CS-330

Memo

To: Prof. Virgil

From: Mitansh Chaudhari

cc: TA. Zhang

Date: Nov- 22- 2022

Re: Security Device Program Implementation

What have I implemented in my project, and How was the implementation done?

I have implemented a program that will simulate a security device similar to those installed in homes and offices. The program reads from a user-entered string and a random number-generated string to decide whether the user or the random number generated has entered the correct access code. There are two parts to this Security Device program.

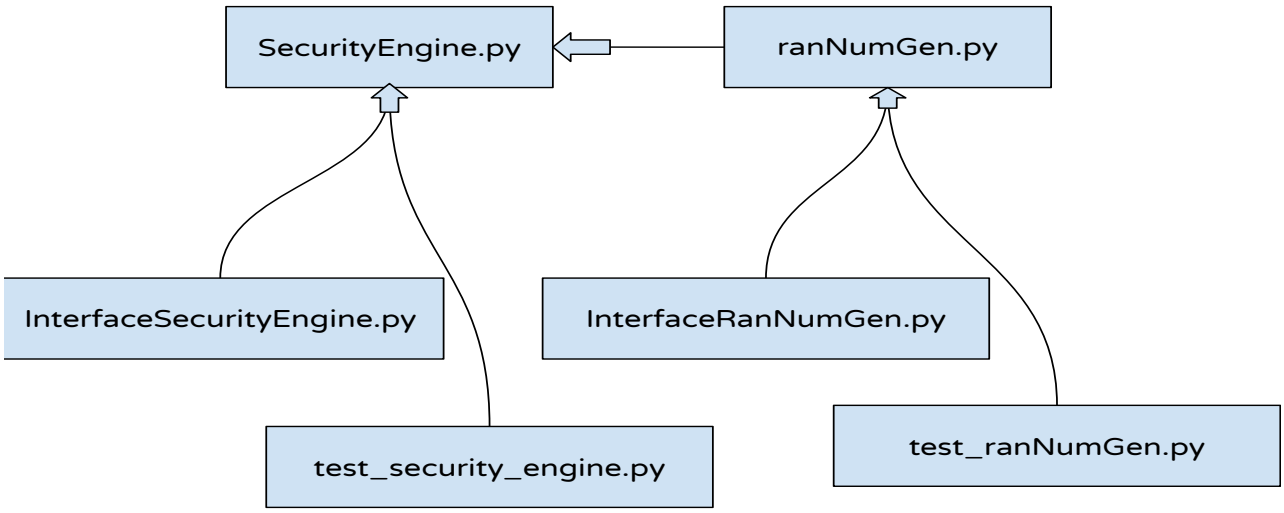
The first part will read user input to unlock the lock if it finds the correct unlocking access code. Likewise, it will also lock the lock if it finds a valid locking code. This part of the program will output the current state, such as Lock and Unlock and give stats on the number of attempts, the number of times unlocked and the number of times locked. The program discards characters other than digits 0 through 9. This implementation can be seen in the SecurityEngine.py file, where the method state_checker returns the engine's current state. The following method, setData(self, userData=""), checks if parameters are given for that method, and if no parameters are given, it asks the user to input data. It then sets that input to self.data so we can use that information in the next method, currentState(self), to calculate the program's current state. setData is a beneficial method because when random numbers are generated, and we need to validate those numbers, we can use setData(ranNum) to calculate if it will unlock the lock. The next method, currentState(self), performs the calculation to check if the data from the previous method contains correct locking or unlocking data. If the current state is lock and the given data has the valid unlocking code,

this method will print("State =Unlock"), set the current state to unlock, and prompt the user, "To stop the Engine, Press S or Try to Lock the Lock: "If the user enters s, the program sets the engine to stop state. The program sets the engine to a lock state if the user enters the correct locking code. If the user enters an incorrect locking code while in the unlock state, the engine remains in the same state. Likewise, when it's in lock state. At the end of the method, it returns the current state using the state_checker(self) method. The last method of the SecurityEngine.py file is the statsOfState(self): which returns the total number of attempts made, of those attempts, how many times you unlocked and locked the lock.

The second part of the program tests how long it takes to unlock the lock using a randomly generated sequence of numerical digits. First, the program generates a random number input string for the security device and tries to unlock the lock while ignoring the one-second wait time. Second, the program calculates the average time to break the lock if one has to wait one second to see if it is unlocked or not. This implementation can be seen in the ranNumGen.py file. This class's __init__ method initializes the variables that will help make the calculations. The next method is bruteForce(self, trails = None) which attempts to unlock the lock while ignoring the one-second wait time. It first checks if the user has provided a number of trials; if not, it defaults to 5 trials. Next, it creates an object of SecurityEngine.py so the program can check if the randomly generated number unlocks the lock. This method returns the list of random numbers that unlocked the lock in given trails. Also, the min, max, and average digits to unlock the lock. The last method of this class calculates the average time to break the lock if one has to wait one second to check the state of the lock. Similarly to the previous method, It first checks if the user has provided a number of trails; if not, it defaults to 5 trails. Next, it creates an object of SecurityEngine.py so the program can check if the randomly generated number unlocks the lock. The method returns a list containing the time(in seconds) it took to unlock the lock. It also returns the average time(in seconds) to unlock.

Other Python files in the app folder of my projects, such as InterfaceSecurityEngine.py, InterfaceRanNumGen.py, test_security_engine.py, and test_ranNumGen.py, are the extensions of the two main files, SecurityEngine.py, and ranNumGen.py.

Following is the Hierarchy:



What are my findings?

On average it would take a human about 7 to 8 days to break the lock if they have to wait one second. But a computer can unlock the lock in seconds, therefore this sheds light on the importance of cyber security in the modern world. We need to design and implement programs that are not easy to break. As the world moves towards quantum computing, the aspects of cyber security and the program's vulnerability will need to be evaluated. The Finite automata in my program is implemented with Python.

What is the regular expression corresponding to the implemented language?

A Regular Expression or also known as RegEx is a sequence of characters that defines a search pattern. For example the search pattern for the unlock code would be: s = “my unlock code is 204801” math = re.search(r’d/d/d/d/d/d’,s) Since Python has built-in support for regular function, we can import re module to search for patterns. However, in my program I use an array with a predefined index to look for the pattern. Which is similar to the Python RegEx module.

Attach the state transition diagram for the FA: