Penetration Testing Report for Relq

Prepared By: Arayik Gabrielyan
Date: December 6, 2024
Email: gabrielyan_2003@bk.ru

Table of Contents

Login Admin(injection)

Access Control Bypass via Unauthorized File Addition

Lack of Account Lockout Mechanism Allowing Password Bruteforce

Sensitive Data Exposure Through Forgotten Sales Backup

Exploitation of Easter.egg Backup File Vulnerability

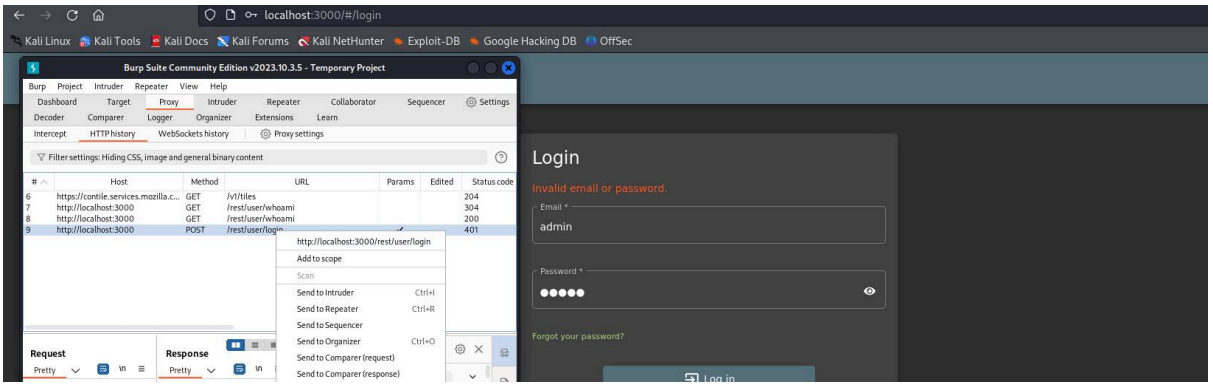
Try Hack Me: Root Me

Creating a Simple Python Network Scanner Similar to Nmap

Login Functionality Analysis

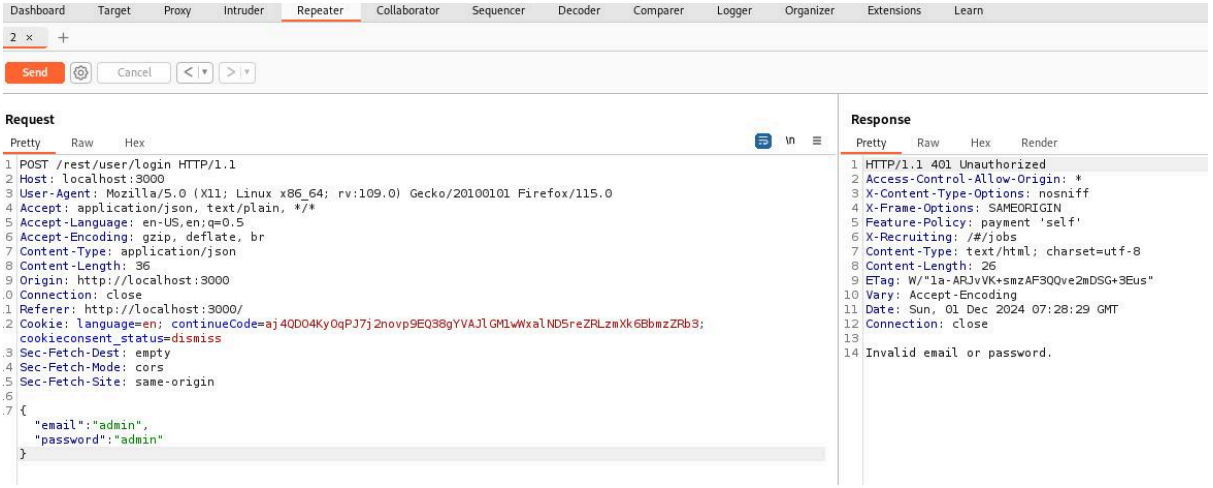
During the penetration testing process, an attempt was made to log in to the Juice Shop application as an admin user. The following steps were taken to analyze the login functionality:

1. **Initial Observation**:
   Upon entering the admin username and password, the application returned an "Invalid" error message.
2. **Intercepting the Request**:
   Using **Burp Suite**, the login request was intercepted through the **Proxy** tab. The intercepted request pointed to the following endpoint: `http://localhost:3000/rest/user/login`.
   This request was located in the HTTP history section of Burp Suite.
3. **Sending to Repeater**:
   The identified request was sent to the **Repeater** tab for further analysis by pressing `Ctrl + R`. This allowed for detailed inspection and manual modifications to test the application's response.
4. **Further Testing and Observations**:
   In the Repeater tab, the request was analyzed, modified, and re-sent to the server to identify potential vulnerabilities in the login process.

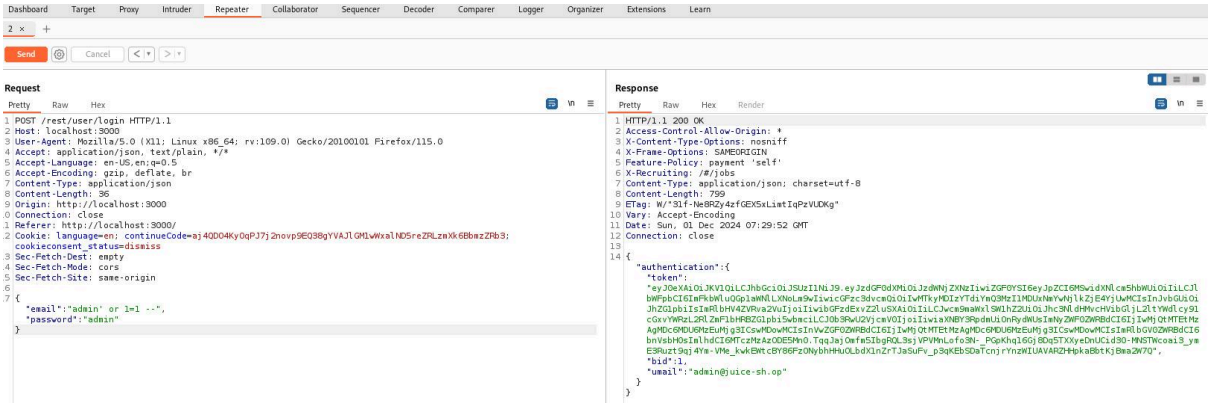


Execution in Repeater:
After sending the intercepted request from the **Repeater** tab by clicking **Send**, the server's response was analyzed. The response revealed the

login credentials that were collected, allowing for further investigation of potential vulnerabilities.

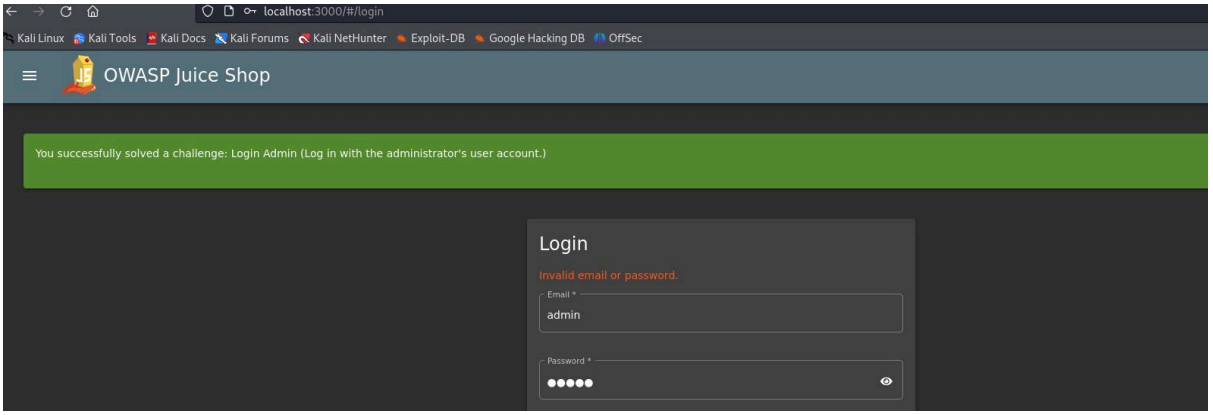


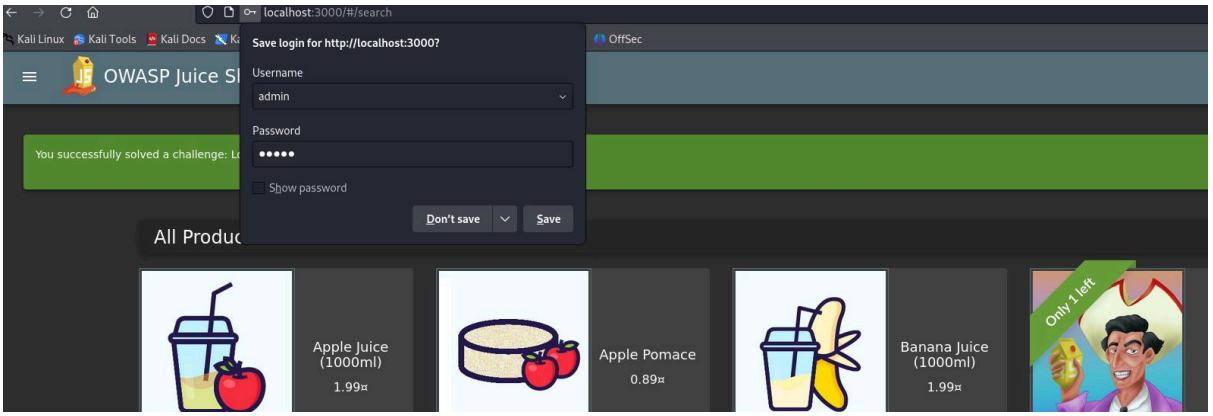
SQL Injection Attempt:
To test for SQL injection vulnerabilities, the following payload was used in the login field:
`admin' OR 1=1 --`.
After sending the modified request, the server responded by providing an access key, confirming the presence of an SQL injection vulnerability.



Challenge Completion:
Following the successful exploitation of the vulnerability, the Juice Shop website displayed the following message:
"You successfully solved a challenge: Login Admin (Log in with the administrator's user account.)"

This confirmed that administrator-level access was successfully achieved, validating the effectiveness of the performed penetration testing steps.

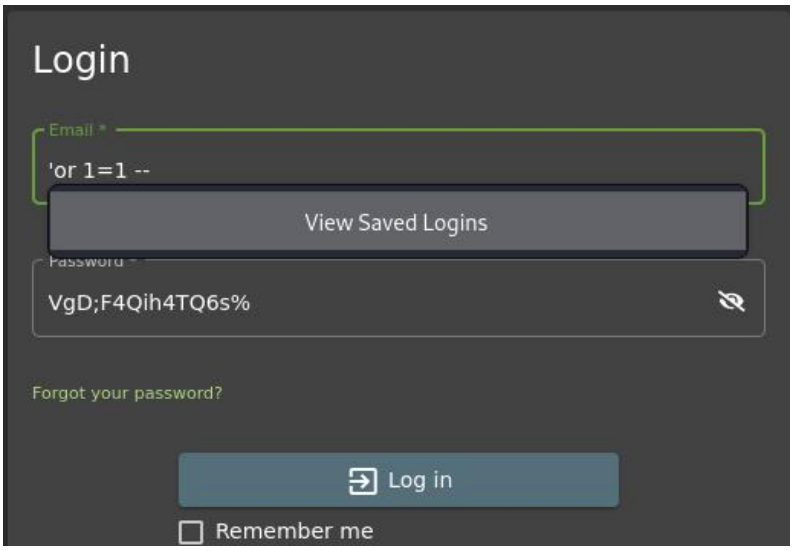


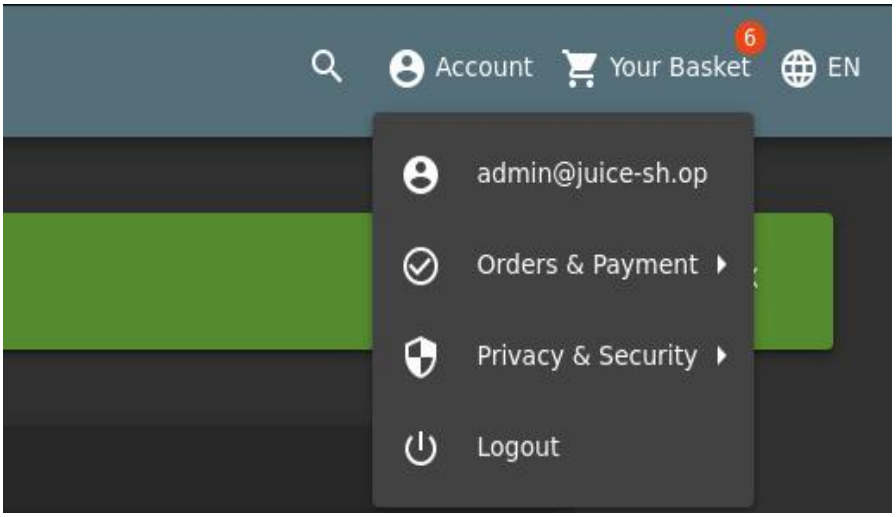


In the **Username** field, enter: `admin' OR 1=1 --`

In the **Password** field, you can leave it empty or enter any random value, since the SQL injection bypasses the need for a valid password.

