Set-Cookie: fpc=d=WmdZ6DzTnE...JAS04jxkD expires=Wed, 18-Sep-2013 17:47:21 GMT; path=/;

domain=www.yahoo.com

Location: <http://www.yahoo.com/tablet/>

Vary: Accept-Encoding

Content-Type: text/html; charset=utf-8

Age: 0

Transfer-Encoding: chunked

Connection: keep-alive

Server: YTS/1.20.10

Modern response

HTTP/2 200 OK

server: nginx

content-type: text/html; charset=utf-8

x-nyt-data-last-modified: Fri, 03 Apr 2020 13:06:36 GMT

last-modified: Fri, 03 Apr 2020 13:06:36 GMT

x-pagetype: vi-homepage

x-vi-compatibility: Compatible

x-xss-protection: 1; mode=block

x-content-type-options: nosniff

content-encoding: gzip

cache-control: s-maxage=30,no-cache

x-nyt-route: homepage

x-origin-time: 2020-04-03 13:07:39 UTC

accept-ranges: bytes

date: Fri, 03 Apr 2020 13:07:39 GMT

age: 31

x-served-by: cache-lga21966-LGA, cache-bos4624-BOS

x-cache: HIT, MISS

x-cache-hits: 5, 0

x-timer: S1585919260.727513,VS0,VE12

vary: Accept-Encoding, Fastly-SSL

set-cookie: nyt-a=jRLIskwL3RTL1Zzn3ifKyg; Expires=Sat, 03 Apr 2021 13:07:39 GMT; Path=/; Domain=.nytimes.com; SameSite=none; Secure

set-cookie: nyt-gdpr=0; Expires=Fri, 03 Apr 2020 19:07:39 GMT; Path=/; Domain=.nytimes.com

x-gdpr: 0

set-cookie: nyt-purr=cfhhcfh; Expires=Sat, 03 Apr 2021 13:07:39 GMT; Path=/; Domain=.nytimes.com

set-cookie: nyt-geo=US; Expires=Fri, 03 Apr 2020 19:07:39 GMT; Path=/; Domain=.nytimes.com

x-frame-options: DENY

x-api-version: F-F-VI

content-security-policy: default-src data: 'unsafe-inline' 'unsafe-eval' https;; script-src data: 'unsafe-inline' 'unsafe-eval' https: blob;; style-src data:

'unsafe-inline' https;; img-src data: https: blob;; font-src data: https;; connect-src https: wss: blob;; media-src https: blob;; object-src https;; child-src

https: data: blob;; form-action https;; block-all-mixed-content;

content-length: 174470

X-Firefox-Spdy: h2

HTTP Request Methods

Most HTTP requests

Verb	Description
<u>GET</u>	Retrieve resource at a given path
<u>POST</u>	Submit data to a given path, might create resources as new paths
<u>HEAD</u>	Identical to a GET, but response omits body
<u>PUT</u>	Submit data to a given path, creating resource if it exists or modifying existing resource at that path
<u>DELETE</u>	Deletes resource at a given path
<u>TRACE</u>	Echoes request
<u>OPTIONS</u>	Returns supported HTTP methods given a path
<u>CONNECT</u>	Creates a tunnel to a given network location

HTTP Response Status Codes

- 3 digit response codes
 - 1XX - informational
 - 2XX - success
 - 200 OK
 - 3XX - redirection
 - 301 Moved Permanently
 - 303 Moved Temporarily
 - 304 Not Modified
 - 4XX - client error
 - 404 Not Found
 - 5XX - server error
 - 505 HTTP Version Not Supported

