CS390R: Project 0

February 7, 2023

Please submit all screenshots in one pdf file with each clearly labeled (The corresponding question) for grading. Code should be handed in as separate py files with the specified names. All other files will be ignored.

1. For this class you will need various tools to complete your projects, including this one. Please setup the following on your devices and provide screenshots of each program running. This will need to be completed for the following parts (20 pts).

   (a) A Linux virtual machine. This can be done with software such as VirtualBox or VMWare Workstation. Note that the running version of Linux must have cpu architecture amd64/x86_64 (these are essentially the same thing). There are many articles online on how to do this, feel free to share ones you found helpful on Piazza for others (5 pts).

   (b) Python3 and the package *pwntools* on your virtual machine. This package allows your python script to interact with the files that we will be exploiting and has functionality for exploit creation & program analysis that we will be taking advatage of in this class. Installation instructions can be found here https://pypi.org/project/pwntools/ (5 pts).

   (c) GDB and the extension *pwndbg* on your virtual machine. This is an extension to GDB which gives cleaner syntax, helpful commands, and better introspection when debugging programs. You can find installation instructions at https://github.com/pwndbg/pwndbg (5 pts).

   (d) The reverse engineering software Ghidra (this can be installed on your host machine). You can find the program here at https://ghidra-sre.org/ (5 pts).

2. We will now learn the basics on how to use Ghidra. This is an open source tool made by the NSA and various other contractors and agencies, which will take a compiled program and try to give the best estimate of what the original source code was for the project. This process is called decompilation (And Ghidra is a decompiler, there are other industry standard compilers such as Binary Ninja and Ida/HexRays but we will be using Ghidra as it is free) (30 pts).

   Run Ghidra and make a new project. Then enter the code browser by clicking the dragon icon. In the new windows go to File → Import File, and select the executable given in the project files. Then hit "Ok", "Yes" to analyzing it now, and just do the default settings before hitting "Analyze". You will not need to change any settings.

Explore various windows in Ghidra by browsing the Window tab at the top. The main ones that we use in this class are the ”Functions” windows which just lists functions that Ghidra has identified in the code (this is primarily done by identifying stack based opcode patterns and where the call function is used), the ”Listing” window which shows the assembly of the file (this is useful if Ghidra fails to properly guess the original source code, as reading the assembly can often be more straightforward), and the ”Decompile” window which will show the source code of the function you currently have selected in the ”Listing” window.

3 Note that from the ”Decompile” window, if you click a piece of code it will take you to the corresponding assembly in the ”Listing” window, and if you double click a function in the decompilation, it will take you to the function. This makes traversing code execution very practical when reversing instead of looking through the ”Functions” window every time. This double click can also be done with global variables as they are stored in different parts of the file. You can test this in the part1 function.

Please provide screenshots/answers to the following:

(a) A screenshot of you having the Functions window, Listing window, and Decompile window open all at once with you having clicked on the ”main” function in the Functions window, so the ”main” functions assembly is in the Listing windows and its decompilation is in the Decompile window (7 pts).

(b) A screenshot of you opening the ”main” function, then opening the ”Function Graph” and ”Function Call Graph” windows. Explain in a quick sentence what the difference of these two windows are and their purpose (7 pts).

(c) A screenshot of function ”part1” after renaming the variable used in the scanf and strcmp function calls to what you think is appropriate. This can be done using the ’l’ key after clicking it in the Decompile window. Renaming variables is necessary when working with larger pieces of code to properly understand what you’re looking at. Note you can also rename function names using ’l’, and change data types of functions and variables with ’ctrl + l’ (8 pts)

(d) A brief description of what you would need to input to the program to get the ”main” function to print the congratulations text. (8 pts)

3. We will now learn how to use the Python library pwntools. This library is an exploit development library and has many useful functionalities to interact and exploit programs. The following code allows you to start a process and read/write data to it. We have used the command ’ls’ as an example.

```
from pwn import *

#Create the process, check the documentation for how to add command line
                                                arguments
p = process('/bin/ls')

#Read a line from the file, you can also use .read(), or .readuntil() with
                                                parameters to have more control on
                                                how much data you read
data = p.readline()
print(data)

#Likewise you can send data through stdin using p.{sendline/send}()
```

Please make a pwntools script that will start the challenge file, read and write data to it to solve part1 and part2, and print the congratulations text. For part2 you can just write a for loop and brute force the random number. Also note that pwntools needs to read and write all strings as bytes so if you want to send ”test”, you must do sendline(b”test”). Submit a screenshot of the code working alongside the source code in a file named LastName_FirstName_p0p3.py (30 pts).

4. We will now use GDB with the pwntools to dynamically solve this challenge without brute forcing the solution to part2. Load the program in gdb, and set a break point at part2. Next use the disassemble command to find the offset of the scanf function call and set a breakpoint there. Hit continue, enter the solution to part1, and note the pause at the scanf now.

   Submit a screenshot of the current state of GDB, noting the registers, current disassemble window, stack, and function backtrace. All of this information is very useful in live debugging, and pwndbg also has many other commands that we will take advantage of later in the class (6 pts).

   Now, using the disassemble command or ghidra to read the source code, see the offset (relative to the base pointer), where we compare the input and use the 'x $rbp-{offset}' command to see the value stored there. Note that you should get the same value that was listed on the stack by the PwnDbg output. Submit a screenshot of you getting this information from both sources and that they match (6 pts).

   You should now be able to just continue the program and submit the value from the stack to complete the program. Please submit a screenshot of you completing the challenge in GDB (8 pts).