Cyber Attacks & Defense

Preliminary Rev. Engineering #1

Dr. Yeongjin Jang

Objective for Week 1

- Learn how to read Intel (x86) assembly
- Understand how CPU uses memory
  - Local variables and function arguments in stack
  - Array access
- Learn how to use GDB to see program's state at runtime
- Understand the program's behavior!
  - **Reverse engineering**!

Registration

- Did you all finish these three registrations?
  - Web page: https://ctf.unexploitable.systems/
  - SSH: ssh [your_id]@vm-ctf1.eecs.oregonstate.edu (on flip)
  - Piazza: https://piazza.com/class/kmwe0l3rmbg3vf
  - Discord: https://discord.gg/bMk3z767FX

Challenges for Week 1

- Visit challenge director (or fetch)

```
blue9057@blue9057-vm-ctf1 ~ ‹ruby-head›
$ cd /home/labs/week1
blue9057@blue9057-vm-ctf1 /home/labs/week1 ‹ruby-head›
$ ls
example  level0  level1  level2  level3  level4  level5  level6  level7  level8  level9
```

- We have 10 challenges!

What Do We Need to Do?

- You need to read the ‘flag’

```
blue9057@blue9057-vm-ctf1 /home/labs/week1/level0 <ruby-head>
$ ls -als
total 24
4 drwxr-xr-x  2 week1-level0-solved week1-level0-solved 4096 Sep 26  2019 .
4 drwxr-xr-x 13 root                root                4096 Oct  1  2019 ..
4 -r--r-----  1 week1-level0-solved week1-level0-solved   30 Sep 26  2019 flag
8 -rwxr-sr-x  1 week1-level0-solved week1-level0-solved 7632 Sep 26  2019 level0
4 -rw-r--r--  1 week1-level0-solved week1-level0-solved  189 Sep 26  2019 README
```

- But, cat fails…

```
blue9057@blue9057-vm-ctf1 /home/labs/week1/level0 <ruby-head>
$ cat flag
cat: flag: Permission denied
```

WHY?

Oregon State University

Filesystem Permissions in UNIX

- You do not have privilege to read the `flag` file

```
4 -r--r-----  1 week1-level0-solved week1-level0-solved   30 Sep 26  2019 flag
```

- -r--r----- <- what does this mean?
  - - **r-- r-- ---**
    - **r-- : Owner of the file can read this (week1-level0-solved)**
    - **r--: User who belongs to the group of the file can read this (week1-level0-solved)**
    - **---: Any other user can't do anything**
- **Quiz**
  - **rw-rw-r-- :** Owner can read/write, group members can read/write, others can only read
  - **rwx------ :** Owner can read/write/execute while others can't do anything

Runtime Privilege in UNIX

- Privilege level of flag

```
4 -r--r-----  1 week1-level0-solved week1-level0-solved   30 Sep 26  2019 flag
```

  - Only the user or group member of week1-level0-solved can read the flag

- Take a look at the challenge binary program

```
8 -rwxr-sr-x  1 week1-level0-solved week1-level0-solved 7632 Sep 26  2019 level0
```

- rwx: Onwer can read/write/execute
- **r-s**: Group member can read/execute, and during running, it gets the group member privilege!
- r-x: any other user can read/execute the program

How To Read FLAG?

- Flag is available only to week1-level0-solved…

```
4 -r--r-----  1 week1-level0-solved week1-level0-solved   30 Sep 26  2019 flag
```

- Your group privilege will be week1-level0-solved during running level0

```
8 -rwxr-sr-x  1 week1-level0-solved week1-level0-solved 7632 Sep 26  2019 level0
```

- Find a way to run `cat flag` while you are having the privilege!

```
blue9057@blue9057-vm-ctf1 ~/week1/level0 <ruby-head>
$ ./level0
What's the password?
PaSSw0Rd
Correct!
Spawning a privileged shell
[blue9057@blue9057-vm-ctf1 ~/week1/level0$] id
uid=1001(blue9057) gid=10000(week1-level0-solved) groups=10000(week1-level0-solv
ed),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),113(lpadmin),128(sambashare),9
99(docker),1001(blue9057)
[blue9057@blue9057-vm-ctf1 ~/week1/level0$]
```

Level0

The C code looks like this

`get_a_shell()` inherits the group privilege and execute `/bin/bash`!