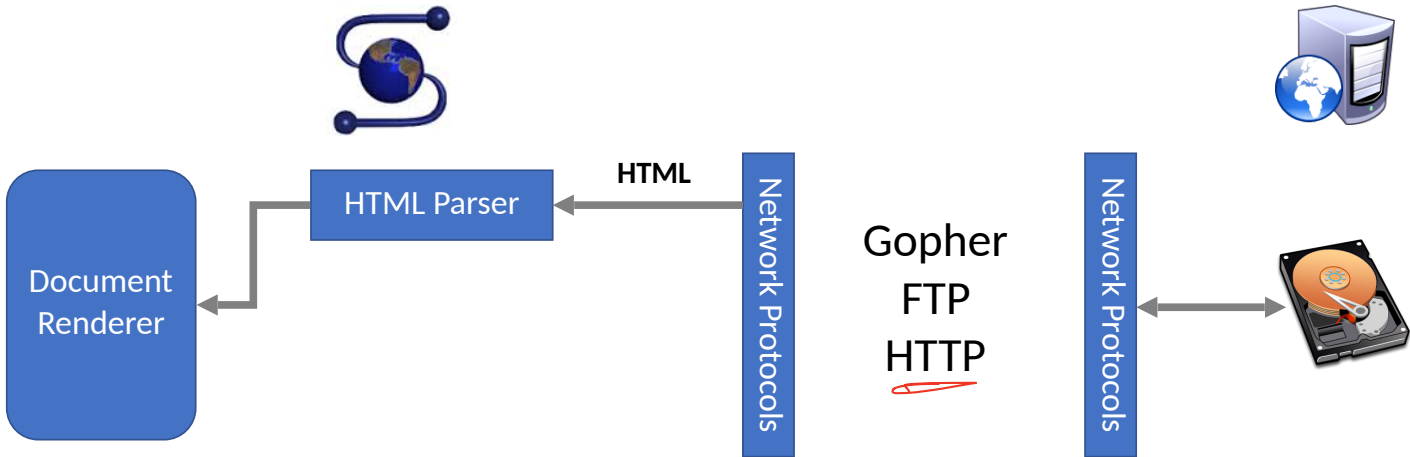
302

Web Architecture circa-1992

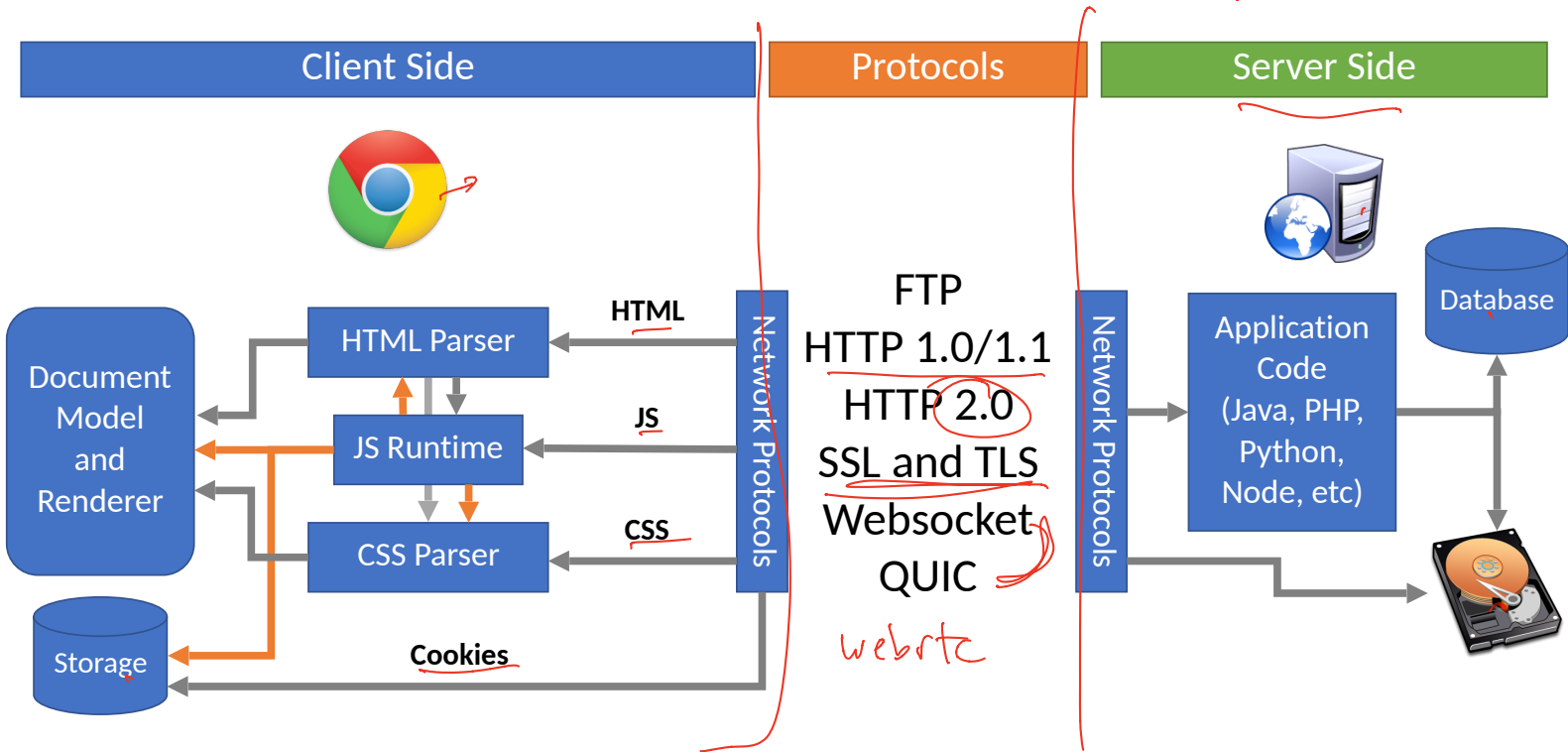
Client Side

Protocols

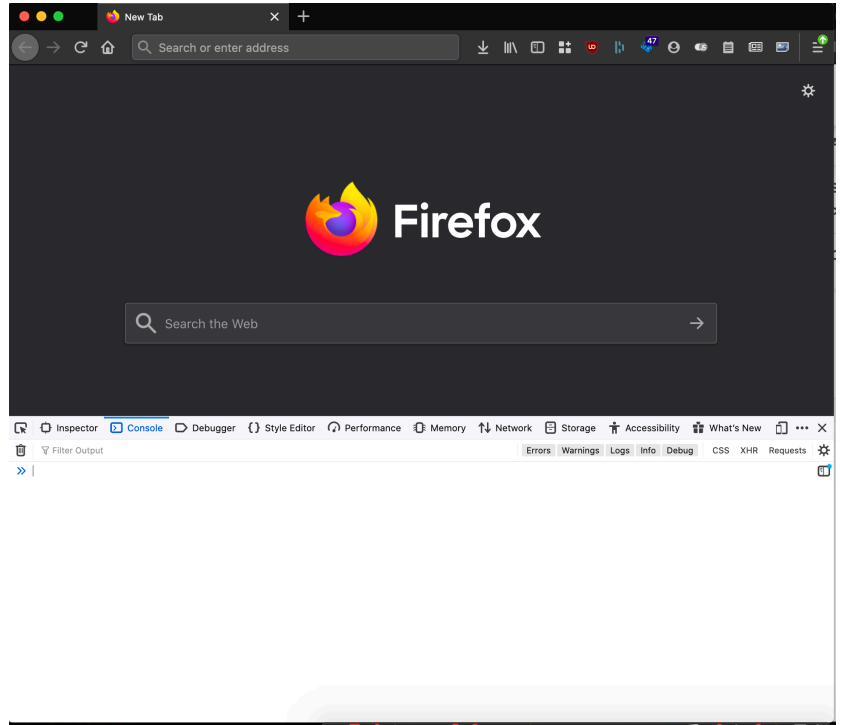
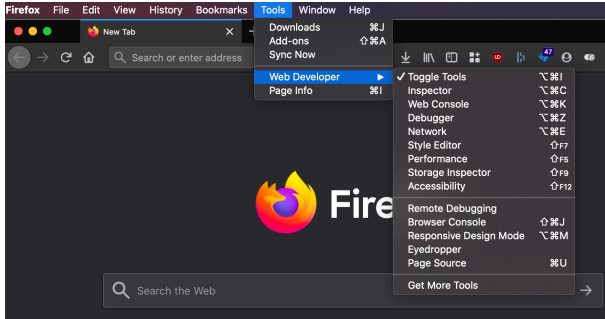
Server Side



