Timo Virtanen

HANDS-ON WEB SECURITY

Mitigating Vulnerabilities in DVWA

COMP.SEC.300

May 2025

ABSTRACT

Timo Virtanen: Hands-on web security
Tampere University
May 2025

This paper's purpose is getting hands on experience from securing vulnerable website. The damn vulnerable web application (DVWA) by digininja was chosen as the web application to secure in this paper. DVWA is purposefully vulnerable website for teaching and learning purposes.

DVWA has a list of vulnerabilities in column right side of the web site. Each vulnerability module or tab has information on how to exploit that vulnerability. To observe the code of each vulnerability, user should go to DVWA files, then to vulnerability folder and there each vulnerability is listed. Each vulnerability has 4 levels of security, and each one is listed in this folder as low, medium, hard and impossible. This paper mitigates four of low security vulnerabilities which are brute force, command injection, cross site request forgery and SQL injection.

Due to the scope of this assignment, only four vulnerabilities were resolved, and one vulnerability was found without a vulnerability module. This vulnerability is usage of MD5 hashing algorithm for password storing.

This paper highlights the importance of understanding web security fundamentals. Even at a basic level, securing a web application requires understanding of different types of attacks and how to prevent them.

USE OF AI IN THESIS

I have utilised AI tools in my thesis:

- ☐ No
- ☒ Yes

The AI tools utilised in my thesis and their purposes are described below:

Names and versions of AI tools: GPT4o

Purpose of using AI tools: Helping with some sentence wording and some text generation, which was overseen by me afterwards.

Sections where AI tools were used: 3.4 and 5. Conclusion

I acknowledge that I am fully responsible for the entire content of my thesis, including the parts generated by AI, and accept accountability for any violations of ethical standards in publications.

CONTENTS

1. INTRODUCTION

Damn Vulnerable Web Application (DVWA) is an opensource web application that can be downloaded from git freely. Its main purpose is to aid security professionals to test their skills and tools in legal environment, help web developers better understand the processes of securing web applications and do aid both students and teachers to learn about web application security. DVWA has four different difficulty types: Low, medium, high and impossible. In practice, this means the difficulty of hacking but in this case it means how much we must implement to make the website as secure as possible. In this written work, we are using low difficulty, which means that the website is very vulnerable and requires more work to make it secure. Our primary goal in this project is to identify and resolve these vulnerabilities to improve the overall security of the application.

DVWA lists all vulnerabilities and explains how to abuse them. This makes it easy to test our implementations. The website has numerous vulnerabilities, and resolving every single one would require significant effort. We are focusing on addressing about half of them to keep the scope match this assignment.

In DVWA, locating the code that contains vulnerabilities is straightforward. The vulnerable code is stored in the “vulnerabilities” folder at the root of the website. For example, brute force vulnerability has a subfolder called “source”, which contains four .php files named according to the difficulty levels explained earlier. Modifying a .php file does not require restarting the Apache server that the website uses and makes testing intuitive and efficient.

2. SETTING UP DVWA

DVWA is a php/MariaDB web application that can be accessed at Digininja's github repository [1]. DVWA lists its installation requirements as follows: Operating system must be Debian-based system (Kali, Ubuntu, Kubuntu, Linux Mint, Zorin OS) and privileges include that it must be executed as root user. All instructions for installations can be found at README.md. It includes automated installations and manual installations. The web application runs on apache2 server and requires setting up MariaDB database. DVWA also requires few PHP configurations. [1]

Youtube has video guides for setting up DVWA and one was used in this written work as well. CryptoCat's channel has a thorough video called *0 – Intro/Setup – Damn Vulnerable Web Application (DVWA)* explaining how to set up the web application and other videos in playlist explaining how to abuse vulnerabilities in the code. [2]

Once DVWA has been set up and the main view is open, the vulnerabilities and their explanations can be found on the left side of the main view. Each vulnerability is explained and how to exploit them. For example, brute force lets you try different username and password combinations without any delays, nor does it use CSRF tokens in trying to access an account.

Security levels and their explanations are included in figure 1. The security level is set to low, allowing for more extensive inspection and the implementation of additional security features.

