SQL Injections

Greater Omorose
Northumbria University
Newcastle upon Tyne, United Kingdom
Greateromorose1@gmail.com

***Abstract*— This paper explores the vulnerabilities of web applications to SQL injection attacks. Furthermore, it will discuss the importance of protecting SQL databases, types of SQL injection, prevention strategies, and a demonstration that shows how to set up a simple login page on Windows and Linux servers by using Apache. After full setup, there will be an SQL injection attack demonstration and a discussion of possible countermeasures.**

Keywords—SQL injection, Prevention strategies

I. INTRODUCTION

In today's digital age, computer software has revolutionized countless aspects of our lives, from conducting business transactions to accessing healthcare services, and even online teaching was made online during a time when humanity was at its lowest, technology is a constant in these turbulent times. However, alongside these advancements, some seek to take advantage for malicious purposes. One of the most common and devasting attacks is SQL Injection.

Large amounts of sensitive data from companies and their customers are stored in special databases within the organization's servers. Databases have become essential pillars of modern organizations for this reason they need to be protected against unauthorized access since any leaks would have a huge impact on the privacy of the users and the reputation of the institution.[1]

SQL(Structured Query Language) is a standard computer language used to access and manipulate databases. It allows database administrators to perform multiple actions such as retrieving, storing, creating, updating, or deleting data from a database. These databases are considered relational databases because there is a relationship between tables. For example, in a database for an online store, there will be tables for customers and another for orders, with each order linked to a specific customer, creating a link between these two tables. Various software manages this RDB such as PostgreSQL, Oracle Database, and SQLite but in this paper, MySQL is what has been installed to carry out the webserver setup and demonstration attack.[2]

SQL databases are commonly used in web applications to store data related to websites and users. This data is managed through queries within specific commands. When visitors interact with the website their actions are translated into SQL commands. When these commands are executed they modify or update data, while also ensuring that visitors do not directly interact with these databases. Unfortunately, attackers have found ways to target and exploit databases to get access to data that they should not be allowed to access. One of the most common attacks on databases is SQL injection [1]. This attack has been so damaging to organizations, that it has been on OWASP's Top 10 in 2010,2013, 2017, and 2021 furthermore according to a blog[3]:

- 42% of hackers attempt to break into web applications using SQL injection-based techniques
- 21% of organizations are still at risk of attack involving SQL.
- The biggest SQL attack ever stole over 1 billion usernames and passwords.
- Hackers took 130 million credit card details using SQL-based attacks
- Although there has been a technological evolution in prevention against this attack, such as web application firewalls, these prevention strategies are still being bypassed by attackers.

There have been so many companies that fell victim to this attack, for example in 2008 Heartland Payment System fell victim to a major SQL attack which caused approximately 130 million credit and debit card numbers to be compromised. Similarly, In 2011, Sony Pictures received a huge SQL injection attack which caused approximately 77 million PlayStation Network accounts to be compromised. This damaged Sony's reputation and cost them over 170 million dollars. Also in 2021, Ubiquiti fell victim to this attack. These companies should have invested more resources in finding prevention strategies for this attack because if better precautions had been taken, then these attacks could have been prevented furthermore, these cases demonstrated the catastrophic effects of SQL injections such as[6]:

- Loss of confidentiality: which happens when a hacker gains unauthorized access to the data stored in the database
- Loss of Authentication: Occurs when an attacker gains access to a system, without authenticated credentials.
- Loss of authorization: Occurs when an attacker leaks personal information in a system.
- Loss of integrity: This happens when a hacker gains access to a database and potentially alters the information stored in these databases, which would compromise its reliability.
- Loss of availability: An attacker may delete essential data from the database, causing significant losses and rendering the data inaccessible.
- Remote Code Execution: After gaining access to the database attackers might add new accounts with full user rights and then use it to compromise existing OS.

For these reasons, my paper aims to focus on SQL injection attack types, their execution methods, and prevention strategies. Additionally, I will provide a practical demonstration of these attacks to demonstrate their execution and potential consequences. This research will be very detailed but it is important to acknowledge the limitations, which are:

- The prevention strategies provided in this paper are based on information available up to 2024. As cyber threats evolve, these strategies might become outdated in the future, particularly from 2025 onwards.
- The effectiveness of the prevention strategies discussed in this paper could vary depending on the specific scenarios and environments in which they are implemented.

- The simulation attack might not replicate real-world conditions
- The findings might be based on a specific set of conditions and might not be generalizable to all systems.

