CSE 127 Discussion 1

Patrick Liu
Adapted from Ariana Mirian's slides

This discussion is being recorded

Goals of Discussion

- Assignment-oriented
  - Give you assignment-specific help
- Focus on information pertinent to lecture and assignments
- Will be run by different TAs throughout the quarter
- Ground Rules
  - Keep yourself muted
  - Ask questions in the chat
  - Be respectful and kind to each other

A Note on Remote Learning

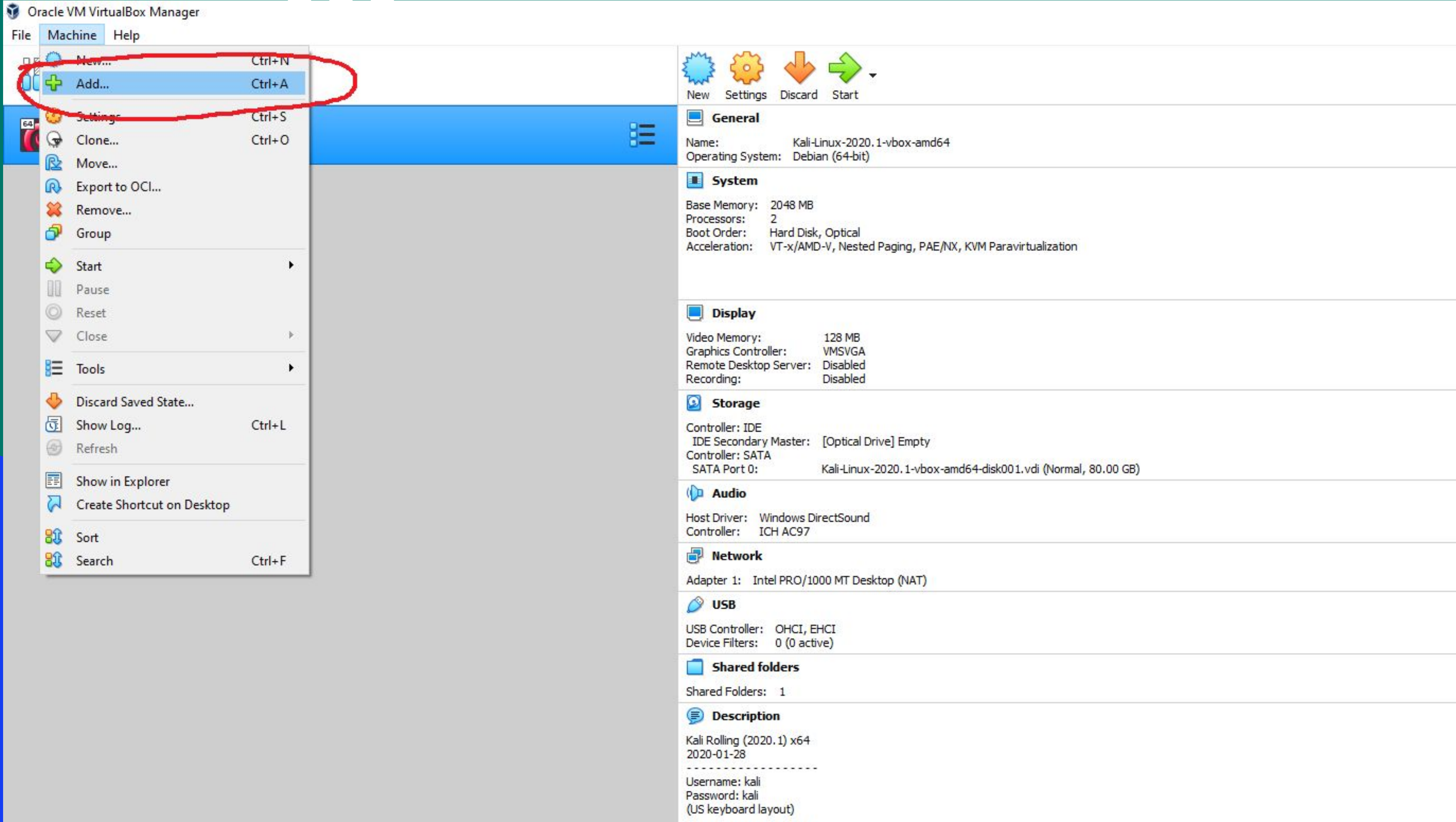
- Remote learning is hard on all of us
- If you have any issues or need help with anything, course related or otherwise, please feel free to reach out to any of the instructional staff
  - We're here to help!

