Challenge URL: /dvwa/vulnerabilities/sqli_blind/
Objective: Find the version of the SQL database software through a blind SQL attack.
Tools needed: None

Did you remember to read this section's README?

The Guide

THIS IS A WORK IN PROGRESS, TO BE COMPLETED LATER.

Examining the Form

Here's our form:

We know from the related Challenge 7 that there are five user IDs: 1, 2, 3, 4, and 5. So let's test the form by inputting "1":

Now just for fun, let's try a user ID that we know doesn't exist, like "6".

So we get a message of "User ID exists in the database." for valid, "true" entries, and "User ID is MISSING from the database." for invalid, "false" entries.

Discovering the SQL Injection

In our last section, we discovered that the returned message differs based on whether the input was "valid" or not. That's incredibly important information. You see, the key issue with blind SQL injection (SQLi) attacks is that we can't directly view the result of the query; it usually gets filtered or processed before something is returned to the end user. So by having clearly defined "true" and "false" messages, the server will still leak information. As long as we can craft a SQLi exploit that can return a a true/false answer, we can leverage that to pull arbitrary information. It'll take longer than your basic SQLi, but is still effective. So let's start building a query that gives is what we want.

For basic SQLi attacks, we could have a query like:

```
1' or (select version()) #
```

But since we can only get a "true"/"false", we can approach this in two ways:

1. Iterate through every version number possible and do a string comparison. For example, we could do something like `test 5.0.0.0, 5.0.0.1, ... , 5.1.0.0, ... until we get a match`. In my mind, this isn't feasible since we don't know the exact format of the version number. What if it's X.X.X.X, or XX.XX.XX.XX, or XX.XXXXXX? Since we don't really have a way of knowing when doing a blind test, this option doesn't seem effective.
2. Iterate through each character in the version number. We could try something like `grab the first character of the version number, then iterate through all alphanumeric ASCII characters until we find a match. We then move to the second character of`

`the version number, then third, and so on until we get a blank space (implying the end of the version string)`. Since this is format-agnostic, this seems like the best place to start.

Further, since we don't know the version length, plus since we do know the range of alphanumeric ASCII characters is somewhat large, this sounds like a prime opportunity to build a script to automate this process!

THIS IS A WORK IN PROGRESS, TO BE COMPLETED LATER.