COMP3911 Coursework 2

Safwan Chowdhury - ed18src@leeds.ac.uk
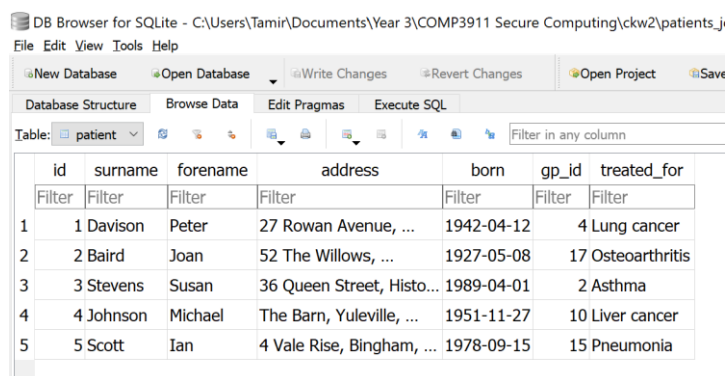Tamir Cohen - sc22tc@leeds.ac.uk

Analysis of Flaws

1. The database is not encrypted so patient data is vulnerable
2. No certificates used, HTTPS not available
3. Weak passwords are allowed making the website insecure
4. There is no timeout, captcha, or max login attempts making the website susceptible to brute force attacks
5. Passwords are not hashed and therefore vulnerable
6. The website is susceptible to SQL injection attacks

Encryption of the Database

The first flaw to be discussed is the lack of encryption in the database. If someone were to gain access to the database they would be able to view patients' sensitive personal and medical data, namely their first and last names, addresses, DoB and what they are being treated for. It is crucial that this information is encrypted so that if an attacker gains access to the database through a malicious attack they are not simply able to open and view this data. This flaw was discovered by using a database inspection tool DB Browser for SQLite and was apparent when examining the entries in the database. This is shown in Figure 1.

Figure 1: Unencrypted patient table

SQL Injection

The second flaw is that the website is susceptible to SQL injection attacks. We are able to inject using the phrase 'or '1' = '1 to attack the website and gain access to the data in the database without the need for a valid username, password or even patient surname. Upon further inspection of the server code we can identify that the query made to the server for the authentication system is as follows:

```
select * from user where username='%s' and password='%s'
```

Therefore by inserting 'or '1' = '1 into the brackets where the password entry would go, as shown in Figure 2, we create an extended SQL query:

```
select * from user where username='%s' and password=''or '1' = '1'
```

The resultant SQL query is extended by the or statement and is such that all data from the patient table will be returned for while the password is an empty string or 1 is equal to 1. Since '1'='1' always evaluates to true, the statement selects all the records from the table so long as that username exists in the table. This bypasses the authentication of the website. Further, inputting the same value in the patient surname extends the search query to

```
select * from patient where surname=''or '1' = '1' collate nocase
```

Which will similarly always return true and select all the data in the patient table as shown in Figure 2.

Figure 2: inputting an SQL Injection attack (left) and data exposed by attack (right)

Password Hashing

The third flaw is that the user passwords are stored vulnerably as plaintext in the database and are not hashed, so anyone who gains access to the database would be able to steal users' login data, undermining the security of the system. This was discovered through the inspection of the database as shown in Figure 3.

Figure 3: Users' passwords are stored unhashed in the database

Fixes Implemented

Database Encryption:

To fix database encryption we first generated a keyPair using the RSA standard and stored both these keys as private.key and public.key. We take the private key and encrypt it using a

secret key to create private.des and delete the private.key file. This provides us with an encrypted private key file. To initialise the database we ran an encryption function once which took all the patient data, bar the surnames, and encrypted them using the public key. This function was executed, and as a result the data in the database was encrypted. This function is now redundant. During execution, when a valid request is made to the server, it runs a decryption function for the private key and loads the private key to memory. It then uses the cipher decrypt mode and the private key to decrypt the requested data and display it to the user.

	id	surname	forename	address	born	gp_id	treated_for
	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	1	Davison	BLOB	BLOB	BLOB	BLOB	BLOB
2	2	Baird	BLOB	BLOB	BLOB	BLOB	BLOB
3	3	Stevens	BLOB	BLOB	BLOB	BLOB	BLOB
4	4	Johnson	BLOB	BLOB	BLOB	BLOB	BLOB
5	5	Scott	BLOB	BLOB	BLOB	BLOB	BLOB

Figure 4: Encrypted patient table shown as Binary Large Object (BLOB)

SQL Injection:

To fix SQL injection we implemented some simple changes to the SQL queries and the way they are handled. We replaced the %s fields with ? which when used in SQL ensures only the number of arguments specified will be passed to the query and executed. Therefore we can no longer pass a cleverly constructed SQL query extension to the field and expect it to execute. To ensure this worked we replaced the statement with a preparedStatement and used setString to set the variables passed to the query.

Password Hashing:

To fix the issue of passwords being stored vulnerably in the database as plaintext, we implemented password hashing using the SHA-512 algorithm in combination with a 16 byte random salt for each password. The inclusion of a salt is an additional security measure used to prevent rainbow table attacks, as precomputed rainbow tables for SHA-512 are commonly available. The resulting hash and the salt for each password is converted from binary data to a string through Base64 encoding, and these strings are then stored in the database. The database schema for the user table was modified to accommodate this. Converting the plaintext passwords was done manually with help of the Hashing.java class that we implemented. In its current form, the database contains the same three users as in its initial form, each with their passwords unchanged, as well as some additional test users. To facilitate authentication, the authenticated function in AppServlet.java was modified such that users are queried by their username, and their hashed password and salt are retrieved. The plaintext password entered is hashed with this salt (decoded back into binary data), and if the resulting hash matches the hash retrieved, the user is authenticated. It is assumed that usernames are unique, although implementing this in the database schema was beyond the scope of this coursework.

	id	name	username	password	salt
	Filter	Filter	Filter	Filter	Filter
1	1	Nick Efford	nde	Y9w+b86AdtvOS+GD...	ipz21iBZ31EYFL3gboY...
2	2	Mary Jones	mjones	IQbtUhFvPJ6tCPEpML...	bN7z1nR3yrlF4gvaagl...
3	3	Andrew Smith	aps	3YS6o5ueSWv4nfo3o...	V0B1EewcxMtkdHsJb0...
4	4	Tom Jones	tjones24	jJwOLngZZ7QWRtEJ9f...	eTwA6e83sMXhZzU4L...
5	5	Tamir Cohen	tc02	HJR1VI5U4Cv6jzaxruf...	qYW0zzF0CwMXV8oKJ...
6	6	Safwan Chowdhury	safw1	ilnFbnUgd1zc2kG4VIo...	jcmnI44XeBRBHv9kkx...
7	7	Matt Smith	matts	PkkNh8jst4RYfWdJEQ...	CrE0QK+An9BB5DeiPr...

Figure 5: Hashed passwords and salts stored in user table in database