After entering the payload in the login fields, submit the form. The payload modifies the SQL query to always evaluate the condition `1=1`, which will always be true and allow you to bypass the authentication.





Countermeasures to Prevent SQL Injection

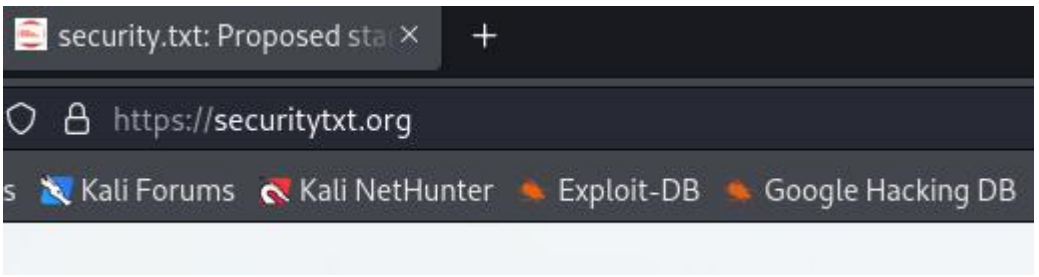
To prevent SQL injection vulnerabilities like the one identified in the Juice Shop application, the following best practices and mitigation steps should be implemented:

1. **Use Prepared Statements (Parameterized Queries)**:
   Ensure that all SQL queries are executed using prepared statements with bound parameters, which separate the SQL logic from user input. This prevents attackers from manipulating the query structure.
2. **Implement Input Validation and Sanitization**:
   Validate and sanitize all user inputs before processing. Any input that will be used in SQL queries should be strictly checked for type, length, format, and range. Reject or sanitize any input containing special characters such as `'`, `--`, and `;`.
3. **Least Privilege Principle**:
   Database accounts used by the application should only have the necessary permissions. The application should use a non-administrative database user to connect to the database, limiting the potential impact of any SQL injection attack.
4. **Use ORM (Object-Relational Mapping)**:
   Implement an ORM framework that automatically escapes and sanitizes input, reducing the risk of SQL injection by abstracting raw SQL queries.
5. **Error Handling and Reporting**:
   Avoid exposing detailed error messages to users. Ensure that errors related to SQL queries do not display stack traces or database information, as these can provide attackers with valuable insights into the system's structure.
6. **Regular Security Audits and Penetration Testing**:
   Conduct regular security assessments, including automated vulnerability scans and manual penetration testing, to identify and remediate potential SQL injection vulnerabilities.
7. **Web Application Firewall (WAF)**:
   Use a WAF to help detect and block common SQL injection attempts in real time. A WAF can provide an additional layer of defense while other measures are implemented.

Access Control Bypass via Unauthorized File Addition

The **security.txt** website (https://securitytxt.org) is a project aimed at improving communication related to website security. It was developed as a standard to help website owners provide information about vulnerability discovery and reporting.

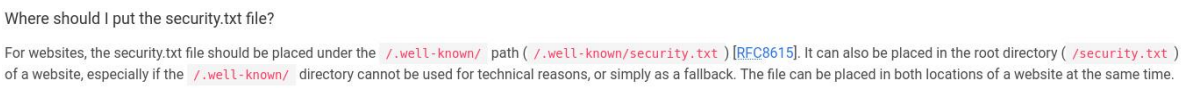
I placed the **security.txt** link in the



Placing security.txt as a Continuation of the Main Website Link

website's link field as a continuation of the main website link.

The **security.txt** file was correctly placed under the `/.well-known/` directory, making it publicly accessible.


Where should I put the security.txt file?

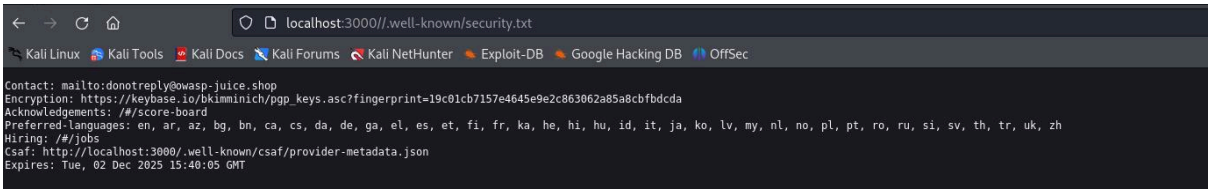
For websites, the security.txt file should be placed under the `/.well-known/` path (`/.well-known/security.txt`) [RFC8615]. It can also be placed in the root directory (`/security.txt`) of a website, especially if the `/.well-known/` directory cannot be used for technical reasons, or simply as a fallback. The file can be placed in both locations of a website at the same time.

Correct Placement of security.txt in the `/.well-known/` Directory

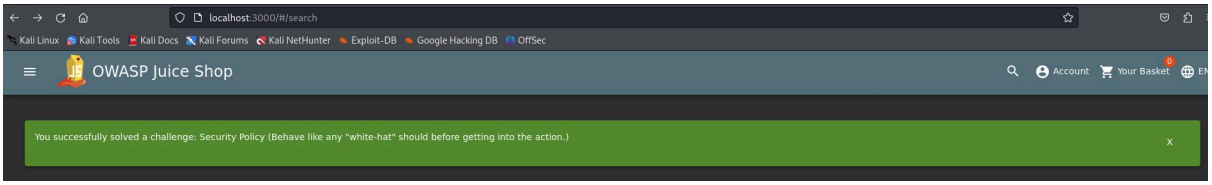
By accessing the link `http://localhost:3000//.well-known/security.txt`, I was able to retrieve contact details and other security-related information from the file.

Accessing and Retrieving Security Information from security.txt

This file provides an easy way for security researchers to contact the security team or report any vulnerabilities.



Navigated back to the application and observed access to restricted sections without authentication.



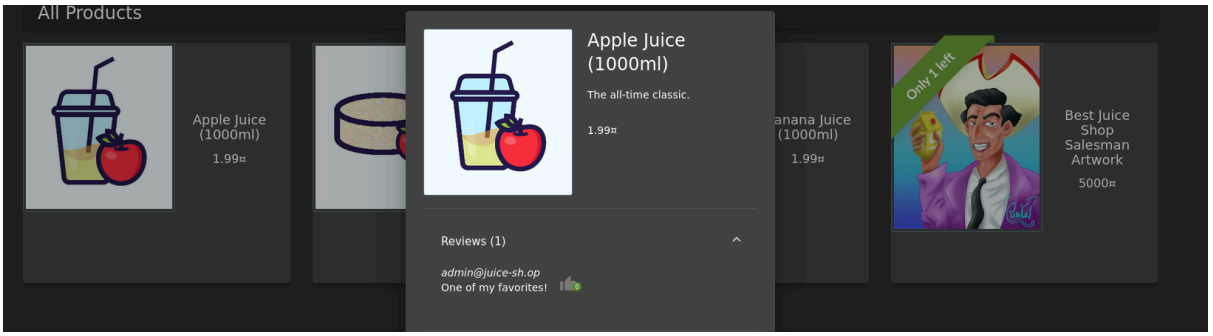
Countermeasures:

1. **Restrict Unauthorized File Additions in the `/.well-known/` Directory:**
   It is important to implement proper access control to prevent unauthorized users from adding files to the `/.well-known/` directory, which could expose sensitive information or provide unauthorized access.
2. **Strengthen Access Control Mechanisms for All Application Sections:**
   Access control should be enforced uniformly across all sections of the application to ensure that only authorized users can access restricted resources. This can be achieved by implementing proper authentication and authorization mechanisms.
3. **Improve Session Management to Prevent Unintended Permission Escalation:**
   Session management mechanisms should be strengthened to avoid situations where users gain unintended elevated permissions. Proper session timeouts, secure cookies, and re-authentication for sensitive actions can help mitigate this risk.

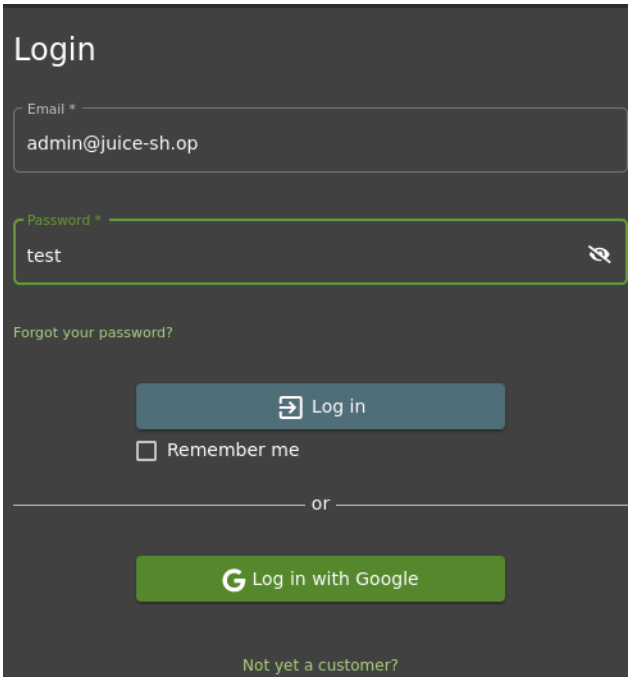
Lack of Account Lockout Mechanism Allowing Password Bruteforce

Initial Steps to Set Up a Bruteforce Attack

First, select any product available for sale on the Juice Shop website. Copy the administrator's email address displayed with the product details. Next, navigate to the login page where users can access their accounts.

