Cyber Matters

7. Implementation: Authentication

Now that I have a semi-working web application that gets information from a very limited range of sources and then displays it, it's time to return to the authentication framework I wrote about in Post #5. In order to make things easier, I'm going to work on the Twitter side of things as a proof of concept, and then maybe I'll get to authenticating the voter data if I have time (I'll explain why the voter data is harder to apply this paradigm to later, even though it's more sensitive than Tweets and should probably be the first thing to be authenticated).

Structure:

If you remember from Post #5, my plan to ensure (or at least improve the chances) that the user of this application is strictly the person whose information they're searching for, was something like this. The user queries my app for data, and I find that data for them. Then, I use that data to give them some multiple choice questions, from which they must identify their own data. If they answer right, they get closer to authenticating, but if they answer wrong, they move closer to getting false data. Throughout, there's a chance that a given question might not contain any of their data at all, and this chance grows larger the more questions they get wrong. As I already mentioned, I'll be working on the Tweets side of this first. So, what do I need in order for this to work?

1. Random Tweets

The first thing I need, of course, is a way to get access to random tweets. I contemplated several different ways to go about this, with the first being using a Python package like Faker to simply generate false sentences and then use those as tweets. After all, what is a tweet but a string of text with a certain time, user, and set of other data associated with it? When I looked deeply into this, though, I realized that text generated truly randomly just doesn't quite work. The main reason is that randomly generated text doesn't make sense. Using a random text generator will give you a "tweet" like the following, for example,

```
find an ballet crunch maker
```

which I personally think would make a hilarious tweet, but probably isn't feasible to use as one for authentication purposes. If I saw three "tweets" like that one and then one coherent sentence, it wouldn't be difficult for me to identify the real tweet in there.

What I really needed was a large set of real tweets, and luckily, I was able to find several on Kaggle, a data repository for educators and researchers who might need data for circumstances just like mine. I settled on a dataset of over 1.6 million tweets assembled by researchers at Stanford and distributed for use by students under the terms here. I then developed a Python program which would find a small random subset of these tweets and then return it in the format that would be most helpful in my application.

2. Question Interface

I then developed a question interface using Vue.js. The gist of this is really quite simple. In place of the section of the interface that would normally show a user's tweets, I overlaid a quiz interface, which displays a table whose rows are each a potential tweet. The user's job is to click the row with the real tweet of theirs, after which their `correct_responses` counter goes up. If they click the wrong one, the counter goes down and the chances of the next question being valid go down as well. Importantly, there are two types of questions:

1. "Valid" questions have three random tweets and one true tweet from the user
2. "Invalid" questions have four random tweets with no true tweet from the user

When the `correct_responses` counter reaches the threshold value, the user's true list of tweets is displayed. If, on the other hand, the counter hits the lower threshold, the system displays a list of fake tweets from the set mentioned above.

3. Relevant Thresholds

Another necessary piece of all of this is deciding the relevant thresholds. I've settled on two correct answers and two incorrect answers being the starting thresholds (though I adjust them in the demo videos to show different aspects of the system clearly), but it's difficult to figure out what the best thresholds should be without empirical research: that is, to answer the question "what thresholds provide the best likelihood of excluding bad actors while also ensuring that almost anyone using the tool with good intentions can authenticate?" If I were to build this tool further, I would design a research schema to answer this question.

Now, check out the system in action!

VIDEO HERE