II. Related Work

A. History

The vulnerability was first documented by Jeff Forristal in December 1998 in the Phrack magazine. Even though it was first officially documented in 1998, this attack only started gaining popularity from 2002[4]. As developers began to use SQL to interact with databases, vulnerabilities arose due to inadequate input validation and insufficient security measures against this attack. The first major attack occurred in 2002 on guess.com's database which led to 200.000 customer data, such as credit cards, being compromised [5]. Ever since the large-scale attack, there have been multiple types of research on possible prevention strategies against these attacks such as using SQL scanning security tools like WAVES(in 2003), AMNESIA(in 2005) etc, but unfortunately, SQL attacks have continued to evolve at a rapid pace, which made these software more ineffective over time[4].

B. SQL injection types

As mentioned earlier, an SQL injection attack is a vulnerability that targets a database. These vulnerabilities are usually found in PHP and ASP applications, and they are mainly caused by bad coding practices, invalid input, generous privileges, uncontrolled variable size, error messages, variable orphism, dynamic SQL, client-side control, multiple statements etc. For my methodology practical demonstration, I took inspiration from a paper called: SQL Injection: Classification and Prevention by Aditya Ray [7] in which he explained how to perform a login bypass. According to this paper, a website that has a login page usually requires a Username and a password parameter. When these parameters are set, they get sent to a server and a SQL query is generated. The parameters are then compared with the data present in the database and if the comparison is successfully true, then the user is logged in. The SQL query that allows comparison may look or be like this: SELECT * FROM users WHERE username='admin' and password='strongpassword' ; unfortunately this query can be exploited by inputting the value ' OR '1'='1' which would cause the SQL query to be SELECT * FROM users WHERE username ='administrator' and password = 'aaa' OR '1' = '1'; This query will always be true, so an unauthenticated user, with no knowledge of the password can log in. The use of true statements to bypass a security system is very common for this reason in my practical I will demonstrate how to set up this attack and its potential damages [7]. Unfortunately, Ray's paper [7] only provides insight into the technical aspect of the attack but it does not talk about the success rate of the attack on a Windows server vs a Linux server and it fails to address potential countermeasures against this attack. Not considering the attack on the Linux vs Windows server is crucial because each server has its own vulnerabilities, which means that they could have different responses in reacting against an SQL injection attack. For these reasons in my practical demonstration I will perform the attack on both server to understand if they would react differently to the same type of attack and possible mitigation strategies.

There are other types of SQL injection attacks:

- In-band SQL injection
- Blind SQL Injection
- Out-of-band SQL injection

I *IN-band SQL injection*

This attack uses the same channel to launch and receive results of an attack, there are 2 types: Error-based, and Union-based. Error-based is when the attacker intentionally sends unexpected input to generate some errors. These errors provide valuable information regarding the target system, which can be used to craft payloads. Union-based attacks are when an attacker uses the 'Union' command to merge 2 queries and present the results.[7]

II *Blind SQL injection*

Unlike an IN-Band SQL attack, this attack doesn't display the output of an injection made by the user on a web page, so an attacker has to send malicious SQL queries, and based on how the application behaves, they can determine if their attack was successful or not. There are 2 subtypes of this attack: Boolean-based blind SQL and Time-based SQL. Boolean-based injection is a type of injection that requires the use of true and false conditions. To perform this attack, the attacker needs to inject false payloads and then true payloads and then compare the results.[7] If the application behaves differently for each scenario, then the target is vulnerable to injections. For example, by querying the database with conditions that reveal one character at a time, attackers can ultimately obtain complete access to sensitive accounts, such as the Administrator account. Time-based SQL injection is a type of SQL injection attack where the attacker exploits the application's response time to deduce information about the database. For example, if the attacker thinks the first character of the Administrator's password is 'p', they can inject a query that makes the database wait for 5 seconds. If the query is correct, the database will wait and then respond after 5 seconds, indicating that the injected condition was true.[7]

III *Out-of-band SQL injection*

This attack occurs when the database cannot directly communicate its responses to the webserver due to restriction and security measures. During this attack, the attacker forces the database to send its responses to the specified server they control rather than the web server by using specific payloads designed to trigger out-of-bond network interactions to check for out-of-band network interaction.[7]

C. SQL attack stages

Depending on the target, the SQL injection attack stages might be different from each other, but they usually follow the same pattern[8]:

1. Identification of vulnerable inputs: The attacker first identifies inputs that could be vulnerable to SQL injections. For example, URL parameters or text fields in a form.
2. Crafting the malicious SQL query: Once a vulnerable input has been found, a malicious SQL query is crafted with the purpose of modifying the original query to perform malicious actions.
3. Bypassing application security measures: Even if the attacker crafted a functional query, there could be security measures like input validation that may prevent this crafted query from functioning. For these reasons, in this stage, the attacker needs to find ways to bypass these security measures like string concatenation.
4. Executing the malicious query: As the application runs the SQL query, it incorporates the harmful input provided by the attacker.
5. Extracting or manipulating data: This is the stage in which the attacker can alter existing data, add new data etc.
6. Exploiting database server vulnerabilities: In this stage, the attacker might decide to target not just the database but the OS of the server or even the file systems.[8]

