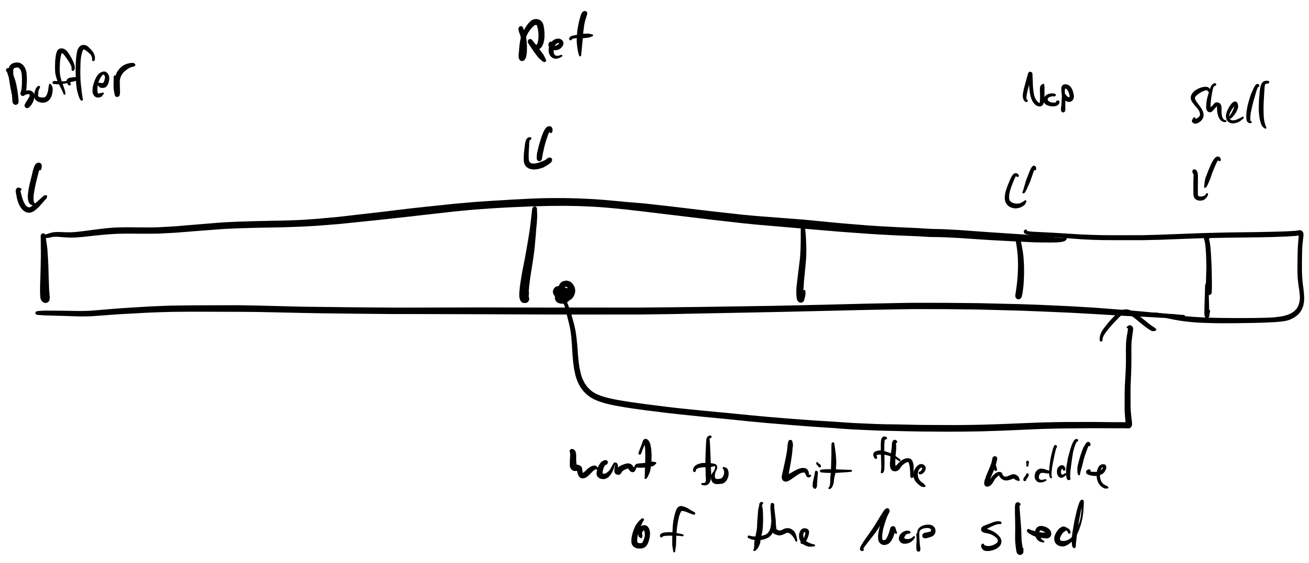
# ☆ Stack 4-64 Explainer ☆



☆ Always point to the

☆ Can put nop sled in an Envr Variable

## Stack 4-32 Notes

- Epiloge of Function call matters here
  - ↳ reset the registers
  - ↳ ex: stack pointer

ret → pops 4 bytes off stack

lea → esp , [ecx -0x4]

