Challenge

Question 1

What is the name of the only Orthopedist?

Question 2

What is Katie Cain’s profession?

Challenge (cont.)

Question 3

How many medical professionals can be found in this registry?

Question 4

What is the name of the person who has a password of "greyblob"?

Challenge (cont.)

Question 5

What is Mike Torres' password?

Analysis

Determine Normal Usage

User can search for names which then returns a table containing the name and the profession of the employee.

Web App Exploit Checklist (for NCL but these are good to check in any web app exploit test)

- [ ] Robots.txt
- [ ] Sitemap.xml
- [ ] Cookies
- [ ] Javascript code

Nothing again? This is getting ridiculous

Looking at the network data when we use as intended, we can see we are redirected to the page `/professionals.html?q=[userInput]`

Seeing the `?q=[Input]` let's us know that this website uses queries. Combined with the fact we're getitng tables, probably a SQL query.

We can confirm this by passing in bad queries like `1"` and looking at the errors given.

Exploiting

Now we can move onto the real exploit by looking at all the information we have determined.

Information

- Site contains a database of employee information
- The site uses SQLite

Using the Information

Since the only thing we can interact with on this page is a search bar that runs a SQL query based on our input, odds are we will be crafting a SQL injection.

There are many ways to do this and is often a pain since we have to do this manually.

Designing a SQL Injection

Breaking Out of the Query

A good place to start is attempting to break out of the currenty query.

Most vulnerbale SQL queries (when searching for strings as we are) encase the query inside of either `"` or `'`.

If we try making our search `1"` or `1'` we get two results. The `1"` gives us an error for a bad query, so we may be closer to the right track here.

Next we can add a `;` as that ends statements in SQLLite.

Crafting Advanced Injections

Now that we have broken out of the query by finishing it and starting a new line, we can try to get our information.

We will `SELECT` all fields `*` `FROM` a table `WHERE` any character (`%`) is `=` to any information in the table by leaving the statement open for the databse to process (ending in an `"`).

Final Query

Putting it all together results in a query that looks like.

```
1"; SELECT * FROM USERS WHERE "%"="
```

We use the `USERS` table as that is a common name.

We can query other information as well with queries like `1"; select sqlite_version();`

This is a good repo for example SQL injections: swissky/PayloadsAllTheThings/SQL Injection

Result

Entering our designed query

```
1"; SELECT * FROM USERS WHERE "%"="
```

will reveal all the information in the database, including columns we couldn't previously see before.