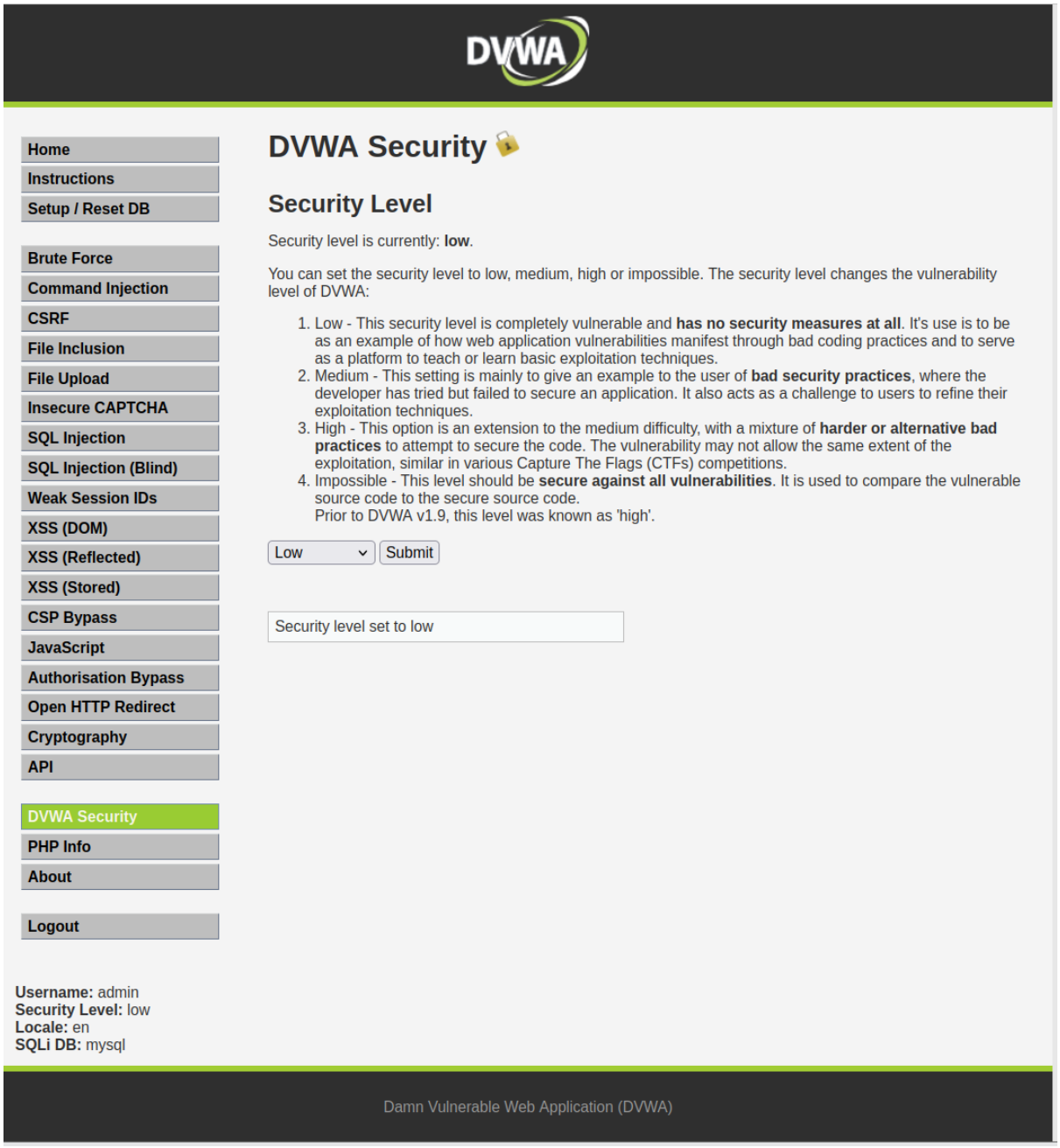

Figure 1. *DVWA security tab explaining the differences between security levels and allows the user to change the security level at will.*

Once DVWA is set up and vulnerability has been set to low, we are focusing on addressing the vulnerabilities of the web application.

3. IMPLEMENTATION OF SECURITY

3.1 Brute Force

Brute force attack is a password-guessing attack. This attack is an attempt to discover a password by systematically trying every possible combination of letters, numbers and symbols until the password is discovered. A common way of block these attacks is simply locking out account. This however is not the best solution, since attacker could easily abuse this security measure and lock out multiple accounts and deny the access to the service with lockouts. This also enables attacker to harvest usernames, since only valid account names will lock. [3] Also locking out users after too few tries will annoy legitimate users and locking out after too many tries will help attackers to brute-force passwords. [4]

Easy solution to slow down brute force attacker is to inject random pauses when checking a password. Adding few seconds' pause can greatly slow down the attackers attempts but it still wouldn't bother most legitimate users when they are trying to log in. This was implemented in Program 2. line 39.