D. Legal and Ethical considerations

Ethical consideration:

- Privacy: People's privacy should be respected but unfortunately unauthorized SQL injections can violate this principle because they can potentially expose sensitive information. For these reasons, the primary objective should always be to protect this sensitive data.
- Security: Organizations should take responsibility for protecting their customer's data and if they fail to protect against SQL injections they should take full responsibility for the consequences.
- Potential harm: SQL injections can have severe consequences such as financial loss and identity theft. So, for these reasons, it is important to consider the potential harm that these attacks might cause.
- Informed consent: SQL injection attacks involve manipulating systems and data so before performing this attack against a system, you should get informed consent from the system admins.

Failure to follow this ethical consideration and performing an SQL injection attack against an organization without consent is considered illegal due to many laws. For example, the Computer Misuse Act 1990 protects personal data held by organizations. This act safeguards personal data held by organizations and prohibits unauthorized access or modification of computer systems [11]. For these reasons, unauthorized SQL injection attacks are affected by this law. Failure to comply with this act can lead to fines and penalties. For example [11]:

- Unauthorized access to computer material can lead to up to six months in prison and/or up to a £5,000 fine.
- Unauthorized access to computer materials with intent to commit a further crime can cause up to a five-year prison sentence and/or an unlimited fine.
- Unauthorized modification of data up to a five-year prison sentence and/or an unlimited fine.

III. Methodology

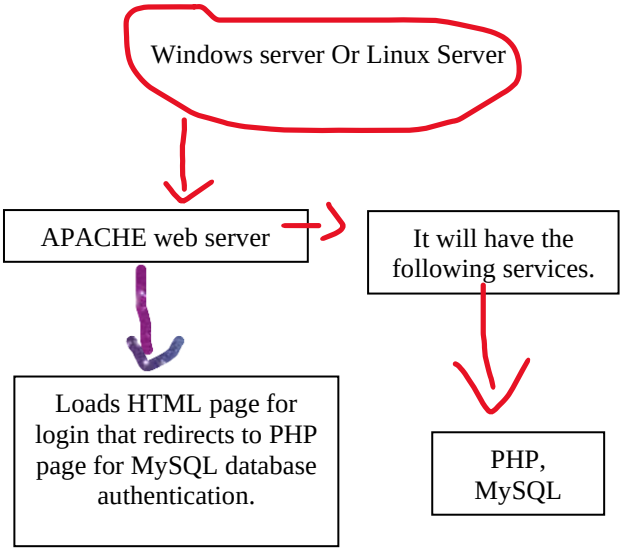
A. Attack Selection

The login function is a very important aspect of a website, for this reason, it requires a robust security measure because if individuals can bypass login functions without being authorized then it puts all application users at risk of losing their data, etc. For these reasons, the attack simulation I will perform will be an SQL Login bypass attack to gain access to the admin user account. Usually, a simple login page with a Username and Password field will have an SQL query that might look like this: “SELECT * FROM users WHERE username = ‘$user’ AND password = ‘$pass’. An attacker could bypass this login system by using a malicious string such as ‘OR ‘1’=’1’--. Inputting this string in the username field will cause the query to change and become: “SELECT * FROM users WHERE username = ‘’ OR ‘1’=’1' — ‘ AND password = ‘[pass]’, this query will always output a true result because there is an Or condition and 1=1 is always true, which will allow the attacker to bypass the login and gain access to the system.

In my simulation attack, I will use the same logic, but I will assume that the username field is not injectable and that I already used OSINT to find out the name and username of the admin. The attack vector used is the login page's input fields, specifically the username and password fields and the vulnerability I will exploit will be the SQL query used for user authentication which is not properly sanitized. If for example the admin username was “Admin”, and we input this value in the username field and type a random password followed by ' or '1'='1'--, it will cause the SQL query to be SELECT * FROM users WHERE username = 'admin' AND password = 'randomPass' or '1'='1'-- '. This injected string would trick the website into thinking that we are the admin, and it would grant us access to his user account and possibly steal customer data or send phishing emails to the colleagues to trick them into sending money etc.