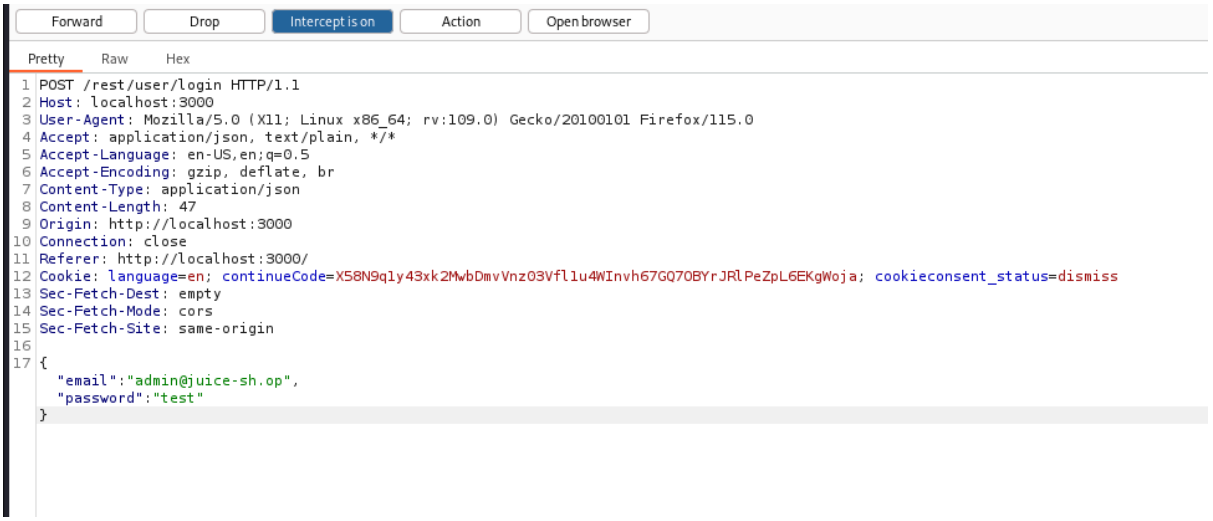


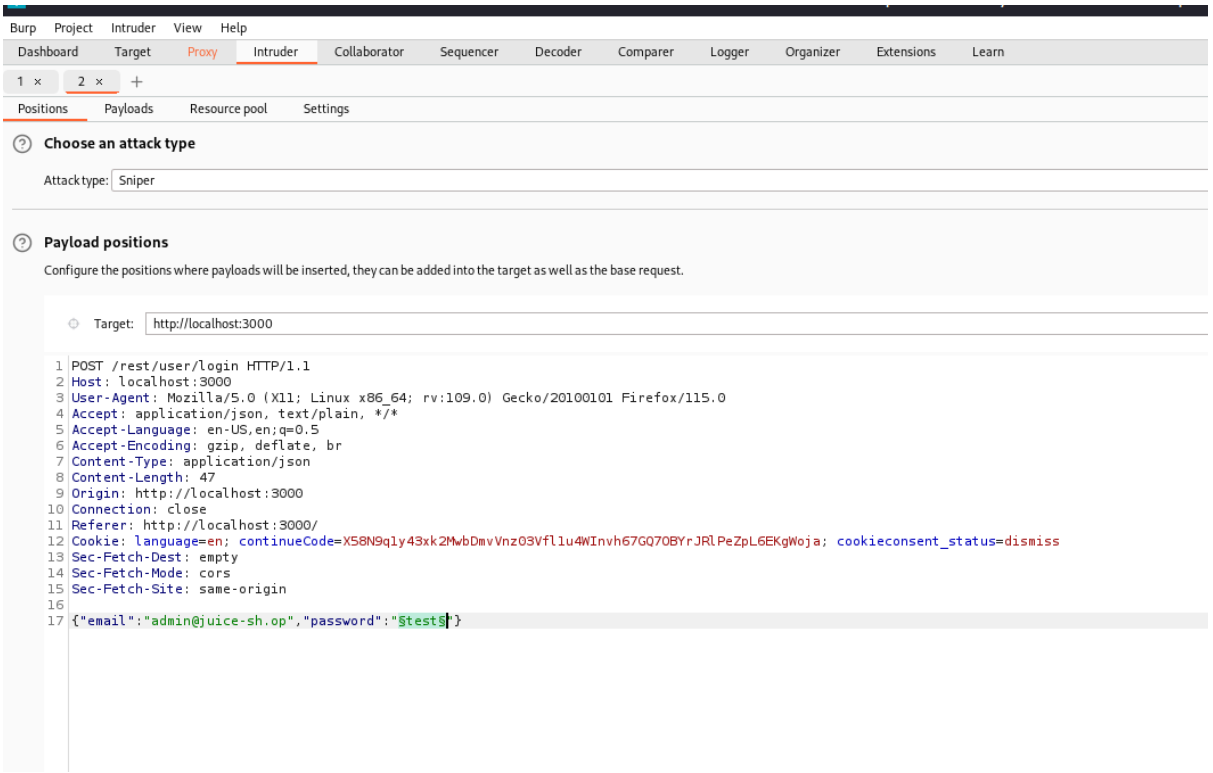
On the login page, enter the email address you previously copied into the Email field. In the Password field, input an incorrect value, such as "test." Before proceeding with the login attempt, connect the proxy and open the Burp Suite tool. Once everything is set up, click "Log In" and wait for Burp Suite to intercept the request.



Using Intruder to Identify the Correct Password

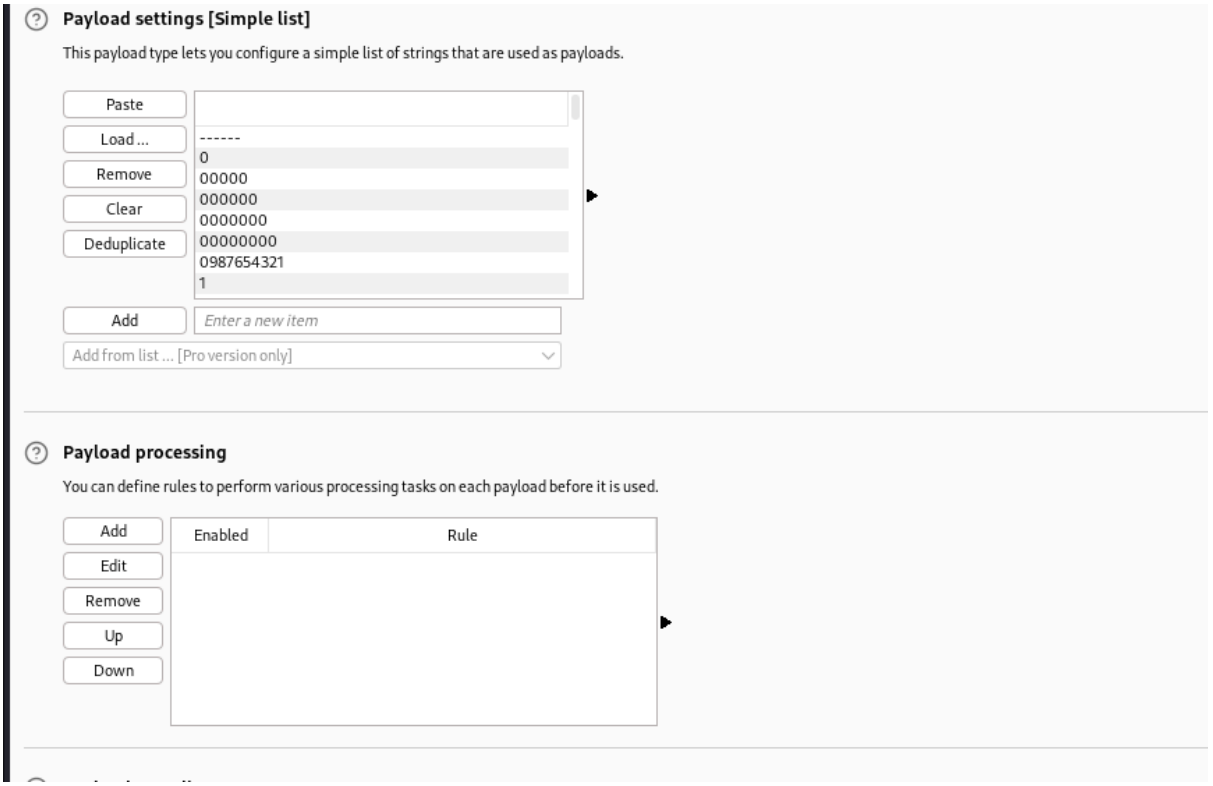
Once Burp Suite intercepts the login request, navigate to the tool and forward the request to the Intruder module. In Intruder, specify the appropriate field where the password will be replaced. Then, upload a list of 1000 potential passwords as the payload. Burp Suite will systematically test each password until the correct one is identified.





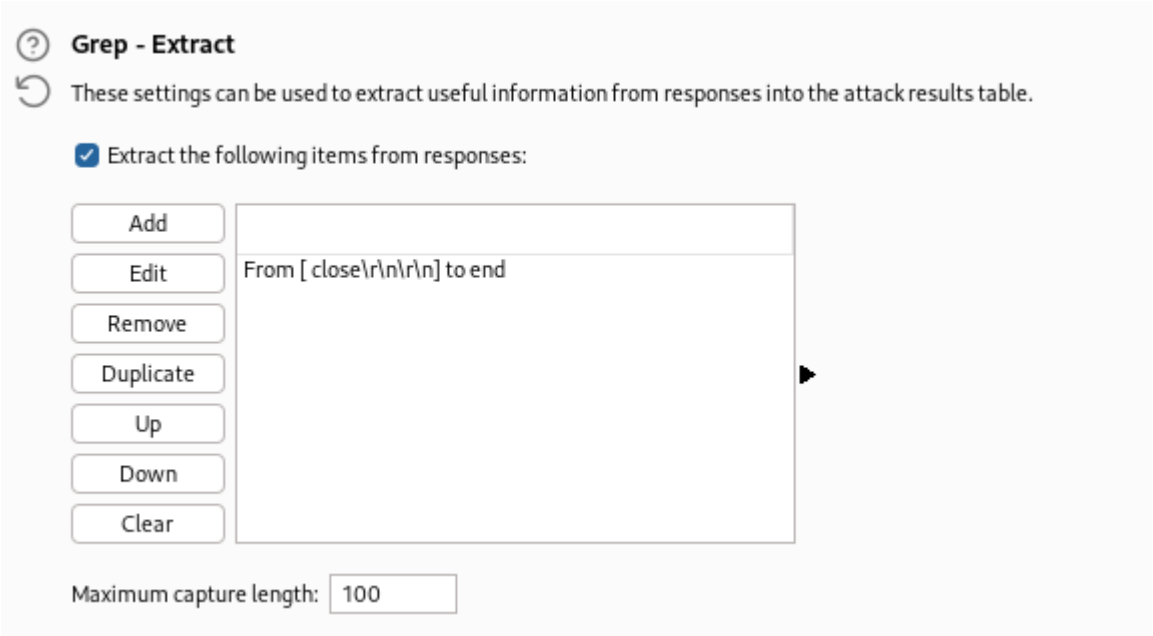
Importing a Password List for Intruder Payloads

Next, use a browser to search for a list of 1000 commonly used passwords, for example, from GitHub. Copy the list and upload it to Burp Suite Intruder's **Simple List** as the payload for the attack.

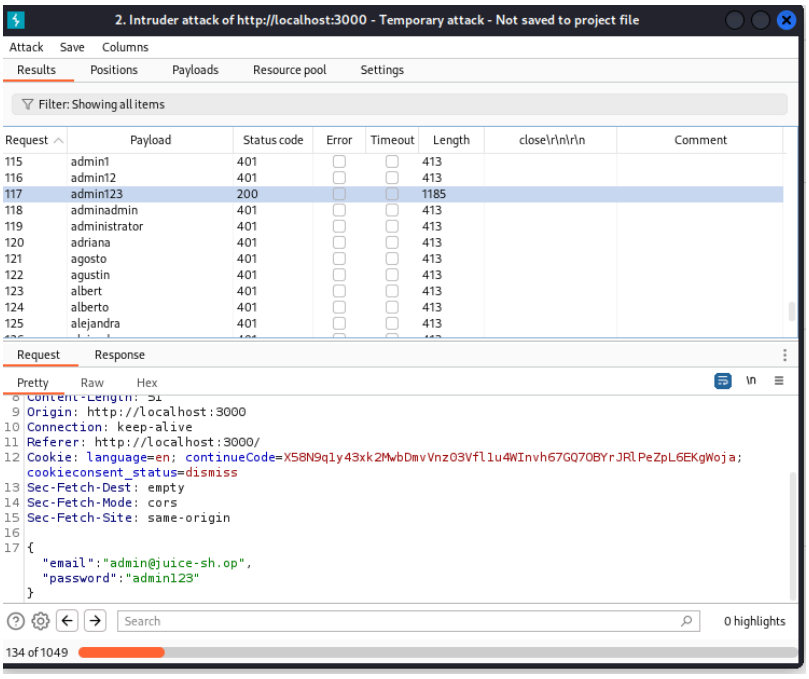


Configuring Grep - Extract for Error Response Tracking

Then, navigate to the **Intruder** settings and open the **Grep - Extract** section. Add the response phrase **"Invalid email or password"** to track and identify failed login attempts.