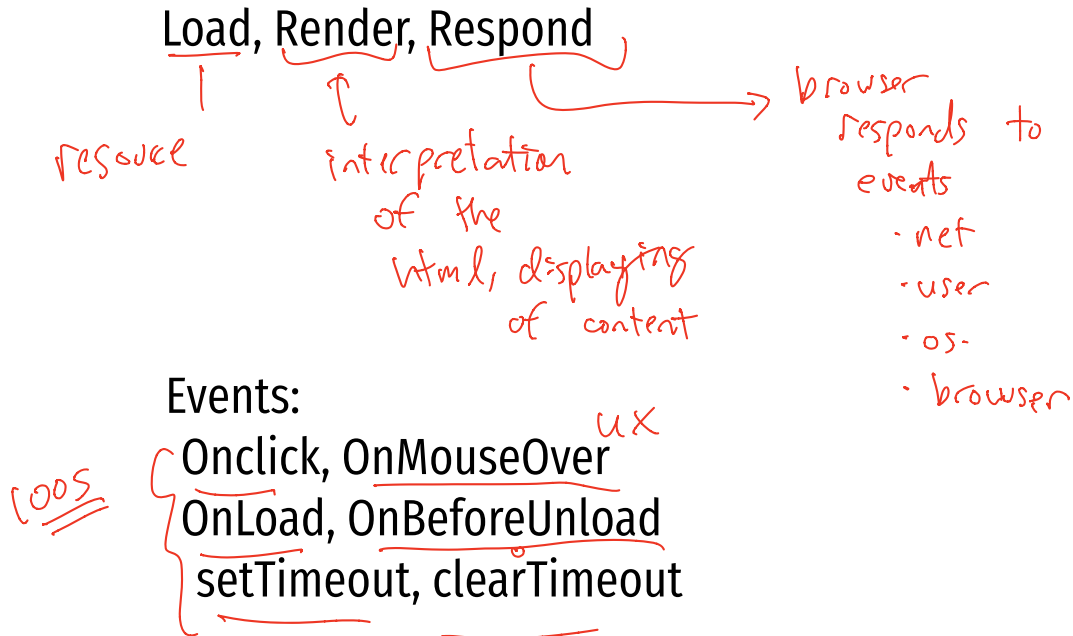
Web Architecture circa-2018



Console



Browser Execution Model



Web Pages (HTML)

- Multiple (typically small) objects per page
 - E.g., each image, JS, CSS, etc. downloaded separately
- Single page can have 100s of HTTP transactions!
 - File sizes are heavy-tailed
 - Most transfers/objects very small

markup language

```
<!doctype html>
```

```
<html>
<head>
  <title>Hello World</title>
  <script src="../../jquery.js"></script>
</head>
<body>
  <h1>Hello World</h1>
  </img>
  <p>
    I am 12 and what is
    <a href="wierd_thing.html">this</
a>?
  </p>
  </img>
</body>
</html>
```


Web Pages (HTML)

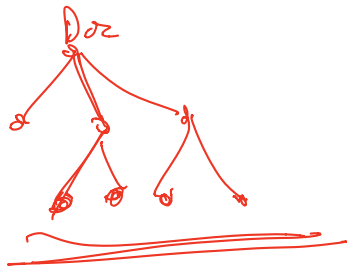
- Multiple (typically small) objects per page
 - E.g., each image, JS, CSS, etc. downloaded separately
- Single page can have 100s of HTTP transactions!
 - File sizes are heavy-tailed
 - Most transfers/objects very small

```
<!doctype html>

<html>
<head>
  <title>Hello World</title>
  <script src="../../jquery.js"></script>
</head>
<body>
  <h1>Hello World</h1>
  </img>
  <p>
    I am 12 and what is
    <a href="wierd_thing.html">this</
a>?
  </p>
  </img>
</body>
</html>
```

4 total objects:
1 HTML,
1 JavaScript,
2 images

Document Object Model (DOM)



- A web page in HTML is structured data.
DOM provides an abstraction of this hierarchy.

RWX

Properties: document.alinkColor, document.forms[]

Browser objects: window, document, frames, history

A webpage can modify itself in clever ways using the DOM.

like a program

What About JavaScript?

- Javascript enables dynamic inclusion of objects

```
document.write('`

Security issue?

`"GET /<secret info>"`

? `<secret info>`

— server that responds

400  
"error  
&  
1ms

or

an `200 OK??` at `imagelibrary.com`  
sending network data.  
learns `<secret info>`  
~ 10s ms.



# Example attack: port scanning

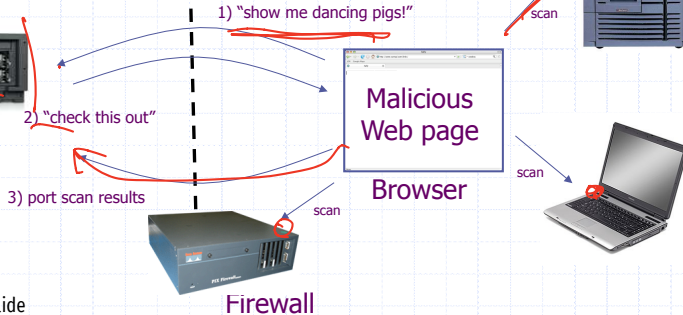
Security consequence

## 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

*malicious*  
Server



Credit: John Mitchell for slide

Firewall

*neu.jpg*

*[192.168.0.1:80] / foo.jpg*

*0.2*

*0.4*

*:*

*0.25%*

# Security: Isolation

GOAL

Safe to visit an evil site:



Safe to browse many sites concurrently:



Safe to delegate:



iframe

# Windows, Frames, Origins



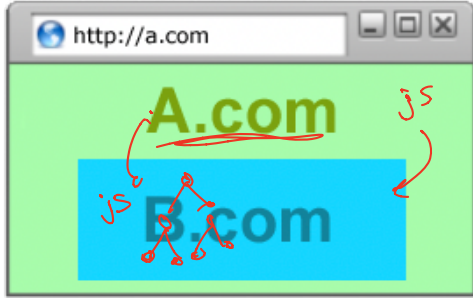
Each page of a frame has an origin

*important concepts*

*for isolation:*

Frames can access resources of its own origin.

# Windows, Frames, Origins



Each page of a frame has an origin

Frames can access resources of its own origin.

Q: can frame A execute javascript to manipulate DOM elements of B?

# Same origin policy

(SOP)

http https

nytimes.com

80

Origin: scheme + host + port

Pages with different origins should be “**isolated**” in some way.

# Same Origin Policy

Origin = <protocol, hostname, port>

- The Same-Origin Policy (SOP) states that subjects from one origin cannot access objects from another origin
- This applies to JavaScript
  - JS from origin  $D$  cannot access objects from origin  $D'$ 
    - E.g. the iframe example
  - However, JS included in  $D$  can access all objects in  $D$ 
    - E.g. `<script src='https://code.jquery.com/jquery-2.1.3.min.js'></script>`

# huge Except for:

<img> → security attacks possible.

<form> submit to different origins

<script> - libraries

<jsonp> →

# Same Origin Policy

- The Same-Origin Policy (SOP) states that **subjects** from one origin cannot access **objects** from another origin
  - SOP is the basis of classic web security
  - Some exceptions to this policy (unfortunately)
  - SOP has been relaxed over time to make controlled sharing easier
- In the case of cookies
  - Domains are the origins
  - Cookies are the subjects



# Mixing Origins

