

A REPORT

ON

**WEB APPLICATION SECURITY AND PENETRATION TESTING
BASED ON OWASP**

BY

Name of the student :

Bibek Bhandari

Registration No.

AP22110011483

Prepared in the partial fulfillment of the
Summer Internship Course

AT

SRM University AP



SRM UNIVERSITY, AP

(July, 2024)

Internship Completion Certificate

Date:

I hereby certify that the project titled “**Web Application and Penetration Testing Based on OWASP**” which is submitted by Bibek Bhandari (AP22110011483) for the partial fulfillment of the Summer Internship Course is a record of the project work carried out by him under my guidance & supervision. To the best of my knowledge, this work is original and has not been submitted previously elsewhere.

.....

Dr. Sriramulu Bojjagani

Assistant Professor

Department of Computer Science and Engineering

SRM University AP, Andhra Pradesh

JOINING REPORT

Name of the Student	Bibek Bhandari
Roll No	AP22110011483
Programme (BTech/ BSc/ BA/MBA)	BTECH
Branch	CSE
Name and Address of the Internship Company [For research internship, it would be SRMAP]	SRM University AP Mangalagiri Mandal, Guntur, Andhra Pradesh
Period of Internship	15 June – 30 July

I hereby inform that I have joined the summer internship on 12th June 2024 for the In-plant Training/ Research internship in the industry.

Date : 12-07-2024


 Signature of the Student

CERTIFICATE FROM INDUSTRY MENTOR/HR (FACULTY MENTOR FOR RESEARCH INTERNSHIP)

Certified that the above-mentioned student has joined our organization for the INTERNSHIP / INDUSTRIAL TRAINING / ACADEMIC ATTACHMENT in the industry / Organization.

Name of the Industry Mentor	Dr. Sriramulu Bojjagani
Designation	Assistant Professor
Phone No (If any)	9000303647
Email	Sriramulu.b@srmap.edu.in
Signature & Date	

Acknowledgements

I would like to express my deepest gratitude to the following individuals who have supported and guided me throughout my internship. I am profoundly thankful to **Prof. Manoj K Arora**, Vice Chancellor of SRM University, for providing the opportunity and resources necessary for the completion of this internship. I also extend my sincere thanks to **Prof. CV Tomy**, Dean of School of Engineering and Sciences (SEAS) at SRM University AP. Finally, I am deeply grateful to **Dr. Sriramulu Bojjagani**, Faculty Mentor and Assistant Professor at SRM University, for his continuous support, mentorship, and encouragement. His feedback and direction were crucial in shaping my research and achieving the objectives of the internship.

I acknowledge with gratitude all those who have directly or indirectly contributed to the completion of this project. Your support and encouragement have been immensely appreciated.

Abstract

In today's ever-changing world of cybersecurity, keeping web applications safe remains a key worry as threats become more complex. The following research project tries to delve into the issues of web application security based on OWASP guidelines for identifying, assessing, and mitigating vulnerabilities within a web application. This research will give practical examples on the OWASP Top 10 2021 vulnerabilities and employ various penetration testing tools available in Kali Linux in detecting and addressing these vulnerabilities.

This literature collection thus informs the methodology and approaches taken in our research to ensure everything pegs under standard industry practices. Practical applications of the study are underlined through hands-on experience with tools like Burp Suite, used to address problems in Damn Vulnerable Web Application. Major vulnerabilities explored in this research will include Cross-Site Request Forgery, Command Injection, Brute Force Attack, XSS, DOM, and SQL Injection, all in relation to the application in the real world and how to remediate them.

These results are important and useful to both the academic community and practitioners. They provide a sound framework for understanding VAPT within web applications, hence adding value to the insights obtained by cybersecurity practitioners and researchers who seek to improve the security postures of information systems based on the Web.