PA1

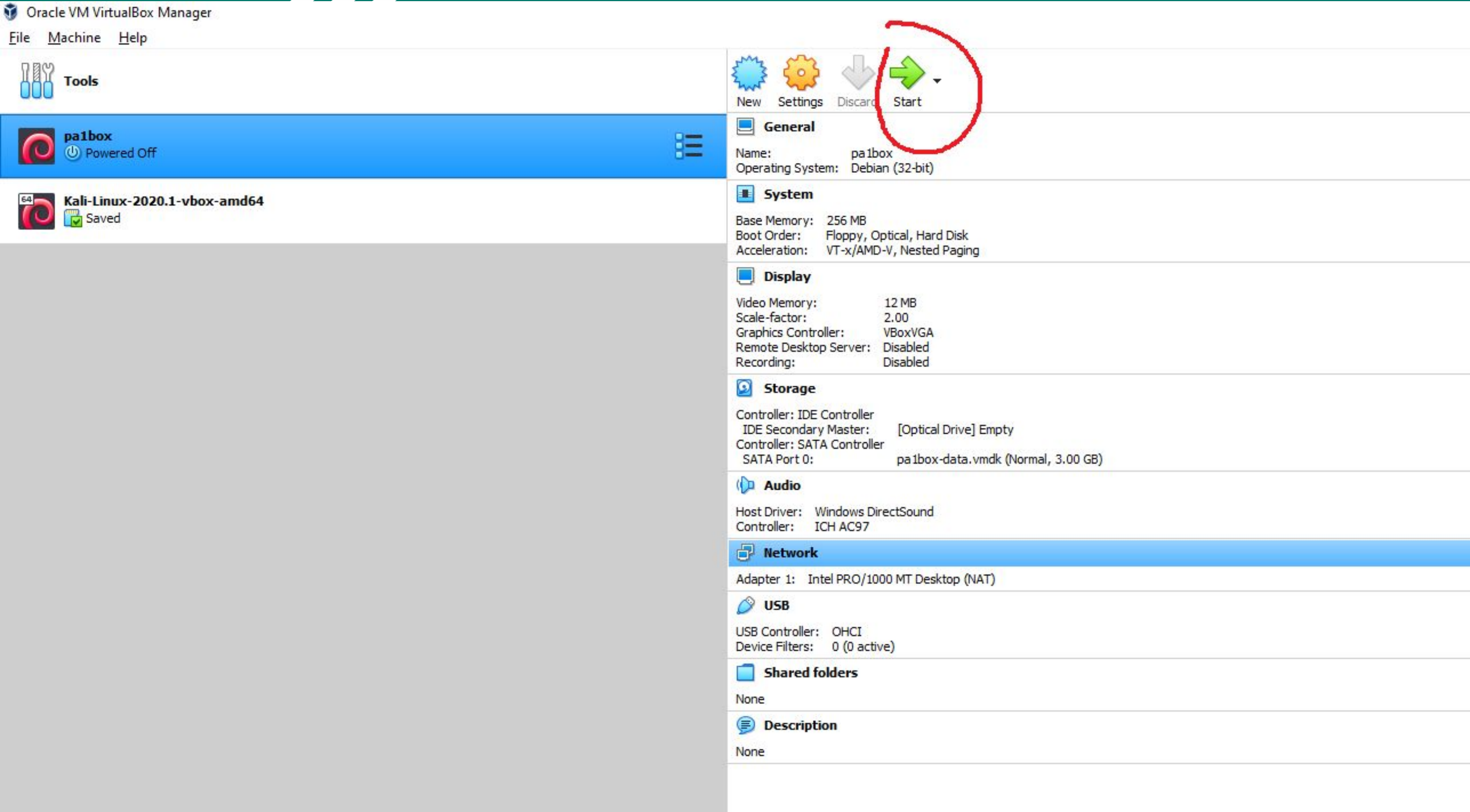
- Goals:
  - Get you comfortable with using local VM images and VirtualBox
  - (Re)introduce GDB and how to use it
  - Introduce some x86 assembly
- Due Tuesday, April 13