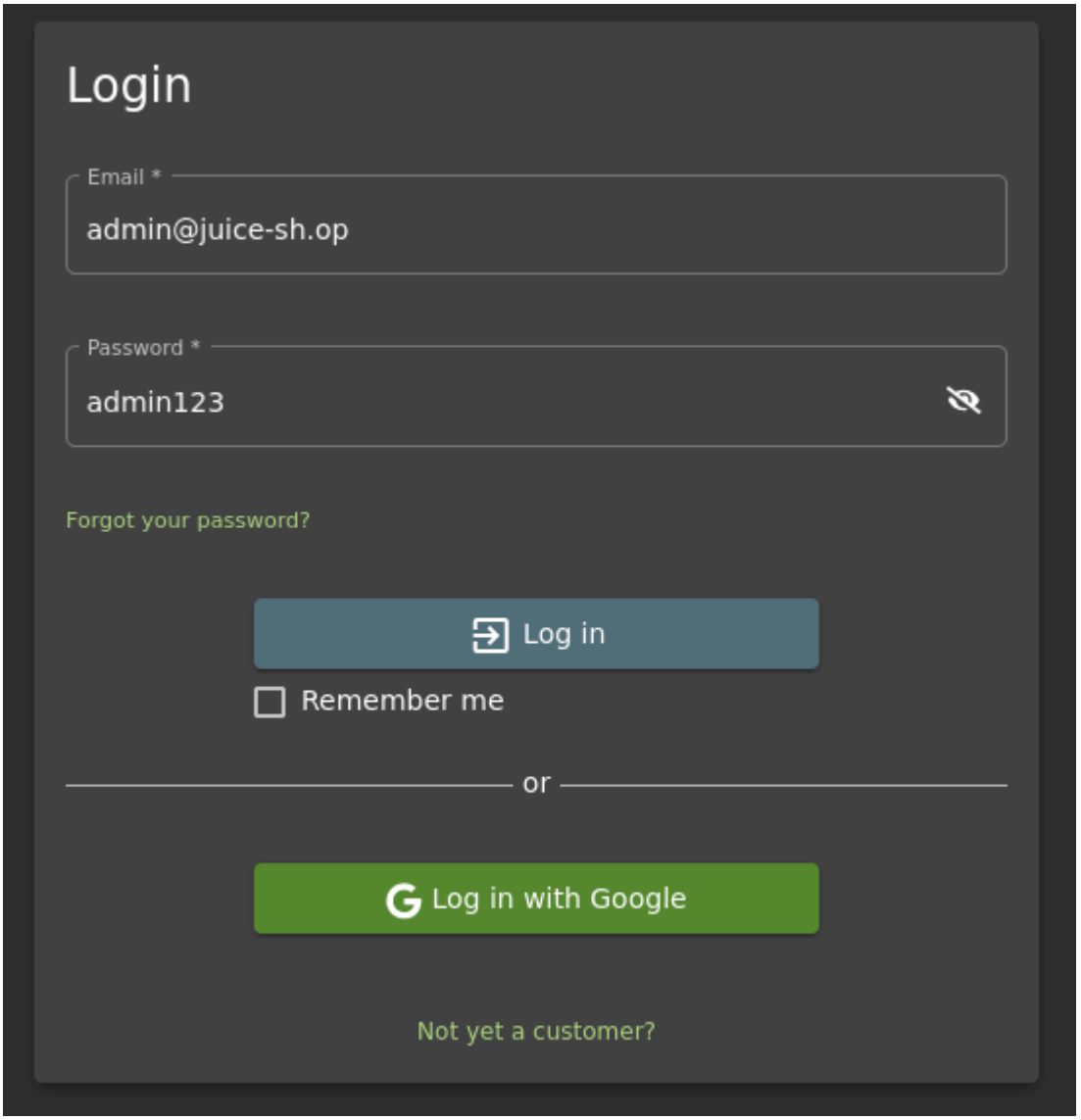


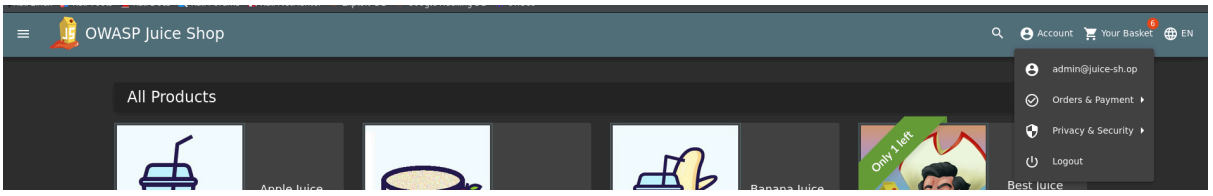
After completing the setup, click **Start Attack** in Intruder and wait for the results. On line 117 of the output, observe a **200 OK** status code, indicating a successful login. The correct password is displayed as **admin123**.



Administrator Access Achieved Successfully

Finally, we use the identified password to log in to the website. Upon entering the credentials, we successfully gain access to the site as an administrator.



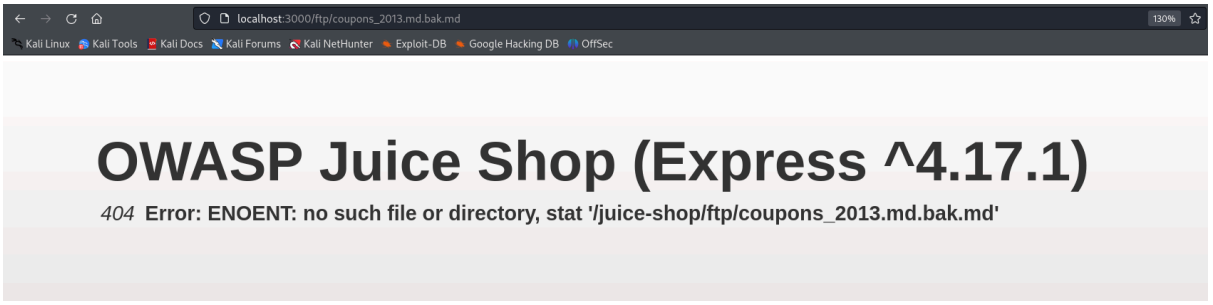


Sensitive Data Exposure Through Forgotten Sales Backup

Appending Malicious Paths to Access Backup Files

To exploit this vulnerability, we append the following path to the Juice Shop URL:
`http://localhost:3000/ftp/package.json.bak%00.md`

This allows us to access a backup file stored on the server, potentially exposing sensitive information.



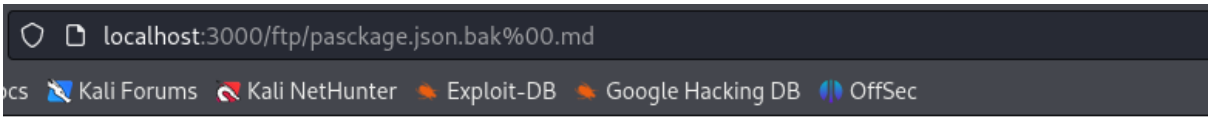
Exploiting URL Encoding to Access Backup Files

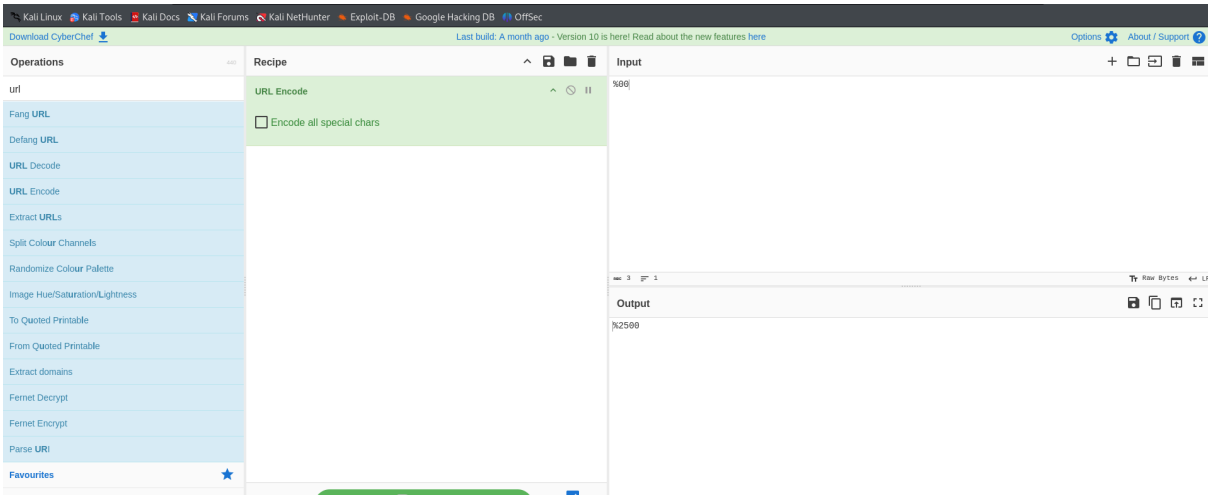
Next, open the following URL in your browser:

CyberChef URL Encoding Tool

Use the tool to URL encode %00, which will convert it to %2500. Replace %00 with %2500 in the URL. After making this change and refreshing the page, a

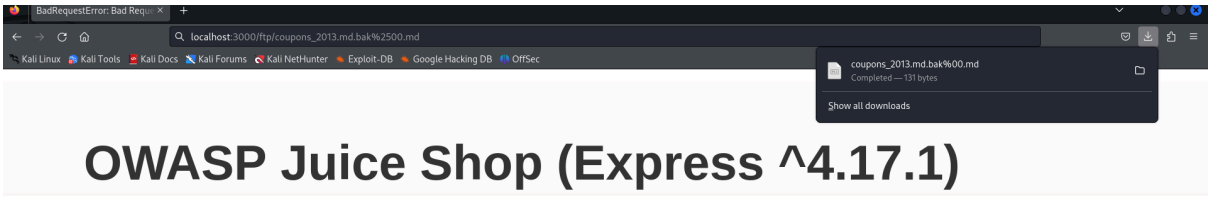
file download will be triggered, indicating access to the backup file.

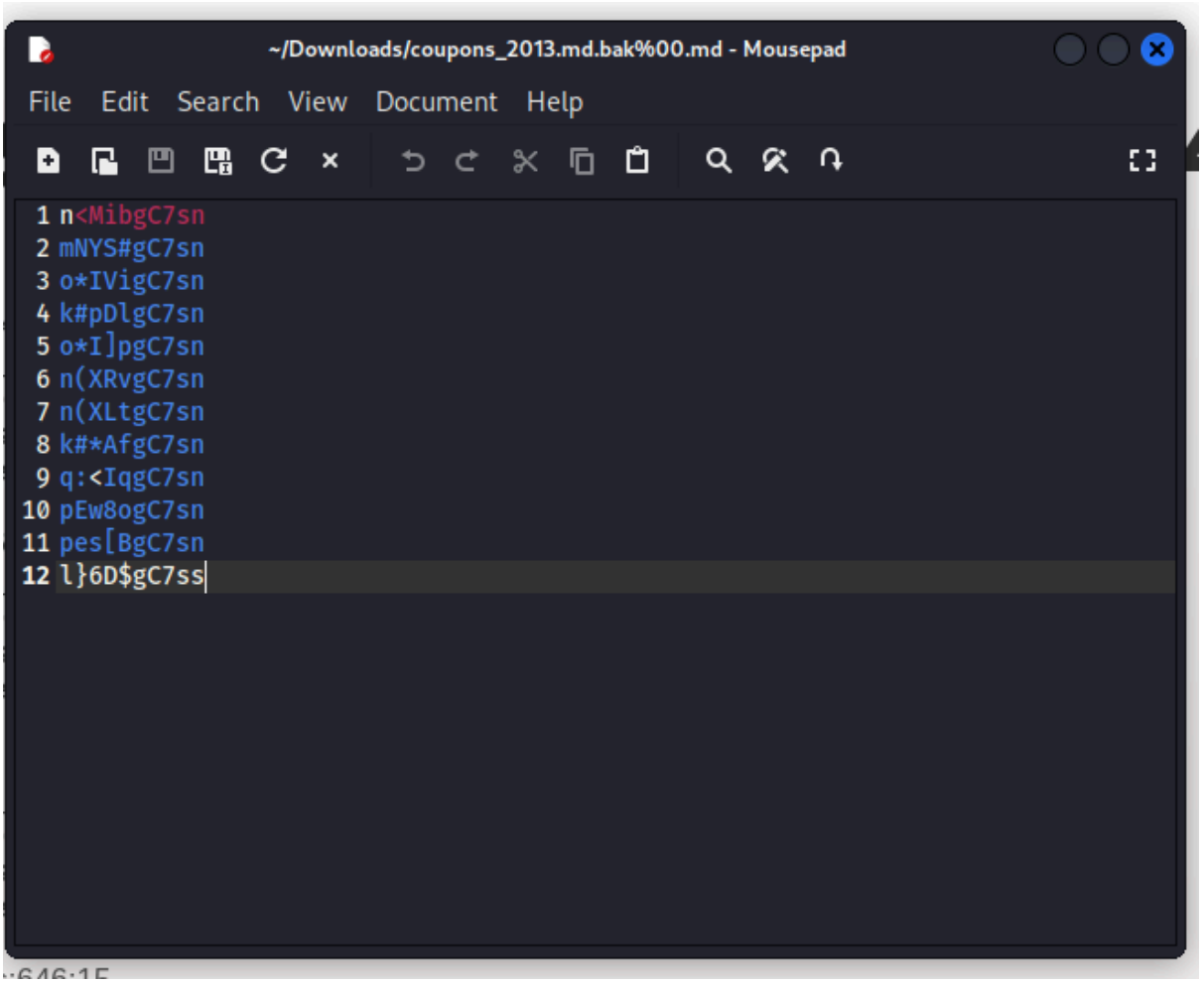




Sensitive Data Extracted from Backup File

Finally, upon opening the downloaded file, we confirm that it contains sensitive data, demonstrating unauthorized access to the server's backup.



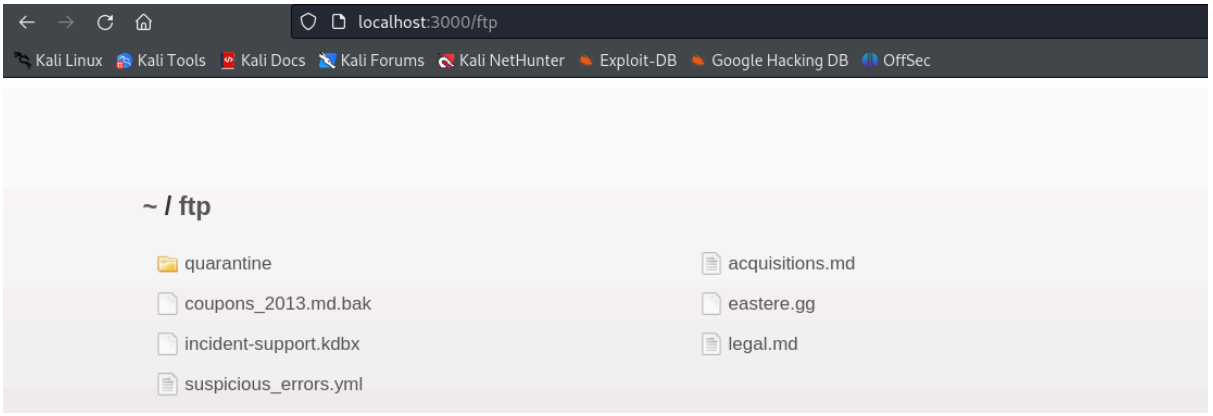


Exploitation of Easter.egg Backup File Vulnerability

Accessing Unprotected Backup Files Through Malicious Paths

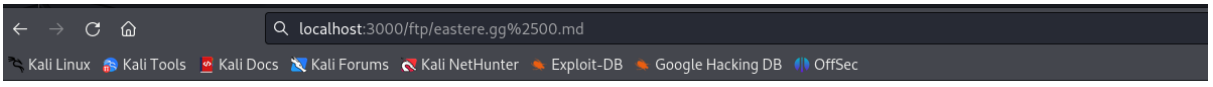
To exploit this vulnerability, we append the following path to the Juice Shop URL:
`http://localhost:3000/ftp/easter.egg`

This manipulation allows access to a backup file stored on the server. These backup files can contain sensitive information such as configuration details, user data, or credentials, which, if exposed, can lead to unauthorized access or further exploitation of the system.