Table of contents

1.	INTRODUCTION.....	7
2.	MAIN TEXT	9
2.1	ASSUMPTIONS MADE	9
2.2	EXPERIMENTAL WORK/DATA COLLECTION.....	9
2.3	DESCRIPTION OF ACTIVITIES	9
2.3.1	Reflected Cross-Site Scripting (XSS):.....	9
2.3.2	Cross-Site Request Forgery (CSRF):	10
2.3.3	Command Injection:.....	11
2.3.4	SQL Injection:.....	11
2.3.5	Brute Force Attack:	13
3.	RESULTS OBTAINED.....	14
4.	OUTCOMES:.....	17
5.	CONCLUSION:	19
6.	REFERENCES:	20

1. Introduction

Web applications have quickly found their place within business operations and personal activities in the wake of digital transformation. The increased reliance on web technologies, however, has opened up applications to a host of security threats. Malicious actors exploit these vulnerabilities to gain unauthorized access to confidential information, making off with losses in finance and reproach in character. Security assurance of web applications thus holds much more significance than it ever did.

The Open Web Application Security Project aids developers and security experts through detailed documents and tools to detect vulnerabilities in a web application. OWASP mainly concentrates on the Top 10 list, highlighting the most relevant security issues that need improvements for the security of a web application. The OWASP Top 10 for 2021:



Figure 1.1 : OWASP Top 10 Web Vulnerabilities 2021

Even with these resources at one's disposal, most web applications remain vulnerable because the security measures in place have not been either properly put into action or tested. The DVWA, Damn Vulnerable Web Application, is a resource used as a means to learn about and practice various web application vulnerabilities in a controlled environment.

The focus of this report shall be the domain of web application security, based on the OWASP standards. Much emphasis is laid on hands-on experiences with practical applications for these guidelines. This project looks at implementing some of the OWASP Top 10 2021 vulnerabilities and demonstrating efficient techniques on finding, assessing, and mitigating such risks using the tools available within Kali Linux.

Excluding the outline and appendix, over these years, so has the OWASP Top 10 evolved to show changes in the landscape of web application security. New threats are found; older ones develop new vectors, and priorities of security professionals shift. Trend analysis expresses the most important changes and developments in Web Application Security, reflecting why it is important for organizations to keep current with best practices and emerging threats.

