Bryce Jensen
CS-405
2-2 Activity: SQL Injection Coding
9-12-24

Process for preventing a SQL Injection in the run_query function:

1. Clear any prior results
`records.clear();`

2. Create a lowercase version of the query for case-insensitive matching
`std::string lower_sql = sql;` `std::transform(lower_sql.begin(), lower_sql.end(), lower_sql.begin(), ::tolower);`

3. Use a regular expression (regex) to detect the "or value=value" pattern	
`std::regex injection_pattern(R"(\sor\s+['"]?\w+['"]?\s*=\s*['"]?\w+['"]?\s*;?)");`	
• `\sor\s+` ○ `\s` ○ `or` ○ `\s+`	"or" surrounded by whitespace. ← Matches any whitespace character. ← Matches the word "or". ← Matches one or more whitespace characters.
• `['"]?\w+['"]?` ○ `['"]?` ○ `\w+`	A word surrounded by single or double quotes. ← Matches a single or double quote, optionally. ← Matches one or more words.
• `\s*=\s*` ○ `\s*` ○ `=`	An equal sign surrounded by 0 or more spaces. ← Matches 0 or more spaces ← Matches a literal equal sign (=)
• `\s*;?` ○ `\s*` ○ `;?`	A semicolon followed by an optional semicolon. ← Matches 0 or more spaces ← Matches an optional literal semicolon (;)

4. Use regex_search to check if the SQL query matches the injection pattern—place it in an if statement.

`if (std::regex_search(lower_sql, injection_pattern)) {`
`  // Step 5 here`
`}`

5. If a potential SQL injection is detected, print a message and cancel the query by returning false—place in the if statement from step 4.
`std::cout << "Potential SQL Injection detected. Stopped: " << sql << std::endl;` `return false;`

Bryce Jensen
CS-405
2-2 Activity: SQL Injection Coding
9-12-24

6. Execute the SQL Query using `sqlite3_exec`.
   If unsuccessful, print a message, free the error, and return failure.

```
char *error_message;
if (sqlite3_exec(db, sql.c_str(), callback, &records, &error_message)
      != SQLITE_OK) {
    std::cout << "Data failed to be queried from USERS table. ERROR =
           " << error_message << std::endl;
    sqlite3_free(error_message);
    return false;
}
```

7. Return successful

```
return true;
```

Output to the console:

```
SQL Injection Example
Connected to the database.
USERS table created.

SQL: SELECT * from USERS ==> 4 records found.
User: Fred [UID=1 PWD=Flinstone]
User: Barney [UID=2 PWD=Rubble]
User: Wilma [UID=3 PWD=Flinstone]
User: Betty [UID=4 PWD=Rubble]

SQL: SELECT ID, NAME, PASSWORD FROM USERS WHERE NAME='Fred' ==> 1 records found.
User: Fred [UID=1 PWD=Flinstone]
Potential SQL Injection detected. Stopped: SELECT ID, NAME, PASSWORD FROM USERS WHERE NAME='Fred' or 1=1;
Potential SQL Injection detected. Stopped: SELECT ID, NAME, PASSWORD FROM USERS WHERE NAME='Fred' or 'hi'='hi';
Potential SQL Injection detected. Stopped: SELECT ID, NAME, PASSWORD FROM USERS WHERE NAME='Fred' or 'hack'='hack';
Potential SQL Injection detected. Stopped: SELECT ID, NAME, PASSWORD FROM USERS WHERE NAME='Fred' or 2=2;
Potential SQL Injection detected. Stopped: SELECT ID, NAME, PASSWORD FROM USERS WHERE NAME='Fred' or 'hi'='hi';

C:\Users\bryce\OneDrive - SNHU\(CS-405)\CS-405-M2-SQLInjection_code_files\x64\Debug\CS-405-M2-SQLInjection_code_files.exe (process 11480) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```