Getting Started

- Download VirtualBox from here(if you don't already have it)
- Install VirtualBox
- Download the `pa1box` image from the assignment website'
- Start the virtual machine
- SSH into the VM from an ssh client of some kind



Oracle VM VirtualBox Manager
File
Machine
Help
New...
Ctrl+N
Add...
Ctrl+A
Settings
Ctrl+S
Clone...
Ctrl+O
Move...
Export to OCI...
Remove...
Group
Start
Pause
Reset
Close
Tools
Discard Saved State...
Show Log...
Ctrl+L
Refresh
Show in Explorer
Create Shortcut on Desktop
Sort
Search
Ctrl+F
New
Settings
Discard
Start
General
Name: Kali-Linux-2020.1-vbox-amd64
Operating System: Debian (64-bit)
System
Base Memory: 2048 MB
Processors: 2
Boot Order: Hard Disk, Optical
Acceleration: VT-x/AMD-V, Nested Paging, PAE/NX, KVM Paravirtualization
Display
Video Memory: 128 MB
Graphics Controller: VMSVGA
Remote Desktop Server: Disabled
Recording: Disabled
Storage
Controller: IDE
IDE Secondary Master: [Optical Drive] Empty
Controller: SATA
SATA Port 0: Kali-Linux-2020.1-vbox-amd64-disk001.vdi (Normal, 80.00 GB)
Audio
Host Driver: Windows DirectSound
Controller: ICH AC97
Network
Adapter 1: Intel PRO/1000 MT Desktop (NAT)
USB
USB Controller: OHCI, EHCI
Device Filters: 0 (0 active)
Shared folders
Shared Folders: 1
Description
Kali Rolling (2020.1) x64
2020-01-28
Username: kali
Password: kali
(US keyboard layout)


Oracle VM VirtualBox Manager
File
Machine
Help
Tools
pa1box
Powered Off
Kali-Linux-2020.1-vbox-amd64
Saved
New
Settings
Discard
Start
General
Name: pa1box
Operating System: Debian (32-bit)
System
Base Memory: 256 MB
Boot Order: Floppy, Optical, Hard Disk
Acceleration: VT-x/AMD-V, Nested Paging
Display
Video Memory: 12 MB
Scale-factor: 2.00
Graphics Controller: VBoxVGA
Remote Desktop Server: Disabled
Recording: Disabled
Storage
Controller: IDE Controller
IDE Secondary Master: [Optical Drive] Empty
Controller: SATA Controller
SATA Port 0: pa1box-data.vmdk (Normal, 3.00 GB)
Audio
Host Driver: Windows DirectSound
Controller: ICH AC97
Network
Adapter 1: Intel PRO/1000 MT Desktop (NAT)
USB
USB Controller: OHCI
Device Filters: 0 (0 active)
Shared folders
None
Description
None

Common Issues

- Weird stack trace on startup and system doesn't start
  - In advanced boot options, try booting using sysvinit
- VirtualBox throws an error on startup
  - This varies, but on windows it is most likely because you haven't enabled Hyper-V, which there are resources to do here