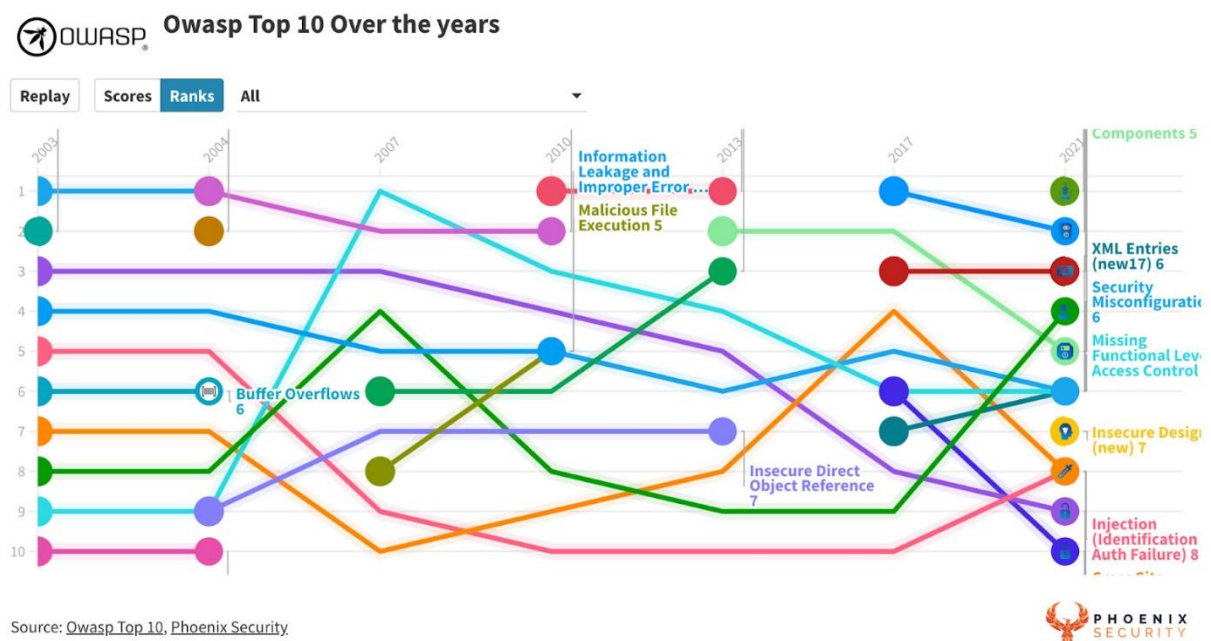


Figure 1.2 : OWASP Top 10 Web Vulnerabilities Over the years

2. Main Text

2.1 Assumptions Made

For the analysis of various vulnerabilities in DVWA, it was assumed that the vulnerable application is very practical in exposing common web security vulnerabilities for the sake of conducting analysis on various vulnerabilities in DVWA, this research also assumed that the system is perfectly configured for all levels of difficulty. All vulnerabilities, as well as different protections, were assumed to work perfectly.

2.2 Experimental Work/Data Collection

In the research performed, several vulnerabilities were tested in DVWA: Brute Force Attack, Reflected XSS, Cross-Site Request Forgery, Command Injection, and SQL Injection. All of these vulnerabilities were tested at different levels of difficulty. The idea was to see how changing security measures would affect these attacks' successes. The testing included attempts at attacks with automated tools and manual testing of their results.

2.3 Description of Activities

2.3.1 Reflected Cross-Site Scripting (XSS):

- **Objective:** To find and analyze vulnerabilities associated with reflected cross-site scripting, where user input is reflected in the response from the application without proper sanitization.

Medium Reflected XSS Source

```
<?php
header ("X-XSS-Protection: 0");

// Is there any input?
if( array_key_exists( "name", $_GET ) && $_GET[ 'name' ] != NULL ) {
    // Get input
    $name = str_replace( '<script>', '', $_GET[ 'name' ] );

    // Feedback for end user
    echo "<pre>Hello {$name}</pre>";
}

?>
```

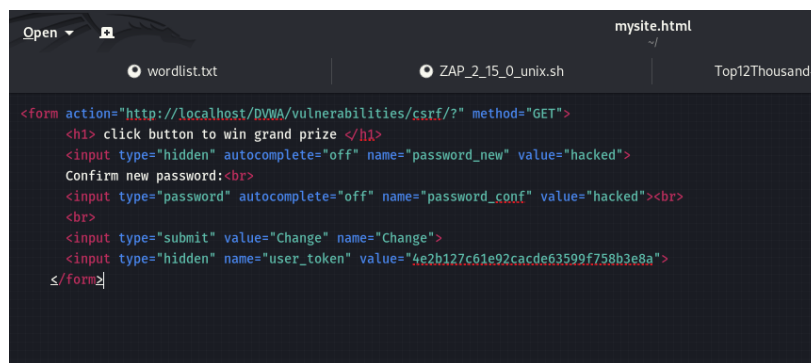
Figure 2.1 : Reflected XSS Source Code in DVWA

- **Activities:**
 - **Form Input Testing:** Entered unique strings into input fields on the DVWA application
 - **Response Analysis:** The application handling of the input has been observed through the source view of the page in the browser itself by pressing CTRL+U. The searched for a unique string in the page source using CTRL+F and looked for that string to be in the response for all levels.
 - **Payload Injection:** Injected known XSS payloads into input fields to test for vulnerabilities at various security levels:

- **Low-Level Security:** Payload injected
`<script>alert(document.cookie)</script>`.
 - **Observation:** Executed JavaScript alert showing cookies.
- **Medium-Level Security:** Payload used `<Script>alert("hello you are hacked")</Script>`.
 - **Observation:** The payload was executed because script tags had been replaced with null in the source code.
- **High-Level Security:** Payload used ``.
 - **Observation:** Since all the instances of the tag `<script>` were denied, a different payload that used the event `onerror` inside the element `` was injected and triggered the JS alert.
- **Testing for Reflection:** It checked whether the injected payloads were reflected and executed in the browser's response, hence a vulnerability.