The reason why the attack works is because the injected SQL query starts with ‘Select * From users’, which lets Mysql select all the columns from the ‘users’ table in the database. The ‘Where’ filters the rows from the ‘user’ table based on certain conditions. In this case the condition is that the ‘username’ column should be equal to ‘admin’ and the ‘password’ column should be equal to ‘randomPass’. However, the injection attempt adds an OR '1'='1' to the SQL query. The ‘OR’ statement introduces an alternative condition ‘1’=’1’, and the reason why this works is because the ‘1’=’1’ will always be TRUE. So when the SQL query is executed, the database checks both the original condition (‘username’ = ‘admin’ and ‘password = ‘randomPass’) and condition (‘1’ = ‘1’) which causes the entire Where statement to be true which effectively bypasses the original password check.
[10]

B. Simulation setup



In this attack simulation, I configured a Linux and Windows server and then installed Apache on each server. and used Kali to perform the attack. My ip network is 192.168.232.0, but after subnetting the network to allow 14 hosts connection, I found that the IP range for this would be 192.168.232.1 - 192.168.232.14 /28. I decided to set the default gateway to 192.168.232.2.

I Kali

When configuring Kali Linux, make sure to change the Ip address. To do so, open Kali search bar and look for “Advanced Network Configuration” and select “wired connection 1”, then navigate to IPv4 settings. Change the method to “manual” and add Ip address 192.168.232.6, netmask 28 and Gateway 192.168.232.2 and save changes. With these configurations, Kali Linux will be on the same network as the Linux and Windows server, so it will be able to display them on his web browser.

II Linux server

- Stage 1: Install Ubuntu Live Server 22.04.4 AMD 64 on VMware Workstation Player 15, and then follow on-screen instructions..
- Stage 2: After installation, Run “sudo apt update && upgrade” and then type the command “Sudo nano /etc/netplan/00-installer-config.yaml” which will allow us to edit the 00-installer-config.yaml and assign an Ip address to the Linux server. By using this file, disable DHCP and assign the IP 192.168.232.3 /28 and the default gateway 192.168.232.2. For the nameserver just insert [8.8.8.8 and 8.8.4.4]
- Stage 3: Install Apache by using the command sudo apt install apache2(the Apache version is 2.4.59) then install other packages like php (version 8.3.6) by using the command sudo apt-get install php libapache2-mod-php php-mysq1 php-gd php-curl and then install MySQL(version 8.3.0) by using the command sudo apt-get install mysql-server. The PHP installation is important because it allows the website to connect to the MySQL database. MySQL installation is important because we will use it to store user credentials. At the end of this stage, when you navigate to 192.168.232.3 on your web browser, the default Apache page should load.

- Stage 4: Install security settings on MySQL by typing the command “sudo mysql_secure_installation” and follow the screen prompts by replying to them with Y(yes) or N(no). After successful installation, use the command sudo systemctl start mysql and then log into MySQL by using the command sudo mysql -u root -p. It is important to login with a root account because it will always have full database permission.

- Stage 5: Create a new database by using the command Create DATABASE [database name] (e.g. name it feedback_db), then select the database by using the command “use [**database**]”. Then create a table that stores password and username. To create this table type the command CREATE TABLE user (id INT AUTO_INCREMENT PRIMARY KEY, username VARCHAR(255) NOT NULL, password VARCHAR (255) NOT NULL);

- Stage 6: Insert entries into the table, type INSERT INTO user (username, password) VALUES ('greater' , 'northumbria'); At the end of this stage, the database, table, and credentials to login, should be all functional

- Stage 7 is the stage where you create a custom HTML page and connect you page to MySQL by using PHP. Create a directory in /var/www/ by using the mkdir -p command and name the new directory w21007692. Navigate to this directory (e.g. cd /var/www/w21007692) and create an html page called index.html by typing the command “sudo nano index.html” and create a PHP file named login.php by typing the command “sudo nano login.php”. Create an HTML file with this code:

```
<!DOCTYPE html>
<html>
<head>
<title>Login Page</title>
</head>
<body>
<form action="login.php" method="post">
<label for="username">Username:</label><br>
<input type="text" id="username" name="username" required><br><br>
<label for="password">Password:</label><br>
<input type="text" id="password" name="password" required><br><br>

<button type="submit">Login</button>

</form>
</body>
</html>
```

Then write this code in a file with extension .php:

```
<?php
if ($_SERVER["REQUEST_METHOD"] == "POST") {
        $username = $_POST['username'];
        $password = $_POST['password'];

        $host = 'localhost';
        $dbUsername = 'w21007692';
        $dbPassword = 'northumbria';
        $dbName= 'feedback_db' ;
        $conn = new mysqli($host, $dbUsername , $dbPassword , $dbName);

        if ($conn->connect_error) {
                die("Connection failed: " . $conn-> connect_error);
        }
        $sql = "SELECT * FROM user WHERE username = '$username' AND password = '$password'";
        $result = $conn->query($sql);

        if ($result->num_rows == 1) {
                echo 'Login successful';
        } else {
                echo 'invalid username or password';
        }

        $conn->close();
}
?>
```