The GNU Debugger

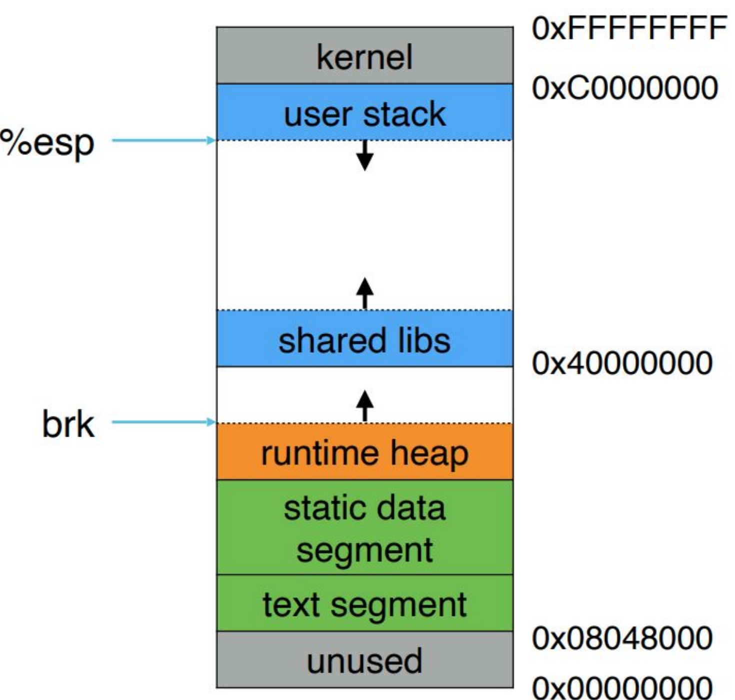
- https://www.gnu.org/software/gdb/
- Helpful docs here
- Allows you to "see" inside your program
    - See registers, memory access, instructions
    - Breakpoints allow you to pause execution at any point
- This will be your best friend for PA2

echo

- Write a simplified version of the `echo` utility using the example code provided
- Use only raw x86 assembly code
- Here is a list of syscalls that you may find useful

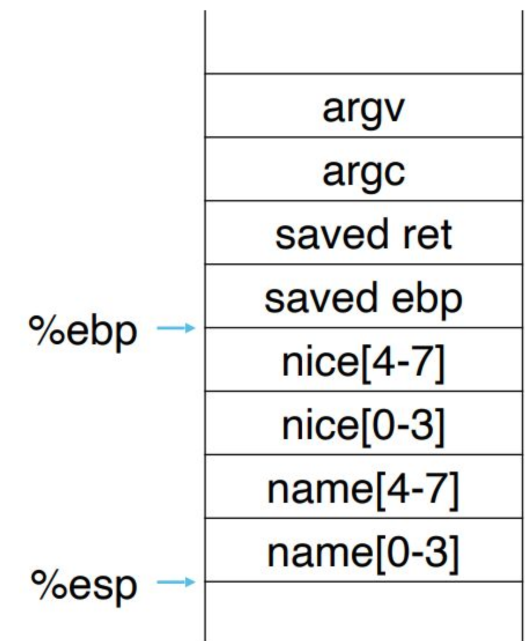
Process Memory Layout

- Text
  - Executable code
- Data
  - .data, .rodata, .bss
- Stack
- Heap



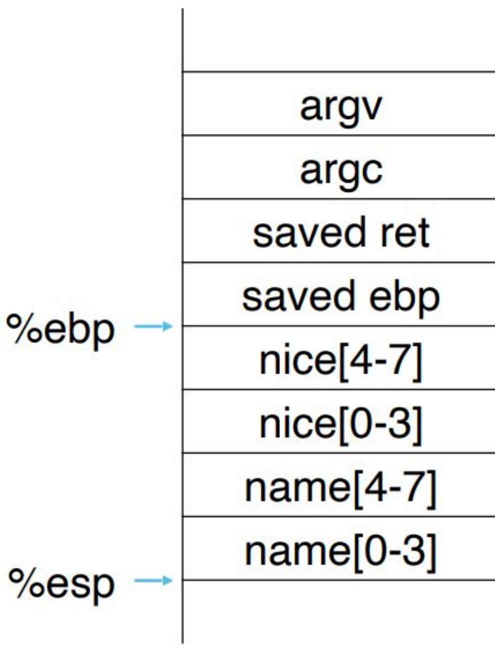
Registers to know

- %esp, or the Stack Pointer
  - Designates the top of the stack
  - Grows from high to low memory addresses
- %ebp, or the Frame Pointer/Base Pointer
  - Points to middle of stack frame(to the saved base pointer)
  - Doesn't move as function calls are made
  - Why does it exist?(why can't we just offset from stack pointer?)



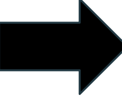
Registers to know

- %eip, or the Instruction Pointer
  - Holds the address of the next instruction to be executed



Function calls

```
1   int foobar(int a, int b)
2   {
3       int x = 1;
4       int buf[12];
5       buf[4] = 10;
6       return x;
7   }
8
9   int main()
10  {
11      return foobar(77, 88);
12  }
```



```
1   foobar(int, int):
2           pushl   %ebp
3           movl    %esp, %ebp
4           subl    $64, %esp
5           movl    $1, -4(%ebp)
6           movl    $10, -36(%ebp)
7           movl    -4(%ebp), %eax
8           leave
9           ret
10  main:
11          pushl   %ebp
12          movl    %esp, %ebp
13          pushl   $88
14          pushl   $77
15          call    foobar(int, int)
16          addl    $8, %esp
17          nop
18          leave
19          ret
```

https://godbolt.org/z/xMejhv

Basic Buffer Overflows

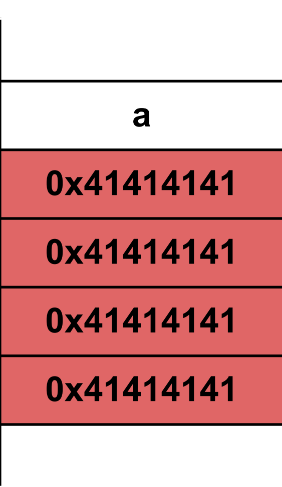
- Let's get hacking!
- What's the problem with this program?

```
int foo(int a){
    char buf[8];
    gets(buf);
    return 0
}
```

a
return address
saved ebp
buf[4-7]
buf[0-3]

Basic Buffer Overflows

- An input larger than 8 characters will begin to overwrite the stack!
- Suppose we simply put a bunch of As
  - What happens when the program returns?
  - Program will attempt to pop 0x41414141 into eip
    - Is the data at 0x41414141 a valid instruction?
  - Crash!



Basic Buffer Overflows

- What if we wanted to do something other than crash?
  - Can we make the program execute instructions of our choice?

a
return address
saved ebp
buf[4-7]
buf[0-3]

0xfffeabcd ⇨ buf[0-3]

Basic Buffer Overflows

- What if we wanted to do something other than crash?
  - Can we make the program execute instructions of our choice?
- Let's make some NOPs, and make our program jump to them
  - Now, our input is 12 bytes of 0x90, followed by 0xcdabfeff(why?)
  - What will happen when this program returns?

a
0xfffeabcd
0x90909090
0x90909090
0x90909090

0xfffeabcd ⇨ (points to the bottom 0x90909090 slot)

Basic Buffer Overflows

- What if we can't overwrite the instruction pointer?
  - What's the first thing we can overwrite?

a
return address
saved ebp
buf[4-7]
buf[0-3]

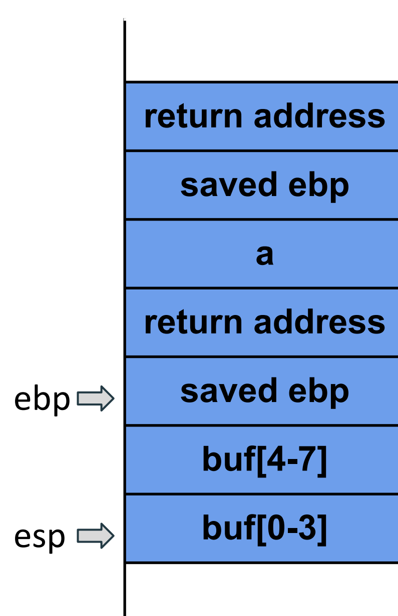
Basic Buffer Overflows

- What if we can't overwrite the instruction pointer?
  - What's the first thing we can overwrite?
  - Can you still control the instruction pointer via the saved base pointer?