The original authentication code for DVWA is as follows:

```
1.  <?php
2.  if( isset( $_GET[ 'Login' ] ) ) {
3.          // Get username
4.          $user = $_GET[ 'username' ];
5.
6.          // Get password
7.          $pass = $_GET[ 'password' ];
8.          $pass = md5( $pass );
9.
10.         // Check the database
11.         $query  = "SELECT * FROM `users` WHERE user = '$user' AND password = '$pass';";
12.         $result = mysqli_query($GLOBALS["___mysqli_ston"],  $query ) or die( '<pre>' .
    ((is_object($GLOBALS["___mysqli_ston"])) ? mysqli_error($GLOBALS["___mysqli_ston"]) : (($___mysqli_res
    = mysqli_connect_error()) ? $___mysqli_res : false)) . '</pre>' );
13.
14.         if( $result && mysqli_num_rows( $result ) == 1 ) {
15.                 // Get users details
16.                 $row    = mysqli_fetch_assoc( $result );
17.                 $avatar = $row["avatar"];
18.
```

```
19.                     // Login successful
20.                     $html .= "<p>Welcome to the password protected area {$user}</p>";
21.                     $html .= "<img src=\"{$avatar}\" />";
22.             }
23.             else {
24.                     // Login failed
25.                     $html .= "<pre><br />Username and/or password incorrect.</pre>";
26.             }
27.
28.             ((is_null($___mysqli_res = mysqli_close($GLOBALS["___mysqli_ston"]))) ? false : $___mysqli_res);
29.   }
30.   ?>
```

Program 1. *Original low security brute force vulnerability.*

This code lets you try different username and password combinations without any delays, nor does it use CSRF tokens in trying to access an account. In our implementation the third unsuccessful attempt of authenticating the user, the web application locks the user out for one minute. This could easily be modified to lock the session for longer periods of time after the first minute, but this code was made to be a proof of concept.

The sleep timer on line 39. is the better solution to prevent brute force attacks. This lockout mechanism doesn't lock out the users or IP addresses, which are often more harmful than useful methods [3]. This method locks the session, meaning it can be circumvented with clearing cookies or using terminal.

```
<?php
1.  session_start(); // Start session to track failed login attempts and timestamps


2.  // Initialize failed attempts and last failed attempt time if not already set
3.  if (!isset($_SESSION['failed'])) {
4.      $_SESSION['failed'] = 0;  // Track number of failed login attempts
5.      $_SESSION['last'] = 0;    // Track the last failed attempt time
6.  }


7.  $html = ""; // Initialize HTML output variable


8.  // Check if the login form has been submitted
9.  if (isset($_GET['Login'])) {
10.     // Get username and password from the form
11.     $user = $_GET['username'];
12.     $pass = md5($_GET['password']); // Hash the password (note: MD5 is not secure for production)
```

```
13.     // Check if user is locked out (3 or more failed attempts within 60 seconds)
14.     if ($_SESSION['failed'] >= 3 && (time() - $_SESSION['last']) < 60) {
15.         // Display lockout message if the user is locked out
16.         $html .= "<pre><br />You are locked out. Please try again in " . (60 - (time() - $_SESSION['last'])) . "
seconds.</pre>";
17.     } else {
18.         // Reset failed attempts if lockout time has expired (more than 60 seconds)
19.         if ($_SESSION['failed'] >= 3 && (time() - $_SESSION['last']) >= 60) {
20.             $_SESSION['failed'] = 0;
21.         }

22.         // Query the database to check the username and password
23.         $query  = "SELECT * FROM `users` WHERE user = '$user' AND password = '$pass';";
24.         $result = mysqli_query($GLOBALS["___mysqli_ston"], $query) or die('<pre>' .
25.             (is_object($GLOBALS["___mysqli_ston"]) ? mysqli_error($GLOBALS["___mysqli_ston"]) :
26.             (($___mysqli_res = mysqli_connect_error()) ? $___mysqli_res : false)) . '</pre>');

27.         // Check if the login credentials are correct
28.         if ($result && mysqli_num_rows($result) == 1) {
29.             // Login successful, reset failed attempts counter
30.             $_SESSION['failed'] = 0;

31.             // Fetch user details from the database
32.             $row    = mysqli_fetch_assoc($result);
33.             $avatar = $row["avatar"];

34.             // Display welcome message and user avatar
35.             $html .= "<p>Welcome to the password protected area {$user}</p>";
36.             $html .= "<img src=\"{$avatar}\" />"; // Display user's avatar
37.         } else {
38.             // Login failed, increment failed attempts counter and update last attempt time
39.             sleep( rand( 2, 4 ) );
40.             $_SESSION['failed']++;
41.             $_SESSION['last'] = time(); // Record the time of the failed attempt

42.             // Display error message for incorrect credentials
43.             $html .= "<pre><br />Username and/or password incorrect.</pre>";
44.         }

45.         // Close the database connection
46.         ((is_null($___mysqli_res = mysqli_close($GLOBALS["___mysqli_ston"]))) ? false : $___mysqli_res);
47.     }
48. }
?>
```

Program 2. *Locking out implementation in unsuccessful authentication of user.*

Anti-CSRF token would make this more secure, but it has been implemented in the base code and could be just made with *generateSessionToken();* but this was left out of this implementation since the function was already included in the base code. More of this in CSRF section.

