# COMP3911 - Secure Computing

**Leyna Tiji - sc23lt - 201712295**
**Mario Manini - sc23mm2 - 201729419**
**Jonghyun Kim - sc23j3k - 201743821**
**Finlay Bridges - sc23fb - 201759422**
**Jake Lowther-Spittlehouse - ll22jls - 201660983**

# Analysis of Flaws

1. By inputting an SQL command into the User ID and Patient Surname field, we can retrieve the user account information, including passwords.

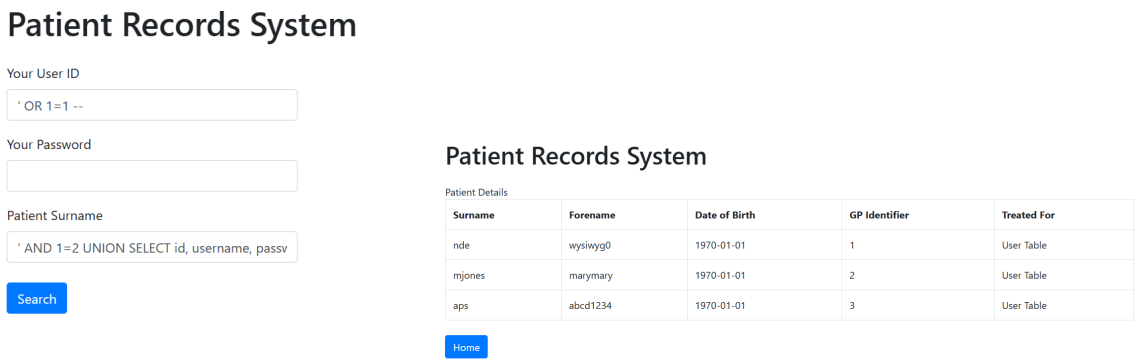
2. If “ ‘ OR 1=1 – “ is entered into the patient surname field with a correct user ID and password in the other fields, a list of all the patient’s records is shown instead of a single patient.

3. Passwords are stored in the database in plain text, rather than being encrypted.

4. After entering certain SQL commands (INSERT or DROP) into a field, the website takes you to an error page which contains sensitive information such as info on the file structure of the server and the services used.

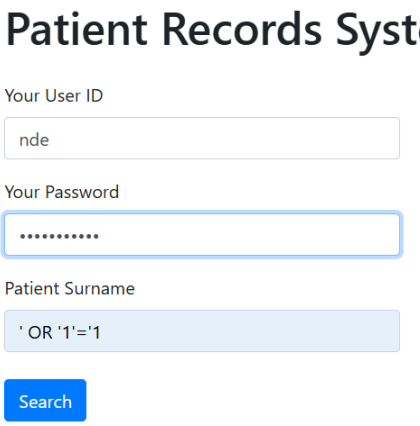
5. A user who has the authority to create or edit records is able to insert HTML or Javascript into patient data that is rendered in its entirety by a client browser viewing records, this allows malicious javascript to be run on the client.

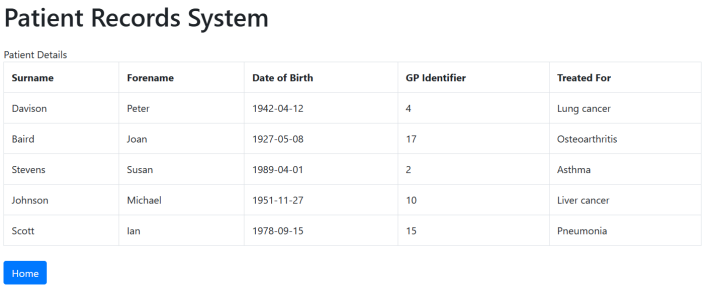
### Flaw 1: Retrieving all User Accounts




The screenshots above illustrate that the two SQL commands paired together perform an SQL injection that is able to retrieve all user accounts held in the database, including the usernames, passwords, user IDs and further personal information. This also violates data protection laws as data is not sanitised.