- Stage 8: type the URL of the server in Kali(in my case 192.168.232.3) and everything should be ready to be attacked

III Windows server

- Stage 1: Install Windows Server 2019 essential on VMware workstation player version 15. I named my Windows server as “Greateromorose”
- Stage 2: Press Windows + X and choose Network Connections, right-click on the network adapter you want to configure, and choose ‘properties’. In the properties window select ‘Internet Protocol Version 4’. Select “use the following IP address” and type 192.168.232.8, with subnet mask 255.255.255.240, and default gateway 192.168.232.2.
- Stage 3: For “preferred DNS option” type 8.8.8.8 and 8.8.4.4
- Stage 4: Navigate to apachlounge.com/download/ and install the latest version of Apache (in my case version 2.4.59). When you press the download button, the zip file will be saved in the Download folder (in my case it was saved in C:\Users\Greateromorose\Downloads)
- After unzipping the downloaded file, navigate to C:\Users\Greateromorose\Downloads\Apache24\ and edit the “conf” file, by changing SRVROOT and adding the path to the Apache file which in my case is

C:/Users/Greateromorose/Downloads/Apache24, then change the listen to 0.0.0.0:8080 which will allow the Apache server to listen to all ipv4 connections on port 8080. Then add in the “Server name section” the IP address 192.168.232.8:8080.

- Stage 5: Install PHP by navigating to php.net and clicking download PHP version 8.3.6. Upon installation extract the zip file.
- Stage 5: Open the windows search bar and look for “edit the system environment variables”. In the advanced window select environment variables, select Path, click edit and add a new path. The new path should match the location of the php extracted file, in my case “C:\php-8.3.6. To verify if this works go to the command prompt and type the command php -v and if you get an output that shows the PHP version then it means that PHP was successfully configured.
- Stage 6: Since our website will use PHP to connect to MySQL, the PHP module needs to be enabled. So, scroll at the bottom of the conf file and enable the PHP module. By typing the command:

  LoadModule php_module "C:/php-8.3.6/php8apache2_4.dll"
  AddType application/x-httpd-php .php
  PHPIniDir "C:/php-8.3.6"
  LoadFile "C:/php-8.3.6/php8ts.dll"

  After saving the file, open the PHP unzipped file and look for a file named php_development and rename it php_ini. Edit php_ini by adding at the bottom line:
  extension_dir = "C:\php-8.3.6\ext
  extension=mysqli.
  The reason why I renamed the php_development, is because Apache will not recognize this file unless its renamed php_ini.
- Stage 7: In this stage you have to install MySQL, to do this you have to navigate to mysql.com and download MySQL version 8.3.0. Upon completion of the installation, follow the screen prompts, e.g. create a root password, and add user. In this case the user will be w21007692 and the password will be Northumbria. This stage is very important, because if not done correctly then the PHP connection will not work due to incorrect credentials. After clicking Next, follow the screen prompts.
- Stage 8: Create a database and table that contains the login credentials. To do this, Open the MySQL 8.3 terminal and type “CREATE TABLE user (id INT AUTO_INCREMENT PRIMARY KEY, username VARCHAR(255) NOT NULL, password VARCHAR (255) NOT NULL); and after executing this command type INSERT INTO user (username, password) VALUES ('greater' , 'northumbria');
- Stage 9: Navigate to the location of htdocs in my case C:\Users\Greateromorose\Downloads\Apache24\htdocs and create a file with html extension and another file with php extension Create an HTML file with this code:

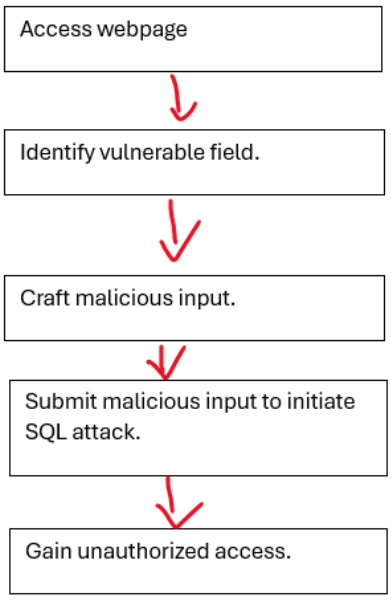
```
<!DOCTYPE html>
<html>
<head>
<title>Login Page</title>
</head>
<body>
<form action="login.php" method="post">
<label for="username">Username:</label><br>
<input type="text" id="username" name="username" required><br><br>
<label for="password">Password:</label><br>
<input type="text" id="password" name="password" required><br><br>

<button type="submit">Login</button>

</form>
</body>
</html>
```