3.2 Command Injection

Command injection attack's goal is to execute arbitrary commands on the host operating system via vulnerable application. Command injection attacks are possible largely due to insufficient input validation. Command injection should not be confused with Code Injection attacks. Code injection allows attacker to add their own code that is then executed by the application. In command injection, the attacker extends the default functionality of the application to execute system commands, without the necessity of injecting code. [5]

To secure a web application from command injection, a proper sanitizing of user input should be included in the code. The low-level security does not include any sanitizing of user input and allow command injection as shown in the figure 2.

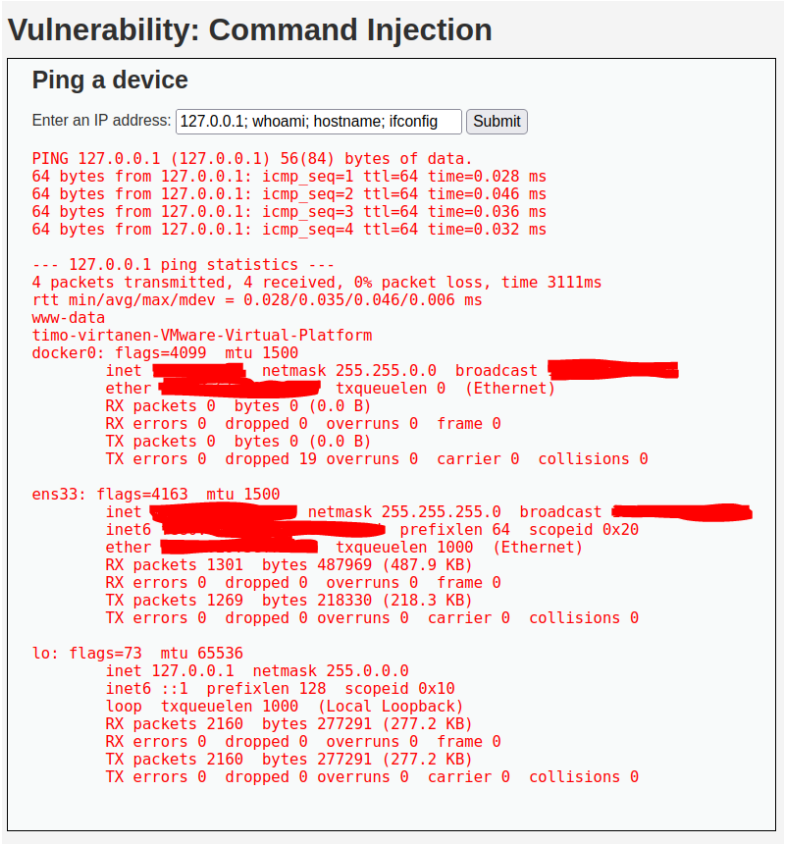

Figure 2. *Pinging a device and injecting command attack*

Figure 2. contains an example of private data being leaked without proper sanitation. This is the original code for command injection vulnerability.

```
<?php
1.  if( isset( $_POST[ 'Submit' ]  ) ) {
2.      // Get input
3.      $target = $_REQUEST[ 'ip' ];

4.      // Determine OS and execute the ping command.
5.      if( stristr( php_uname( 's' ), 'Windows NT' ) ) {
6.          // Windows
7.          $cmd = shell_exec( 'ping  ' . $target );
8.      }
9.      else {
10.         // *nix
11.         $cmd = shell_exec( 'ping  -c 4 ' . $target );
12.     }

13.     // Feedback for the end user
14.     $html .= "<pre>{$cmd}</pre>";
15. }

?>
```

Program 3. *Original low-security code for pinging devices*

Sanitizing the user input can be done with filter_var($target, FILTER_VALIDATE_IP), which validates $target to be a proper IP address. This is used in our solution to address the command injection vulnerability. If user adds anything that is not a valid IP address, the web application will inform about invalid IP address.

```
if( isset( $_POST['Submit'] ) ) {
2.      // Get input
3.      $target = trim($_REQUEST['ip']);

4.      // Validate input (allow only valid IP addresses)
5.      if (filter_var($target, FILTER_VALIDATE_IP)) {
6.          // Determine OS and execute the ping command securely
7.          if (stristr(PHP_OS, 'WIN')) {
8.              // Windows
9.              $cmd = shell_exec('ping ' . escapeshellarg($target));
```

```
10.         } else {
11.             // *nix (Linux/macOS)
12.             $cmd = shell_exec('ping -c 4 ' . escapeshellarg($target));
13.         }

14.         // Store output for display
15.         $html .= "<pre>{$cmd}</pre>";
16.     } else {
17.         $html .= "<p style='color:red;'>Invalid IP address.</p>";
18.     }
19. }

?>
```

Program 4. *User input filtered with filter_var($target, FILTER_VALIDATE_IP)*

Arial 18 pt font is used for the headings in this guide, with 42 pt space above and below. The font size of subheadings is 14, with 18 pt space above and 12 pt space below them.

