Task 1: Command Injection

The vulnerability in Example 1 of the Command Injection section of the website was obvious and also stated in the Pentester Lab pdf. The command that was put into the web server's terminal was located in the url of the website, and there aren't any checks for extra commands if they're placed within the URL and separated using semicolons.

Making the shell was fairly straight forward. Using the skeleton REPL code provided in the assignment along with the python requests library, I was able to make an HTTP GET request to the server and pull out the html response associated with it. With some formatting with split, I isolated the lines associated with the vulnerability using python substrings and joined them together for readability in the shell.

Task 2: SQL Injection

The vulnerability in Example 1 of the SQL Injection was a little trickier. Simply adding a semicolon for extra SQL queries and escaping the ending single quote with a hash wasn't enough. Instead, I utilized UNION queries to gather information about the databases that were being used. Because I've never done this kind of SQL injection before, I played around with the information_schema database as well as looked at several UNION-based SQL Injection lines to figure out how to extrapolate data from the databases as well as possible.

The shell itself was also a bit more complicated. Because of the nature of the information in the html response, all of the information was on a single line (except for dumping the table). Isolating this line and splitting on the pike character ("|"), iterating

through the array (skipping every other item in the array due to the databases/tables/columns being separated by double pikes, leading to empty strings in between), I was able to get the information about the databases, tables, and columns efficiently.

Because the website can only display 3 columns at a time, I had to get creative with how I displayed all of the information in the table in the event that more than 3 columns existed within the table. The shell makes a separate HTTP GET request for each column. It then isolates the information stored in the column from the HTTP response message and stores it in a 2d array. The 2d array is then printed on the shell.