LAB: Shellcode Development (Ubuntu 16.04 and 20.04 VMs)
****

Linux system call table for x86 (32-bit):

https://chromium.googlesource.com/chromiumos/docs/+/master/constants/syscalls.md#x86-32_bit

Files provided for you: mysh.s, mysh2.s, myexit.s, shellcode1.c, shellcode2.c

Files with figures: mybash-fig.pdf, myls-fig.pdf, myenv-fig.pdf, myenv2-fig.pdf

Files you will be asked to create during lab tasks: mybash.s, myls.s, myenv.s, myenv2.s

****
* Task 1: Using the Stack Approach to generating shellcode *
****

****
* Task 1.a: The Entire Process *
****

mysh.s contains assembly code for shellcode that makes a system call to execve()
to run the command "/bin/sh"

Compiling assembly to object code
==========

Background Information on x86 (32-bit) Processor Registers:

    1) Four 32-bit general purpose registers

        EAX, EBX, ECX, EDX: 32-bits each
        AX, BX, CX, DX (refers to the rightmost 16-bits of EAX, BAX, ECX, EDX)
        AL, BL, CL, DL (refers to the rightmost 8-bits of AX, BX, CX, DX)
        AH, BH, CH, DH (refers to the leftmost 8-bits of AX, BX, CX, DX)

    2) EBP: 32-bit base pointer register
       ESP: 32-bit stack pointer register
       EIP: 32-bit instruction pointer register

    3) Two 32-bit index registers (often used as pointers): ESI, EDI

    4) Six 16-bit segment registers: CS, DS, SS, ES, FS, GS
       (used to denote what memory is used for different parts of a program)

Compile mysh.s to object code using the "Netwide Assembler (NASM)" assembler/disassembler
for Intel x86 and x64 architectures.

```
$ nasm -f elf32 mysh.s -o mysh.o
```

If you get an error, you may have to install nasm using the following command:

```
$ sudo apt install nasm
```

Linking object code to generate executable binary
==========

Link the object code in mysh.o to generate executable binary:

```
$ ld -m elf_i386 mysh.o -o mysh
```

Let's run the executable "mysh" to check if it launches a shell.

Use "echo $$" to verify that mysh launches a new shell.

```
$ echo $$
3480        <-- process ID of current shell (might be different for you)
```

```
$ ./mysh
$ echo $$
$ 5940      <-- process ID of the new shell launched by shellcode in mysh
                (might be different for you)

$ exit      <-- exit launched shell

$ echo $$   <-- back to parent shell
3480


Getting the machine code using "objdump" or "xxd" command
=========================================================

Let's inspect the machine code generated:

$ objdump -Mintel --disassemble mysh.o

mysh.o:     file format elf32-i386

Disassembly of section .text:

00000000 <_start>:
   0:   31 c0                   xor    eax,eax
   2:   50                      push   eax
   3:   68 2f 2f 73 68          push   0x68732f2f
   8:   68 2f 62 69 6e          push   0x6e69622f
   d:   89 e3                   mov    ebx,esp
   f:   50                      push   eax
  10:   53                      push   ebx
  11:   89 e1                   mov    ecx,esp
  13:   31 d2                   xor    edx,edx
  15:   31 c0                   xor    eax,eax
  17:   b0 0b                   mov    al,0xb
  19:   cd 80                   int    0x80


$ xxd -p -c 20 mysh.o

FYI: In the output produced, the text that start with "31c0" and ends with "cd80" is
the shellcode.

*********************************************
*  Task 1.b: Eliminating Zeros from the Code  *
*********************************************

For this task, create a new file named "mybash.s"

Copy and paste the following code into mybash.s file and save the file.

;
; CODE: BEGIN
;
; Makes syscall to execve() to execute the command: /bin/bash
; Compile with: nasm -f elf32 mybash.s -o mybash.o
; Link with: ld -m elf_i386 mybash.o -o mybash
; Run with: ./mybash
;
section .text
  global _start
    _start:
      ; store the argument string "/bin/bash" on stack
      mov  edx,"h***"
      shl  edx,24
      shr  edx,24
      push edx
      push "/bas"
      push "/bin"
      mov  ebx, esp           ; ebx = address of command "/bin/bash"
```

```
      ; setup argv[] array on stack
      xor  eax, eax
      push eax                ; argv[1] = 0
      push ebx                ; argv[0] points "/bin/bash"
      mov  ecx, esp           ; ecx = address of argv[]

      ; no environment variables to pass to execve()
      xor  edx, edx           ; edx = 0x00000000

      ; set eax register with system call number for execve()
      xor eax,eax             ; eax = 0x00000000
      mov al,0x0b             ; eax = 0x0000000b

      int 0x80                ; invoke execve() system call

;
; CODE: END
;
```