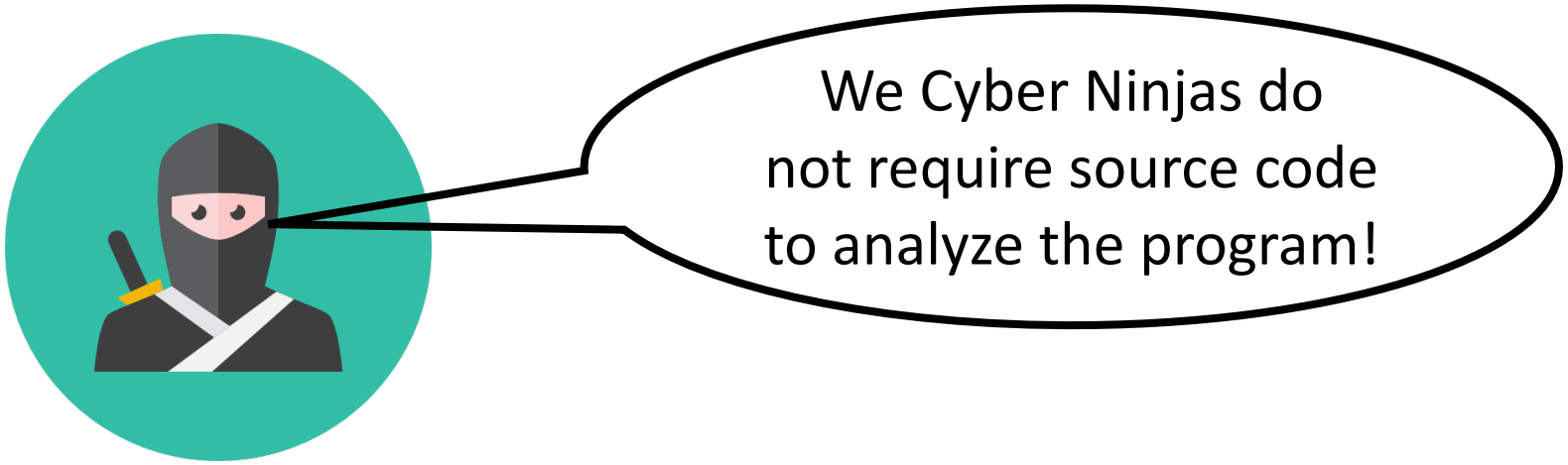
With a correct password supplied, ”PaSSw0Rd”, The program runs `get_a_shell()`

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#include <unistd.h>

void get_a_shell() {
    printf("Spawning a privileged shell\n");
    setregid(getegid(), getegid());
    execl("/bin/bash", "bash", NULL);
}


int main() {
    char buf[512];
    printf("What's the password?\n");
    scanf("%s", buf);
    if(strcmp(buf, "PaSSw0Rd") == 0) {
        printf("Correct!\n");
        get_a_shell();
    }
    else {
        printf("Wrong password!!\n");
    }
}
```

Level0

- But you don’t have the C code

```
blue9057@blue9057-vm-ctf1 /home/labs/week1/level0 ‹ruby-head›
$ ls -als
total 24
4 drwxr-xr-x  2 week1-level0-solved week1-level0-solved 4096 Sep 26  2019 .
4 drwxr-xr-x 13 root                root                4096 Oct  1  2019 ..
4 -r--r-----  1 week1-level0-solved week1-level0-solved   30 Sep 26  2019 flag
8 -rwxr-sr-x  1 week1-level0-solved week1-level0-solved 7632 Sep 26  2019 level0
4 -rw-r--r--  1 week1-level0-solved week1-level0-solved  189 Sep 26  2019 README
```



REVERSE ENGIEERING!

Oregon State University

How To Get Disasser

```
[blue9057@blue9057-vm-ctf1 ~/week1/level0$] gdb level0
pwndbg: loaded 181 commands. Type pwndbg [filter] for a l
pwndbg: created $rebase, $ida gdb functions (can be used
Reading symbols from level0...(no debugging symbols found
```

Open gdb

diasss main