```
<html>
<head></head>
<body>
 <p>This is my page.</p>
 <script>var password = 's3cr3t';</script>
 (<iframe id='goog' src='http://
google.com'></iframe>)
</body>
</html>
```

This is my page.

The Google logo is displayed in its characteristic multi-colored font (blue, red, yellow, blue, green, red) with a trademark symbol.

Google Search

I'm Feeling Lucky

# Mixing Origins

skeleton —

```
<html>
<head></head>
<body>
 <p>This is my page.</p>
 <script>var password = 's3cr3t';</script>
 <iframe id='goog' src='http://
 google.com'></iframe>
</body>
</html>
```

Can JS from google.com read *password*?

This is my page.

The Google logo is displayed in its characteristic multi-colored font (blue, red, yellow, blue, green, red) with a trademark symbol.

Google Search

I'm Feeling Lucky

# Mixing Origins

```
<html>
<head></head>
<body>
 <p>This is my page.</p>
 <script>var password = 's3cr3t';</script>
 <iframe id='goog' src='http://
google.com'></iframe>
</body>
</html>
```

Can JS from google.com read *password*?

Can JS in the main context do the following:  
*document.getElementById('goog').cookie*?

This is my page.

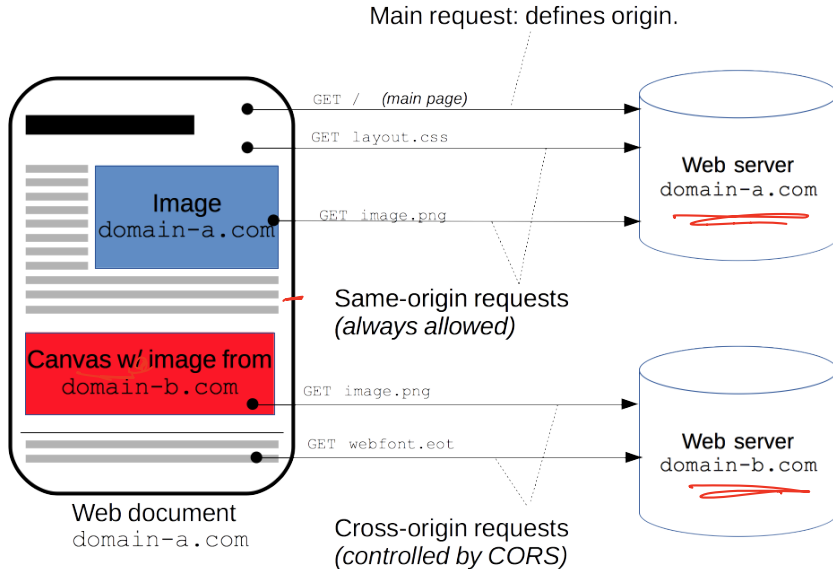
The Google logo is displayed in its characteristic multi-colored font (blue, red, yellow, blue, green, red) with a trademark symbol.

# Another exception: CORS

Cross-origin resource sharing

Access-control-allow-origin: <list of domains>

# Cross-Origin Resource Sharing (CORS)



**Cross-Origin Resource Sharing (CORS)** is a mechanism that uses additional [HTTP](#) headers to tell browsers to give a web application running at one [origin](#), access to selected resources from a different origin. A web application executes a cross-origin HTTP request when it requests a resource that has a different origin (domain, protocol, or port) from its own.

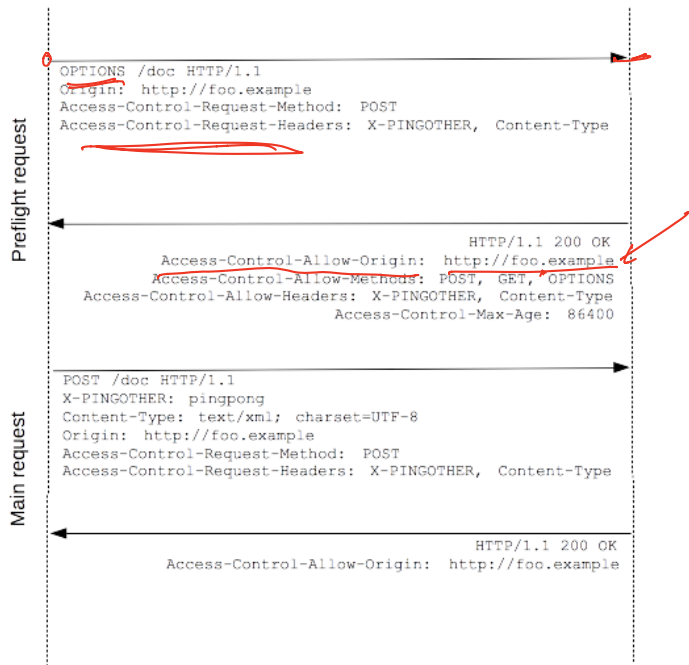
<https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS>

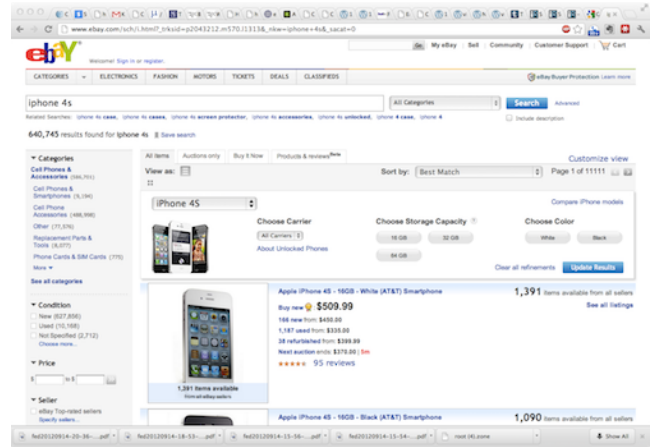
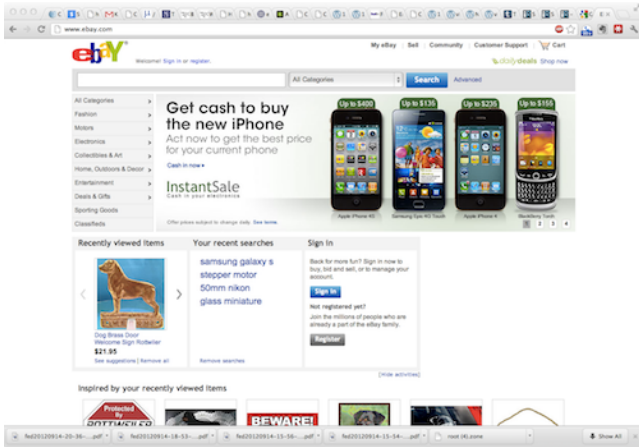
# Pre-flighted request

→ requires

authenticated servers (https)

Client Server





As the user navigates a website, STATE information is generated.

Eg: Authentication information for a session.

**Issue:** How to manage state  
information over **HTTP?**



# Keep state information in the URL?

## FatBrain URL authenticator

Start: <https://www.fatbrain.com/HelpAccount.asp?>

~~t=0&p1=attacker@mit.edu&p2=540555758~~

Try: <https://www.fatbrain.com/HelpAccount.asp?> ✘

t=0&p1=victim@mit.edu&p2=~~540555757~~

Target: <https://www.fatbrain.com/HelpAccount.asp?>

t=0&p1=victim@mit.edu&p2=~~540555752~~

# Storing state in FORMs

