Final

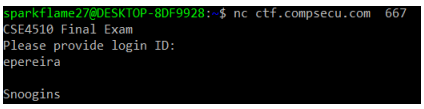
CSE4510: Software Reverse Engineering Section 01
Professor: Christopher Stricklan
Eric Pereira

May 3, 2019

To start, we are given two files for the final and these hold a flag we are supposed to catch. There are two stages to this final so we will start with stage 1:

Stage 1

Stage 1 starts by asking for a login id. The login id is pretty simple, it is our standard Florida Tech usernames, so my username is: **epereira2015**. After that it has an empty line and if you give it another newline character it will go to what is the failcase; `Snoogins`.



I went to explore the binary to see what the issue was. I explored the binary of the `finalexam` executable. I started down the rabbit hole, which started with the `main`. following the `main` led me to the `exam` function, and the first call in this after the login id is the `Stage1Check` function. This function had a peculiar function, the `hash` function, which is displayed below, showing the logic.

Essentially what this does is it loops as many times as there are letters in `arg1`, and `arg1` is the login id you input. What it does in the loop is it gets the ASCII value of each letter in the submitted login id and it adds them together. For my username this means that I need to get the character value of each letter/symbol in the name and add them together. If I add each letter in my login id together I get the value 1045 and upon testing it out I get a result that brings me to a checkerboard!

It is also important to note there is an error if you do not user the right login id. If the login id that is used does not exist as a `.so` file in the libraries it will not pass stage 1, as the example below shows:

```
sparkflame27@DESKTOP-8DF9928:~$ nc ctf.compsecu.com  667
CSE4510 Final Exam
Please provide login ID:
a
97
/home/finalexam/libs/a.so: cannot open shared object file: No such file or directory
```

Stage 2

Stage 2 starts with a game of Connect 4. This game of Connect 4 asks for Player X's input, Player X being me, the user, and once I give a number coordinating to a specific column a response will come from Player Y, an automated bot. This game is actually relatively easy, I discovered that the bot plays the same exact moves each time. This makes it relatively easy to win. Winning provides a screen that looks like:

```
Game Over!

| * * * * * * |
| * * * * * * |
| * * * * * * |
| * * X * * * |
| * * X * * * |
| * * X * * * |
| Y * X * Y Y |
Player  X Wins!
Congratulations! Here's your flag: Tr0ll1ng; you are not l33t
```

This is very clearly a Ben Kenobi "These aren't the flags you are looking for" moment. This meant it was time to go down the rabbit hole of the binary. I went into the `Stage2` function, and immediately saw the check I mentioned in the previous **Stage 1** section, where I realized I could not simply use any username but only login id's with matching `.so` files in the libraries. I then decided that the issue be in my `.so` file, as `Stage2` seems to end right there, and not do too much, so I went to my `epereira2015.so` to see if anything there could give me a hint as to what exactly is going on. I started by going into the `StartGame` function. Reviewing this function I noticed there were calls to an object called `Logger`, which I assumed was a way to log my moves so that maybe the bot could make its decision, but I thought that didn't make sense either because it seemed to play the same moves no matter what I did. I found it strange, but let it continue. Upon the continuation of this I noticed a `success` and scrolling down I saw a string that would print a string with what would be my flag. Strange, I did not see it write when I ran it though, maybe it is a special case? That did not seem to make too much sense. Then I noticed something strange, very strange, and this led me down another rabbit hole. This is when I saw the result was printing to the `Logger`!

```
lea     eax, [esi-0x52bc]  {data_12d44, "Congratulations You got the 100%…"  }
push    eax {var_298}  {data_12d44, "Congratulations You got the 100%…"  }
call    Logger::Write
```

Upon this discovery I was confused as to where exactly the `Logger` was writing to, I needed access to it somehow. I went back to see if there were any hints as to exactly what I needed. In the `StartGame` function I was looking through to see if there were any calls to the `Logger` , and there was one distinct call I noticed, one that seemed it would be important as it opened a new Socket. I went to see in the `OpenSocket` call in `Logger`, and went exploring through each function inside it. I noticed that there was a number being pushed with an string, a string saying open socket.

```
push    eax  {11045, "6Logger10OpenSocketEv"}
call    htons
```

This number, 11045, is the number I believed to be the socket number, and it looked suspicious as well because it was my hash number (1045) added to

10,000. Upon testing however, I ran into a few problems, I believed they may have been server side, I am uncertain as to how many people in the class have also experienced this problem. I got strange output in the log that looked like:

```
  Playing Move at: 3
<PlayMove Begin
  Player Playing Move: 3 X
<CheckFullColumn Begin>
Collecting Player Input
  Human Player Input
  Playing Move at: 3
<PlayMove Begin
  Player Playing Move: 3 X
<CheckFullColumn Begin>
Collecting Player Input
  Human Player Input
  Playing Move at: 3
<PlayMove Begin
  Player Playing Move: 3 X
<CheckFullColumn Begin>
Collecting Player Input
  Human Player Input
  Playing Move at: 3
<PlayMove Begin
  Player Playing Move: 3 X
<CheckFullColumn Begin>
Collecting Player Input
  Human Player Input
  Playing Move at: 3
<PlayMove Begin
  Player Playing Move: 3 X
<CheckFullColumn Begin>
Collecting Player Input
  Human Player Input
  Playing Move at: 3
<PlayMove Begin
  Player Playing Move: 3 X
<CheckFullColumn Begin>
Collecting Player Input
  Human Player Input
  Playing Move at: 3
<PlayMove Begin
  Player Playing Move: 3 X
<CheckFullColumn Begin>
Collecting Player Input
  Human Player Input
  Playing Move at: 3
<PlayMove Begin
  Player Playing Move: 3 X
<CheckFullColumn Begin>
Collecting Player Input
  Human Player Input
  Playing Move at: 3
<PlayMove Begin
  Player Playing Move: 3 X
<CheckFullColumn Begin>
Collecting Player Input
  Human Player Input
  Playing Move at: 3
<PlayMove Begin
  Player Playing Move: 3 X
<CheckFullColumn Begin>
Collecting Player Input
  Human Player Input
  Playing Move at: 3
<PlayMove Begin
  Player Playing Move: 3 X
<CheckFullColumn Begin>
Collecting Player Input
```

This is before I put in anything, not to mention I usually press "2" because my specific bot allows me to win if I type "2" four times in a row. I eventually got 1 working piece with my flag, however I had to use a friends VPS (I believe something is wrong with the connection in some way because it worked through the VPS but not my laptop), the result I found was this:

```
<CheckWinCol Begin>
  We have a Winner!
  Game Over!
<DisplayBoard Begin>
Displaying Board  - ROWS: 7 COLS: 6
DisplayBoard End>
<CheckFullBoard Begin>
<CheckFullColumn Begin>
<CheckFullColumn Begin>
returning win
Congratulations You got the 100%% Flag! Here's your
flag: {d70e1ad6-de10-4e58-8e62-deeaa51783aa}
```

Although going through a friends VPS worked, I was unable to create and write a script including this (I was not able to use his VPS for long, and did not work with the script much there). My script allowed me to pass Stage 1 and get the Stage 2 `TrOlling` flag, but I am unable to really read the log files and as a result my script does not output anything for the second flag. I wrote some information in the notes as to ideally what I would use to print out my results in the log, but as mentioned previously, it seems that I was unable to completely test or write a decent script for this.