```
pwndbg> disass main
Dump of assembler code for function main:
   0x080485a0 <+0>:     push   %ebp
   0x080485a1 <+1>:     mov    %esp,%ebp
   0x080485a3 <+3>:     sub    $0x228,%esp
   0x080485a9 <+9>:     lea    0x80486e7,%eax
   0x080485af <+15>:    movl   $0x0,-0x4(%ebp)
   0x080485b6 <+22>:    mov    %eax,(%esp)
   0x080485b9 <+25>:    call   0x80483c0 <printf@plt>
   0x080485be <+30>:    lea    0x80486fd,%ecx
   0x080485c4 <+36>:    lea    -0x204(%ebp),%edx
   0x080485ca <+42>:    mov    %ecx,(%esp)
   0x080485cd <+45>:    mov    %edx,0x4(%esp)
   0x080485d1 <+49>:    mov    %eax,-0x208(%ebp)
   0x080485d7 <+55>:    call   0x8048410 <__isoc99_scanf@plt>
   0x080485dc <+60>:    lea    -0x204(%ebp),%ecx
   0x080485e2 <+66>:    mov    %esp,%edx
   0x080485e4 <+68>:    mov    %ecx,(%edx)
   0x080485e6 <+70>:    movl   $0x8048700,0x4(%edx)
   0x080485ed <+77>:    mov    %eax,-0x20c(%ebp)
   0x080485f3 <+83>:    call   0x80483b0 <strcmp@plt>
   0x080485f8 <+88>:    cmp    $0x0,%eax
   0x080485fb <+91>:    jne    0x804861f <main+127>
   0x08048601 <+97>:    lea    0x8048709,%eax
   0x08048607 <+103>:   mov    %eax,(%esp)
   0x0804860a <+106>:   call   0x80483c0 <printf@plt>
   0x0804860f <+111>:   mov    %eax,-0x210(%ebp)
   0x08048615 <+117>:   call   0x8048530 <get_a_shell>
   0x0804861a <+122>:   jmp    0x8048633 <main+147>
   0x0804861f <+127>:   lea    0x8048713,%eax
   0x08048625 <+133>:   mov    %eax,(%esp)
   0x08048628 <+136>:   call   0x80483c0 <printf@plt>
   0x0804862d <+141>:   mov    %eax,-0x214(%ebp)
   0x08048633 <+147>:   mov    -0x4(%ebp),%eax
   0x08048636 <+150>:   add    $0x228,%esp
   0x0804863c <+156>:   pop    %ebp
   0x0804863d <+157>:   ret
End of assembler dump.
```

NOT HUMAN FRIENDLY

Level0

EASY TO UNDERSTAND

```c
int main() {
    char buf[512];
    printf("What's the password?\n");
    scanf("%s", buf);
    if(strcmp(buf, "PaSSw0Rd") == 0) {
        printf("Correct!\n");
        get_a_shell();
    }
    else {
        printf("Wrong password!!\n");
    }
}
```