a
return address
saved ebp
buf[4-7]
buf[0-3]

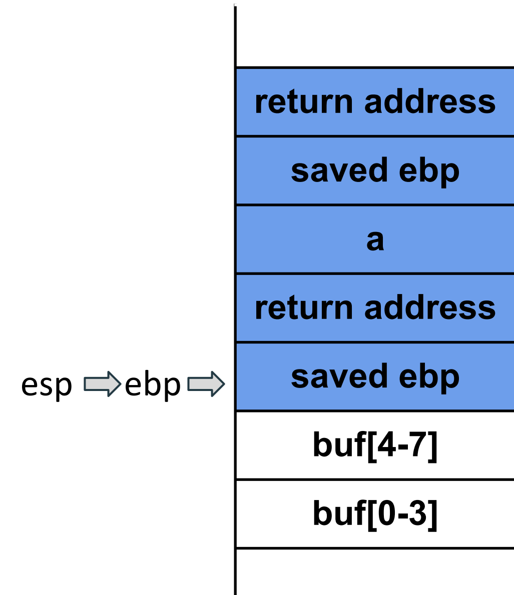
Basic Buffer Overflows

- What happens when a function returns?



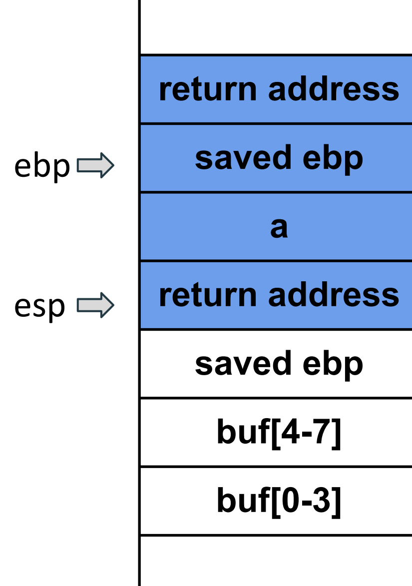
Basic Buffer Overflows

- What happens when a function returns?
  - Sets esp = ebp



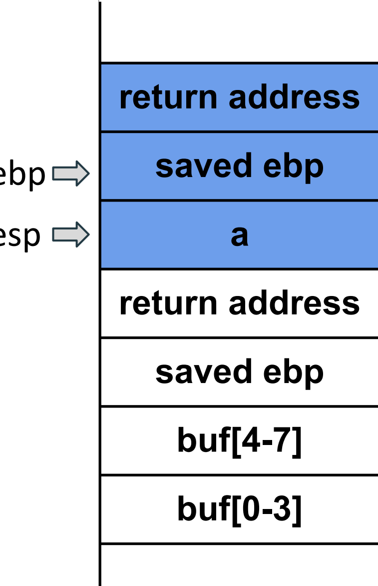
Basic Buffer Overflows

- What happens when a function returns?
  - Sets esp = ebp
  - Pop ebp



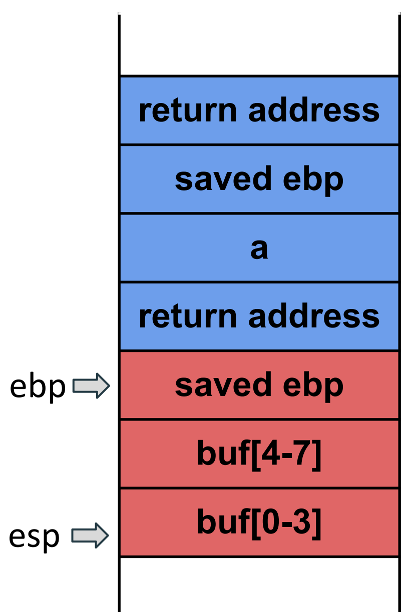
Basic Buffer Overflows

- What happens when a function returns?
  - Sets esp = ebp
  - Pop ebp
  - Pop eip