Compile mybash.s using the following command:

```
$ nasm -f elf32 mybash.s -o mybash.o
```

Generate executable binary using the following command:

```
$ ld -m elf_i386 mybash.o -o mybash
```

Use "echo $$" to verify that mysh launches a new shell.

```
$ echo $$
2471        <-- process ID of current shell (might be different for you)

$ ./mybash
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.
```

It will respond with a shell prompt.

```
$ echo $$
$ 2872      <-- process ID of the new shell launched by shellcode in mybash
                (might be different for you)

$ exit      <-- exit launched shell

$ echo $$   <-- back to parent shell
2471
```

If you are interesting in seeing the shellcode generated, use the command:

```
$ objdump -Mintel --disassemble mybash.o
```

```
************************************************
*  Task 1.c: Providing Arguments for System Calls  *
************************************************
```

For this task, create a new file named "myls.s"

Copy and paste the following code into myls.s file and save the file.

```
;
; CODE: BEGIN
;
; Makes syscall to execve() to execute the command: /bin/sh -c "ls -la"
; Compile with: nasm -f elf32 myls.s -o myls.o
; Link with: ld -m elf_i386 myls.o -o myls
; Run with: ./myls
;
```

```
section .text
  global _start
    _start:
      ; store the arguments on stack
      xor  eax, eax
      push eax                ; use 0 to terminate the string
      push "-la "
      push "ls  "
      push eax                ; use 0 to terminate the string
      push "-ccc"             ; "-ccc" option to /bin/sh is equivalent to "-c" option
      push eax                ; use 0 to terminate the string
      push "//sh"             ; multiple successive slashes are equivalent to one slash
      push "/bin"
      mov  ebx, esp           ; ebx = address of command "/bin//sh"

      ; setup argv[] array on stack
      push eax                ; argv[3] = 0
      mov eax,ebx             ; argv[2] = address of "ls -la"
      add eax,20
      push eax
      mov eax,ebx             ; argv[1] = address of "-ccc"
      add eax,12
      push eax
      push ebx                ; argv[0] = address of "/bin//sh"
      mov  ecx, esp           ; ecx = address of argv[]

      ; no environment variables to pass to execve()
      xor  edx, edx           ; edx = 0x00000000

      ; set eax register with system call number for execve()
      xor eax,eax             ; eax = 0x00000000
      mov al,0x0b             ; eax = 0x0000000b

      int 0x80                ; invoke execve() system call

;
; CODE: END
;
```

Compile myls.s using the following command:

```
$ nasm -f elf32 myls.s -o myls.o
```

Generate executable binary using the following command:

```
$ ld -m elf_i386 myls.o -o myls
```

Run the executable with the command below. This command will display the contents of the current directory in long format.

```
$ ./myls
```

(You should now see a listing of the directory in a long format)

If you are interesting in seeing the machine code generated, use the command:

```
$ objdump -Mintel --disassemble myls.o
```

****
* Task 1.d: Providing Environment Variables for execve() *
****

For this task, create a new file named "myenv.s"

Copy and paste the following code into myenv.s file and save the file.

```
;
; CODE: BEGIN
;
; Makes syscall to execve() to execute the command: /usr/bin/env
```

```
; Compile with: nasm -f elf32 myenv.s -o myenv.o
; Link with: ld -m elf_i386 myenv.o -o myenv
; Run with: ./myenv
;
section .text
  global _start
    _start:
      ; store three environment variables on stack: aaa=1234  bbb=5678  cccc=1234
      xor  eax,eax            ; clear eax
      push eax                ; use 0 to terminate the string
      mov  edx,"4***"
      shl  edx,24
      shr  edx,24
      push edx                ; place "4\0\0\0" on the stack
      push "=123"             ; place "=123" on the stack
      push "cccc"             ; place "cccc" on the stack
      push eax                ; place "\0\0\0\0" on the stack
      push "5678"             ; place "5678" on the stack
      push "bbb="             ; place "bbb=" on the stack
      push eax                ; place "\0\0\0\0" on the stack
      push "1234"             ; place "1234" on the stack
      push "abc="             ; place "abc=" on the stack
      mov  ebx,esp            ; save current value of stack pointer in ebx register

      ; setup envp[] array on stack
      push eax                ; envp[3] = 0
      mov  eax,ebx
      add  eax,24
      push eax                ; envp[2] = address of "cccc=1234"
      mov  eax,ebx
      add  eax,12
      push eax                ; envp[1] = address of "bbb=5678"
      push ebx                ; envp[0] = address of "abc=1234"
      mov  edx,esp            ; edx = envp[]

      xor  eax, eax           ; clear eax
      push eax                ; place "\0\0\0\0" on the stack
      push "/env"             ; place "/env" on the stack
      push "/bin"             ; place "/bin" on the stack
      push "/usr"             ; place "/usr" on the stack
      mov  ebx, esp           ; ebx = address of command "/usr/bin/env"

      ; setup argv[] array on stack
      push eax                ; argv[1] = 0
      push ebx                ; argv[0] = address of "/usr/bin/env"
      mov  ecx, esp           ; ecx = address of argv[]

      ; set eax register with system call number for execve()
      xor  eax, eax           ; eax = 0x00000000
      mov  al, 0x0b           ; eax = 0x0000000b

      int  0x80               ; invoke execve() system call

;
; CODE: END
;
```