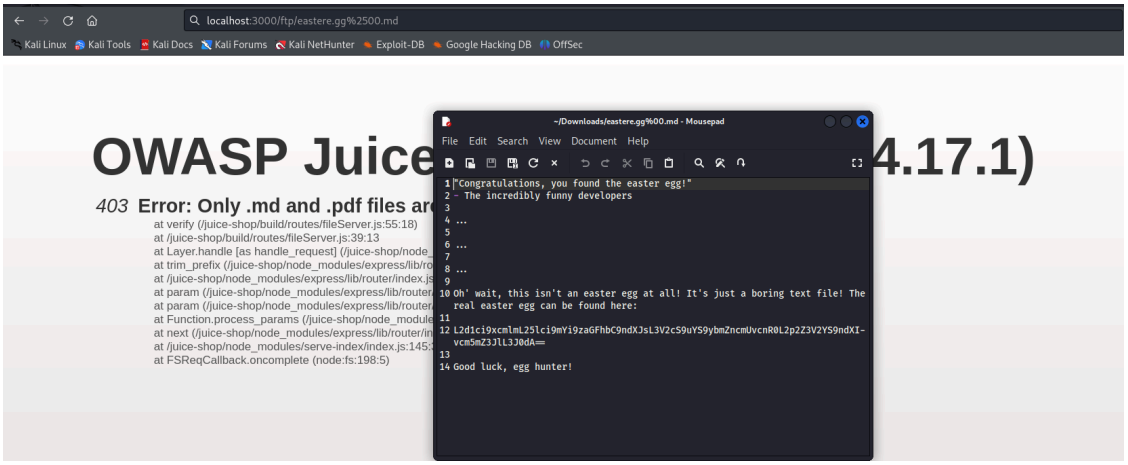
Next, we modify the URL by appending `%2500md` to the path.

This change allows us to bypass certain filters and access the targeted backup file, further demonstrating the server's vulnerability to URL manipulation.



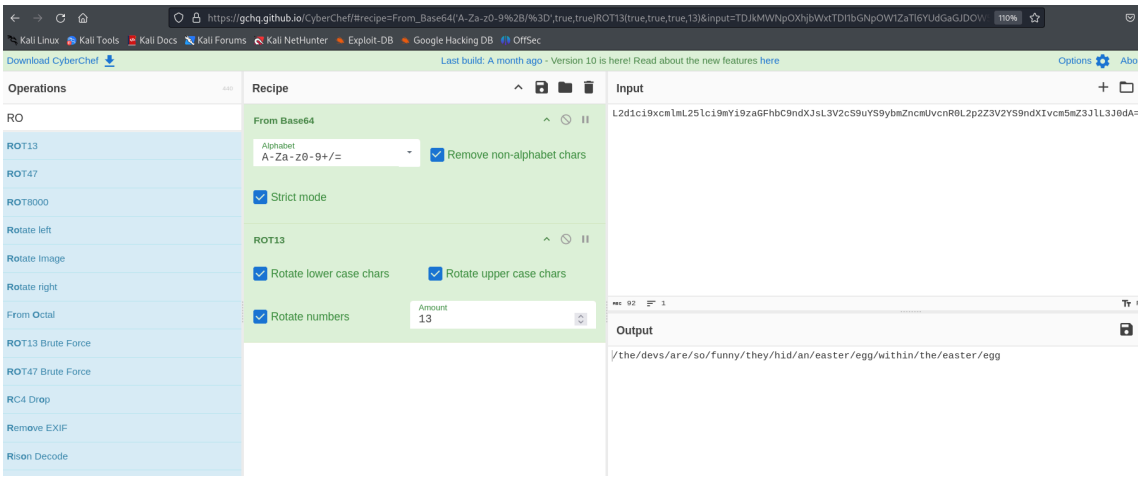
Extracting Data from the Downloaded File

By clicking on the modified URL, a file is downloaded. We open the file, copy the content from line 12, and input it into the CyberChef tool for further processing. This step allows us to analyze and utilize the extracted data for the next stages of the exploitation.



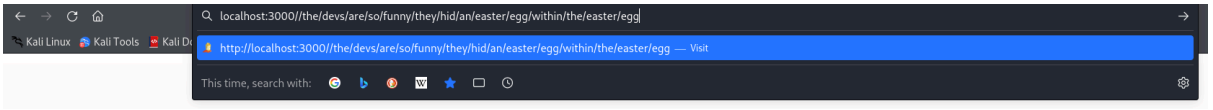
Decoding the Extracted Data

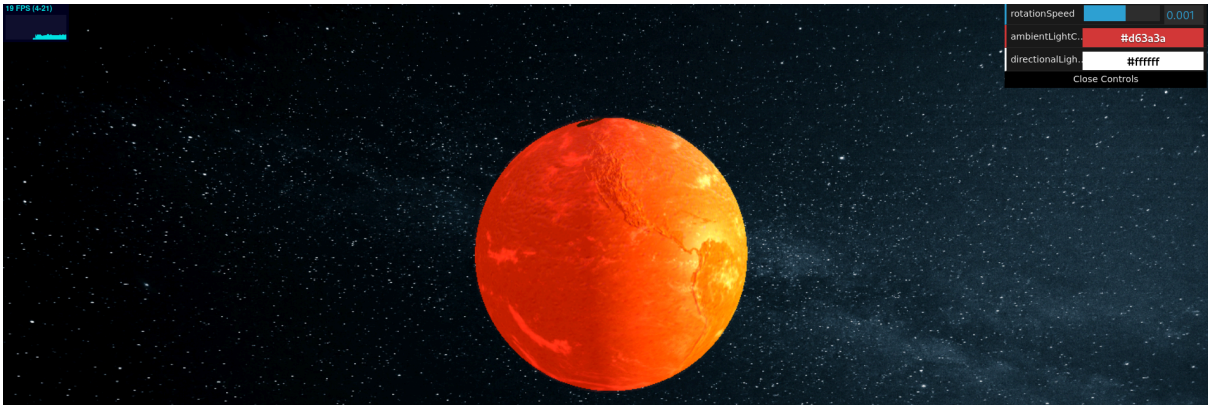
In the CyberChef tool, we paste the copied content into the **Input** section. In the **Recipe** section, we add the following operations: `From Base64` and `ROT13`. After applying the recipe, we copy the decoded result from the **Output** section for further use.



Accessing the Target Website and Modifying Its Appearance

After copying the decoded result, we return to the main Juice Shop URL and append the copied data to it. This leads us to a website that appears different. After making some color adjustments on the site, we go back and observe that we have successfully exploited the vulnerability.

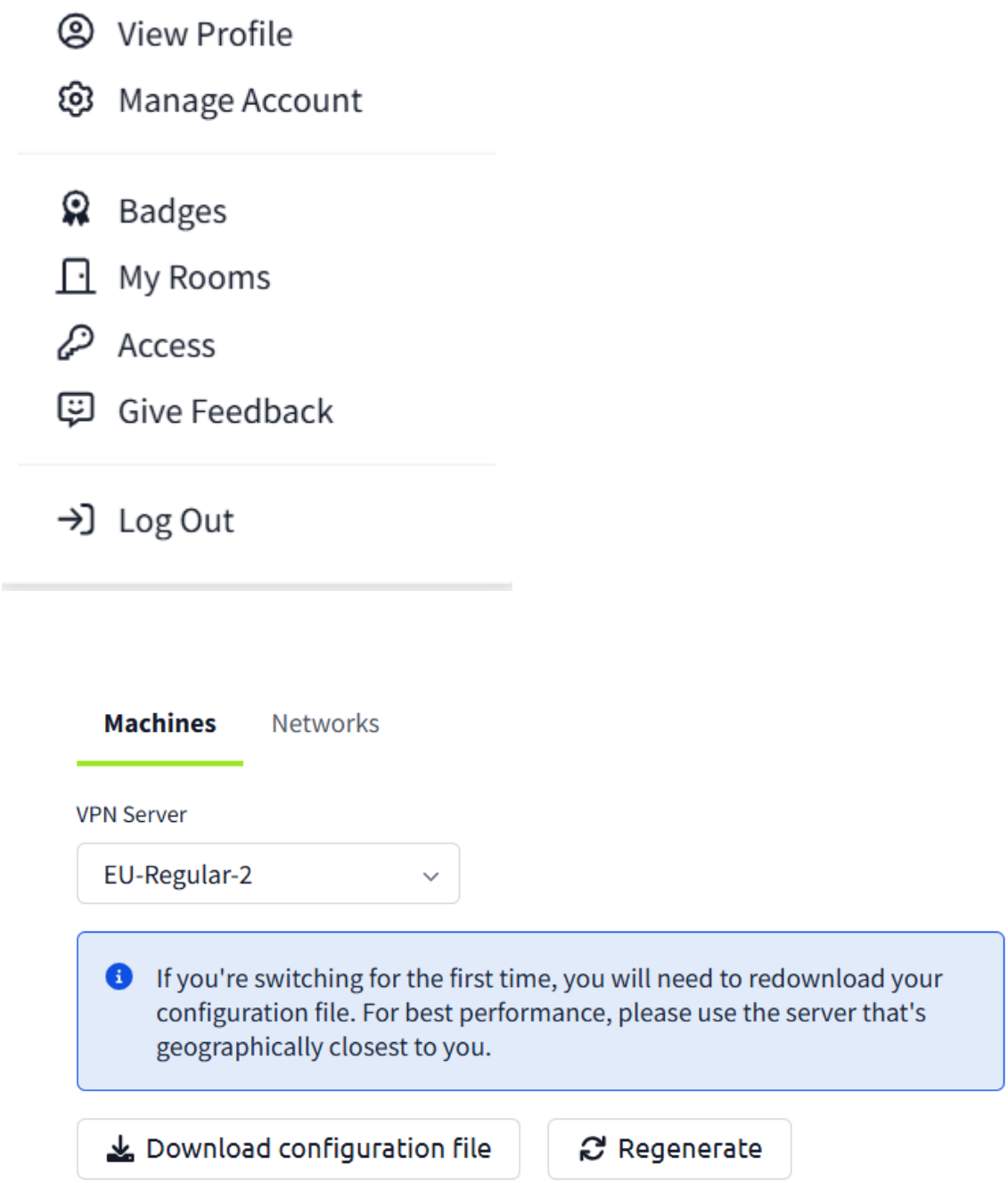




Try Hack Me: Root Me

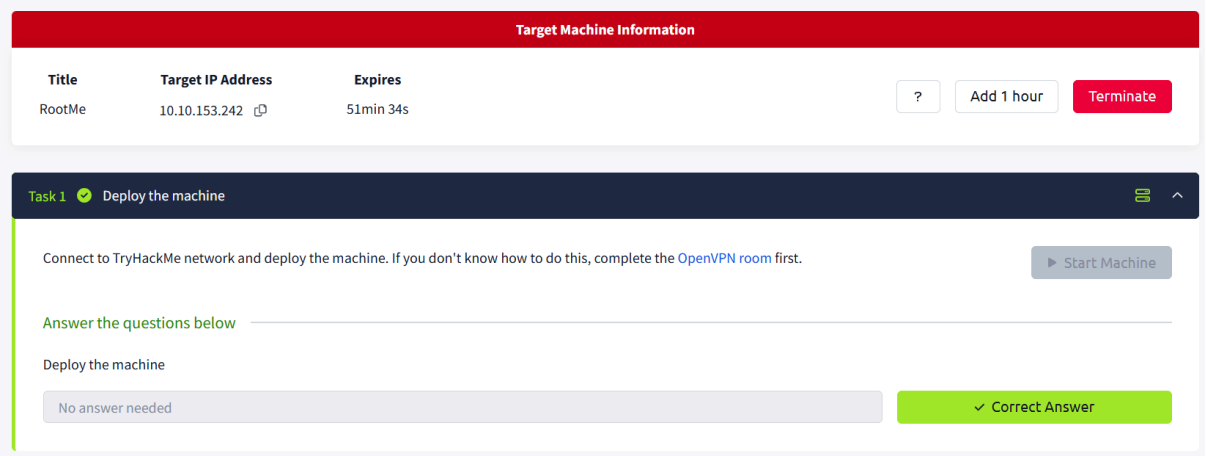
Configuration File Setup

Open the **TryHackMe** website and navigate to the **RootMe** room. First, download the configuration file as shown in the image. This step is necessary to proceed with the upcoming tasks.