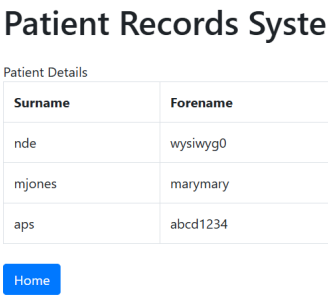
### Flaw 2: Retrieving information about all patients



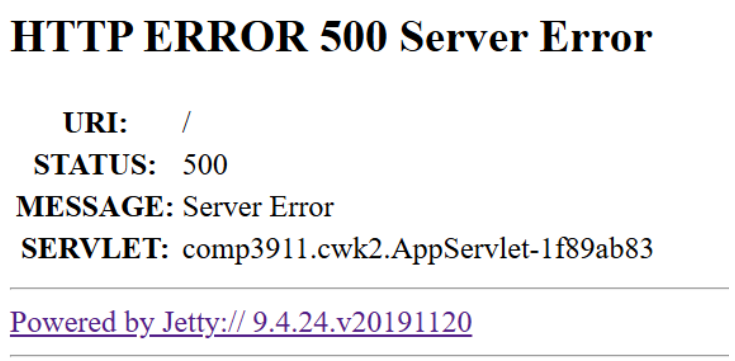

This flaw is a vulnerability to SQL injection, discovered by trying the "' OR 1 = 1 –" in the patient surname section. This is significant because it allows you to see information about any patient, even if you don't know their surname.

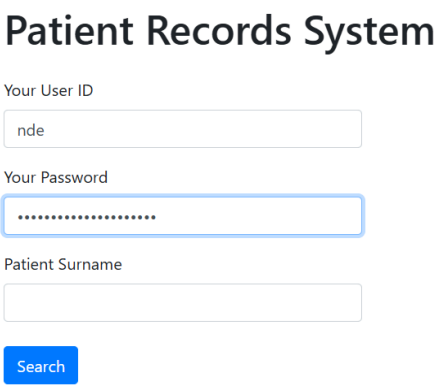
### Flaw 3: Passwords are Unhashed




The previous flaw uncovered a third flaw in the system, passwords are being stored in plain text in the database. This means that when performing an SQL injection and retrieving database information, passwords can be read easily. The passwords should be hashed and encrypted before being stored in the database.