```
<FORM METHOD=POST
```

```
 ACTION="http://www.dansie.net/cgi-bin/scripts/cart.pl">
```

```
 Black Leather purse with leather straps
Price: $20.00

```

```
<INPUT TYPE=HIDDEN NAME=name VALUE="Black leather purse">
```

```
<INPUT TYPE=HIDDEN NAME=price VALUE="20.00">
```

```
<INPUT TYPE=HIDDEN NAME=sh VALUE="1">
```

```
<INPUT TYPE=HIDDEN NAME=img VALUE="purse.jpg">
```

```
<INPUT TYPE=HIDDEN NAME=custom1 VALUE="Black leather purse with
leather straps">
```

```
 <INPUT TYPE=SUBMIT NAME="add" VALUE="Put in Shopping Cart">
```

```
</FORM>
```

# Cookies - origin.

- Introduced in 1994, cookies are a basic mechanism for persistent state
  - Allows services to store a small amount of data at the client (usually ~4K)
  - Often used for identification, authentication, user tracking
- Attributes
  - Domain and path restricts resources browser will send cookies to
  - Expiration sets how long cookie is valid
  - Additional security restrictions (added much later): HttpOnly, Secure
- Manipulated by Set-Cookie and Cookie headers

# Cookie Example

Client Side



GET /login\_form.html HTTP/1.1

HTTP/1.1 200 OK

Server Side



# Cookie Example

Client Side



Server Side



GET /login\_form.html HTTP/1.1

HTTP/1.1 200 OK

POST /cgi/login.sh HTTP/1.1

HTTP/1.1 302 Found

Set-Cookie: session=FhizeVYSkS7X2K

*http*

*origin*

- If credentials are correct:
1. Generate a random token
  2. Store token in the database
  3. Send token to the client

# Cookie Example

Client Side



Store the cookie

GET /login\_form.html HTTP/1.1

HTTP/1.1 200 OK

POST /cgi/login.sh HTTP/1.1

HTTP/1.1 302 Found

**Set-Cookie: session=FhizeVYSkS7X2K**

Server Side



If credentials are correct:

1. Generate a random token
2. Store token in the database
3. Send token to the client

# Cookie Example

Client Side



Store the cookie

Server Side



GET /login\_form.html HTTP/1.1

HTTP/1.1 200 OK

POST /cgi/login.sh HTTP/1.1

HTTP/1.1 302 Found

Set-Cookie: session=FhizeVYskS7X2K

*Same origin*  
GET /private\_data.html HTTP/1.1

Cookie: session=FhizeVYskS7X2K;

HTTP/1.1 200 OK

If credentials are correct:

1. Generate a random token
2. Store token in the database
3. Send token to the client

1. Check token in the database
2. If it exists, user is authenticated

# Cookie Example

Client Side



Store the cookie

Server Side



GET /login\_form.html HTTP/1.1

HTTP/1.1 200 OK

POST /cgi/login.sh HTTP/1.1

HTTP/1.1 302 Found  
Set-Cookie: session=FhizeVYskS7X2K

GET /private\_data.html HTTP/1.1  
Cookie: session=FhizeVYskS7X2K;

HTTP/1.1 200 OK

GET /my\_files.html HTTP/1.  
Cookie: session=FhizeVYskS7X2K;

If credentials are correct:

1. Generate a random token
2. Store token in the database
3. Send token to the client

1. Check token in the database
2. If it exists, user is authenticated



# Managing State

- Each origin may set cookies
- Objects from embedded resources may also set cookies

```
</
img>
```

cookies are also sent for every network request.

fb pixel

# Managing State

- Each origin may set cookies
  - Objects from embedded resources may also set cookies

```
</
img>
```

- When the browser sends an HTTP request to origin *D*, which cookies are included?

# Managing State

- Each origin may set cookies
  - Objects from embedded resources may also set cookies

```

```

- When the browser sends an HTTP request to origin  $D$ , which cookies are included?
  - Only cookies for origin  $D$  that obey the specific path constraints

# Managing State

- Each origin may set cookies
  - Objects from embedded resources may also set cookies

```

```

- When the browser sends an HTTP request to origin  $D$ , which cookies are included?
  - Only cookies for origin  $D$  that obey the specific path constraints

# Managing State

- Each origin may set cookies
  - Objects from embedded resources may also set cookies