Connecting to TryHackMe Successfully

At this point, we have successfully connected to TryHackMe and are ready to begin solving the challenges.




```
┌──(kali㉿kali)-[~/Desktop]
└─$ sudo openvpn game.arm.950.ovpn
[sudo] password for kali:
2024-11-27 10:34:25 Note: --cipher is not set. OpenVPN versions before 2.5 defaulted to BF-CBC as fallback when cipher negotiation fai
led in this case. If you need this fallback please add '--data-ciphers-fallback BF-CBC' to your configuration and/or add BF-CBC to --d
ata-ciphers.
2024-11-27 10:34:25 Note: cipher 'AES-256-CBC' in --data-ciphers is not supported by ovpn-dco, disabling data channel offload.
2024-11-27 10:34:25 OpenVPN 2.6.7 x86_64-pc-linux-gnu [SSL (OpenSSL)] [LZO] [LZ4] [EPOLL] [PKCS11] [MH/PKTINFO] [AEAD] [DCO]
2024-11-27 10:34:25 library versions: OpenSSL 3.3.2 3 Sep 2024, LZO 2.10
2024-11-27 10:34:25 DCO version: N/A
2024-11-27 10:34:25 WARNING: Your certificate is not yet valid!
2024-11-27 10:34:25 TCP/UDP: Preserving recently used remote address: [AF_INET]54.76.30.11:1194
2024-11-27 10:34:25 Socket Buffers: R=[212992→425984] S=[212992→425984]
```

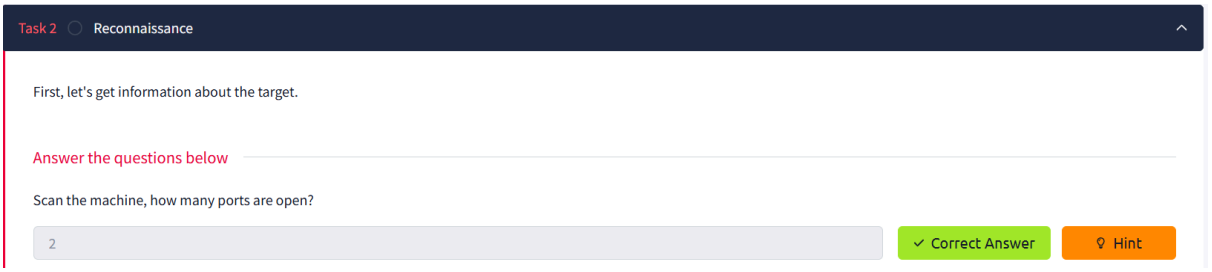
Initial Scanning

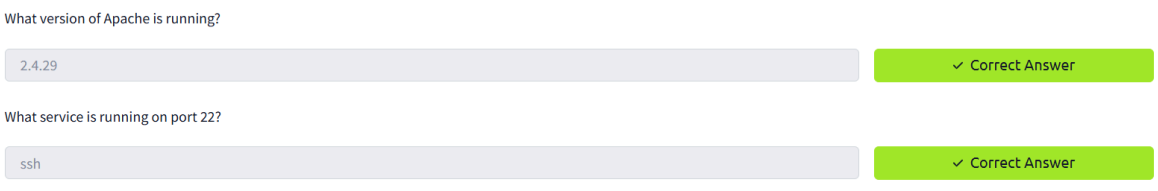
Open the terminal and run the commands shown in the image to perform the scan:

For the first question, identify the number of open ports and enter the result in the answer field.

For the second question, determine the version of the **Apache Server**.

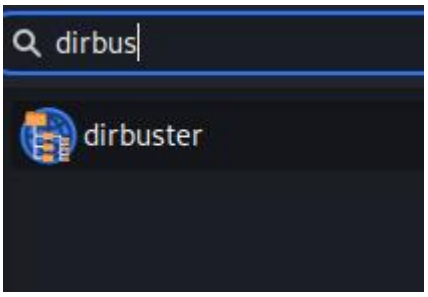
For the third question, find out what service is running on port 22.

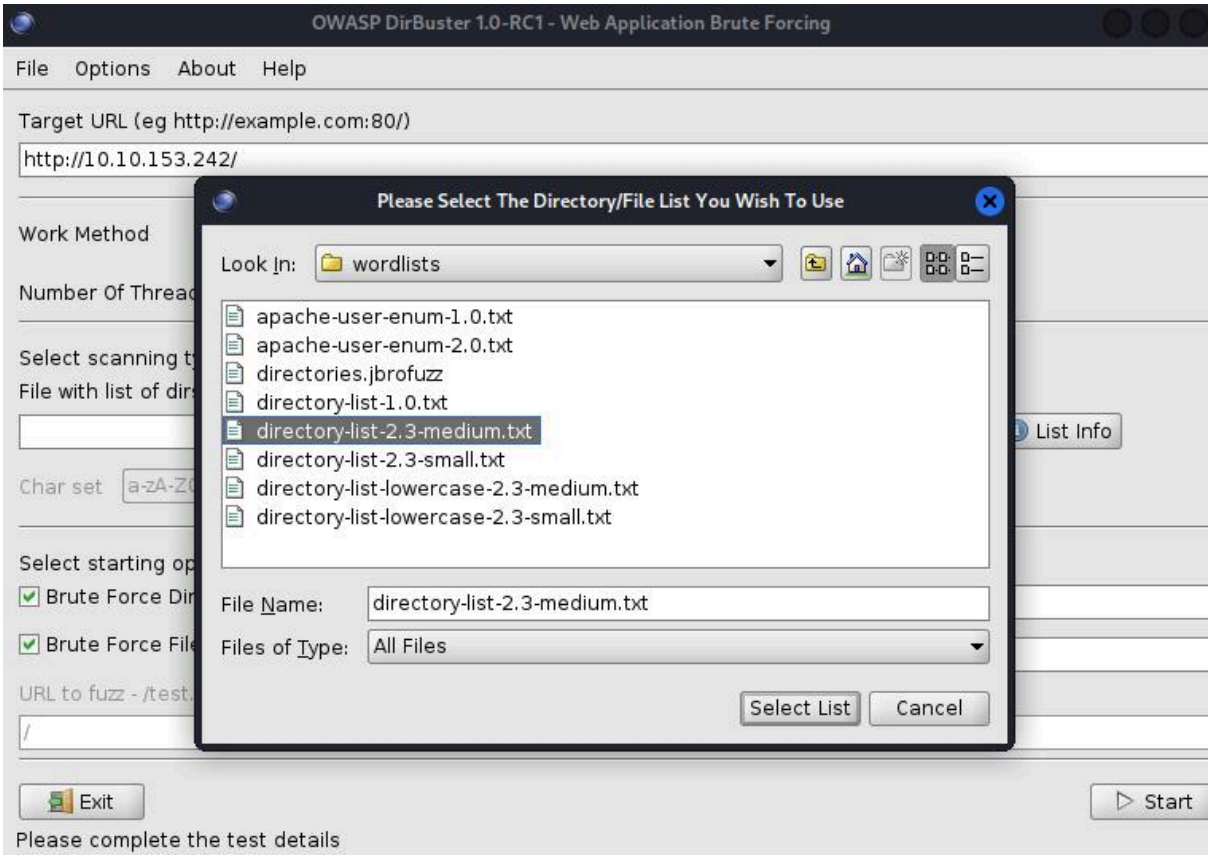


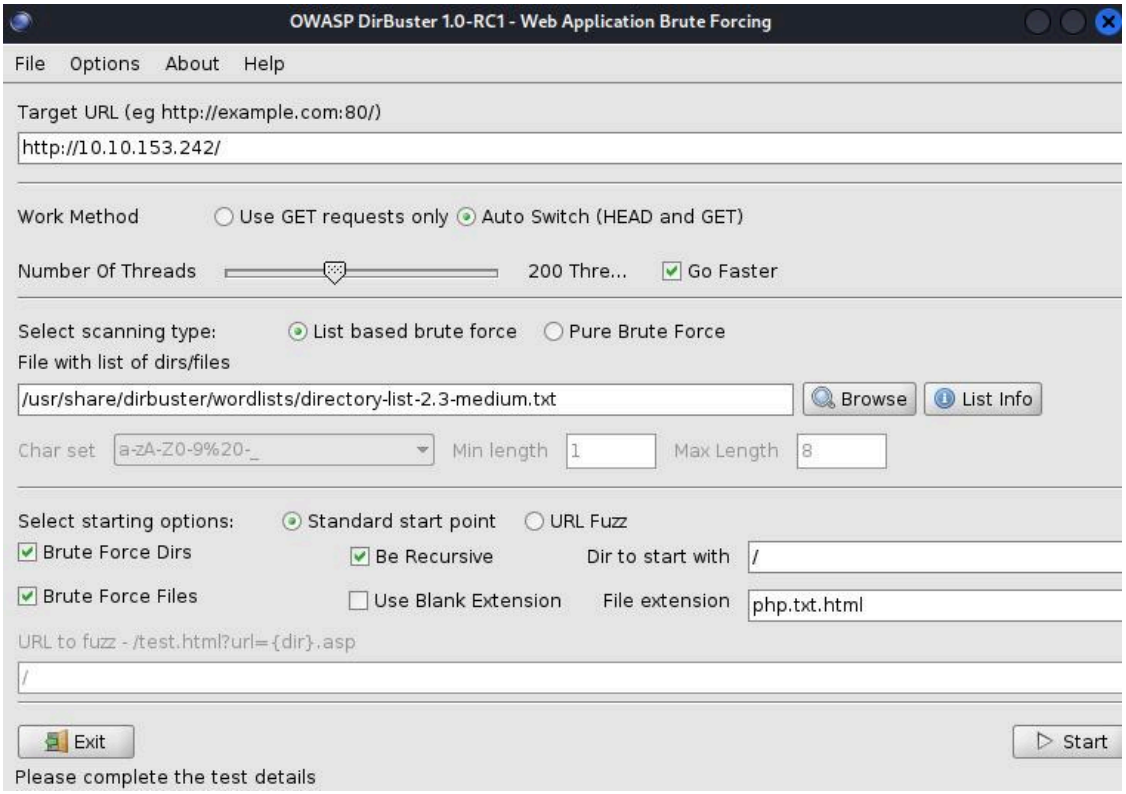


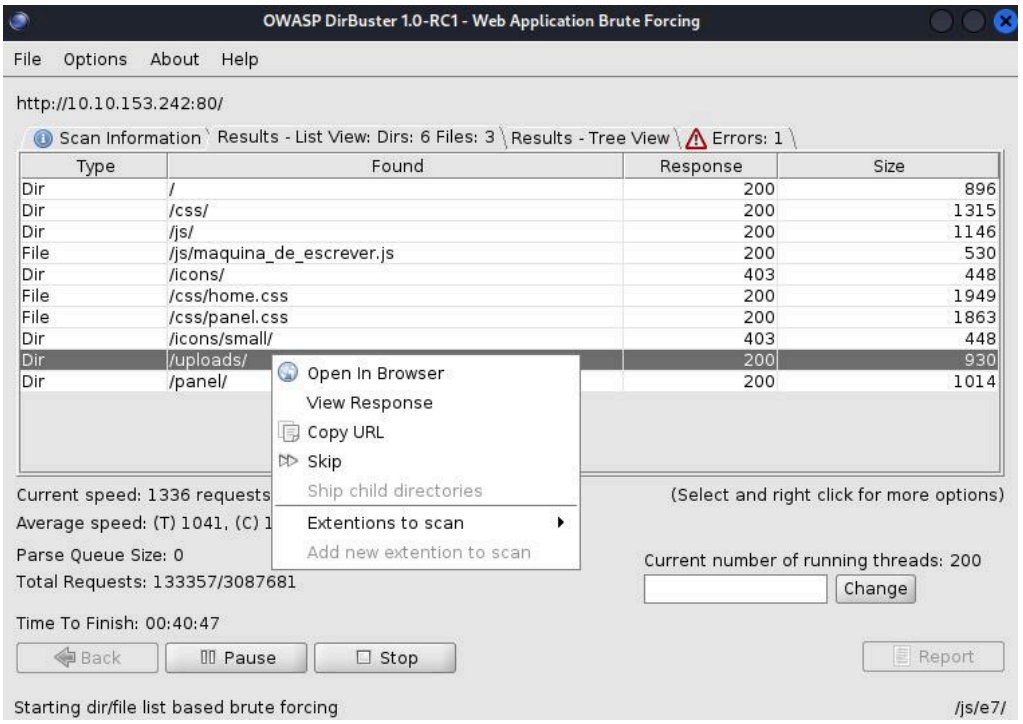
Directory Discovery with DirBuster
For the next tasks, open the **DirBuster** tool and perform a scan based on the instructions in the image:

After clicking **Start**, you will find the `panel` and `uploads` directories.