Create a PHP file with this code:

```
<?php
if ($_SERVER["REQUEST_METHOD"] == "POST") {
        $username = $_POST['username'];
        $password = $_POST['password'];

        $host = 'localhost';
        $dbUsername = 'w21007692';
        $dbPassword = 'northumbria';
        $dbName= 'feedback_db' ;
        $conn = new mysqli($host, $dbUsername , $dbPassword , $dbName);

        if ($conn->connect_error) {
                die("Connection failed: " . $conn->connect_error);
        }
        $sql = "SELECT * FROM user WHERE username = '$username' AND password = '$password'";
        $result = $conn->query($sql);

        if ($result->num_rows == 1) {
                echo 'Login successful';
        } else {
                echo 'invalid username or password';
        }

        $conn->close();
}
?>
```

- After configuring all the necessary Apache extensions and config files, you can fully install Apache by going into the command prompt as administrator and use the command CD to navigate to the bin file within the Apache folder (C:\Users\Greateromorose\Downloads\Apache24\bin). Once the current directory is changed, type the command httpd -k install to install Apache. Open the Windows search bar and type services and open it. After opening it, look for Apache service and click Start.
- In case you are not able to connect to the Apache server it might be that the firewall is blocking the connection from external connections. To disable the firewall, navigate to the Windows search bar and search for Windows Defender Firewall with advanced security. Select “New Rule” and follow the command prompt to “allow connection on port” option. Then select the port you are interested in, which in this case is port 8080. After adding this new rule, all external connections should be allowed on this port
- Finally, everything should be functional and ready to attack.

C. *Execution of the Attack*



To start the attack, I made sure that the Kali, Windows and Linux server were configured with the right ip address, subnet mask and default gateway. Then I started Apache on both servers.
Open 2 tabs on the kali web browser. On one tab type 192.168.232.3(linux server ip address) and on the other tab type 192.168.232.8(windows server ip address). Both tabs displayed the same layout for the login interface that prompts user to type their credentials: the Username and the password.

On the Windows server I performed different attempts with different combinations of usernames and passwords to gain access but only 'Greater' and 'Student' (the admin credentials) granted me access. The fact that the website successfully authenticated me only when I typed the current credentials, means that the authentication system is fully functional. I tried this on Linux server as well, and only the correct admin credential authenticated me.

The aim of this attack is to gain access to the admin account without having knowledge of the password. In the attack I assumed that the OSINT stage of finding the admin username was already done because detailing about OSINT procedures falls outside the scope of my research. The aim of the research is to show how easy it is to bypass a PHP script that uses SQL without prepared statements. For these reasons, I just assumed that I already knew the username of the admin, which is greater, and that the only credential I was missing was the admin password. Furthermore, since my research does not discuss about tools to perform SQL injections, I decided to perform a manual SQL injection without the use of tools like SQL map. I knew that the general SQL script for a login page would look something like this: " SELECT * FROM users WHERE username = '$user' AND password = '$pass'. I used this assumption to my advantage and deduced that if the SQL query was somehow modified to something like this: **SELECT * FROM users WHERE username = 'greater' AND password = '$pass' or '1'='1'-- '** then I would be able to authenticate as the administrator. To be able to do this, I typed in the username field 'greater' and in the password field ' or '1'='1. I then added the comment symbol '-- ' which forces the SQL query to execute only up to 1=1 and not after.
Ultimately this manipulation allowed me to log in as the Admin. From there, I could have performed different types of cyber-attacks such as sending phishing messages to colleagues, customer data theft, and ransom etch.

IV. RESULT/DISCUSSION

The SQL injection attack demonstrated the vulnerabilities present in the login system authentication process. In the initial stage of our SQL Login bypass attack simulation, the preparation and setup involved ensuring all servers and network configurations were correctly established. As expected, all servers, including the Linux and Windows servers, along with their network configurations, were working properly because I was able to view the html sites on Kali Linux.
On the Linux server, the login system was working and communicating with MySQL because I got no error messages when accessing the web page. On the Linux server, I expected the login system to prevent any user who does not have the username 'greater' and the password 'student' from logging in. Upon testing it on my webserver to see if the authentication system works, I tried random combinations but none of them worked, I know this because every time I inserted the wrong credentials it would display the message "invalid username or password. Only when I wrote the correct credentials, did the system successfully grant me access and displayed the message:" Login successful" which means that the login system is functional.
On the Linux Server web page, I typed the username 'greater' and the password **' or '1'='1**. Upon clicking the login button, I expected the system to log me in and as expected the system granted me access to the admin account, indicating a successful bypass of the login mechanism. I know that it was successful because whenever I typed the wrong credentials it would display an error message saying: "invalid Username or password" but when I typed the injected command it displayed the message:" login successful". I made 5 attack attempts on the Linux server and all attempts successfully granted me admin access systems because on all attempts I got the message "Login successful" ,this means that the success rate of this attack was 100%.