```

```

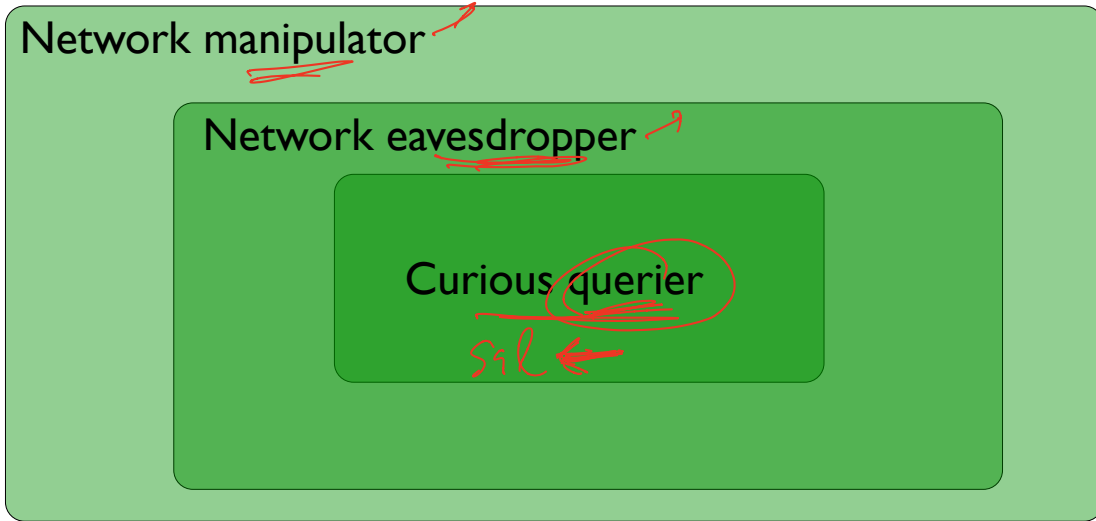
- When the browser sends an HTTP request to origin  $D$ , which cookies are included?
  - Only cookies for origin  $D$  that obey the specific path constraints

- Origin consists of <domain, path>  
*fb*

Site A and Site B have different COOKIE jars.

Javascript from A cannot read/write DOM/cookie/state from B.

# Attacker Model



# Cookie



*WSJ*  
POST /wp-login.php HTTP/1.1



HTTP/1.1 200



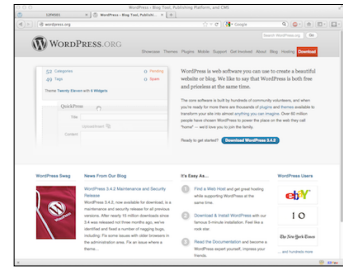
Set-cookie: (X)

GET /admin.php HTTP/1.1



cookie: (X)

website



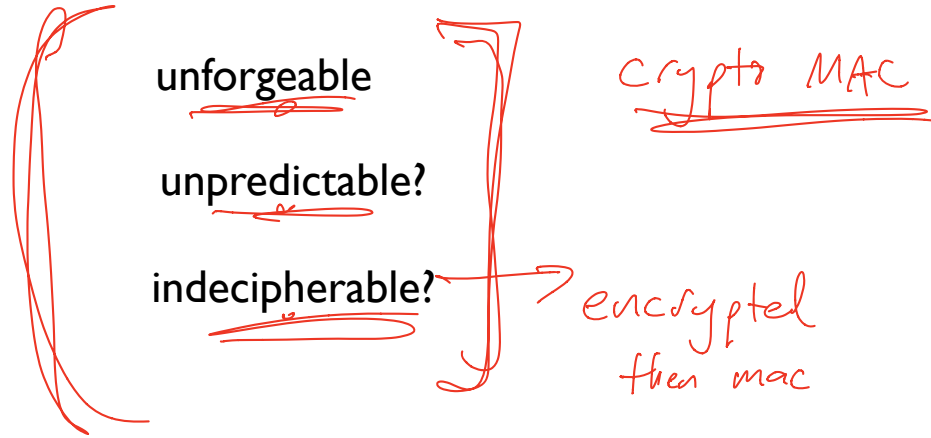
*X "go for admin"*



Set-cookie: .X.

cookie: .X.

# Properties that X should have:



Use a Message Authentication Code (MAC) for this purpose.

Do not attempt to create your own homebrew version.

# WSJ.com analysis

---

- Design: cookie = { user, MAC<sub>k</sub> (user) }
- Reality: cookie =  
user + UNIX-crypt ( user + server secret )  
16

# WSJ.com analysis cont.

---

username	crypt() Output	Authenticator cookie
<u>bitdiddl</u>	<u>MaRdw2J1h6Lfc</u>	<u>bitdiddlMaRdw2J1h6Lfc</u>
bitdiddle	<u>MaRdw2J1h6Lfc</u>	<u>bitdiddleMaRdw2J1h6Lfc</u>

# WSJ.com analysis cont.

---

username	crypt() Output	Authenticator cookie
bitdiddl	MaRdw2J1h6Lfc	bitdiddlMaRdw2J1h6Lfc
bitdiddle	MaRdw2J1h6Lfc	<u>bitdiddle</u> MaRdw2J1h6Lfc

↑  
crypt only reads the first 8 characters of its input

# How to recover WSJ's secret key?

cookie is            USER + crypt(USER + secret key)

8 characters, 128 ascii symbols,

$$128^8 = 72057594037927936$$

Too many guesses for one life time.

# Key peeling, char by char.

username

ABCDEFGH

ABCDEFGG

input to crypt

ABCDEFGH

ABCDEFGG**A**

ABCDEFGG**B**

ABCDEFGG**C**

...

ABCDEFGG**M**

MA

MA MB ... Ma

check website

ok ✓

fail ✓

fail

fail

OK ✓

Embedding state information into a cookie or form.

State, Expiration,  $\text{MAC}_{\text{server secret}}(\text{State}, \text{Expiration})$



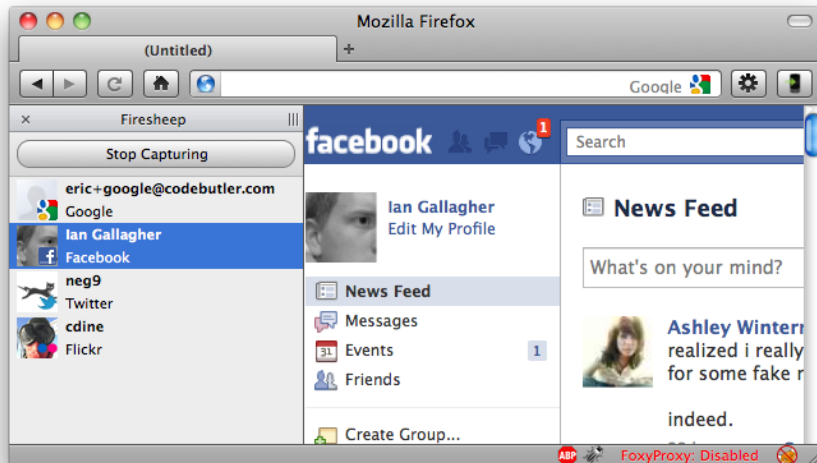
# Session Hijacking

before  
https

If cookies are used to maintain login sessions...



# Firesheep [2010]



# Third-party cookies, tracking

Visit A.com first.



Set cookie A=1

B: 2

# Third-party cookies, tracking

Visit A.com first.



Visit c.com next.



Cookies: {a.com: 1, b.com: 2}

set c: 3

B=2

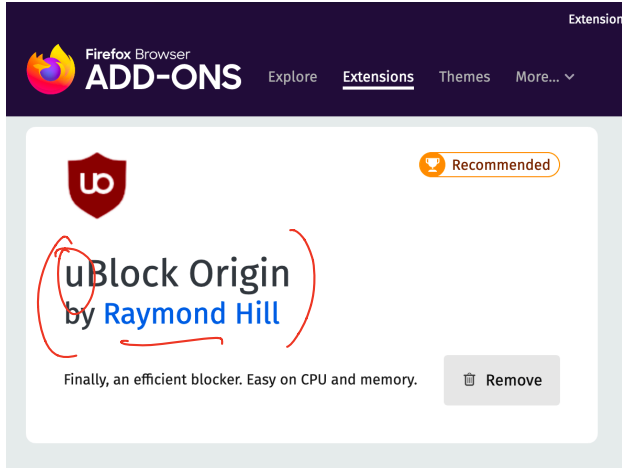
B=2.

tells B that  
user 2  
visited  
A then C

Examples *(next)*

# Blocking


ma block origin



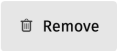
The image shows a screenshot of the Firefox Add-ons page for the uBlock Origin extension. The page has a dark purple header with the Firefox logo and the text "Firefox Browser ADD-ONS". Below the header, there are navigation links: "Explore", "Extensions" (which is underlined), "Themes", and "More...". The main content area features the uBlock Origin logo (a red shield with "uB" in white) and a "Recommended" badge. The text "uBlock Origin" is prominently displayed, with "by Raymond Hill" underneath it. A red circle is drawn around the "uBlock Origin" text, and a red line is drawn under "by Raymond Hill". Below the text, there is a description: "Finally, an efficient blocker. Easy on CPU and memory." and a "Remove" button with a trash icon.

Extension

Firefox Browser  
**ADD-ONS** Explore Extensions Themes More... ▾

 Recommended

**uBlock Origin**  
by Raymond Hill

Finally, an efficient blocker. Easy on CPU and memory. 

# Cross-site Request Forgery (CSRF) attack



www.attacker.com

Victim Browser



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=xyzyz>
</form>
<script>document.forms[0].submit()</script>
```