2.3.2 Cross-Site Request Forgery (CSRF):

- **Objective:** This section tests if the application has any chances of a CSRF attack. This is an attacker-induced action performed on behalf of the user.



```

<form action="http://localhost/DVWA/vulnerabilities/csrf/?" method="GET">
  <h1> click button to win grand prize </h1>
  <input type="hidden" autocomplete="off" name="password_new" value="hacked">
  Confirm new password:<br>
  <input type="password" autocomplete="off" name="password_conf" value="hacked"><br>
  <input type="submit" value="Change" name="Change">
  <input type="hidden" name="user_token" value="4e2b127c61e92cacde63599f758b3e8a">
</form>
  
```

Figure 2.2 : Modified HTML file to simulate CSRF attack

- **Activities:**
 - **Form Modification:** The form was copied from the source code of the DVWA and pasted into a text editor and modified. Logged in to the DVWA, then tampered with the form to do something it wasn't meant to do—for instance, change a user's password.
 - **Simulating Attack:** Saved the modified HTML file and opened it in a web browser. Submitted the form to simulate a CSRF attack, trying to change the user's password without authentication.
 - **Request Observation:** Determine whether or not the application accepted that unauthorized request and performed the action as programmed by the altered form, such as in this case, changing the password.

2.3.3 Command Injection:

- **Objective:** Identify the possibility of command injection vulnerabilities; that is, the execution of operating system commands injected as part of the user input.

```
<?php
if( isset( $_POST[ 'submit' ] ) ) {
    $target = $_REQUEST[ 'ip' ];
    // Remove any of the characters in the array (blacklist).
    $substitutions = array(
        '&&' => '',
        ';' => '',
    );
    $target = str_replace( array_keys( $substitutions ), $substitutions, $target );
    // Determine OS and execute the ping command.
    if (strcasecmp(substr(php_uname('s'), 0, 10), 'Windows NT')) {
        $cmd = shell_exec( 'ping ' . $target );
        echo '<pre>'.$cmd.'</pre>';
    } else {
        $cmd = shell_exec( 'ping -c 3 ' . $target );
        echo '<pre>'.$cmd.'</pre>';
    }
}
?>
```

Figure 2.3 : Command Injection Source Code in DVWA

- **Activities:**
 - **Initial Testing:** The functioning of the application was tested by providing regular IP addresses within the "Ping a device" section of DVWA.
 - **Payload Injection:** Injected additional commands in the IP address field, such as *127.0.0.1; whoami*, and analyzed if arbitrary commands could be executed. The following payloads were used to identify sensitive system files, like *127.0.0.1; cat /etc/passwd*.
 - **Response Analysis:** Noticed that the application is disclosing system information or performing other undesired actions based on the injected commands.

2.3.4 SQL Injection:

- **Objective:** The test was conducted in order to find out if the application was vulnerable to SQL injection attacks, whereby malicious SQL commands were executed in order to gain unauthorized access or manipulate data stored in the database.
- **Activities:**
 - **Initial Testing:** The application's USER ID page was visited and a valid user ID was entered to check whether the application is working fine and whether

there is any data retrieval. Therefore, from the above code, it is clear that the variable \$id is retrieved from the user's input without validation/sanitization, and an attacker may manipulate \$id and can inject malicious SQL code.

- **Payload Injection:** Injected test payloads of SQL injection into the application, such as 2' OR '1'='1' --, to modify the SQL queries and notice if more data was being accessed. The payloads were used in exploring the database and retrieved more information.
- **Data Extraction:** Observed the results to see if the application returns more data than intended, like all the user records, thus indicating a successful SQL Injection attack.