Upon successful login on both servers, no further malicious actions were demonstrated but the successful access to the admin account highlighted the potential for data theft or other malicious activities, emphasizing the severity of the SQL injection vulnerability and the importance of robust security measures to mitigate such risks.

I tried the same method of attack on the Windows server. So, I tested if the login system was functional by intentionally typing the wrong parameters and as expected none of the parameters worked the only thing that happened was that if I got the wrong login credential then it would display an error message saying invalid username or password but when I typed the correct username and password, it would log me in, and display the message "Login Successful". I know it logged me in because it displayed a message that said "login Successful", this means that the login system works. To perform the attack, I inputted the username 'greater' (which is the admin name), and in the password field I typed **' or '1'='1**. Once I clicked login it displayed a message saying, "login successful". I tried this attack 5 times and all attempts were successful because on all attempts I got the message "Login successful", so this means that the success rate of this attack was 100%.

As I mentioned earlier, this attack (both on Windows and Linux) exploited an unsanitized SQL query that was written in the PHP file. Since the username and password parameters were unsanitized, it allowed me to inject these malicious parameters. The fact that I was able to bypass the system with a simple statement, proves the gravity of this attack because it is very easy to perform, and not just the admin accounts could be in danger but an attacker could also be able to bypass any customer account and potentially still their assets such as debit and credit card. Other potential damages are:

- An attacker with admin privileges could modify or delete critical data, alter system configuration, and install malware.
- Admin access could allow the attacker to perform identity theft, which means that they could impersonate legitimate users and access their accounts.
- The attacker could perform privilege escalation and compromise more accounts which would expand the scope of their attack.
- A successful attack will cause reputation damage to the organization because it will cause reduce trust between customers, partners, and stakeholders.

V. Countermeasure and Prevention Strategies

A Countermeasures

The reason why the attack was successful is because my PHP code "put too much trust" in the user input. For these reasons, the attacker was able to use this "trust" to his advantage and perform the login bypass. To prevent this from happening, we need to use prepared statements.
Prepared statements, separate the SQL query from the user input this key characteristic will allow the SQL query execution to be more secure reducing the risk of a successful SQL injection attack.
To make a prepared statement there are several steps:

- Prepare: The SQL query is sent to our database server without any specific parameters. The database will analyze and compile the query to prepare it for execution.
- Parameters: in the SQL query use '?' which is used in the SQL query to indicate where the user input will be inserted later.
- Bind: This is the stage in which the values for the placeholder are bound to the prepared statement.
- Execute: Finally, the prepared statement is executed with the inserted parameters.

This will be my new code:

```
<?php
if ($_SERVER["REQUEST_METHOD"] == "POST") {
        $username = $_POST['username'];
        $password = $_POST['password'];

        $host = 'localhost';
        $dbUsername = 'w21007692';
        $dbPassword = 'northumbria';
        $dbName= 'feedback_db' ;

        $conn = new mysqli($host, $dbUsername , $dbPassword
, $dbName);

        if ($conn->connect_error) {
                die("Connection failed: " . $conn->
connect_error);
        }
        $sql = "SELECT * FROM user WHERE username = ?
AND password = ?";
        $stmt = $conn->prepare($sql);

        $stmt->bind_param("ss", $username, $password);
        $stmt->execute();
        $result = $stmt->get_result();

        if ($result->num_rows == 1) {
                echo 'Login successful';
        } else {
                echo 'invalid username or password';
        }

        $conn->close();
}
?>
```

The highlighted section is what I changed from my previous code. I wrote my code by following all the steps mentioned earlier. Also, I added "ss" to indicate the type of parameters being bound to the placeholder in the SQL query. In this case, it stands for 2 string parameters.

B. SQL injection prevention strategies

There are multiple prevention strategies for SQL attacks [9]:

- Prepared Statements: The issue with SQL is that it allows user's input to be directly inserted into the SQL query, which can be exploited by the attacker, but with prepared statements instead of inserting user input in the SQL query directly, parameters are used. This approach separates the SQL logic from the user input, making it harder for attackers to inject malicious code because you are telling the database exactly where to put the user input.
- Using Stored Procedures: Using Stored Procedures helps separate user input from SQL code to prevent it from being directly executed. Unlike prepared statements where the SQL code template is sent from the application; a Stored Procedure has its SQL code saved on the database server and is called by the application when needed.[9]
- Validating User Input is important because it ensures that the user input matches the accepted type, length, format, etc.
  Only after the input has been validated then it should be permitted to move on and engage with other areas of the system, like the database.
- Limiting privileges: Be mindful of the privileges given to user accounts. Always make sure to give the minimum possible privileges because if an attacker is able to compromise the account, then he would be more limited.
- Encrypting data: Always encrypt data because if attackers are able to compromise and steal system data, then the encrypted data would prevent the attacker from reading the content of sensitive information.