Compile myenv.s using the following command:

```
$ nasm -f elf32 myenv.s -o myenv.o
```

Generate executable binary using the following command:

```
$ ld -m elf_i386 myenv.o -o myenv
```

Run the executable with the command (which displays three environment variables):

```
$ ./myenv
abc=1234
bbb=5678
```

```
cccc=1234
$

If you are interesting in seeing the machine code generated, use the command:

$ objdump -Mintel --disassemble myenv.o


****************************************************************
*  Task 2: Using Code Segment Approach to generating shellcode  *
****************************************************************

File to use: mysh2.s

Compile mysh2.s using the following command:

$ nasm -f elf32 mysh2.s -o mysh2.o

Generate executable binary using the following command:

$ ld -m elf_i386 mysh2.o -o mysh2

Run the executable with the command:

$ ./mysh2      // this should generate segmentation fault

Now, generate executable binary using the following command:

$ ld --omagic -m elf_i386 mysh2.o -o mysh2

Run the executable with the command:

$ ./mysh2
$
$ exit      <- to exit the launched shell

Task: write shellcode to execute /usr/bin/env with environment variables a=11 and b=22:

For this task, create a new file named "myenv2.s"

Copy and paste the following code into myenv2.s file and save the file.

;
; CODE: BEGIN
;
; Makes syscall to execve() to execute the command: /usr/bin/env
; Compile with: nasm -f elf32 myenv2.s -o myenv2.o
; Link with: ld --omagic -m elf_i386 myenv2.o -o myenv2
; Run with: ./myenv2
;
section .text
  global _start
    _start:
        BITS 32
        jmp short two

    one:
        pop ebx                     ; ebx register = address of beginning of db area

        xor eax, eax                ; eax = 0
        mov [ebx+12],al             ; replace the first * in db area with \0
                                    ; this will terminate the string "/usr/bin/env"

        mov [ebx+17],al             ; replace the second * in db area with \0
                                    ; this will terminate the string "a=11"

        mov [ebx+22],al             ; replace the third * in db area with \0
                                    ; this will terminate the string "b=22"

        ; setup argv[] array in db area
```

```
        mov [ebx+24],ebx            ; AAAA in db area is placeholder for argv[0]
                                    ; replace AAAA with address of "/usr/bin/env"

        mov [ebx+28],eax            ; BBBB in db area is placeholder for argv[1]
                                    ; replace BBBB with "\0\0\0\0" (NULLs)

        lea ecx,[ebx+24]            ; ecx register = address of argv[0] in db area

        ; setup envp[] array in db area
        lea edx,[ebx+13]
        mov [ebx+32],edx            ; CCCC is placeholder for env[0]
                                    ; set it to address of "a=11" in db area
        lea edx,[ebx+18]
        mov [ebx+36],edx            ; DDDD is placeholder for env[1]
                                    ; set it to address of "b=22" in db area

        mov [ebx+40],eax            ; EEEE is placeholder for env[2]; set to 0 (NULL)

        lea edx,[ebx+32]            ; edx register = address of env[0]

        ; set eax register with system call number for execve()
        xor eax,eax                 ; eax = 0x00000000
        mov al,0x0b                 ; eax = 0x0000000b

        int 0x80                    ; invoke execve() system call

    two:
        call one
        db '/usr/bin/env*a=11*b=22**AAAABBBBCCCCDDDDEEEE'

;
; CODE: END
;
```

Compile myenv2.s using the following command:

```
$ nasm -f elf32 myenv2.s -o myenv2.o
```

Generate executable binary using the following command:

```
$ ld -m elf_i386 myenv2.o -o myenv2
```

Run the executable with the command:

```
$ ./myenv2      // this should generate segmentation fault
```

Now, generate executable binary using the following command:

```
$ ld --omagic -m elf_i386 myenv2.o -o myenv2
```

Run the executable with the command:

```
$ ./myenv2
a=11
b=22
$
```

If you are interesting in seeing the machine code generated, use the command:

```
$ objdump -Mintel --disassemble myenv2.o
```

******** END OF THE SHELLCODE DEVELOPMENT LAB ***********