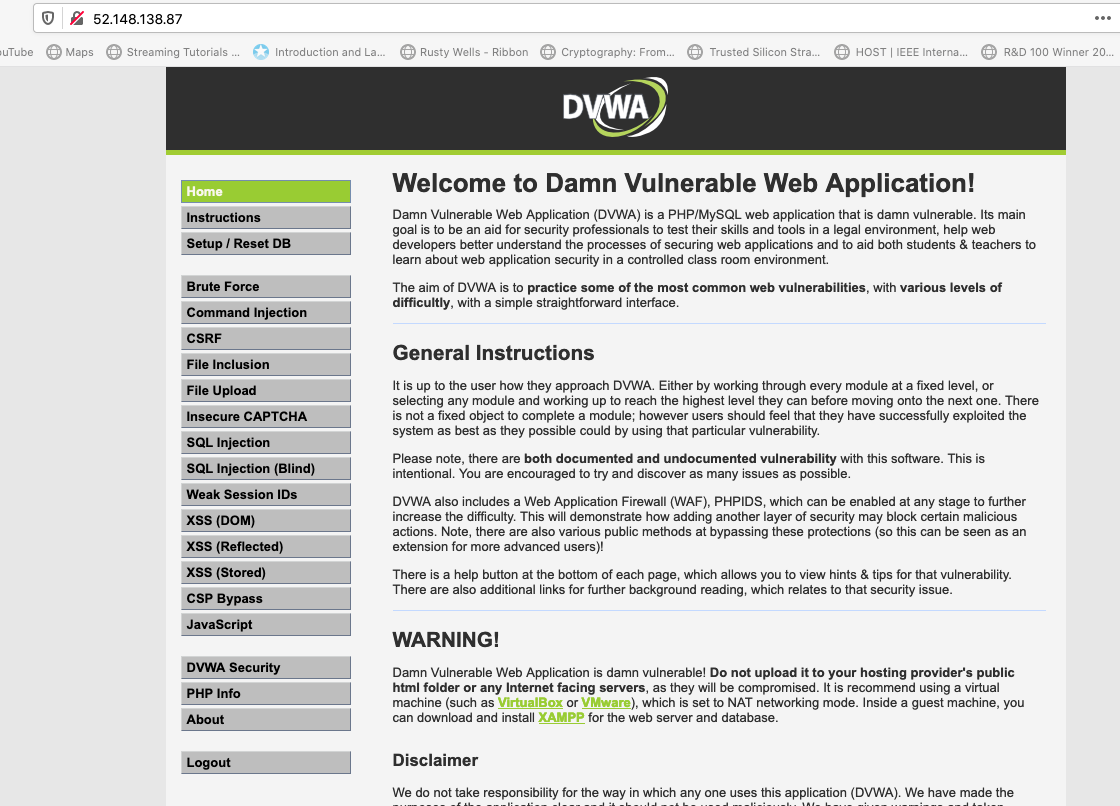
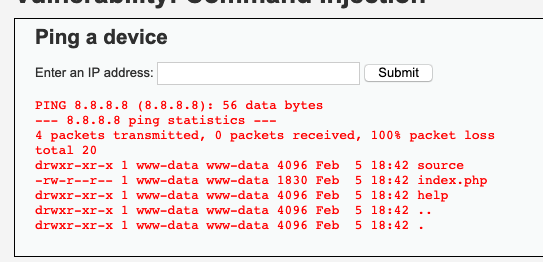
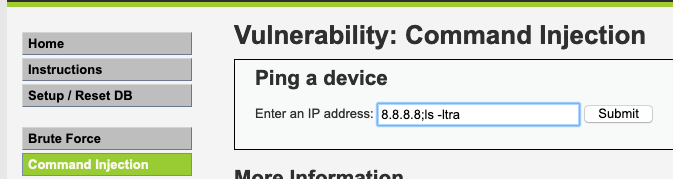
Extra Item: DVWA—Damn Vulnerable Web Application

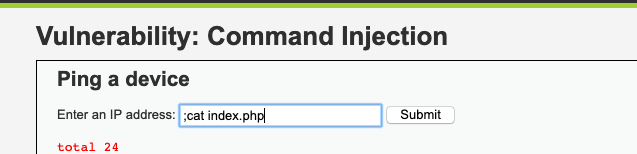
One objective of this exercise was to setup a DVWA instance. The activities didn’t emphasize using it to learn about Web vulnerabilities. So I will explore a few since it provides a valuable hands-on experience with web vulnerabilities.

Landing page: Here’s what you see when the load balancer public IP entered in the browser:

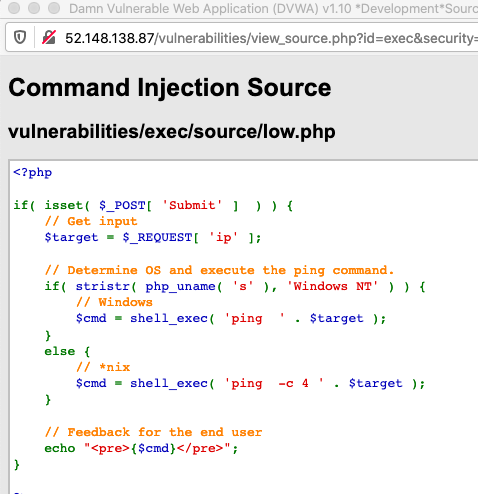
Improper coding leads to potential disasters. Here malicious form input that is not sanitized exposes the web application to hacking. Commands can be chained through the form by putting a semicolon followed by a command. Here the command is a directory listing. It shows things the user should not see.