```
pwndbg> disass main
Dump of assembler code for function main:
   0x080485a0 <+0>:     push   %ebp
   0x080485a1 <+1>:     mov    %esp,%ebp
   0x080485a3 <+3>:     sub    $0x228,%esp
   0x080485a9 <+9>:     lea    0x80486e7,%eax
   0x080485af <+15>:    movl   $0x0,-0x4(%ebp)
   0x080485b6 <+22>:    mov    %eax,(%esp)
   0x080485b9 <+25>:    call   0x80483c0 <printf@plt>
   0x080485be <+30>:    lea    0x80486fd,%ecx
   0x080485c4 <+36>:    lea    -0x204(%ebp),%edx
   0x080485ca <+42>:    mov    %ecx,(%esp)
   0x080485cd <+45>:    mov    %edx,0x4(%esp)
   0x080485d1 <+49>:    mov    %eax,-0x208(%ebp)
   0x080485d7 <+55>:    call   0x8048410 <__isoc99_scanf@plt>
   0x080485dc <+60>:    lea    -0x204(%ebp),%ecx
   0x080485e2 <+66>:    mov    %esp,%edx
   0x080485e4 <+68>:    mov    %ecx,(%edx)
   0x080485e6 <+70>:    movl   $0x8048700,0x4(%edx)
   0x080485ed <+77>:    mov    %eax,-0x20c(%ebp)
   0x080485f3 <+83>:    call   0x80483b0 <strcmp@plt>
   0x080485f8 <+88>:    cmp    $0x0,%eax
   0x080485fb <+91>:    jne    0x804861f <main+127>
   0x08048601 <+97>:    lea    0x8048709,%eax
   0x08048607 <+103>:   mov    %eax,(%esp)
   0x0804860a <+106>:   call   0x80483c0 <printf@plt>
   0x0804860f <+111>:   mov    %eax,-0x210(%ebp)
   0x08048615 <+117>:   call   0x8048530 <get_a_shell>
   0x0804861a <+122>:   jmp    0x8048633 <main+147>
   0x0804861f <+127>:   lea    0x8048713,%eax
   0x08048625 <+133>:   mov    %eax,(%esp)
   0x08048628 <+136>:   call   0x80483c0 <printf@plt>
   0x0804862d <+141>:   mov    %eax,-0x214(%ebp)
   0x08048633 <+147>:   mov    -0x4(%ebp),%eax
   0x08048636 <+150>:   add    $0x228,%esp
   0x0804863c <+156>:   pop    %ebp
   0x0804863d <+157>:   ret
End of assembler dump.
```

Let’s learn x86 to C pseudocode translation!

Intel x86 Assembly Syntax

- AT&T/GNU Syntax
  - [instruction] [source] [destination]
  - mov %esp, %ebp `mov %esp,%ebp`
    - Assign the value of esp to ebp, i.e., `ebp = esp;` in C
  - mov $0x0, %eax `movl $0x0,-0x4(%ebp)`
    - Assign value 0 to eax, i.e., `eax = 0;` in C
    - Assign value 0 to the address pointed by ebp-4, i.e., `ebp[-1] = 0;` (ebp as int *)
  - [instruction] [destination]
  - call 0x80483c0 <printf@plt> `call 0x80483c0 <printf@plt>`
    - Call `printf()`
  - jne 0x804861f <main+127> `jne 0x804861f <main+127>`
    - Jump to 0x804861f if the comparison result was Not Equal (NE)

Glossaries: Instructions

- mov: move value from src to dst
  - mov %esp, %ebp (`ebp = esp;`)
  - movl: move 4 bytes (move long integer)
  - movw: move 2 bytes (move word integer)
  - movb: move 1 byte (move byte integer)
- call: call the destination function
  - call 0x80483c0 <printf@plt>
  - call 0x80483b0 <strcmp@plt>
- ret: return from a function call
  - ret `0x0804863d <+157>: ret`

Glossaries: Instructions

- Arithmetic Operations
  - add %eax, %ebx (`ebx = eax + ebx;`)
  - add $1, %eax (`eax = eax + 1;`)
  - sub $0x228, %esp (`esp = esp - 0x228;`)
  - xor %eax, %eax (`eax = eax ^ eax;`)

Glossaries: Instructions

- Compare
  - cmp $0x0,%eax
    ```
    cmp    $0x0,%eax
    ```
  - Compare the value 0x0 to the value of %eax
  - Comparison result will be stored in the EFLAGS register
    - Will be used in the jump instruction
- Jump
  - je: jump if equal
    - `If (eax == 0)`
  - jne: jump if not equal
    - `if (eax != 0)`

    ```
    jne    0x804861f <main+127>
    ```
  - jg: jump if greater
    - `If (eax > 0)`
  - jge: jump if greater and equal
    - `If (eax >= 0)`
  - Jmp: jump anyway!

    ```
    jmp    0x8048633 <main+147>
    ```

There Are Many Jump Instructions

- http://www.unixwiz.net/techtips/x86-jumps.html

JO	Jump if overflow		OF = 1	70	0F 80
JNO	Jump if not overflow		OF = 0	71	0F 81
JS	Jump if sign		SF = 1	78	0F 88
JNS	Jump if not sign		SF = 0	79	0F 89
JE **JZ**	Jump if equal Jump if zero		ZF = 1	74	0F 84
JNE **JNZ**	Jump if not equal Jump if not zero		ZF = 0	75	0F 85
JB **JNAE** **JC**	Jump if below Jump if not above or equal Jump if carry	unsigned	CF = 1	72	0F 82
JNB **JAE** **JNC**	Jump if not below Jump if above or equal Jump if not carry	unsigned	CF = 0	73	0F 83
JBE **JNA**	Jump if below or equal Jump if not above	unsigned	CF = 1 or ZF = 1	76	0F 86
JA **JNBE**	Jump if above Jump if not below or equal	unsigned	CF = 0 and ZF = 0	77	0F 87
JL **JNGE**	Jump if less Jump if not greater or equal	signed	SF <> OF	7C	0F 8C
JGE **JNL**	Jump if greater or equal Jump if not less	signed	SF = OF	7D	0F 8D
JLE **JNG**	Jump if less or equal Jump if not greater	signed	ZF = 1 or SF <> OF	7E	0F 8E
JG **JNLE**	Jump if greater Jump if not less or equal	signed	ZF = 0 and SF = OF	7F	0F 8F
JP **JPE**	Jump if parity Jump if parity even		PF = 1	7A	0F 8A
JNP **JPO**	Jump if not parity Jump if parity odd		PF = 0	7B	0F 8B
JCXZ **JECXZ**	Jump if %CX register is 0 Jump if %ECX register is 0		%CX = 0 %ECX = 0	E3	

Oregon State University

Glossaries: 8 Registers of x86

- %eax: a 4-byte register that stores return value
- %ebx
- %ecx
- %edx
- %esi: a 4-byte register that indicates a source data address
- %edi: a 4-byte register that indicates a destination data address
- %ebp: a 4-byte register that stores stack frame base (base pointer)
- %esp: a 4-byte register that stores stack frame end (stack pointer)
- %eip: an instruction pointer (hidden)

Glossaries: Immediate Value vs. Address

- $0x8048700
  - A number with '$' prefix is regarded as an immediate value

- movl $0x8048700, %edx
  - `edx = 0x8048700;`

Glossaries: Immediate Value vs. Address

- 0x80486e7
  - A number w/o '$' prefix is regarded as an address
  - In x86 assembly, it refers to the value in that address
- mov 0x80486e7, %eax
  - Move the value stored in the address 0x80486fd to %eax
  - If 0x80486fd stores 32, then `%eax = 32;`
- lea 0x80486e7, %eax