### Flaw 4: Error Page

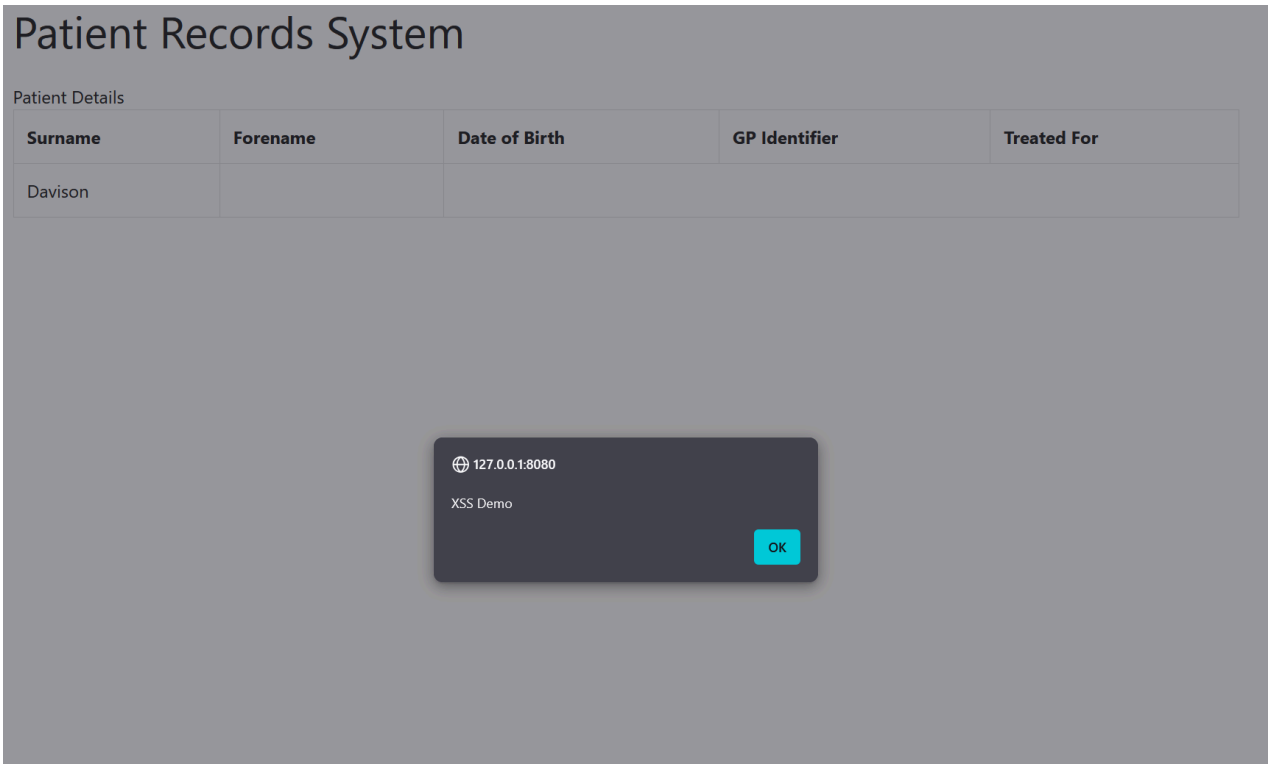


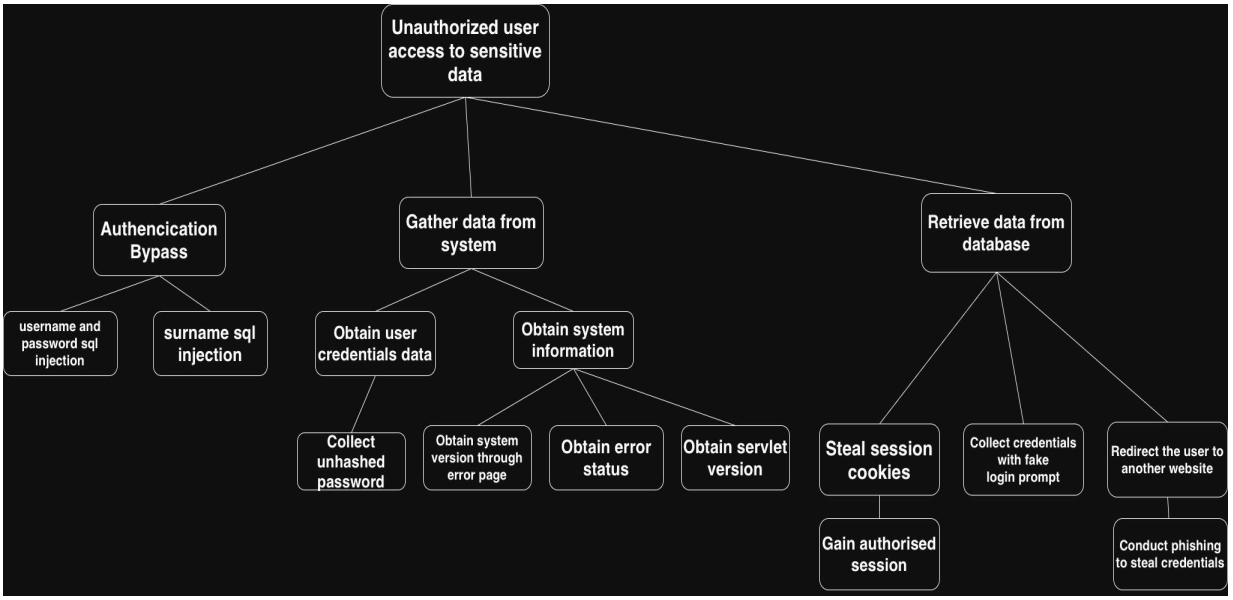

By inputting ' DROP TABLE = Users;' into the password field (works with many other SQL commands), the application produces an error message that exposes sensitive information. Ideally, the code should handle issues like this to prevent an error page from popping up.

**Flaw 5 : XXS vulnerability in the Free Make template engine**

The freemake template engine renders the html pages found in the templates folder with data retrieved from the database, the template engine does not escape special characters in data retrieved from the database so if a patient or user record contains data that could be interpreted by the browser as a script tag such as having a patient's name be set to `<script>window.alert("XXS Demo")</script>` it will be interpreted as a valid HTML tag by the client's browser and in the case of script tags ran as valid Javascript, this allows an attacker to change any aspect of the web page and construct sophisticated attacks to steal passwords or cookies for example.



# Attack Tree




The attack tree has been constructed under what the attacker can achieve through each vulnerability. The attacker's ultimate goal is to gain access to the patient's sensitive data without authorisation. The tree illustrates the main paths leading to this goal, which include bypassing authentication, obtaining information leaked from the system, and acquiring data directly from the database.

For authentication bypass, the attacker can exploit sql injection vulnerabilities to gain access to the system. Through this method, the attacker can view all the patient's sensitive health data and the system administrative data.