POST /login HTTP/1.1  
Referer: http://www.attacker.com/blog  
username=attacker&password=xyzyz

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

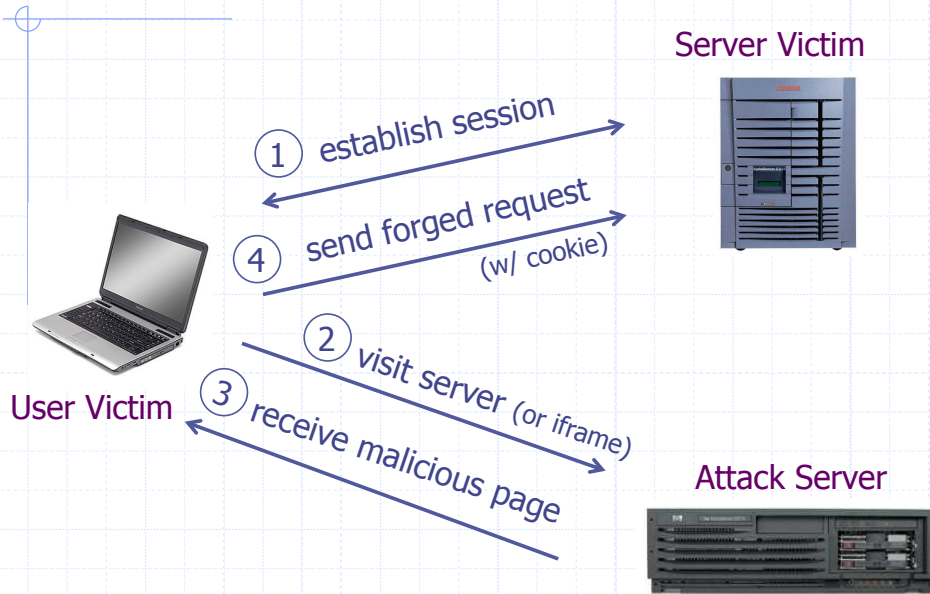
9:20pm

Searched for [llamas](#)

Barth, Jackson, Mitchell 2008



# Basic picture



Q: how long do you stay logged in to Gmail? Facebook? ....

# Cross-Site Request Forgery (CSRF)

1. Assume victim has google/fbook/twitter cookies already setup.

2. Victim visits ATTACKER page.

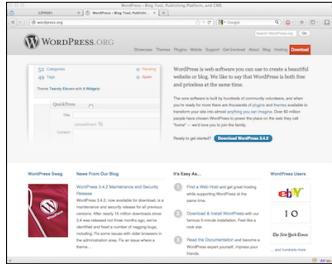
3. ATTACKER page HTML causes a request to google/...

this request uses Victims google/ cookie jar

request **unknowingly** changes state of victim's account

# Cross site RF

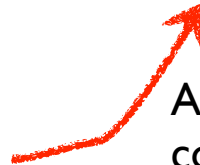
website asks a question  
(sends a form)