3.3 Cross-Site Request Forgery (CSRF)

CSRF is an attack that forces an end user to execute unwanted actions on a web application in which they have already been authenticated. An attacker may trick a web application user into executing actions of the attackers choosing with a little help of social engineering. [6] For example, making another user unknowingly visit this link: "*127.0.0.1/dvwa/vulnerabilities/csrf/?password_new=admin&password_conf=admin&Change=Change*" would change the user's password to admin in DVWA. These links can be disguised as an ordinary links or fake images. [6]

One of the most common methods of mitigating attacks is the use of CSRF tokens. CSRF-tokens are randomly generated values stored in the server-side session and embedded in HTML forms or headers. CSRF tokens are submitted explicitly by the client, usually via form data or request headers, unlike cookies which are sent automatically by the browser. Now when the attacker tries to make the user to click on a CSRF attack link, the server checks for the CSRF token and tries to match it to the session. If the token doesn't match, the action is prevented. [6]

CSRF-vulnerability page is a page where the user may change the user's password. The low security code of CSRF page doesn't ask the user to confirm the current password before change but more importantly it doesn't check or create new CSRF tokens. Code also allows SQL injection, which will be prevented later with prepared statements. This low secure code is shown in Program 5.

```
1.  <?php
2.
3.  if( isset( $_GET[ 'Change' ] ) ) {
4.      // Get input
5.      $pass_new  = $_GET[ 'password_new' ];
6.      $pass_conf = $_GET[ 'password_conf' ];
7.
8.      // Do the passwords match?
9.      if( $pass_new == $pass_conf ) {
10.         // They do!
11.         $pass_new = ((isset($GLOBALS["___mysqli_ston"]) && is_object($GLOBALS["___mysqli_ston"])) ?
mysqli_real_escape_string($GLOBALS["___mysqli_ston"],  $pass_new ) : ((trigger_error("[MySQLConverterToo] Fix the
mysql_escape_string() call! This code does not work.", E_USER_ERROR)) ? "" : ""));
12.         $pass_new = md5( $pass_new );
13.
14.         // Update the database
15.         $current_user = dvwaCurrentUser();
16.         $insert = "UPDATE `users` SET password = '$pass_new' WHERE user = '" . $current_user . "';";
17.         $result = mysqli_query($GLOBALS["___mysqli_ston"],  $insert ) or die( '<pre>' .
((is_object($GLOBALS["___mysqli_ston"])) ? mysqli_error($GLOBALS["___mysqli_ston"]) : (($___mysqli_res =
mysqli_connect_error()) ? $___mysqli_res : false)) . '</pre>' );
18.
19.         // Feedback for the user
20.         $html .= "<pre>Password Changed.</pre>";
21.     }
22.     else {
23.         // Issue with passwords matching
24.         $html .= "<pre>Passwords did not match.</pre>";
25.     }
26.
27.     ((is_null($___mysqli_res = mysqli_close($GLOBALS["___mysqli_ston"]))) ? false : $___mysqli_res);
28. }
29.
30. ?>
```

Program 5. *Low Security password change that allows CSRF*

But as discussed before, we did not implement the generation of CSRF tokens because it is already included in the base code. This function can be found in the folder dvwa/dvwa/includes in the file called *dvwaPage.inc.php*. This function can also be seen in Program 6.

```
1.  function generateSessionToken() {  # Generate a brand new (CSRF) token
2.          if( isset( $_SESSION[ 'session_token' ] ) ) {
3.                  destroySessionToken();
4.          }
5.          $_SESSION[ 'session_token' ] = md5( uniqid() );
6.  }
```