They can also gather data from the system, such as unhashed passwords and system configuration details. When passwords are stored in plain text, the attackers can exploit this weakness and directly steal user credentials. Furthermore, the system's error pages reveal system information such as error status, system version and servlet version, which can assist the attacker to identify vulnerabilities in the system.

Lastly, the web page is vulnerable to XSS such as stealing session cookies, constructing fake login prompts or redirecting users to phishing pages. By stealing session cookies, the attacker can hijack authorised sessions and gain direct access to the database. Through fake login prompts or phishing redirects, the attacker can obtain valid user credentials, which may also lead to direct access to patient records.

# Implementation of Security Fixes

## Fix Identification

**Flaw 1: Retrieving all user accounts**
This fix was chosen because it prevents all forms of SQL injection attacks. By using a prepared statement for the system's authentication query, all values passed in through the login form are treated strictly as data rather than SQL code. This is achieved because the database compiles and parses the structure of the prepared statement before the user's input is substituted into it.

**Flaw 2: Retrieving information about all patients**
By performing an SQL injection instead of inputting a patient surname, we are able to retrieve all patient information and sensitive data stored in the database. An unauthorised user could access all information about patients without actually having authority. We can fix both Flaws 1 and 2 by using prepared statements to protect against SQL injection attacks.

**Flaw 3: Passwords are unhashed**
When retrieving the entire user database using sql injections, we realised that the passwords were stored in plain text in the database, making them readable to hackers. This is vital to fix because in the event that a user with unauthorised access, accesses the database, they should not be able to interpret the passwords without a hashing function.

**Flaw 4: Error Page**
When performing more SQL injections into the input fields, an error page was produced. This is a major flaw as it exposes information about the directory and filename producing the code.

**Flaw 5: XSS in the FreeMake template engine**
The XSS vulnerability in the template engine is severe as it allows a multitude of other attacks to be facilitated such as cookie stealing, fake login prompts, redirects and more. Stopping XSS attacks is therefore of high importance as it can lead to many child nodes in the attack tree. To fix the XSS vulnerability in the FreeMake template engine we can utilise the inbuilt escaping functionality provided by FreeMake. Using the prebuilt escaping methods makes patching the XSS attack quick and doesn't cause any major changes to the flow of the code.

# Fixes Implemented

**Flaws 1: Authentication Vulnerability to SQL injections**
This fix involves changing the code for deciding whether a user is authenticated or not. The function "authenticated" which returns a boolean and the addition of an import for SQL's prepared statement feature are the modifications made to the code. The query which is used for retrieving the user's password is made into a prepared statement which ensures that no SQL can be inserted when querying for the username field. The parameters in the statement are bound to the variables for the username. This has fixed the problem because the database now compiles prepared statements before the user's information is substituted, which ensures that only raw values can be entered by the user and makes the function more secure.

**Flaws 2: Surname Vulnerability to SQL injections**
To fix the surname parameter in order to prevent it from being run. We decided to sanitise the query by escaping quotations (i.e. performing a swap with ‘ for “). If an attacker was to enter in a command it will now be treated like inputting a string. Therefore ‘1 == 1’ is now just the string “1 == 1” as no surname matches this string the sql server will return nothing.

**Flaw 3: Passwords are unhashed**
To fix the flaw, the BCrypt library was imported. Using this function, we were able to hash the original passwords and store them in the database, subsequently removing any plain text passwords from the database. When the program authenticates the user's input, it initially checks the username inputted. If this matches an existing username in the database then the program calls the ‘checkpw’ function from the BCrypt library to compare the entered password with the hashed password stored in the database.

**Flaw 4: Error Page**
By fixing the authentication function, the code no longer processes sql injections. This prevents an error page from ever being produced because the code will handle any type of input.

**Flaw 5: XSS in the FreeMake template engine**
Using FreeMake's own escaping functionality is simple with our current FreeMake configuration function (called on server initialization) where we set up directories, handlers and encoding. Inside the configuration function we can also set our output format to be HTML : `fm.setOutputFormat(HTMLOutputFormat.INSTANCE);`
This tells the template engine which format to match for escaping, since we have set the output format to be HTML it will match our escaping rules to any HTML in all dynamic content `${...}`. To then enable the escaping to automatically apply to all files with HTML file extensions we use `fm.setRecognizeStandardFileExtensions(true);` now all HTML files will have auto escaping applied to the dynamic content such that any data loaded from the database into the page will not be read as executable HTML/Javascript and instead will display as a string literal.