```
lea     0x80486e7,%eax
```

  - LEA: Load Effective Address, load the address of src to dst
  - 0x80486e7 refers to the value stored in the address
  - The address of that value is 0x80486e7
  - Thereby, `%eax = 0x80486e7;`

Dereferencing in x86 Assembly

- Parenthesis means accessing the register as an address
- mov %eax, (%esp)

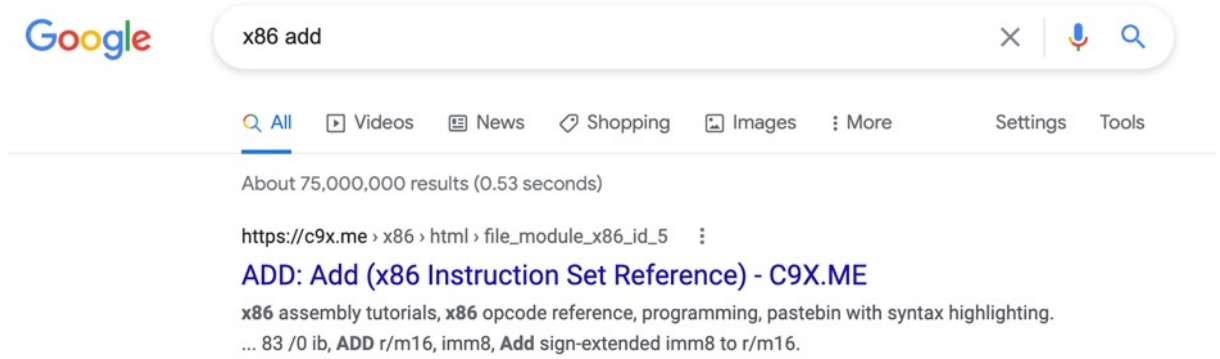
```
mov    %eax,(%esp)
```

  - Regard %esp as int *esp; in C
  - `*esp = eax;` (move the value of eax to the address pointed by %esp)
  - `esp[0] = eax;`
- movl $0x0, -0x4 (%ebp)