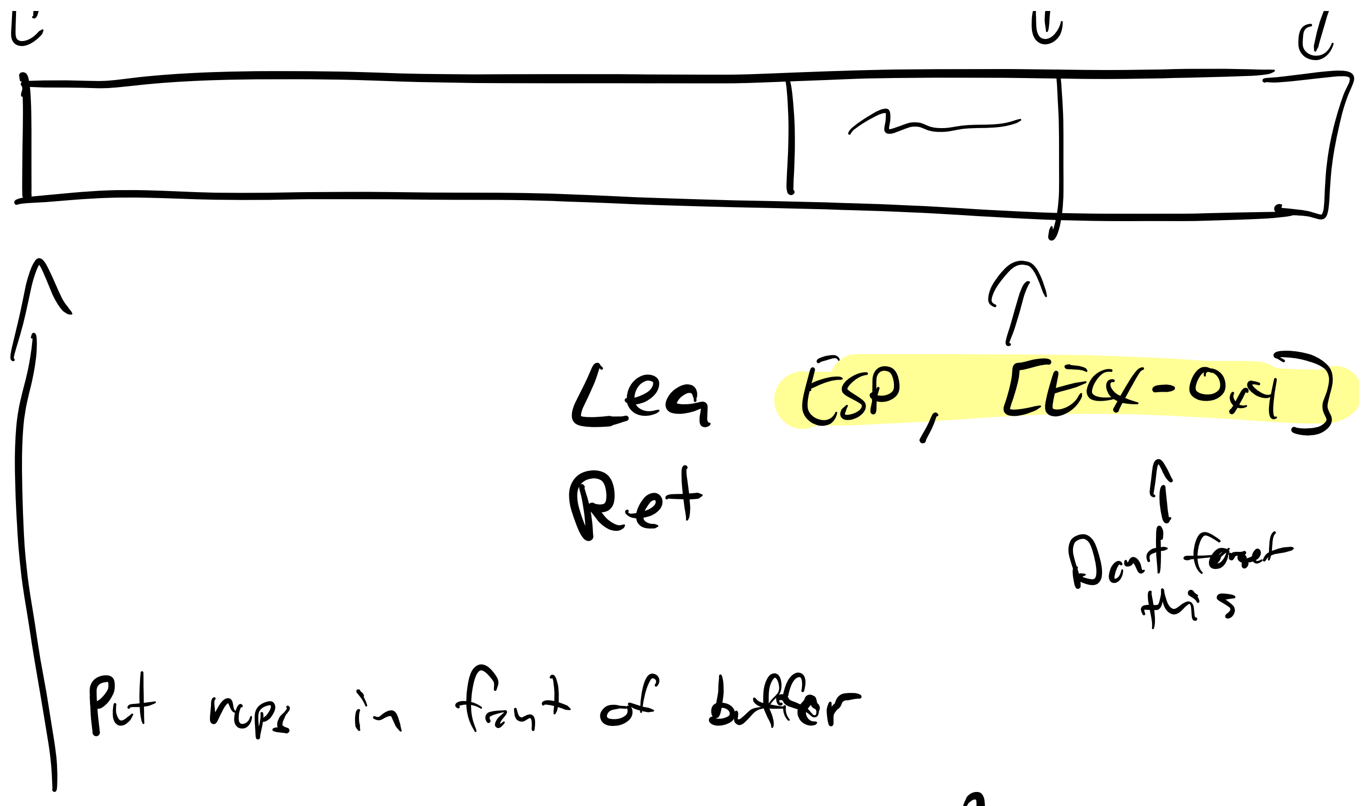
6d 61 61 6e is because the $esp is getting corrupted

**Stack Pivoting:** Controlling where the program thinks the stack is

We want ret in main to go to our nop sled.

We want our ESP to be that address

Buffer ECX Ret



Lea ESP, [ECX-0x4]