The vulnerability arises from directly concatenating user input into the SQL query without proper sanitization or parameterization. The code snippet illustrates this flaw:

```
$id = $_REQUEST['id'];  
$query = "SELECT first_name, last_name FROM users WHERE user_id =  
'$id';";
```

This allows an attacker to manipulate \$id and inject malicious SQL code, potentially leading to unauthorized access, data leakage, or complete data loss.

Example Exploits:

- **Low Level Attack:** The query executed by the database could be

```
1' OR '1'='1'#  
'UNION SELECT table_name, NULL FROM information_schema.tables --  
'UNION SELECT column_name, NULL FROM information_schema.columns WHERE  
    table_name= 'users' --  
'UNION SELECT user, password FROM users --
```

manipulated as follows:

- **Medium Level Attack:** Intercept the request and send it to the repeater. Edit id=1 to the following code and observe the response:

```
1 UNION SELECT user, password FROM users --
```

- **High Level Attack:** After clicking the “here to change your ID” link, insert the following malicious code into the input field and submit:

```
' UNION SELECT user, password FROM users --
```

By exploiting these vulnerabilities, we can obtain usernames and encrypted passwords from the database.

2.3.5 Brute Force Attack:

- **Objective:** To determine whether the application is protected against brute-force attacks involving large numbers of password guesses to gain access.

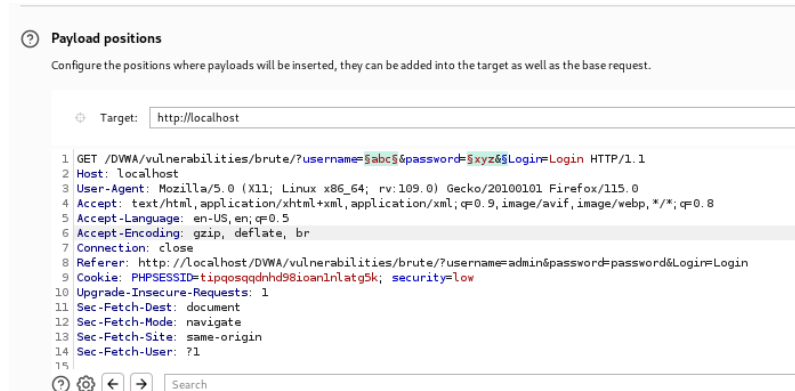


Figure 2.4 : Configuring positions of payload into the target in Burp Suite

- **Activities:**
 - **Setup:** Configured Burp Suite to intercept and analyze HTTP requests. Set up a proxy listener in Burp Suite and configured the browser to use this proxy.
 - **Password Cracking:** Hydra was used to carry out the Brute Force Attack. This is done on the login page and attacked with a list of common passwords. The tool was set up to fire login requests, analyzing the response for successful authentication.
 - **Token Handling:** Handling of Unique User Token generated with every login attempt. Capturing and re-use of this token along with usernames and passwords in following requests to simulate the Brute Force Attack accurately.
 - **Request Analysis:** Monitored the results to analyze the efficiency of this brute-force attack, looking through application responses to different login attempts.

3. Results Obtained

- Brute Force Attack Results:**
 The test confirmed that DVWA was highly vulnerable to brute force attacks at the low level, where no protections were implemented. The medium level provided some mitigation with delayed responses, but persistent attackers could still succeed. The high level, with anti-CSRF tokens and randomized delays, improved security but was not foolproof. Stronger defenses like CAPTCHAs and rate limiting were recommended.

