Databases - SQL Injections

- What is a database - a structured collection of data
- We use Databases because JSON objects are just too large
- We use these to structure and organize information
    - Easier to look at databases than just dumps
- Persistence across restarts
- Large amounts of data can be stored efficiently - postgres - this is where you can go through millions of tables to get information

What does a database look like? In Code

```
Create table person (
        Name text,
        Phone number integer
        Email_address text
);
Create table address (....)
```

A lot of SQL code is self explanatory but it can get complicated very quickly

SQL Commands

- Select - selecting from a table
- Insert - putting into a table
- Update - updating an entry
- Delete - deleting an entry
- Join - combining two tables into one big tables - more columns
- Union - combining two select queries into one long table - more roles
    - Usually when we want to join tables in attacks commonly we want to use union

Filters so it's a little more concise

- Where - add a constraint to a query where the clause is true
- Limit - limit to only n results

SQL is a relational database - there are well defined relationships here between things

- Relationship between fields is clearly defined and clear

Things to note

- When you create a table, you will need to index everything in that table with a key - is an ID with an integer usually
    - Can have unique values that create a key
    - Primary keys - unique values to distinguish a unique entry of a table
    - Unique - similar to primary key but not used for referencing - mostly for cleanliness and data upkeep
    - Foreign keys - used to reference columns of other tables

- Are used to find relationships between things - reference points in the database

Implement SQL connection in flask

```
Import sqlite3
Connection = sqlite.connect('database.db')
With open('schema.sql') as f:
        connection.executescript(f.read())
connection.commit()
connection.close()
```

connection.commit() - important - any time you run a command the command gets stored to a local cache, doesn't get sent unless you commit it (so it uploads it to a remote server)

In command line sqlite3 has their own command line where you can run to run sql commands

* is all fields, not every entry

So an example SQL command is SELECT * FROM grades WHERE email == "admin@admin.com"

The way that you define it in the schema will be the default order

Things can get really finicky really quickly - is things like this that can open up vulnerabilities

If you have something like SELECT * FROM email "' " - opens us up to a sql injection vulnerability because you can send in a ' and then comment out things

{%%} and {% in %} - means in jinja code that means run the python code here instead of html

SQL Injection - occurs when we are able to insert control characters into our data

- Is referred to as injection
- Mixes data and control

E.g.

```
Select username from users where
        Username == 'request.data['username'] and
        Password == 'request.data['password']; --I'll implement hashing eventually
```

Here we can put what we want here - to get around the formatting issues that may come up -- will comment out everything after what you out in there

Injecting the code:
Select username from users where
Username == ‘admin’ and
Password == ‘1’ or ‘1’ = ‘1’;

When it comes to this there is bug after bug after bug

-- does not always need space before and after them. But in our infrastructure we do need the space between them

Datatypes need to be the same if you are unioning - if you’re still getting an error

- Null is all datatypes so the suggestion is to use if you are still getting errors

All of the infrastructure that we know of is done in mysql

- In situations where you don't have the source code
- Select * from inforamation_schema.table

Sql - LIKE does string comparison
LIKE “%QUERY%”;

SELECT * FROM staff WHERE id == 3332654 AND date LIKE "%QUERY%" - what we think payroll looks like

SELECT * FROM staff WHERE id == 3332654 AND date LIKE "%” OR 1=1; --test%"

Start with 1=1 for select statements
Could check for SQLi by going 1=2 which will be false so if nothing comes up then this is vulnerable to SQLi vulnerability

COMP6443{SQLiIsPowerful}

When we find that something is vulnerable to these injections we want to go look for what other tables are

We add this to the query to get more information - we want a union select to get more information in the table
To select from another table we want a union

“ UNION SELECT 1,1,1,1,1,1, table_name FROM information_schema.tables; --test” %”

- To make the union work we need the same number of columns which is what the 1’s are for

- Test out different numbers of 1 here to

SQLMAP is forbidden - no reason to run it, it will dos the infrastructure - is noisy - people would know that you are doing ddosing

There is no situation in a sql injection where you would want to do sql injections for this instance

There is another table called upcoming_layoffs - look into a sql query to select that table

Injections using inserts

Anywhere you think there is going to be a database - check for a SQL database

- Don't use DROP TABLE grades; - could break things down the line because SQL doesn't have checks on top of things

Remediating against SQLi - prepared statements and parameterized queries

- Current modern standard is to use prepared statements and parameterized queries
    - Giving placeholders in to ensure that you have what you are expecting
    - Anytime you are getting query that doesn't suit the specified parameters break
- Parameterized queries - instead of defining it as a string, you can in SQL a get users statement looks like - if someone tries to inject a different type of SQL command then the system will error out instead of actually executing the command

Weaker method

- Escape all control characters - this is good practice regardless but there's always a way around it (e.g. ', ", %')
    - Problems with this is if you have a way to escape it, the attacker has a way of escaping the escaping
    - There are github example where we can get around escaping control characters
- Allow listing - you explicitly define which users can access not as helpful as you can data leak within the same table - e.g. user passwords for example
    - Always a way around whitelisting though

Parameterised query - this is what the statement is going to look like - building this into the SQL - becomes difficult for people to interact with it because its inside the database

Blind sqli - when there is no feedback from the front end

- Some people think that one way to patch this out is to not have error messages
    - But there are ways in which you can structure sql queries to extract some information
- Is an extended topic but it will be mentioned

Xp_cmdshell - just runs the SQL as an injection