Ret

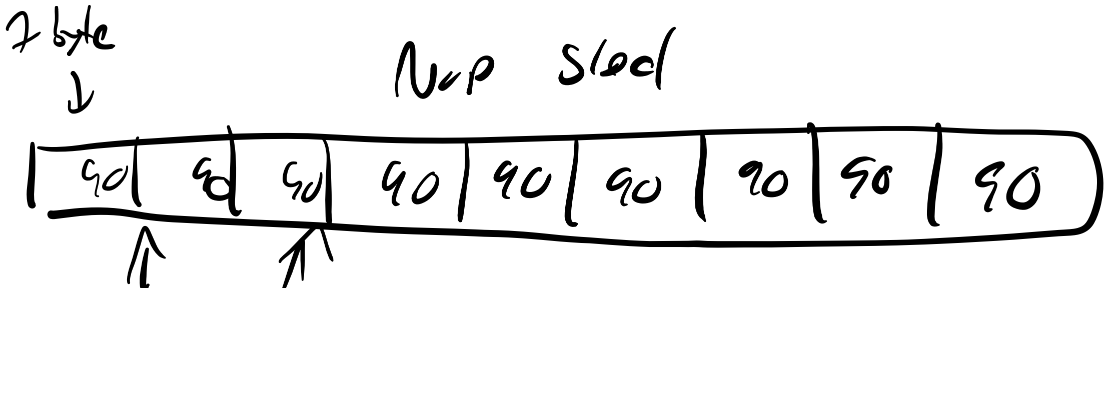
Don't forget this

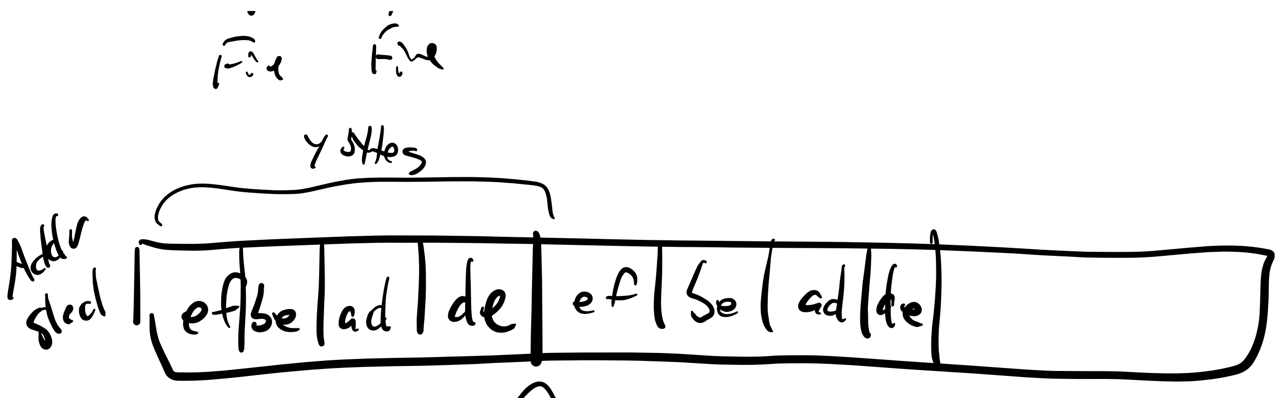
Put nops in front of buffer

Guess where start of Buffer is?

Can make an address sled instead of a nop sled to hit the address of of buffer

- the address itself must be aligned properly





0x deadbeef

Need to make sure to hit the start of the alignment in an addr sled

Guess Addr & 0x FFFFFFF0

will ensure it hits every 16 bytes

Putting Addr_sled in buffer ensures that it will start on a proper alignment

Ecx = our Buffer start address but it must be 4 higher to account for the lea -4 in the assembly.

## Defending Against Code Injection

Hardware added the **NX - Bit**

**NX-Bit:** Declaring if a page in memory can be executed or not

DEP + W ⊕ X are other names for this same technique.

This allows us to make the stack unexecutable.

