Secure Computing Coursework 2

Security Flaws Found

Unencrypted Database

The first issue that I found was that the database was not encrypted at all. All the data was easily accessible and was in plain text. I uses a website called http://inloop.github.io/sqlite-viewer/ that allowed me to be able to view the database in an easy to read format. I was able to see that all the data in the database was unencrypted. I was able to see that the passwords, user name and patients surname were all in a plain text. This is a massive security flaw as if someone gets a copy of the database then all the data is there. If the database is encrypted this is an extra thing they will need to crack in order to get this data.

Password in Plain Text

Another issue I found was that when I looked at the database I saw the passwords were in plain text. This is a problem as they have to be hashed and don't need to use a regular cypher like the other parts of the database. This should be hashed in order to make it harder for an attacker to be able to get the password even if they get the database. This way you can only check if a password entered is the same as the one in the database, yet it won't decrypt the password in the database. This will reduce the risk of the password in the database being cracked.

SQL Injection Attacks

The current database is very susceptible to different SQL injection attacks. The first thing I uncovered was that when you entered *' or '1'='1* for the user name or password. It will then allow you to access the website and find a patients records. Another thing I discovered was that if you do *' or '1'='1* for both password and user name, but then add *' or surname is not'*, for the surname field, then this will allow to access all the records for the patients. This is a massive security flaw as this will allow you to access any records without actually entering any valid details. Also if you can enter invalid details and be able to access a particular persons records, then this will be seen as a massive security flaw.
Another slightly different error is that if you enter a valid password and user name then put for surname *'SELECT * from user'*. This will come up with an error page as you have tried to access a part of the database you are not meant to. This error page will display the jetty version, which will disclose information to the attacker about what server software and version you are using. The attacker could then look for previously found exploits with the server version and attempt to utilise this for further attacks.

HTTP not HTTPS

Currently the servlets that make the website page load make use of basic HTTP servlets. This isn't as secure as using HTTPS servlets which have an extra layer of security. HTTPS makes sure that everything sent and received via the chosen port is encrypted this means that if anybody intercepts this data can't read it. This can be dangerous if there is sensitive information going back/forth through this port. I found this as when I looked at the source code for the AppServer.java and AppServlet.java, I found they were creating basic servlet handlers which isn't secure at all. This needs to be made with a secure port.

Modification Made

Parameterised Queries

The first modification that I made was to add parameterised queries. This was to combat the security flaw of the SQL injection. I modified it so that it would use a prepared statement instead of a normal SQL statement to be entered. I also changed the query so that it would not use “%s” but “?” instead. This meant that it would take in values entered in a prepared statement, which is like a compile template for SQL. This effectively means that you this will remove the threat of an SQL injection into this website. This means that the disclosure of information will no longer occur. Also this removes the error of revealing the server version to the user when incorrect login details are entered. This means it makes it a lot harder for an attacker to found out potential other attack techniques for this website.

Encrypted the Database

Another thing that I did was to encrypt the contents of the database. I used Advanced Encryption Standard (AES) to encrypt the database, which made use of a secret key that is in the source code. I made 2 functions decrypt() and encrypt(), which take in a string as a parameter then output the new decrypted/encrypted string. This will allow me to encrypt and decrypt the data using the AES cypher. I also encrypt the surname and the username entered and compare it to the values in the database. This is safer than decrypting the values in the database and then comparing them. When the data is shown it is has been decrypted from the database as soon as a valid login request is sent. I encrypted all the data apart from the ID fields. I chose AES as I feel it is as a very robust security protocol as it is used very widely for many different applications. Encrypting the database will fix the problem as even if an attacker was able to steal the database, they still will not be able to read it unless they are able to decrypt the data. This will require additional knowledge of what the secret key is, the encryption method we used and if we added any extra security data to the data.

Hashing the Password

The other thing that I didn’t encrypt was the passwords. This didn’t need to be encrypted as I decided to hash the password. This meant that there is no possible way to be able to decrypt the password from the hash in the database. The only way to check is to see if they entered the correct password is by hashing the value they entered. Then comparing that value with the passwords in the database. I hashed the passwords with SHA-256 making use of the messageDigest function to create an hashed password. This will send the hashed password to the authenticated function which will subsequently send this on as an SQL query. I made use of the SHA-256 as it is quite fast to compute and is seen as a standard for encryption currently. The reason that this fixes the problem of plain text is that it means that the password is never seen in plain text form at all. The only way of finding out if it is a correct password is by doing a brute force attack of all the combinations. This mitigates the chances of the password being cracked. Also if an attacker gets the database then they will not be able to see a plain text password inside, but a hashed one that is hard to crack.