Don’t even need to bother to enter the IP address. Just enter a semi-colon followed by a command. This means the developer did not even check to see if a valid IP was entered. The below command shows the contents of that file.



Here is the source code for that page



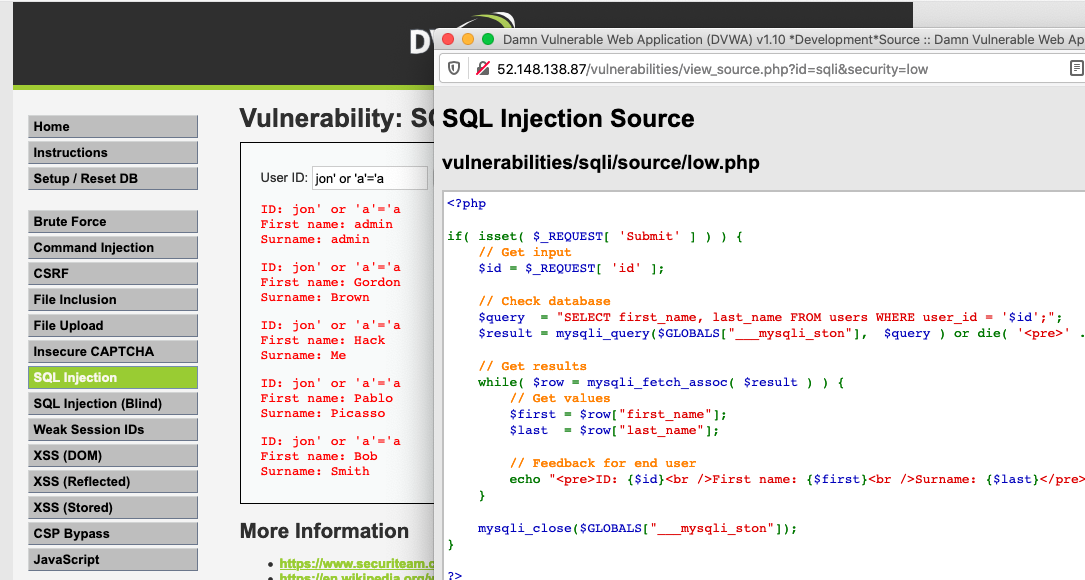
The php simply takes whatever the user enters and pre-prends ping thus running ping on the IP the user is supposed to enter. The user can run any command he desires after the ping. This could lead to entering some command such as what was demonstrated in class:

;echo “This is hacked” > ../../index.php

Which effectively hijacks the site by replacing the landing page with the hacked message.

Mitigate by coding the php so it insures only an IP address and nothing else gets passed along to shell\_exec.

**SQL Injection:** This is similar to code injection but operates on the database with SQL commands.

Here is the php code and page for the SQL Injection page when jon’ or ‘a’=‘a is entered into the User ID box

The entire contents of the user table is displayed. This is not secure.

The developer’s intention is simply that the user only enters a user ID into the box, for example 1. Then the web page will return whether or not that user\_id is in the DB and the details about that user\_id. This would be mitigated by the developer only allowing integers values to be passed from the form.

The query is formed and captured into the $query php variable:

$query  = "SELECT first\_name, last\_name FROM users WHERE user\_id = '$id';";

The result the developer expected would be a query like this:

SELECT first\_name, last\_name FROM users WHERE user\_id = 1

However here with the malicious input the query becomes

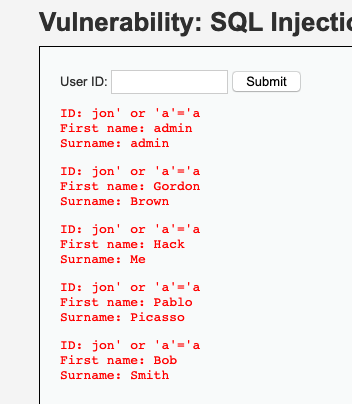
SELECT first\_name, last\_name FROM users WHERE user\_id = ‘jon’

Or ‘a’ = ‘a’;

The above is always true so it will fetch all table rows.

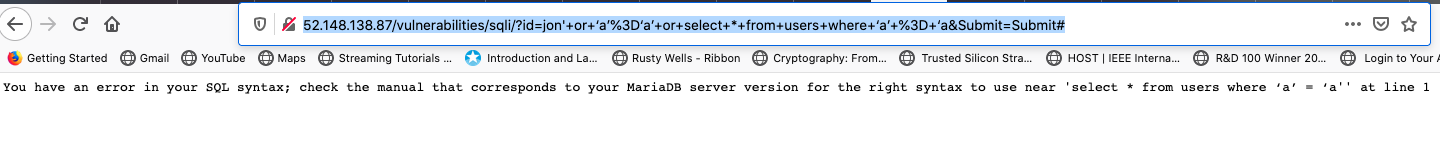
You can imagine even more malicious users here such as injecting a SHOW TABLES comment to learn all the tables in the DB. Then the hacker can also pull info from them. Or injecting DROP \* which wipes out the DB.

This DVWA site is very interesting and I learned a lot from it and plan to work through more examples.



Entering this lead to a SQL error:

jon' or ‘a’=‘a’ or select \* from users where ‘a’ = ‘a



Now with that URL it’s even easier to attack.

URL: http://52.148.138.87/vulnerabilities/sqli/?id=jon%27+or+%E2%80%98a%E2%80%99%3D%E2%80%98a%E2%80%99+or+select+\*+from+users+where+%E2%80%98a%E2%80%99+%3D+%E2%80%98a&Submit=Submit#