website



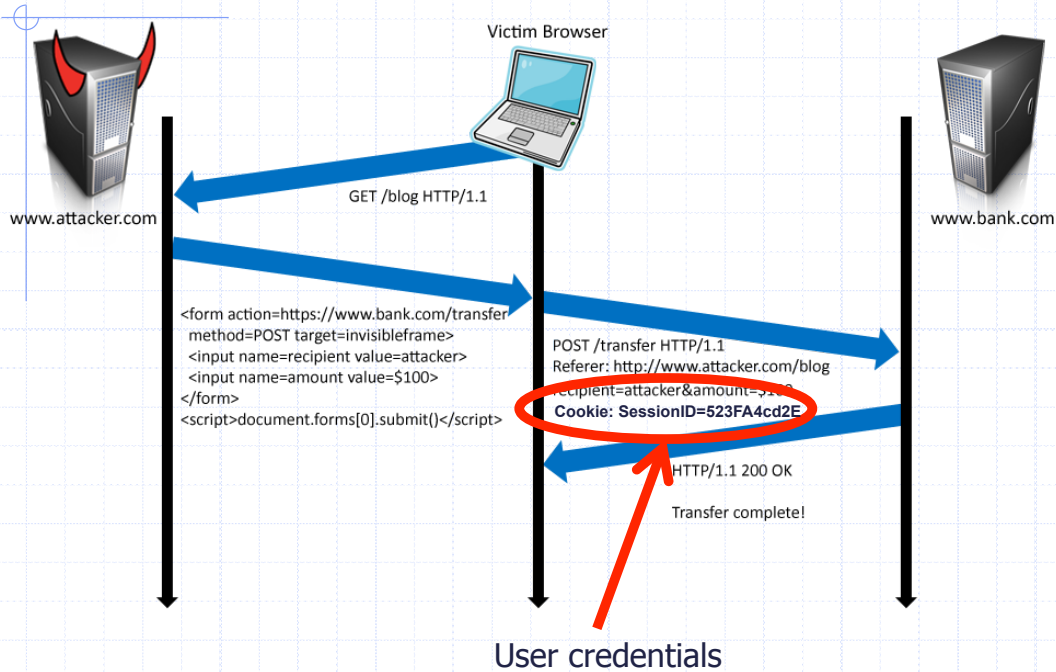
to offer tainted answer



Attacker site  
convinces victim  
browser...

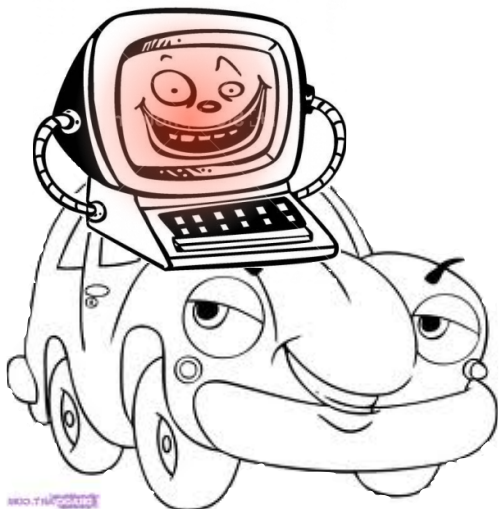


# Form post with cookie



# Drive-by Pharming

(Stamm & Ramzan)



Looking for the Linksys WRT54G default password? You probably have little reason to access your [router](#) on a regular basis so don't feel too bad if you've forgotten the WRT54G default password.

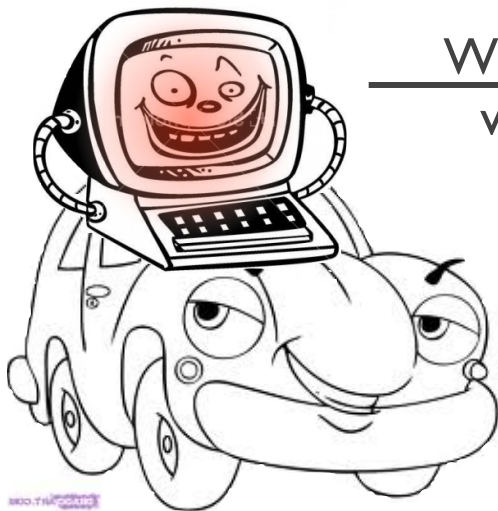
...

For most versions of the Linksys WRT54G, the default password is *admin*. As with most passwords, the WRT54G default password is [case sensitive](#).

In addition to the WRT54G default password, you can also see the WRT54G default username and WRT54G default [IP address](#) in the table below.

# Drive-by Pharming

(Stamm & Ramzan)



Wireless nvram  
value setting



“Use DNS 1.1.1.1”



# National Vulnerability Database

automating vulnerability management, security measurement, and compliance checking

Vulnerabilities	Checklists	800-53/800-53A	Product Dictionary	Impact Metrics	Data Feeds	Statistics
Home	SCAP	SCAP Validated Tools	SCAP Events	About	Contact	Vendor Comments

## Mission and Overview

NVD is the U.S. government repository of standards based vulnerability management data. This data enables automation of vulnerability management, security measurement, and compliance (e.g. FISMA).

## Resource Status

### NVD contains:

52799 [CVE Vulnerabilities](#)  
 202 [Last updated:](#)  
 221 [TIP-SEP Alerts](#)  
 14,393 [EDT](#)  
 2636 [US-CERT Vuln Notes](#)  
 8140 [DVAL Queries](#)  
 60357 [CVE Publication rate:](#) 29.0

## Email List

NVD provides four mailing lists to the public. For information and subscription instructions please visit

## Search Results ([Refine Search](#))

There are **563** matching records. Displaying matches **1** through **20**.

[1](#) [2](#) [3](#) [4](#) [5](#) [6](#) [7](#) [8](#) [9](#) [10](#) [11](#) > >>

### **CVE-2012-4893**

[VU#788478](#)

**Summary:** Multiple cross-site request forgery (CSRF) vulnerabilities in file/show.cgi in Webmin 1.590 and earlier allow remote attackers to hijack the authentication of privileged users for requests that (1) read files or execute (2) tar, (3) zip, or (4) gzip commands, a different issue than CVE-2012-2982.

**Published:** 09/11/2012

**CVSS Severity:** [6.8](#) (MEDIUM)

### **CVE-2012-4890**

**Summary:** Multiple cross-site scripting (XSS) vulnerabilities in FlatnuX CMS 2011 08.09.2 and earlier allow remote attackers to inject arbitrary web script or HTML via a (1) comment to the news, (2) title to the news, or (3) the folder names in a gallery.

**Published:** 09/10/2012

**CVSS Severity:** [4.3](#) (MEDIUM)

### **CVE-2012-0714**

**Summary:** Cross-site request forgery (CSRF) vulnerability in IBM Maximo Asset Management 6.2 through 7.5, as used in SmartCloud Control Desk, Tivoli Asset Management for IT, Tivoli Service Request Manager, Maximo Service Desk, and Change and Configuration Management Database (CCMDB), allows remote attackers to hijack the authentication of unspecified victims via unknown vectors.

**Published:** 09/10/2012

**CVSS Severity:** [6.8](#) (MEDIUM)



# CSRF defenses

Secure Token:

Referer Validation:

Custom Headers:

```
<input type="hidden" id="ipt_nonce" name="ipt_nonce" value="99ed897af2">
```

```
<input type="hidden" id="ipt_nonce" name="ipt_nonce" value="99ed897af2" />
```

# 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