Program 6. *Simple generation of CSRF token, with MD5 hashing.*

This token would be generated in the end of CSRF vulnerability code and checked before any other vulnerable code. For example, in brute force the CSRF token would be checked using *checkToken([PARAMS);* function*.* This function can also be found where the CSRF token is generated. Program 7 has modifications for more secure code. It includes prepared statements to prevent SQL injections and confirming the current password to prevent CSRF attack.

```
01: <?php
02:
03: if( isset( $_GET[ 'Change' ] ) ) {
04:     $mysqli = $GLOBALS["___mysqli_ston"];
05:
06:     // Get inputs
07:     $pass_curr = $_GET[ 'password_current' ];
08:     $pass_new  = $_GET[ 'password_new' ];
09:     $pass_conf = $_GET[ 'password_conf' ];
10:
11:     // Escape and hash current password
12:     $pass_curr = mysqli_real_escape_string($mysqli, $pass_curr);
13:     $pass_curr_hashed = md5($pass_curr);
14:
15:     // Get current user
```

```
16:     $current_user = dvwaCurrentUser();
17:
18:     // Check if current password is correct using a prepared statement
19:     $stmt = mysqli_prepare($mysqli, "SELECT password FROM users WHERE user = ? AND password = ? LIMIT
1;");
20:     mysqli_stmt_bind_param($stmt, "ss", $current_user, $pass_curr_hashed);
21:     mysqli_stmt_execute($stmt);
22:     mysqli_stmt_store_result($stmt);
23:
24:     // Do the passwords match?
25:     if (mysqli_stmt_num_rows($stmt) === 1 && $pass_new === $pass_conf) {
26:         // Escape and hash new password
27:         $pass_new = mysqli_real_escape_string($mysqli, $pass_new);
28:         $pass_new_hashed = md5($pass_new);
29:
30:         // Update password using prepared statement
31:         $update = mysqli_prepare($mysqli, "UPDATE users SET password = ? WHERE user = ?");
32:         mysqli_stmt_bind_param($update, "ss", $pass_new_hashed, $current_user);
33:         mysqli_stmt_execute($update);
34:
35:         $html .= "<pre>Password Changed.</pre>";
36:     } else {
37:         $html .= "<pre>Passwords did not match or current password incorrect.</pre>";
38:     }
39:
40:     // Close SELECT and UPDATE
41:     mysqli_stmt_close($stmt);
42:     mysqli_stmt_close($update);
43: }
44:
45: ?>
```

Program 7. Modified CSRF page vulnerability code.

Changes to the original code include the prepared statements and confirming current password.

3.4 MD5 – Insufficient cryptography

DVWA uses insufficient cryptography called MD5 [7]. This is the case for all security levels. This cryptography algorithm has been proven to contain collisions [8], which allows two different inputs to produce the same hash value. In practice this means that even though attacker has brute forced login informatio6n, it just might be that it is not

the user's real password but just a hash that is identical to the one that was brute forced. MD5 also uses fixed-sized 128-bit hash value, which is shorter than modern secure hash functions. 128-bit fixed-sized also reduces the resistance against brute-force and collisions attacks. [9] However, hashing algorithm vulnerability, along with the usage of CSRF tokens, was not implemented and was only acknowledged due to time constraints and the primary focus being on practicing secure programming. This could just be solved with changing the hashing algorithm, which would require the change to be added to multiple places in the web application. If MD5 was replaced, it should be done by using bcrypt hashing algorithm. It results in robust password security and is effective in warding off brute force attacks [10].

3.5 SQL Injection

SQL injection page allows users to find users' first name and surname by their account ID. The result will show users ID, first name and surname. However, the submission box doesn't sanitize user input and allows attackers to inject SQL code into the text box. For example, injecting: "*' UNION SELECT user, password FROM users#" to* the text box results in ID tab showing the following text: "*' UNION SELECT user, password FROM users#* ", first name showing the users' first names and the surname showing all users' hashed passwords. After receiving hashed passwords, the attacker may use hashid for example to find all possible hashing algorithms that might have been used to hash the password. With other tools available, the possible algorithm list can be narrowed down even more. If the attacker finds the algorithm used, they may start running it on their own computers without any delays until they find the match for a password. If they website is not using salt for password hashing, they may find multiple accounts passwords at the same time.

