CSE 5474 — Software Security Spring 2022

CTF Writeups

Week 3 babyrev CTF *Hezhuang Tian*

1 Overview

In CTF1, we have four warm up questions to practice. The difficulty level are increasing as the level up. Generally speaking, using ghidra to analyze and observe source codes of four binary files is the method of doing CTF1.

General Steps:

1. type **file <filename>** to check what the file type is(Although it is given by the CTF1 description, checking file type is a good starting point).
2. type **strings <filename>** to check if there's any obvious hint.
3. import binary file to ghidra and analyze its source code.
4. set a break point at main function to figure out what the general control flow
5. compare source code in ghidra and assembly code in gdb to find where the EXPECTED_RESULT get evaluated

2 Steps of how to Solve the Challenges

2.1 Level 1

In level1 babyrev, I used the **perl** command to input expected hex value which was given by the program itself.

2.2 Level 2

In level2 babyrev, its structure is very similar with level1 babyrev challenge. However, we need to fill first 20 bytes with junk characters, since the program will take the 21th as the start of input.

2.3 Level 3

In level3, I noticed every input input was implied XOR 0x12 by the program, so I wrote a python program to convert first 8 bytes back. The program asked us to input by environment variable. I found environment variable by using **ltrace** command. Then I typed in converted 8 bytes into environment variable. However, it's not the correct answer. To solve that, I made a break point at **memcpy** function and use x\8bx $rdi. Then I understood we need to skip three bytes of input.

2.4 Level 4

Babyrev challenge 4 is very similar with challenge 3. The only difference is we need to store our input in a file named by the program correctly. However, we don't have permission to create files under challenge directory. So I **cd** to the root directory, then use **ll** to check what folder enables us to create a input file. Then I repeated the steps in level3 to find the correct license.

2.5 Level 5

In challenge 5's source code, it swap value at position i with value at position 11-i. I wrote a python program to automatically do swapping. However, simply swapping the EXPECTED_RESULT is not enough. Because when I checked the $rdi register, which stored the input value and would be compared with EXPECTED_RESULT, the first several bytes are 0x00. So I guess it skipped some bytes at beginning or even read the file from tail to head. After several attempts on making small change on input file, I finally got the correct license.

2.6 Level 6

In challenge 6, we are given right to change 5 position of assembly code. We only need to change the evaluation condition of **cmp** function. I use **cmp** function name and test-jne structure to locate where the conditional statement we need to change is. Then we first type 0 to find where the base address is. Finally we change the value of jne to 74 which is opcode of je.

2.7 Level 7

Challenge 7 is very similar with challenge6 except for the information of base address. I use gdb to open the executable file. The gdb on my local machine automatically calculate the offset between the jne operation and base address. Finally I repeated the steps in level6 to get the correct license.