## Ret 2 libc

↳ Use libc to call system
then call exec with bin//sh

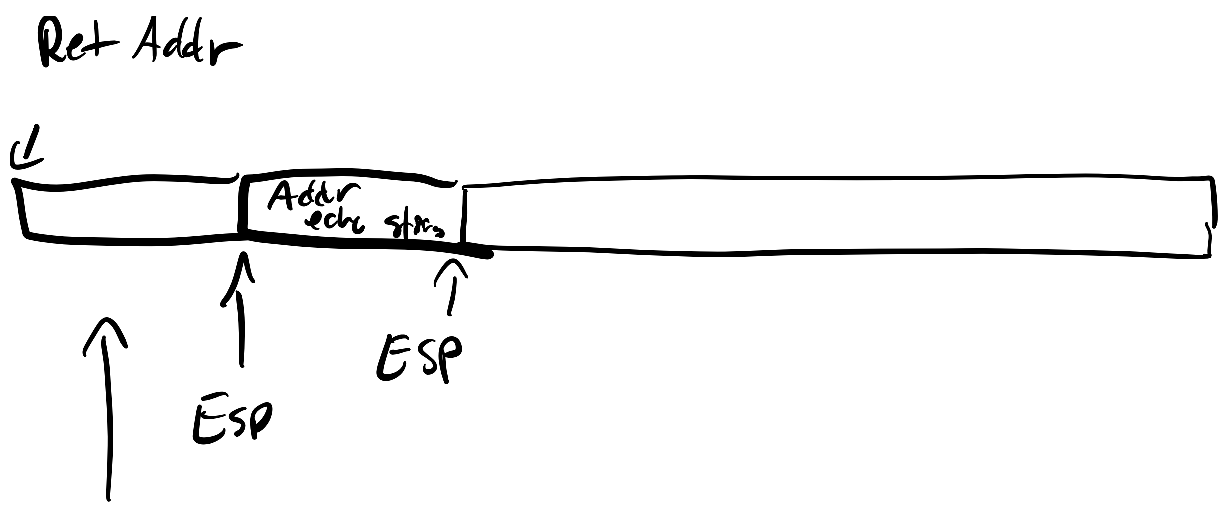
Create c program that uses the function call to examine how to set up the parameters

32 bit binaries pass arguments using the stack
64 bit binaries use the registers

```
gcc -m32 -o <name> c file -g
```

System creates a child process that it runs the command in

```
lea eax, [ebx-0x1ff8]
push eax
```
↝ puts address on the stack



call instruction of system will push another 4 bytes to stack

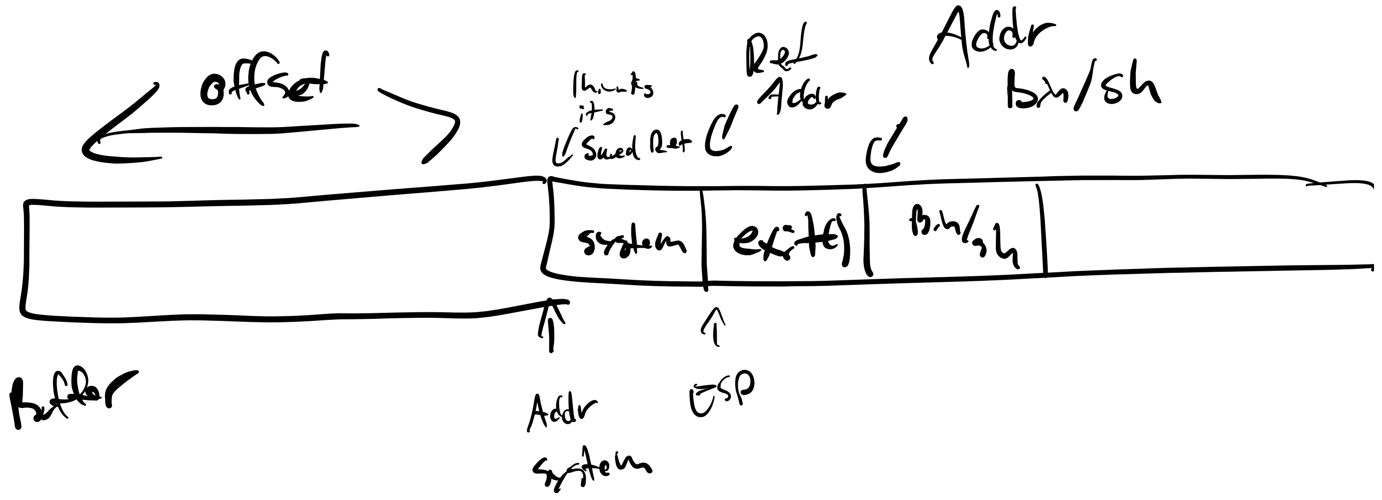
System expects a 4 byte return address followed by address of command

## Stack 5

Cant use Code Injection

Checks if return address is on

the stack



Nop sled with //////

bin/sh lives in libc