Lab 04 Meet Ghidra

This lab will give you an opportunity to start working with Ghidra and look at some of the coding constructs that we've worked with. The lab should be completed on your Windows machine.

Instructions

1. Open up Ghidra and create a new Non-Shared project with the name of **LName_Lab04** (replace "LName" with your actual last name). You can save the project directory out to your Desktop.

2. Once you have your new project, import the **lab04.exe** binary.

3. After importing the binary, open it up in the CodeBrowser and let Ghidra auto analyze it with the default settings.

4. Uh oh! This binary was compiled *without* debug symbols in it. That means we don't get the friendly function names or anything like that. You'll need to find the main function. One way to tackle this is to use the Symbol Tree (left side of CodeBrowser) and locate the **entry** function. Within **entry**, you should see a call to another function:

```
2 void entry(void)
3
4 {
5   __security_init_cookie();
6   FUN_140001438();
7   return;
8 }
```

   Our main function gets assigned to a **var2** you are looking at the decompiler. Careful, though, it gets assigned to two different functions based on some security context; one of the two is correct.

5. Once you've found main, rename the function to be **LName_main**. You'll notice in the Symbol Tree that your function has been renamed and should now be easy to find.

6. The **main** function calls two other user-defined functions. Rename the first one called as **LName_f1** and the second one as **LName_f2**.

7. Function **f2** returns something back to main, but it's not hex. Find the value that is being returned and convert it into a **Char Sequence**. Once you've changed it, add an end of line (**EOL**) comment that says "FName changed this!" replacing FName with your first name.

8. In Ghidra, you will find static values and strings in the **.rodata** section of memory. Use the program tree to go to that section of memory. You should see some familiar-looking strings if you've run the program before. There's also a secret string hiding there. Add a **Pre Comment** above the secret string that says "**FName found the string**".

9. When you find something interesting, it's useful to see *where* in the code that object is referenced. Right-click on the memory address of the secret string and go to **References → Show References to Address**. The Location References Provider window will show you every instruction that references that address in some way. All of them will be within **f1**.

10. The Function Graph view helps you understand any logic/code flow in a function. Our **f2** function has a loop and some if/else logic. Go to **f2** and view the Function Graph.

11. In the CodeBrowser, navigate back to the **main** function.

12. Ghidra has a powerful scripting engine built into it. Open up the Script Manager and browse around some of the sections to get a feel of what's already provided. In the Strings folder there is a script, **RecursiveStringFinder.py**. The script will start at your currently selected function in the CodeBrowser and recursively search through all of the calls to gather the strings that are used. Run it on **main**.

13. Save your work in the CodeBrowser and close the window.

14. Save your Ghidra Project as well.