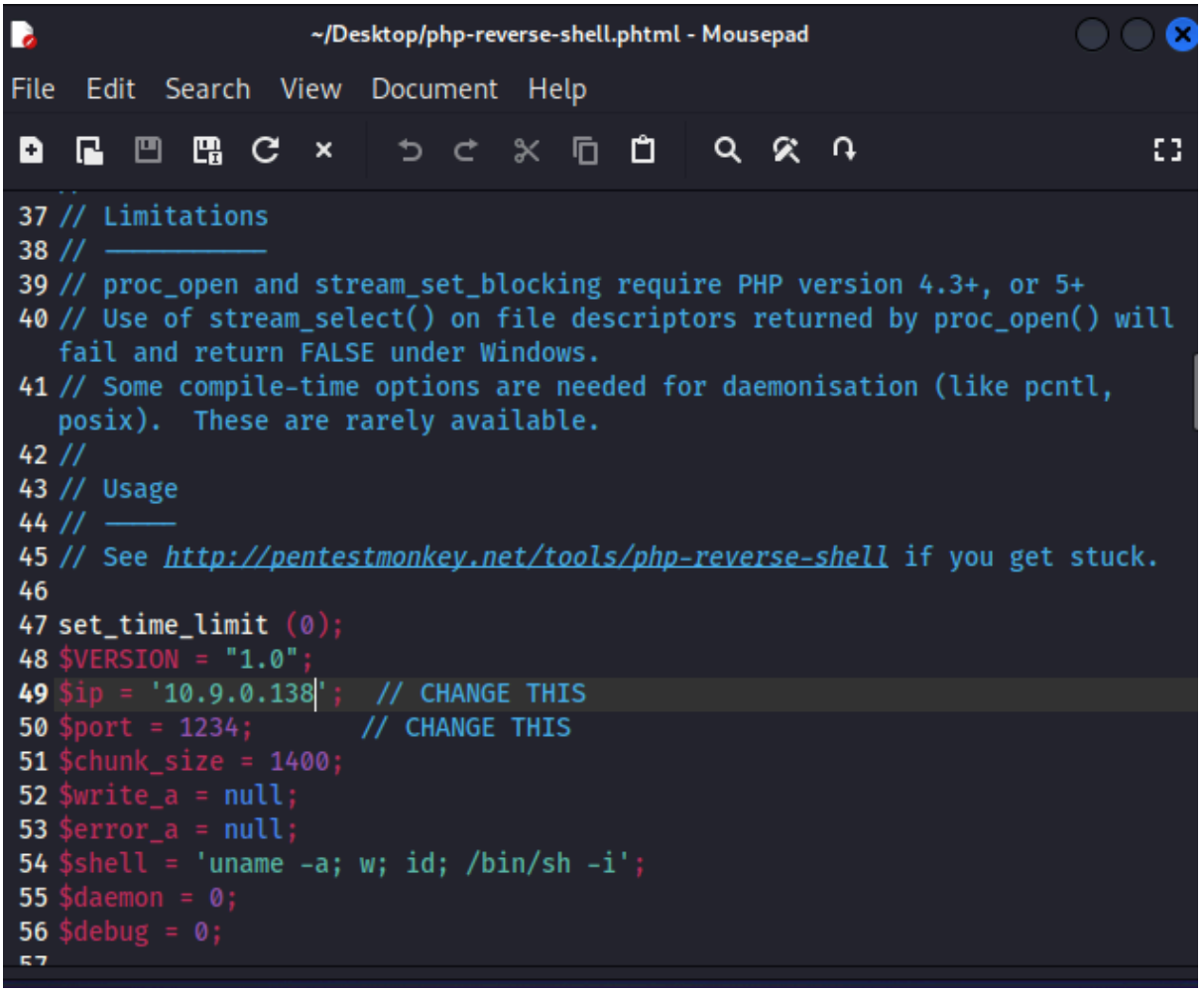



OWASP DirBuster 1.0-RC1 - Web Application Brute Forcing
File Options About Help
Target URL (eg http://example.com:80/)
http://10.10.153.242/
Work Method
Use GET requests only
Auto Switch (HEAD and GET)
Number Of Threads
200 Thre...
Go Faster
Select scanning type:
List based brute force
Pure Brute Force
File with list of dirs/files
/usr/share/dirbuster/wordlists/directory-list-2.3-medium.txt
Browse
List Info
Char set
a-zA-Z0-9%20-_
Min length
1
Max Length
8
Select starting options:
Standard start point
URL Fuzz
Brute Force Dirs
Be Recursive
Dir to start with
/
Brute Force Files
Use Blank Extension
File extension
php.txt.html
URL to fuzz - /test.html?url={dir}.asp
/
Exit
Start
Please complete the test details


OWASP DirBuster 1.0-RC1 - Web Application Brute Forcing
File Options About Help
http://10.10.153.242:80/
Scan Information
Results - List View: Dirs: 6 Files: 3
Results - Tree View
Errors: 1
Type Found Response Size
Dir / 200 896
Dir /css/ 200 1315
Dir /js/ 200 1146
File /js/maquina_de_escrever.js 200 530
Dir /icons/ 403 448
File /css/home.css 200 1949
File /css/panel.css 200 1863
Dir /icons/small/ 403 448
Dir /uploads/ 200 930
Dir /panel/ 200 1014
Open In Browser
View Response
Copy URL
Skip
Ship child directories
Extentions to scan
Add new extention to scan
Current speed: 1336 requests
(Select and right click for more options)
Average speed: (T) 1041, (C) 1
Parse Queue Size: 0
Total Requests: 133357/3087681
Current number of running threads: 200
Change
Time To Finish: 00:40:47
Back
Pause
Stop
Report
Starting dir/file list based brute forcing
/js/e7/

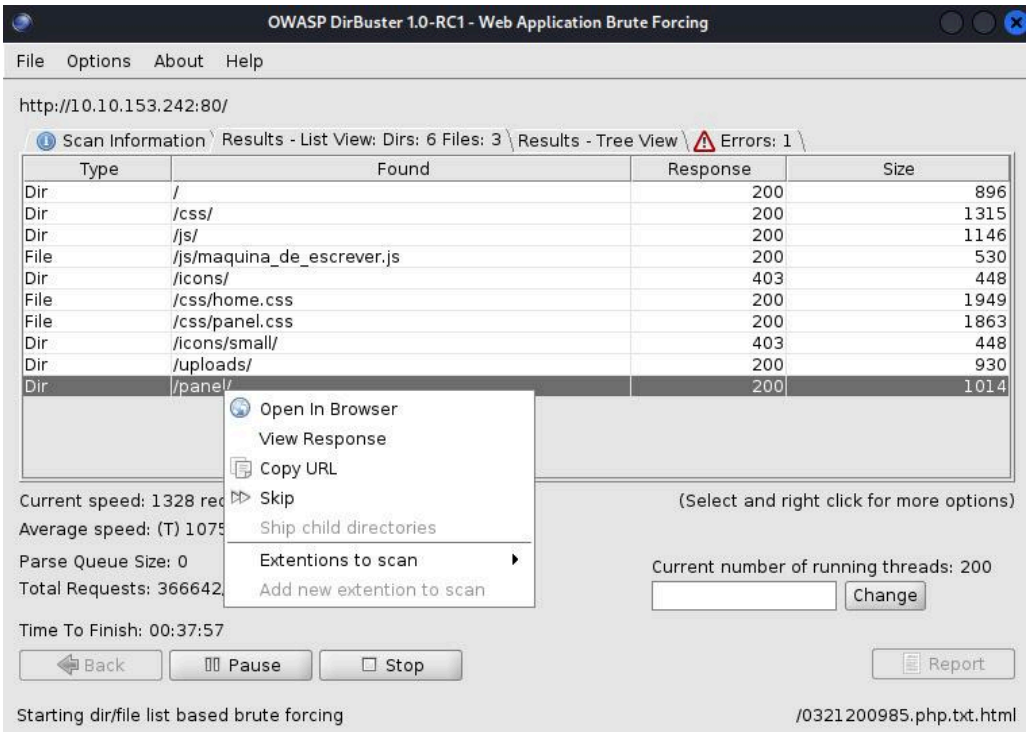
Exploiting Web Shell Upload via `shell.phtml`

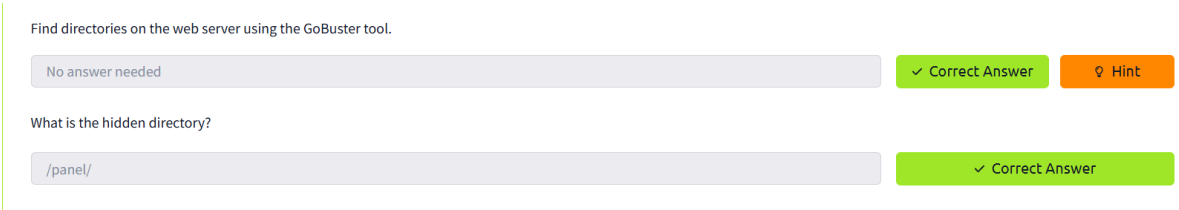
Open these directories in your browser and upload the pre-prepared **shell.phtml** file to the `uploads` folder. In the file, specify your **IP address** and **port number** for establishing a connection.



Using DirBuster to Discover Hidden Directory

DirBuster was used to perform a brute force scan for hidden directories and files on the server. By applying an appropriate wordlist, the scan revealed a hidden directory: `/panel/`. This directory potentially contains tools for server management or additional functionality.

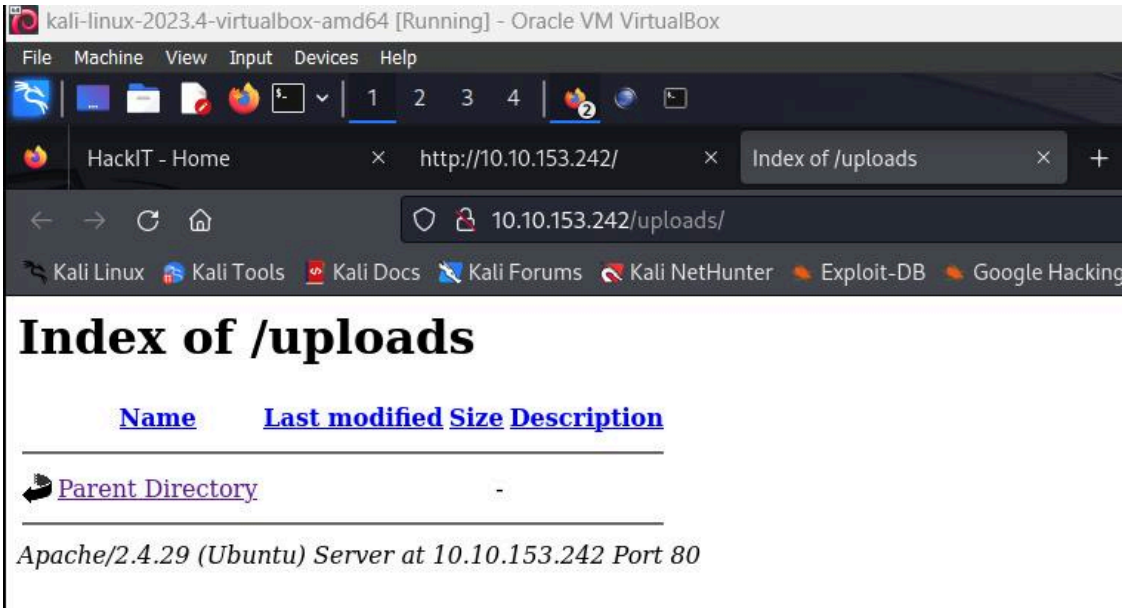


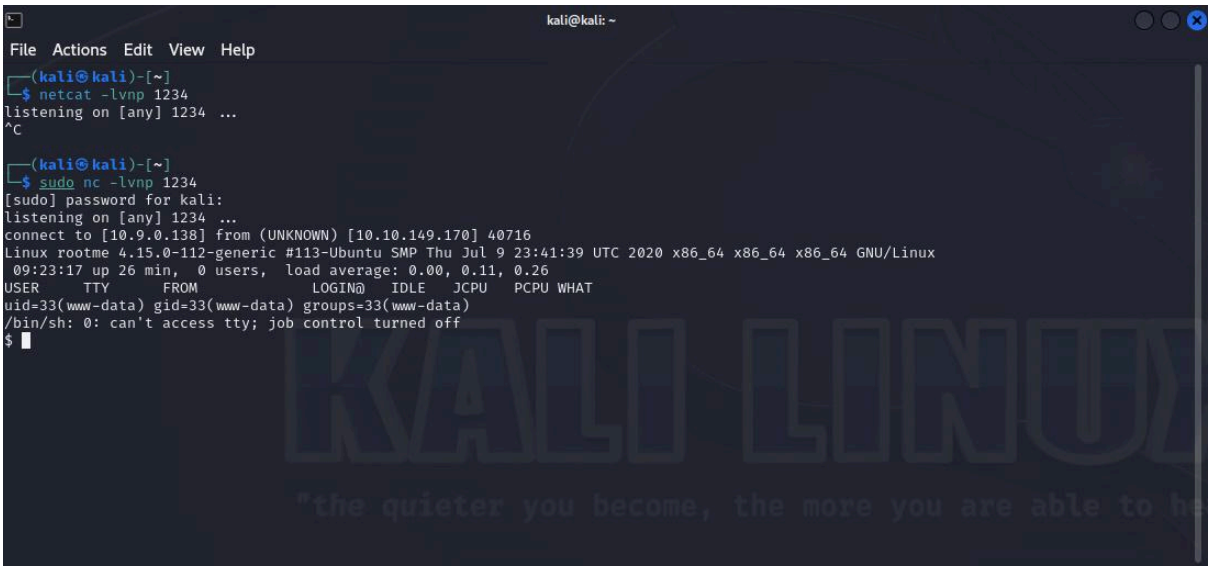


Shell Access through Uploaded File

In the terminal, execute the commands shown in the image.