The low security SQL injection vulnerability is shown in program 8. Everything SQL injection related attacks are possible in the low security code.

```
1. <?php
 2.
 3. if( isset( $_REQUEST[ 'Submit' ] ) ) {
 4.     // Get input
 5.     $id = $_REQUEST[ 'id' ];
 6.
 7.     switch ($_DVWA['SQLI_DB']) {
 8.         case MYSQL:
 9.             // Check database
10.             $query  = "SELECT first_name, last_name FROM users WHERE user_id = '$id';";
```

```
11.             $result = mysqli_query($GLOBALS["___mysqli_ston"],  $query ) or die( '<pre>' .
((is_object($GLOBALS["___mysqli_ston"])) ? mysqli_error($GLOBALS["___mysqli_ston"]) : (($___mysqli_res =
mysqli_connect_error()) ? $___mysqli_res : false)) . '</pre>' );
12.
13.             // Get results
14.             while( $row = mysqli_fetch_assoc( $result ) ) {
15.                 // Get values
16.                 $first = $row["first_name"];
17.                 $last  = $row["last_name"];
18.
19.                 // Feedback for end user
20.                 $html .= "<pre>ID: {$id}<br />First name: {$first}<br />Surname: {$last}</pre>";
21.             }
22.
23.             mysqli_close($GLOBALS["___mysqli_ston"]);
24.             break;
25.         case SQLITE:
26.             global $sqlite_db_connection;
27.
28.             #$sqlite_db_connection = new SQLite3($_DVWA['SQLITE_DB']);
29.             #$sqlite_db_connection->enableExceptions(true);
30.
31.             $query  = "SELECT first_name, last_name FROM users WHERE user_id = '$id';";
32.             #print $query;
33.             try {
34.                 $results = $sqlite_db_connection->query($query);
35.             } catch (Exception $e) {
36.                 echo 'Caught exception: ' . $e->getMessage();
37.                 exit();
38.             }
39.
40.             if ($results) {
41.                 while ($row = $results->fetchArray()) {
42.                     // Get values
43.                     $first = $row["first_name"];
44.                     $last  = $row["last_name"];
45.
46.                     // Feedback for end user
47.                     $html .= "<pre>ID: {$id}<br />First name: {$first}<br />Surname: {$last}</pre>";
48.                 }
49.             } else {
50.                 echo "Error in fetch ".$sqlite_db->lastErrorMsg();
51.             }
52.             break;
53.     }
54. }
55.
```

```
56. ?>
```

Program 8. Low security SQL injection code.

SQL injection attacks also allow attackers to spoof identity and tamper with existing data for example [11]. To avoid SQL injection flaws, the developers need to stop writing dynamic queries with string concatenation or prevent malicious SQL input from being included in executed queries [12].

In our solution, we implemented prepared statements and parameterized queries to safeguard against SQL injection. Secure version of this code is shown in *program 9*, with prepared statements on lines 15 and 45. However, the original implementation did not account for HTML-specific formatting such as the <pre> tag, which can affect how the output is displayed. Because of this, numeric input validation was added at line 7 to ensure only valid numbers are processed, improving both security and output formatting.

```
01: <?php
02:
03: if (isset($_REQUEST['Submit'])) {
04:     $id = $_REQUEST['id'];
05:
06:     // Validate that ID is a number
07:     if (!is_numeric($id)) {
08:         $html .= "<pre>Invalid ID: Must be a number.</pre>";
09:     } else {
10:         switch ($_DVWA['SQLI_DB']) {
11:             case MYSQL:
12:                 $mysqli = $GLOBALS["___mysqli_ston"];
13:
14:                 // Use a prepared statement
15:                 $stmt = mysqli_prepare($mysqli, "SELECT first_name, last_name FROM users WHERE user_id = ?");
16:                 if (!$stmt) {
17:                     $html .= "<pre>Error preparing statement: " . mysqli_error($mysqli) . "</pre>";
18:                     break;
19:                 }
20:
21:                 $id = (int)$id; // Ensure it's an integer for binding
22:                 mysqli_stmt_bind_param($stmt, "i", $id);
23:                 mysqli_stmt_execute($stmt);
24:                 mysqli_stmt_bind_result($stmt, $first, $last);
25:
26:                 // Fetch and display results
27:                 $found = false;
```

```
28:                 while (mysqli_stmt_fetch($stmt)) {
29:                     $found = true;
30:                     $html .= "<pre>ID: {$id}<br />First name: {$first}<br />Surname: {$last}</pre>";
31:                 }
32:
33:                 if (!$found) {
34:                     $html .= "<pre>No user found with ID: {$id}</pre>";
35:                 }
36:
37:                 mysqli_stmt_close($stmt);
38:                 mysqli_close($mysqli);
39:                 break;
40:
41:             case SQLITE:
42:                 global $sqlite_db_connection;
43:
44:                 // Use SQLite prepared statement
45:                 $stmt = $sqlite_db_connection->prepare("SELECT first_name, last_name FROM users WHERE user_id =
:id");
46:                 if (!$stmt) {
47:                     $html .= "<pre>Error preparing SQLite statement.</pre>";
48:                     break;
49:                 }
50:
51:                 $stmt->bindValue(':id', (int)$id, SQLITE3_INTEGER);
52:
53:                 try {
54:                     $results = $stmt->execute();
55:                 } catch (Exception $e) {
56:                     $html .= "<pre>Database error: " . htmlspecialchars($e->getMessage()) . "</pre>";
57:                     break;
58:                 }
59:
60:                 $found = false;
61:                 while ($row = $results->fetchArray(SQLITE3_ASSOC)) {
62:                     $found = true;
63:                     $first = $row['first_name'];
64:                     $last  = $row['last_name'];
65:                     $html .= "<pre>ID: {$id}<br />First name: {$first}<br />Surname: {$last}</pre>";
66:                 }
67:
68:                 if (!$found) {
69:                     $html .= "<pre>No user found with ID: {$id}</pre>";
70:                 }
71:
72:                 break;
```

```
73:         }
74:     }
75: }
76: ?>
```

Program 9. Modified SQL injection code with prepared statements.