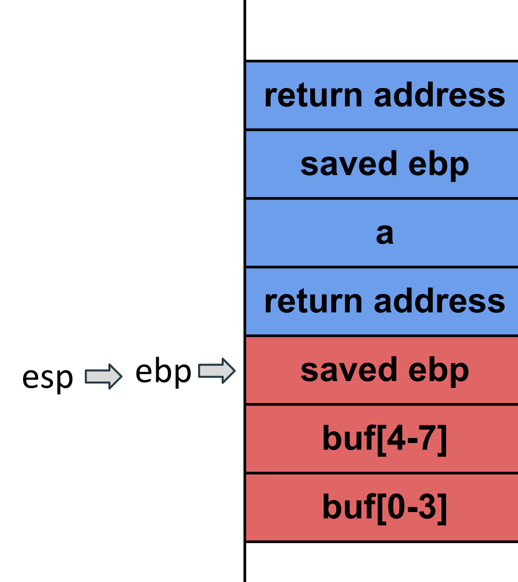
Basic Buffer Overflows

- What if we control ebp?
  - Let's try to return again



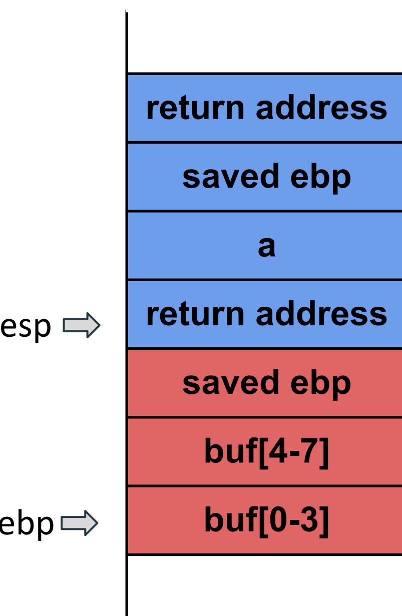
Basic Buffer Overflows

- What if we control ebp?
  - Let's try to return again
  - Setting esp = ebp functions normally...



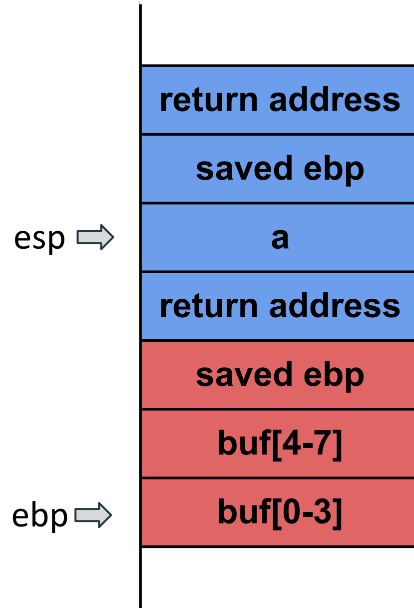
Basic Buffer Overflows

- What if we control ebp?
  - Let's try to return again
  - Setting esp = ebp functions normally…
  - What happens when we try to pop ebp?
    - We can make it point anywhere!



Basic Buffer Overflows

- What if we control ebp?
    - Let's try to return again
    - Setting esp = ebp functions normally…
    - What happens when we try to pop ebp?
        - We can make it point anywhere!
    - Popping eip also functions normally…



Basic Buffer Overflows

- What happens when the next function tries to return?
  - Set esp = ebp...

return address
saved ebp
a
return address
saved ebp
buf[4-7]
esp ⇨ ebp ⇨ buf[0-3]

Basic Buffer Overflows

- What happens when the next function tries to return?
  - Set esp = ebp…
  - Pop ebp...

return address
saved ebp
a
return address
saved ebp
buf[4-7]
buf[0-3]

esp ⇨ ebp ⇨

Basic Buffer Overflows

- What happens when the next function tries to return?
  - Set esp = ebp…
  - Pop ebp…
  - And finally pop eip…
- buf[0-3] became ebp, and buf[4-7] went into eip!