```
movl   $0x0,-0x4(%ebp)
```

  - Move 0 to (%ebp-4)
  - Regard %ebp as int *ebp; in C, -1 to int* is indeed -4 in address
  - `*(ebp-1) = 0;`
  - `ebp[-1] = 0;`

Instructions you don't know?



x86 Instruction Set Reference

ADD

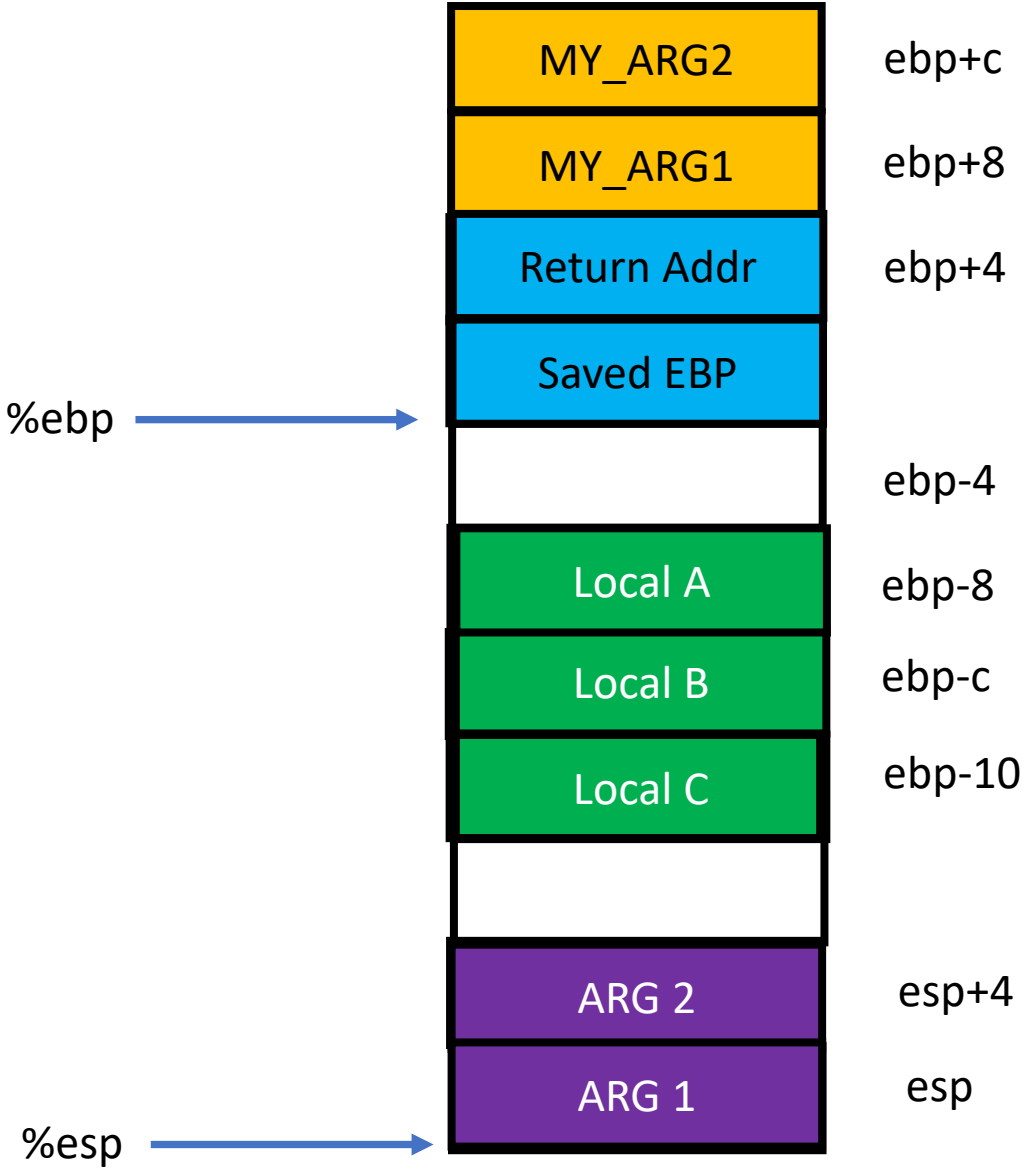
Add

Opcode	Mnemonic	Description
04 ib	ADD AL, imm8	Add imm8 to AL
05 iw	ADD AX, imm16	Add imm16 to AX
05 id	ADD EAX, imm32	Add imm32 to EAX
80 /0 ib	ADD r/m8, imm8	Add imm8 to r/m8
81 /0 iw	ADD r/m16, imm16	Add imm16 to r/m16
81 /0 id	ADD r/m32, imm32	Add imm32 to r/m32
83 /0 ib	ADD r/m16, imm8	Add sign-extended imm8 to r/m16
83 /0 ib	ADD r/m32, imm8	Add sign-extended imm8 to r/m32
00 /r	ADD r/m8, r8	Add r8 to r/m8
01 /r	ADD r/m16, r16	Add r16 to r/m16
01 /r	ADD r/m32, r32	Add r32 to r/m32