Request	Payload 1	Payload 2	Status code	Error	Timeout	Length
66	matrix	password	200	<input type="checkbox"/>	<input type="checkbox"/>	4620
50	admin	password	200	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	4663
0			200	<input type="checkbox"/>	<input type="checkbox"/>	4620
2	bjjb@@@	password	200	<input type="checkbox"/>	<input type="checkbox"/>	4620
4	password	password	200	<input type="checkbox"/>	<input type="checkbox"/>	4620
6	12345678	password	200	<input type="checkbox"/>	<input type="checkbox"/>	4620
8	qwerty	password	200	<input type="checkbox"/>	<input type="checkbox"/>	4620
10	111111	password	200	<input type="checkbox"/>	<input type="checkbox"/>	4620
12	1234567	password	200	<input type="checkbox"/>	<input type="checkbox"/>	4620
14	1q2w3e4r	password	200	<input type="checkbox"/>	<input type="checkbox"/>	4620
16	654321	password	200	<input type="checkbox"/>	<input type="checkbox"/>	4620
18	1234	password	200	<input type="checkbox"/>	<input type="checkbox"/>	4620
20	666666	password	200	<input type="checkbox"/>	<input type="checkbox"/>	4620

Figure 3.1 : Password Cracked with Brute Force Approach

- Reflected XSS Results:**
 When the payload `<script>alert(document.cookie)</script>` was executed in DVWA's XSS reflected vulnerability:
 - Low-Level Security: Displayed a JavaScript alert box with the content of document.cookie.
 - Medium-Level Security: Displayed a JavaScript alert box with the content of document.cookie, but the `<script>` tags were altered or nullified.
 - High-Level Security: No alert was displayed as the payload was rejected.

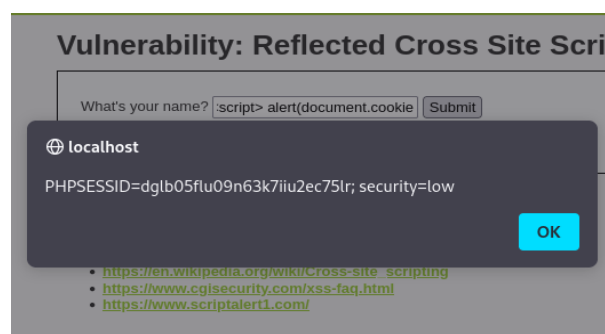


Figure 3.2 : Web Cookie obtained indicating Reflected XSS vulnerability

- **CSRF Results:**

The test showed that CSRF vulnerabilities could be exploited to perform unauthorized actions, such as changing a user's password. Implementing anti-CSRF tokens and validating request origins were identified as essential measures to prevent such attacks.

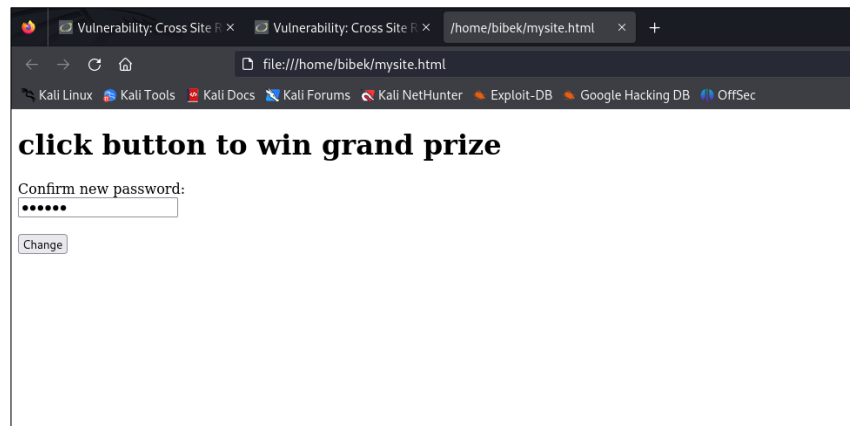


Figure 3.3 : Simulated a CSRF attack, attempting to change the user's password without proper authentication

- **Command Injection Results:**

Command Injection was successfully exploited, allowing the execution of arbitrary system commands. The ability to read sensitive files from the server demonstrated the severity of this vulnerability. The recommended mitigation included avoiding OS command execution and sanitizing user inputs.

