VCU RE Software Challenge Final Write Up

Summary

> I have a more in depth write up about code functionality from a previous challenge. In this write up I am focusing on the difference in password seed generation instead of rehashing the same steps I used to figure out what the program does.

- The password was seeded with time which meant that it is being dynamically created. It would change every second so there was no way I could extract it with static analysis. It used the epoch time using the c time library.
   seed being used
- Through using ghidra I was able to get a sense of the work flow of the program. I first began to try and use the rand() in python and try to predict the seed. This proved more difficult as I was not using the same rand() library and not able to get the correct results.
  - At this time I would constantly step through ida and analyze the rand() function to see what I could be missing. I tried using the ctypes library in python but was not able to get the same rand() vales even with the correct seed.
    seed function
- I tried recreating the program in C while compiling with gcc and could not get the same results. I took a few days break to try and figure out what to do.
  - Reading online I could see people keep talking about how vulnerable it is to seed with time. This made me think of trying to synchronize and predict a time or try and brute force a bunch of guesses.
- I write my own program and looked at the disassembly to see how my version of rand() work. I realized that it looked way different then the one in ida/ghidra. I realized that it was calling rand from the **msvcrt.dll**. Upon more research I realized that it is using a different algorithm in its rand() function.
  - I spent a bunch of time trying to figure out different ways to compile with visual studio. I was so happy to figure out what my issue was. I see that it is very easy to over look such things when analyzing.
     msvcrt.dll
- I decided to decipher the program all in C just to be consistent with the compiler since rand() was very specific.
  - I created a python script that would run the c code to decipher the password and then use that password as an input for the executable file. It takes a few attempts to make sure that the epoch time matches up but it is possible for the password to be cracked.
     solution

Code Appendix

run.py

> run `python run.py` (windows machine)

```
import subprocess

password_cracker = "password_cracker.exe"


executable_name = 'softwarechallenge_expert.exe'
input_file = 'correct_password.txt'

subprocess.run(password_cracker, shell=True)
with open(input_file, 'r') as file:
    input_data = file.read()
subprocess.run(executable_name, input=input_data, text=True, shell=True)

print("\nInput Data: " , input_data)
```

password_cracker.c

> Using *x64 Native Tools Command prompt* run `cl password_cracker.c`

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main() {
    time_t current_time = time(NULL);

    srand((unsigned)current_time);

    int seeded_values[8];
    int answer[12];
    int buffer_values[12] = { 59, 25, 89, 18, 63, 60, 32, 28, 95, 74, 11, 0
};

    for (int i = 0; i < 8; ++i) {
        seeded_values[i] = rand();
    }
    for (int i = 0; i < 8; ++i) {
```

```
        seeded_values[i] = seeded_values[i] % 94 + 33;
    }

    for (int i = 0; i < 12; ++i) {
        answer[i] = buffer_values[i] ^ seeded_values[i % 8];

    }
    printf("\n");

    FILE *file = fopen("correct_password.txt", "w");
    if (file == NULL) {
        fprintf(stderr, "Error opening file.\n");
        return 1;

    for (int i = 0; i < 12; ++i) {
        fputc((char)answer[i], file);
    }

    fclose(file);

    return 0;
}
```