With these changes shown in program 9, vulnerabilities have been mitigated, and SQL injection attacks is impossible or at least very hard to carry out do to the use of prepared statements and parameterized queries , which ensure user input is treated as data rather than executable code.

4 DISCUSSION

The DVWA was selected for this project due to its intentionally insecure design. While the application contains numerous vulnerabilities, only four of them were addressed due to time constraints. The implemented changes significantly improved the security of the application, rendering these specific low-security vulnerabilities effectively the same level as “impossible”

It is important to note that the applied changes or code do not mirror the exact implementations as DVWA’s build in “impossible” security level, but they do use similar practices. The primary references used in this project were OWASP guidelines and limited number of scholarly articles.

This report combines the elements of exercise work and written work. Due to the challenges of demonstrating the differences between regular DVWA and these mitigated vulnerability DVWA, a written format was chosen to be the better alternative to demonstrate the modifications to the application. This modified

5 CONLUSION

This project explored the DVWA in its low security setting, focusing of identifying and resolving a selection of critical vulnerabilities. By selecting an intentionally insecure web application, we were able to gain hands-on experience in understanding vulnerabilities and how to implement effective security measures to address them.

While we only addressed few of the many vulnerabilities due to the scope of this assignment, the applied changes significantly improved the security of the application. The mitigated vulnerabilities now reflect a level of protection comparable to DVWA’s "impossible" setting, even if the exact implementations differ. Our solutions followed best practices outlined in OWASP guidelines and were informed by a limited number of scholarly sources.

Ultimately, this exercise highlights the importance of understanding web security fundamentals. Even at a basic level, securing a web application requires understanding of different types of attacks and how to prevent them. Platforms like DVWA are valuable tools for students, developers and even security professionals to practice and improve their ability to secure real-world applications.

REFERENCES

[1] Damn Vulnerable Web Application. (n.d.). *DVWA - Damn Vulnerable Web Application* [Source code]. GitHub. https://github.com/digininja/DVWA/

[2] CryptoCat. (n.d.). *CryptoCat* [YouTube channel]. YouTube. https://www.youtube.com/@_CryptoCat

[3] OWASP Foundation. (n.d.). *Blocking brute force attacks*. OWASP. https://owasp.org/www-community/controls/Blocking_Brute_Force_Attacks

[4] Blocki, J., & Zhang, W. (2020). *DALock: Distribution aware password throttling*. arXiv. https://arxiv.org/abs/2005.09039

[5] OWASP Foundation. (n.d.). *Command injection*. OWASP. https://owasp.org/www-community/attacks/Command_Injection

[6] OWASP Foundation. (n.d.). *Cross-Site Request Forgery (CSRF)*. OWASP. https://owasp.org/www-community/attacks/csrf

[7] OWASP Foundation. (2016). *M5: Insufficient cryptography*. OWASP Mobile Top 10 2016. https://owasp.org/www-project-mobile-top-10/2016-risks/m5-insufficient-cryptography

[8] Black, J., Cochran, M., & Highland, T. (2006). A study of the MD5 attacks: Insights and improvements. In D. Wagner (Ed.), Advances in Cryptology – CRYPTO 2006 (pp. 262–277). Springer. https://doi.org/10.1007/11799313_17

[9] Datadog. (n.d.). *Import MD5: Static analysis rule for Go*. Datadog Documentation. https://docs.datadoghq.com/security/code_security/static_analysis/static_analysis_rules/go-security/import-md5/

[10] Batubara, T. P., Efendi, S., & Nababan, E. B. (2021). Analysis performance BCRYPT algorithm to improve password security from brute force. *Journal of Physics: Conference Series, 1811*(1), 012129. https://doi.org/10.1088/1742-6596/1811/1/012129

[11] OWASP. (n.d.). *SQL Injection*. OWASP. https://owasp.org/www-community/attacks/SQL_Injection

[12] OWASP. (n.d.). *SQL Injection Prevention Cheat Sheet*. OWASP Cheat Sheet Series. https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html

[13] https://github.com/Virtanenn/hands_on_DVWA/