COMP3911 Coursework 2

By Adam Brown (sc20asb) and Ynyr Evans (sc20ye)

Analysis of flaws

1. Passwords are stored using CHAR not VARCHAR, meaning that every password is exactly 8 characters.
2. Passwords are stored without any form of hashing or encryption.
3. Individual persons are matched to their addresses.
4. Individual persons are matched to their illness/conditions, which may be a privacy breach.
5. The server uses HTTP requests rather than HTTPS.
6. You can view any person's record if you know just their surname. A user's login does not have to be related to the patient (I.e.., the user doesn't have to be the patient's GP)
7. You can inject SQL into the password field and patient surname field, which allows for the attacker to carry out several malicious activities, such as viewing all patient details, dropping the patient database, and so on.
8. Cross-site scripting vulnerabilities identified in the jQuery, lodash, Angular and buefy libraries, which is due to the versions not being upgraded.
9. You can have unlimited login attempts, there is no timeout method implemented.

Flaw 1: Poor password security.

The main security issue present in the database is the password field. Mainly that it uses CHAR (8) to store passwords that must be exactly 8 characters. This means that the possible password combinations are severely limited, making it easier to brute-force. An even more problematic issue is the complete lack of password hashing.

If an attacker gains access to the records in the database, without any sort of password hashing they will be able to see users' passwords in plaintext, and thus can access the system impersonating the user. This leaves a patient's full name, address, date of birth and diagnoses all as vulnerable. The attacker could also attempt to use these passwords on other applications or sell them on the dark web.

This was discovered by inspecting db.sqlite3's schema and records in the user table. We discovered the CHAR (8) flaw through the schema of the password field, and the lack of encryption in the password field through an inspection of the records in the table.

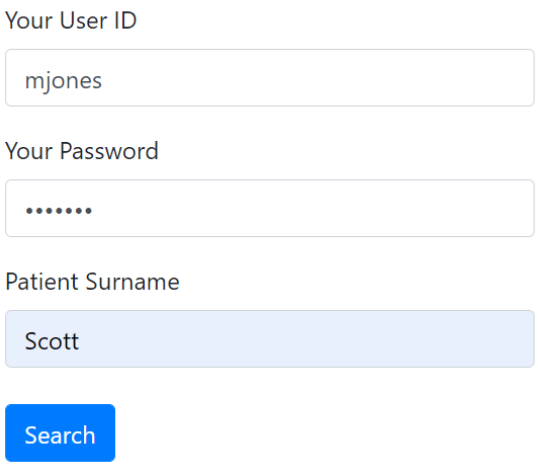
user		CREATE TABLE user (id integer not n
id	integer	"id" integer NOT NULL
name	varchar(30)	"name" varchar(30) NOT NULL
username	varchar(12)	"username" varchar(12) NOT NULL
password	char(8)	"password" char(8) NOT NULL

	id	name	username	password
	Filter	Filter	Filter	Filter
1	1	Nick Efford	nde	wysiwyg0
2	2	Mary Jones	mjones	marymary
3	3	Andrew Smith	aps	abcd1234

Flaw 2: Lack of trust boundaries

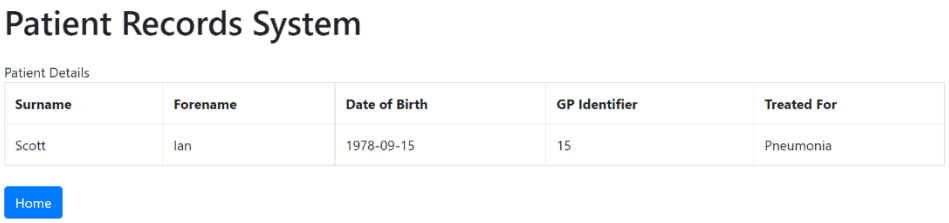
Another serious security flaw is how there are no trust boundaries in the system, excluding the initial trust boundary to log in. For example, in this program there are no checks to see if the user you are currently trying

to log in as has permission to see the record of a patient. Therefore, if you supply a valid username and password, you can see a patient's full name, address, date of birth and diagnosis by only supplying a surname, without necessarily being their GP.



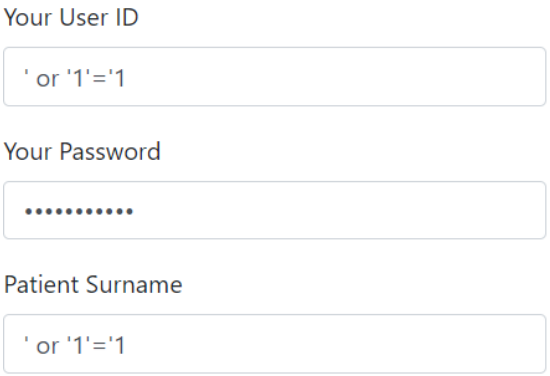
A brute force attack could therefore be used within the surname field to retrieve all information of all users by entering all possible surnames. Additionally, GP's will also receive all patients with the same surname, regardless of if they are their patients.

We discovered this flaw by analysing the code and seeing that there was no check between what user login was provided and which patient record was retrieved. We then tested this in the interface and to demonstrate that it was possible. The accompanying screen shots show how a GP can access the record of a patient that is not theirs.



Flaw 3: SQL injections

This solution leaves itself open to SQL injections within all the input fields on the login page. This allows a malicious user to bypass the user authentication, which then allows them to view any patient's details that they'd like. The password field in the example below has the same entry as the User ID field, which is an SQL statement that is injected.



In addition to this, the user can also bypass the searching for user feature entirely and instead have the entire table of users' information returned to them, as shown below. This was discovered by examining the code, seeing that prepared statements weren't used for the SQL commands and instead commands were merely concatenated strings.

Patient Details

Surname	Forename	Date of Birth	GP Identifier	Treated For
Davison	Peter	1942-04-12	4	Lung cancer
Baird	Joan	1927-05-08	17	Osteoarthritis
Stevens	Susan	1989-04-01	2	Asthma
Johnson	Michael	1951-11-27	10	Liver cancer
Scott	Ian	1978-09-15	15	Pneumonia

Fixes implemented

Fix one

To address the insecure way in which passwords were stored, we decided to instead store the SHA-512 hash of the user's password. SHA-512 was chosen as it is sufficiently long in length that it is secure and very hard to crack. We also salted the password and stored each user's unique salt within the database. This helps to guard against rainbow tables being used to reverse engineer our users' passwords form the stored hashes, as the salt will change the dictionary required for this.

In addition to this, we also added an SHA-256 pepper to the user's password before salting and hashing. This pepper value is only stored within our source code, which adds another layer of security as an attacker would be unable to use rainbow tables to easily crack our hashes even if they had full access to our database, due to them not having access to the pepper value.

To do all of this, we also updated the database so that the password field is data type char(512) and added the salt field.

Fix two

For the trust boundary issue, a user can only see their own patients from now on (we are assuming that the 'user' table means GP). This was implemented by creating an INNER JOIN command on the user table with the patient information, with the query only returning patient records where there is both the surname matches the input, and the GP ID of that record is the same as value as the ID of the user who's logged in.

We changed the database to make GP ID in the patient table a foreign key of id in the user table, thus linking the 2 tables together.

Fix three

We fixed the issue of SQL injection with the use of prepared statements. This means that all data input by the user is passed into our SQL queries as strictly strings and therefore cannot be run as SQL code.