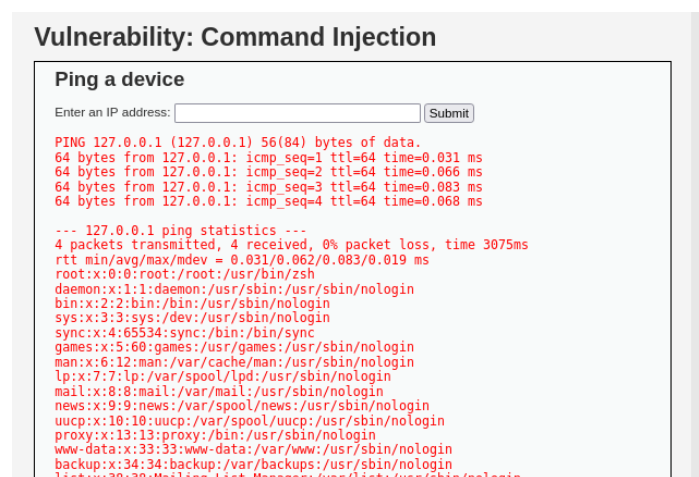


Figure 3.4 : Access to sensitive files using payload 127.0.0.1; cat /etc/passwd

- **SQL Injection Results:**
SQL Injection vulnerabilities were successfully exploited, revealing all user data from the database. The use of parameterized queries and input validation was identified as critical for preventing SQL Injection attacks.

Vulnerability: SQL Injection

User ID:

```
ID: ' UNION select user, password from users -- -  
First name: admin  
Surname: 5f4dcc3b5aa765d61d8327deb882cf99  
  
ID: ' UNION select user, password from users -- -  
First name: gordonb  
Surname: e99a18c428cb38d5f260853678922e03  
  
ID: ' UNION select user, password from users -- -  
First name: 1337  
Surname: 8d3533d75ae2c3966d7e0d4fcc69216b  
  
ID: ' UNION select user, password from users -- -  
First name: pablo  
Surname: 0d107d09f5bbe40cade3de5c71e9e9b7  
  
ID: ' UNION select user, password from users -- -  
First name: smithy  
Surname: 5f4dcc3b5aa765d61d8327deb882cf99
```

Figure 3.5 : Manipulated \$id and injected malicious SQL code to get all user IDs and Passwords

4. Outcomes:

1. Brute Force Attack:

- **Low Level:** With no protection mechanisms in place, this opens up the opportunity for unlimited login attempts and is highly susceptible to brute force attacks. It lacks throttle or lockout of an account opened to a kind of password guessing attack by the attacker with automated tools.
- **Medium Level:** The introduction of a fixed delay, 2 seconds between failed log in attempts, resulted in slowing down the brute force attack. However, it did not prevent it from happening. It simply allowed attackers to still brute force, although at an extremely reduced speed as compared to a low-level scenario.
- **High Level:** Further complicating this was the inclusion of a unique user token, which changes for every login attempt, and randomised delays of 2 to 4 seconds before responding to requests. This put in an additional layer of security, making the attack very hard and extremely time-consuming; however, it did not totally drive out the risk. This added complexity in capturing and utilisation of user token with every request proved the requirement of even more tightening of security measure.
- **Mitigation Observed:** Implementing account lockout mechanisms after a certain number of attempts to login significantly reduce the chances of brute force attacks. Additionally, rate limiting on login attempts from the same IP address and CAPTCHA to identify whether a user is a human or bot are highly critical mitigation measures for brute force vulnerabilities.

2. Cross-Site Request Forgery (CSRF):

- The CSRF vulnerability essentially allowed unauthorized actions to be performed in the name of authenticated users without their consent. Trick a user into submitting a form with malicious requests, and we could change user settings or perform some other unintended action.
- **Mitigation Observed:** This risk was mitigated by introducing the Anti-CSRF token, validation of request whether it is coming from valid users. Validation of the request and whether it is coming from valid users are the important things in mitigating CSRF vulnerabilities.