Navigate to the `uploads` page in your browser and click on the uploaded file. This action establishes a shell connection to your computer.

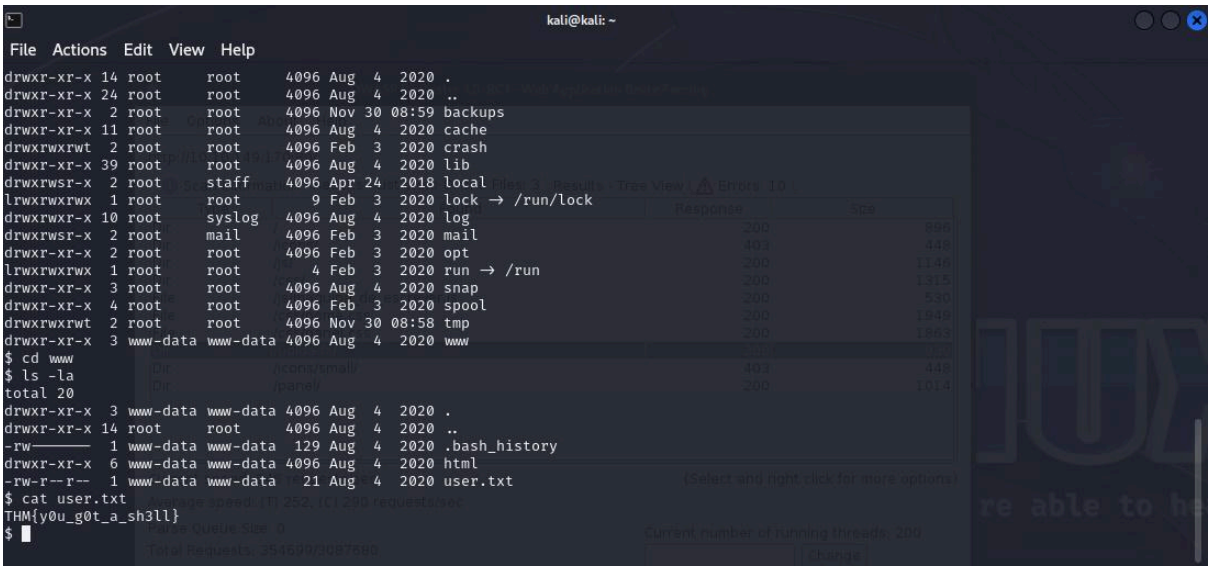


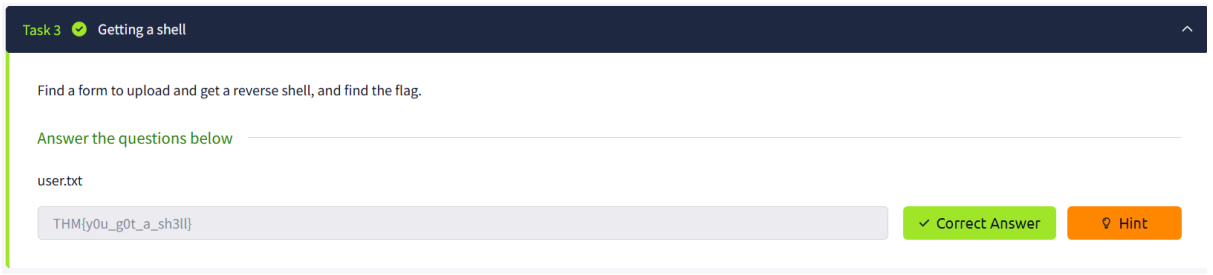


Finding and Reading User.txt File

Use the appropriate command in the terminal as shown in the image to locate the **user.txt** file.

Open the file with the `cat` command and copy its contents into the answer field.

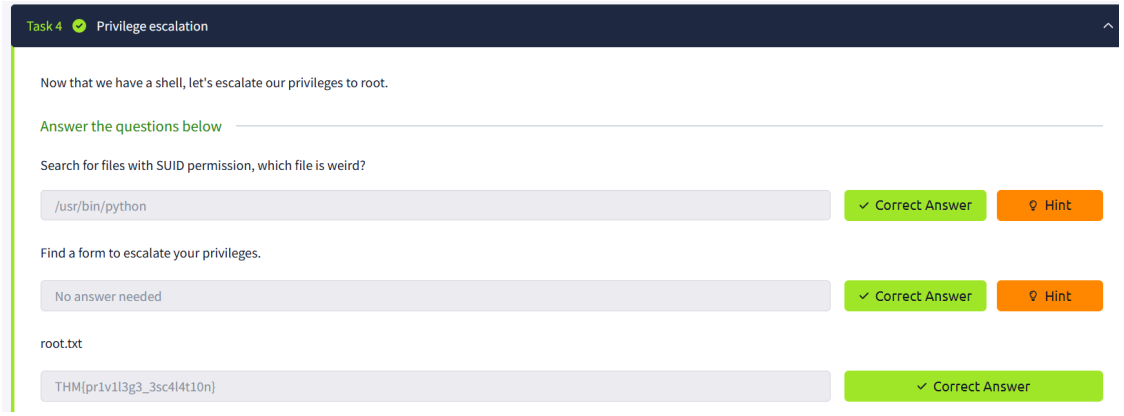




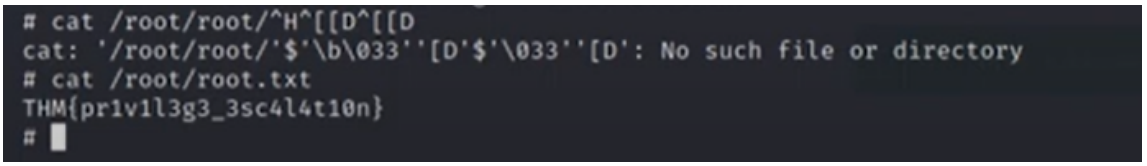
Privilege Escalation using SUID Files

Search for SUID files using the command provided in the image.

Use the identified file for the next steps in privilege escalation.

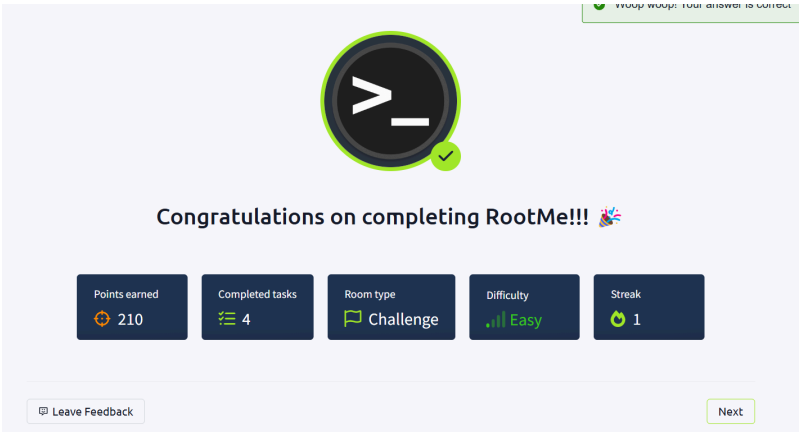


Next, we locate the `root.txt` file, which contains the flag. Using appropriate commands, we access the file, extract its contents, and submit the flag as the final answer.



Successfully Completing the RootMe Challenge

Finally, we have successfully completed the RootMe challenge, having found and submitted the flag from the `root.txt` file.



Creating a Simple Python Network Scanner Similar to Nmap

Main Program Structure (`main()` function)

The program starts by accepting a domain name or IP address input from the user. It then performs the following actions:

```
host = input("Enter the host (IP or domain): ")
try:
    ip = resolve_host(host)
    print(f"Resolved IP: {ip}")
```

The domain name is resolved to an IP address using the `resolve_host` function.

```
ping_host(host)
```

A ping request is made to check if the host is reachable using the `ping_host` function.

```
ports = [80, 443, 53, 123]
scan_tcp_ports(ip, ports)
scan_udp_ports(ip, ports)
```

TCP (80, 443) and UDP (53, 123) ports are scanned using the `scan_tcp_ports` and `scan_udp_ports` functions.

```
os_info = detect_os(ip)
print(f"Detected OS: {os_info}")
```

The operating system is detected based on the TTL (Time to Live) value using the `detect_os` function.

Port Scanning (`scan_tcp_ports` and `scan_udp_ports`)

These functions are responsible for scanning TCP and UDP ports to check whether they are open or closed.

TCP Port Scanning

```python
def scan_tcp_ports(ip, ports):
    print(f"Scanning TCP ports for {ip}...")
    for port in ports:
        try:
            with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
                s.settimeout(1)
                result = s.connect_ex((ip, port))
                if result == 0:
                    print(f"Port {port} is open")
                else:
                    print(f"Port {port} is closed")
        except Exception as e:
            print(f"Error scanning port {port}: {e}")
```

> TCP ports are scanned using sockets. If `connect_ex` returns 0, the port is open; otherwise, it is closed.

UDP Port Scanning

```python
def scan_udp_ports(ip, ports):
    print(f"Scanning UDP ports for {ip}...")
    for port in ports:
        try:
            with socket.socket(socket.AF_INET, socket.SOCK_DGRAM) as s:
                s.settimeout(1)
                result = s.sendto(b"", (ip, port))
                if result:
                    print(f"Port {port} is open")
                else:
                    print(f"Port {port} is closed")
        except Exception as e:
            print(f"Error scanning UDP port {port}: {e}")
```

UDP ports are scanned by sending empty data to check if the port responds.

Ping Request (`ping_host`)

A ping request is made to the user-input IP address or domain to check if the system is reachable and to measure the Round-Trip Time (RTT).

```
def ping_host(host):
    command = ["ping", host, "-n", "3"] if platform.system().lower() == "windows" else ["pi
    try:
        process = subprocess.Popen(command, stdout=subprocess.PIPE, stderr=subprocess.PIPE)
        for line in iter(process.stdout.readline, b""):
            decoded_line = line.decode("utf-8").strip()
            if "time=" in decoded_line.lower():
                print(decoded_line)
        process.stdout.close()
        process.wait()
    except Exception as e:
        print(f"An error occurred while pinging: {e}")
```

A ping request is executed via `subprocess.Popen`. This request is made three times, as 10 pings would be too many.

Operating System Detection (`detect_os`)

This function detects the operating system based on the TTL value obtained from the ping response.

```
def detect_os(ip):
    command = ["ping", ip, "-n", "1"] if platform.system().lower() == "windows" else ["pin
    try:
        process = subprocess.Popen(command, stdout=subprocess.PIPE, stderr=subprocess.PIPE
        output, _ = process.communicate()
        if process.returncode != 0:
            return "Error: Unable to execute ping."

        for line in output.decode("utf-8").splitlines():
            if "ttl=" in line.lower():
                ttl = int(line.split("ttl=")[1].split()[0])
                if ttl <= 64:
                    return "Linux/Unix-based OS"
                elif ttl <= 128:
                    return "Windows"
                elif ttl <= 255:
                    return "Network device (e.g., Cisco Router)"
                else:
                    return "Unknown OS"
        return "Error: TTL value not found in ping response."
    except Exception as e:
        return f"An error occurred during OS detection: {e}"
```

The operating system is identified based on the TTL (Time to Live) value. The TTL value indicates whether the system is running Linux, Windows, or another type of device.

Domain Name to IP Address Resolution (`resolve_host`)

This function resolves a domain name to an IP address using the `socket.gethostbyname` method.

```python
def resolve_host(host):
    try:
        return socket.gethostbyname(host)
    except socket.gaierror:
        raise ValueError(f"Error: Unable to resolve host '{host}'.")
```

The domain name is resolved to an IP address using the `gethostbyname` method.

Terminal Output of Python Network Scanner

The following terminal output demonstrates the execution of the Python network scanner script when targeting `google.com`. The program first resolves the IP address of the host, pings it to check connectivity, scans both TCP and UDP ports, and finally attempts to detect the operating system based on TTL values.

Ping Test: The program performs a simple ping test to `google.com` and reports the round-trip time (RTT) for each packet.

TCP Port Scanning: The script checks the common TCP ports (80 and 443) for their availability. In this case, both are open.

UDP Port Scanning: It checks the UDP ports (53 and 123) for availability, reporting them as closed.

OS Detection: The TTL (Time-to-Live) value from the ping response is used to make an educated guess about the operating system. In this case, it identifies the system as Windows