CSE 5474 — Software Security Spring 2022

CTF3 Writeups

Week 5 babyshell CTF *Hezhuang Tian*

1 Overview

In CTF3, we have 14 shellcode injection challenges to practice. Generally speaking, using ghidra and gdb to analyze and observe source codes, register values and control flow of 14 binary files is a good practice to solve challenges .

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 babyshell, I directly used asm = pwnlib.shellcraft.amd64.linux.execve('/bin/sh') to get control of the shell.

2.2 Level 2

Level1 shellcode works for level2 as well.

2.3 Level 3

In level3, the filter wouldn't allow any 0x48 byte get injected. First I generate shellcode of readfile("flag",1). Then I print its binary value and locate all the 0x48. After using gdb to figure out what those 0x48 instructions are doing, I replace by instructions with same purpose or just eliminate them.

2.4 Level 4

Level1 shellcode works for level4 as well.

2.5 Level 5

Using ghidra to find how many bytes we need to skip, then I solve the challenge by adding some nop bytes.

2.6 Level 6

Level6 is similar to level5. But the numbers of bytes we need to skip are different. We need to find it from ghidra. Then I solve it by simply changing number of nop bytes.

2.7 Level 7

In level7, we cannot inject 0x0f byte. The hex value of syscall is 0x0f 0x05. Then I used the example professor Lin showed us in class to dynamically change 0x05 to 0x0f which solves the challenge.

2.8 Level 8

In level8, the filter will mangle with our shellcode in unit of 8 bytes.We need to either inject two-stage shellcode to bypass the constraints or use nop padding to bypassing mangler. I injected two-stage code to solve the problem. In the first shellcode, I use
mov rax,0
mov rdi,0
lea rsi, [rip]
mov rdx, 1000
syscall to invoke sysread. Then I set second shellcode as readfile('flage',1). It needs to be noticed that we have to add some nop in front of second shellcode to skip some bytes right after first shellcode which avoids segmentation fault.

2.9 Level 9

Level9 program will closes stdin after injecting shellcode. I use chmod to solve it. First we create a symbolic link between "/flag" and "a", using "ln -s /flag a". After injecting follwoing shellcode:
push 0x61
push rsp
pop rdi
push 4
pop rsi
push 0x5a
pop rax
syscall , then I type "cat a" to get the flag.

2.10 Level 10

In level10, the binary byte must be at each location different from each other. I solved level14 before solving level10, and the shellcode of level14 works for level10. I will explain how level14 shellcode works with details in level14 section.

2.11 Level 11

First we create a symbolic link between "/flag" and "a", using "ln -s /flag a". Then I use shellcode of "chmod('a',4)" to solve the challenge.

2.12 Level 12

Level12 is similar to level11. First we create a symbolic link between "/flag" and "a", using "ln -s /flag a". Then I use shellcode of "chmod('a',4)" to solve the challenge.

2.13 Level 13

Level12 is similar to level11. First we create a symbolic link between "/flag" and "a", using "ln -s /flag a". Then I use shellcode of "chmod('a',4)" to solve the challenge.

2.14 Level 14

In level14 we need to inject two-stage shellcode. But we can only inject 8 bytes shellcode. To eliminate size of the first shellcode, we need to eliminate any unnecessary instruction, such as setting a register to zero which is already zero. Then we need to make register size smaller such as changing rax to eax. Finally we need to find the position of second shellcode then store it in rsi.