02 /r	ADD r8, r/m8	Add r/m8 to r8
03 /r	ADD r16, r/m16	Add r/m16 to r16
03 /r	ADD r32, r/m32	Add r/m32 to r32

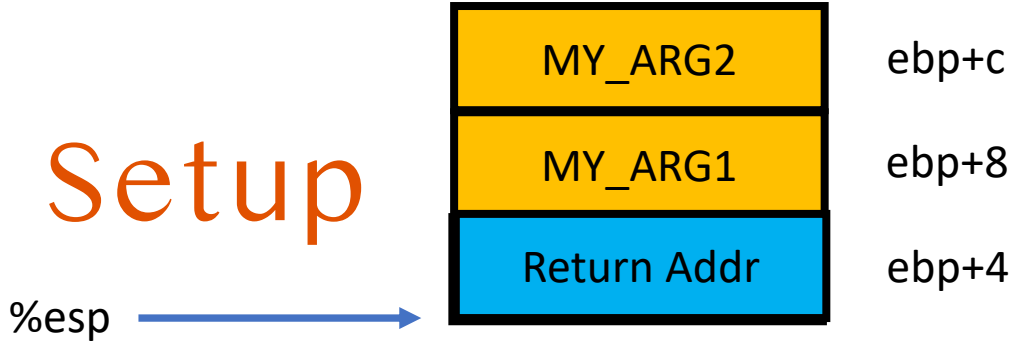
r8 : register 8 bit
r16: register 16 bit
r32: register 32bit
m8: memory 8 bit
m16: memory 16 bit
m32: memory 32bit
imm8: immediate value 8
imm16: immediate value 16
imm32: immediate value 32

Stack Usage

- CPU Stack (grows down)
  - Starts with %ebp
  - Ends with %esp
- Negative indexing over %ebp
  - Local variables
- Positive indexing over %ebp
  - Function arguments from caller
- Non-negative indexing over %esp
  - Function arguments to callee



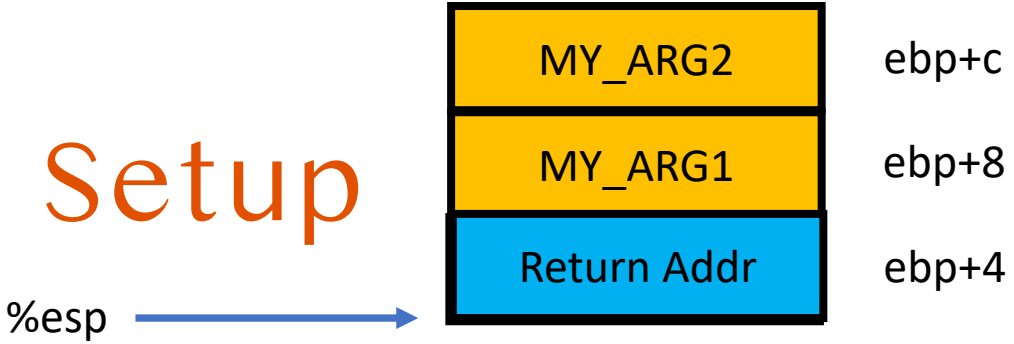
Function Prolog: Stack Setup



- CPU Stack (grows down)
  - Starts with %ebp
  - Ends with %esp

