Tom McLaughlin

Steve Dimitriadis

Ayman El-sayed

Group Project Spec

Our idea was to create a twitter bot, or some other way to communicate from clients to a server. By creating such a channel we can use it to illustrate the way that servers sometimes validate requests based on timestamps.

The bot, or “listener” will wait for a signal from the student, or “client.” At that point the client will have a set time limit to enter a string of integers in correct sequence to get the answer.

Our proposed approaches are the following, but if we find a more elegant way to do it then we will choose that approach.

1. Twitter chat bot – this may not work depending on the supported complexity of client inputs
2. A twitter chat bot but through the browser, using python and selenium to automate the tasks
3. A socket communication program that I (Tom) wrote in my wireless networks and security class to allow arbitrary connections to the server. Traffic between the client and server have built in encryption. I have a preference for this because it removes some of the complexity around client/server communications.

Ultimately the information that we use is arbitrary and could be the whole task, or just a part of a larger game. For example, it could be that the student has a file that has information about what server to connect to in the metadata.

Once the student finds the metadata, connects to the server, and successfully authenticates for the password to steghide the image that had the server info in its metadata for a flag.

Our team will think about what, if anything else, that we want to do to make the game a little longer and more complicated.

Update – Included as comment on Canvas

Final Product –

We were having some trouble getting our dependencies to work and to decide on a way that we could distribute the project to people easily.

We couldn’t find a way to package the project well and still get the kind of functionality that we wanted, so we changed our approach to be single player on a single machine, played through the terminal.

Schools can change the logo that they want to display by changing the Ascii.txt file to include their school logo in Ascii text along with their own welcome message.

Questions are chosen at random from the list of questions in questions.json - a json structured file that holds key value pairs of questions and answers.

The question is chosen, presented to the user, then the user input is checked against the answer that is stored with the question. We did our best to take care of case sensitivity and other input normalization considerations to prevent users from not getting points when they know the answer.

Once all the questions have been asked the program updates the leaderboard and exits.

Some notes about configuration:

The default number of questions asked is 15 and can be changed by altering the “num\_questions” variable in the game.py file.

We could probably figure out the socket communications and make it so that it works as originally described, but we just didn’t have enough time. Maybe for extra credit or as a summer project we could do that. It would also probably work pretty well as a simple web app, which would also allow for easier configuration and question editing.