3. XSS (Reflected):

- Reflected XSS vulnerabilities were demonstrated by injecting malicious scripts in form fields and observe execution in a browser. Successful execution of payloads like `<script>alert("hacked")</script>` or ``, proved the vulnerability of the application to reflected XSS attacks.
- **Mitigation Observed:** It is thus that proper input validation and escaping user inputs are essential in the prevention of XSS attacks. It is also important in the mitigation of this vulnerability that the data reflected in the responses does not contain the executable script or other harmful payloads

4. Command Injection:

- This command injection vulnerability has been exploited by the injection of extra commands through user inputs. It is capable of executing any arbitrary command on the server. This has resulted in unauthorized access to sensitive information and probably control over the server.
- **Mitigation Observed:** Input validation at a stronger level and avoidance of the direct execution of system commands are the major strategies for preventing command injection attacks. Language-specific libraries dealing with system operations, coupled with sanitization of user input, are pretty useful.

5. SQL Injection:

- SQL injection attacks manipulated SQL queries so that various data could either be retrieved or modified from the database. Very often, a malicious SQL code allows access to sensitive information and makes it possible to modify records.
- **Overall Impact:** The exercises demonstrated the critical importance of implementing robust security measures across various vulnerability types. While some mitigations, such as rate limiting and token-based protections, helped add layers of defense, they certainly weren't infallible. Hence, full security practices—including the development of robust input validation, secure coding techniques, and advanced protection mechanisms like CAPTCHAs—will need to be in place to protect web applications against those vulnerabilities.
- **Mitigation Observed:** Security against SQL injection risks would involve parameterized queries, input sanitization. It's always critical to ensure that queries are securely constructed, and any user inputs are properly validated before execution is allowed.

5. Conclusion:

DVWA exposed vulnerabilities gave deep insight into most of the security weaknesses common web applications face. In-depth analysis of vulnerabilities such as Brute Force Attacks, Cross-Site Request Forgery, Reflected Cross-Site Scripting, Command Injection, and SQL Injection brought about important points concerning the nature of these threats and the effectiveness of several mitigation strategies.

These vulnerabilities, researched and practically exploited, gave valuable insight into the complexities of web application security. Each pointed out certain risks and proved that it's important to have comprehensive security measures in place. This goes on to prove that while rate limiting, input validation, and following secure coding practices are effective mitigations, they need to be part of a greater security strategy. Combining these measures with regular security assessments and updating procedures will help build robust defenses against emerging threats.

The researchers closed by stating that security in web applications is a process: it involves vigilance, following best practices, and continuous improvement. Developers and security experts who understand these vulnerabilities can protect the web applications from further attacks and keep user data safe. This study improved our understanding of the current threats and provided a foundation for future improvements in the security of web applications.

6. References:

- [1] Shah, S., & Mehtre, B. M. (2015). An overview of vulnerability assessment and penetration testing techniques. *Journal of Computer Virology and Hacking Techniques*, 11, 27-49.
- [2] Bojjagani, S., & Sastry, V. N. (2017, October). VAPTai: a threat model for vulnerability assessment and penetration testing of android and iOS mobile banking apps. In *2017 IEEE 3rd international conference on collaboration and internet computing (CIC)* (pp. 77-86). IEEE.
- [3] Cordella, A., Bononi, L., & Crin, F. (2018). Web application penetration testing: an analysis of a corporate application according to owasp guide-lines. *ALMA MATER STUDIOUM UNIVERSITY OF BOLOGNA*.
- [4] Qaderi, M., Sinha, G., & Sinha, D. K. (2023, October). Vulnerability Detection and Security Enhancing Using XAMPP, OWASP and DVWA. In *2023 International Conference on Computer Science and Emerging Technologies (CSET)* (pp. 1-4). IEEE.
- [5] Kumar, A., & Taterh, S. (2016). Analysis of various levels of penetration by SQL injection technique through DVWA'. *Journal of Advanced Computing and Communication Technologies*, 4(2), 28-32.
- [6] Sedek, K. A., Osman, N., Osman, M. N., & Kamaruzaman, J. H. (2009). Developing a Secure Web Application Using OWASP Guidelines. *Comput. Inf. Sci.*, 2(4), 137-143.