VI. Reflection

Conducting this research enabled me to understand the types of SQL injection attacks, the execution strategies, and the prevention strategies. The simulation setup for both Linux and Windows servers gave me more experience and confidence in configuring server environments, installing software such as Apache, PHP, and MySQL, and executing SQL injection attacks. I described each step to set up the attack in great detail so it can be easily replicated by people with less experience in performing SQL injections.
Although my report talked about countermeasures against SQL injection, it doesn't address what to do in the case of an SQL injection being successful so, for example, it doesn't mention about incident response plan or what to do in case the "prepared statement security" has been bypassed and the organization system has been compromised.
Furthermore, my attack simulation doesn't address how OSINT works, instead it assumes that the attacker already knows the admin username. I believe it would have been more beneficial to talk about OSINT and prevention strategies.
I also noticed that my server setup (Linux and Windows) took too much time to configure because I started it without doing a setup

plan, this caused me to perform a lot of troubleshooting due to incorrect commands etc. The reason why I didn't make a setup plan was because I underestimated the difficulty in setting up a webserver, so for future projects, I should not underestimate the setup process, instead, I should always make a plan in advance. Making a plan would have allowed me to manage my time better. Overall, the SQL injection attack simulation provided me with valuable insight into SQL injection attacks, for these reasons I believe I will be able to apply this knowledge in performing ethical penetration testing for Organizations that gave me consent to pentest their systems against SQL injections.

References

[1] Nasereddin, M., ALKhamaiseh, A., Qasaimeh, M., & Al-Qassas, R. (2023). A systematic review of detection and prevention techniques of SQL injection attacks. Information Security Journal: A Global Perspective, 32(4), 252–265. Available at: https://www.tandfonline.com/doi/full/10.1080/19393555.2021.1995537?scroll=top&needAccess=true

[2] Celko, J. (2005). *The Language of SQL, 2nd Edition*. Elsevier. Available at: http://www.simplestemporal.com/AAS/pdf/The%20Language%20of%20SQL,%202nd%20Edition.pdf

[3] 10 SQL Injection Attacks Statistics To Know in 2023 By Selma Citakovic (May 23, 2023). Available at: https://securityescape.com/sql-injection-attacks-statistics/#:~:text=A%20single%20malicious%20injection%20can%20steal%20a%20startling,stole%20over%201%20billion%20user%20IDs%20and%20passwords.

[4] Alenezi, M. (2020). SQL Injection Attacks: Countermeasures & Assessments. Available at: https://www.researchgate.net/profile/Mamdouh-Alenezi-2/publication/344597081_SQL_Injection_Attacks_Countermeasures_Assessments/links/5fcc5c6345851568d142b19a/SQL-Injection-Attacks-Countermeasures-Assessments.pdf

[5] Horner, Matthew and Hyslip, Thomas (2017) "SQL Injection: The Longest Running Sequel in Programming History," Journal of Digital Forensics, Security and Law: Vol. 12 : No. 2 , Article 10.. Available at: https://commons.erau.edu/cgi/viewcontent.cgi?article=1475&context=jdfsl

[6] N. Singh, M. Dayal, R. S. Raw and S. Kumar, "SQL injection: Types, methodology, attack queries and prevention," *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*, New Delhi, India, 2016, pp. 2872-2876. Available at: https://ieeexplore.ieee.org/abstract/document/7724789

[7] A. Rai, M. M. I. Miraz, D. Das, H. Kaur and Swati, "SQL Injection: Classification and Prevention," *2021 2nd International Conference on Intelligent Engineering and Management (ICIEM)*, London, United Kingdom, 2021, pp. 367-372. Available at: https://ieeexplore.ieee.org/abstract/document/9445347

[8] SQL Injection Attack: How It Works, Examples and Prevention *BrightSec*. Available at: https://brightsec.com/blog/sql-injection-attack/

[9] Z. C. S. S. Hlaing and M. Khaing, "A Detection and Prevention Technique on SQL Injection Attacks," *2020 IEEE Conference on Computer Applications(ICCA)*, Yangon, Myanmar, 2020, pp. 1-6, doi: https://ieeexplore.ieee.org/abstract/document/9022833

[10] A Beginner's Guide to SQL Injection: Bypassing Login by Chamindu Nayantha. Available: https://medium.com/@chamindunayantha/a-beginners-guide-to-sql-injection-bypassing-login-177fff394f8a

[11] Ethical, legal and environmental impact - CCEA Legislation, How does life, society and the law react to the evolution of digital technology? available: https://www.bbc.co.uk/bitesize/guides/z8m36yc/revision/