```
0x080485a0 <+0>:     push   %ebp
0x080485a1 <+1>:     mov    %esp,%ebp
0x080485a3 <+3>:     sub    $0x228,%esp
```

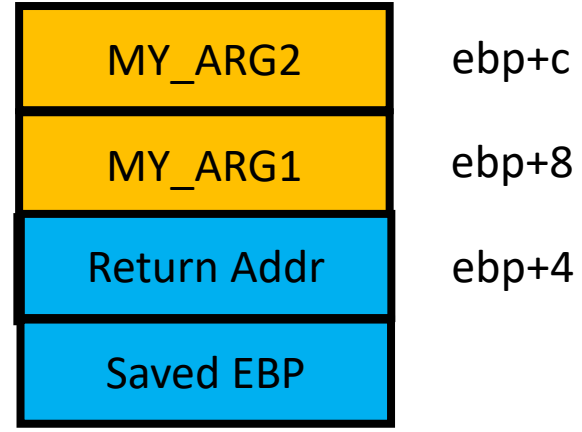
Function Prolog: Stack Setup



- CPU Stack (grows down)
  - Starts with %ebp
  - Ends with %esp

```
0x080485a0 <+0>:     push   %ebp
0x080485a1 <+1>:     mov    %esp,%ebp
0x080485a3 <+3>:     sub    $0x228,%esp
```

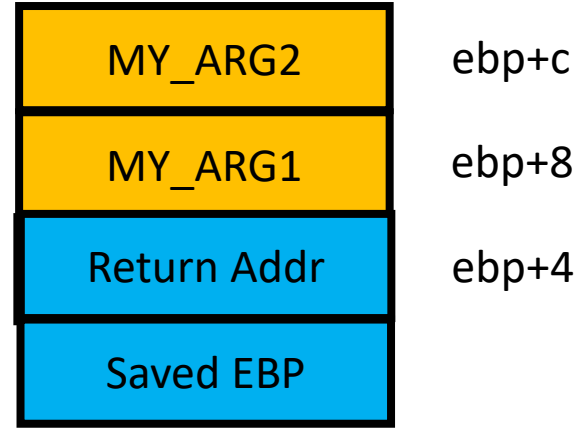
Function Prolog: Stack Setup



- CPU Stack (grows down)
  - Starts with %ebp
  - Ends with %esp

```
0x080485a0 <+0>:     push   %ebp
0x080485a1 <+1>:     mov    %esp,%ebp
0x080485a3 <+3>:     sub    $0x228,%esp
```

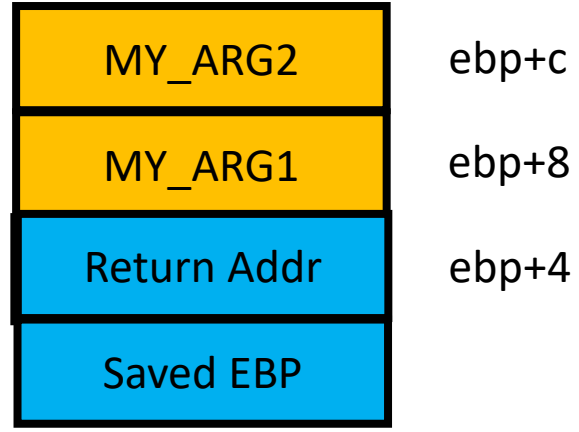
Function Prolog: Stack Setup



- CPU Stack (grows down)
  - Starts with %ebp
  - Ends with %esp

```
0x080485a0 <+0>:     push   %ebp
0x080485a1 <+1>:     mov    %esp,%ebp
0x080485a3 <+3>:     sub    $0x228,%esp
```

Function Prolog: Stack Setup

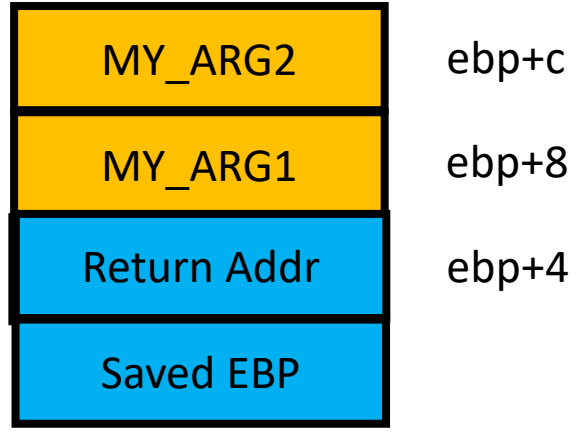


- CPU Stack (grows down)
  - Starts with %ebp
  - Ends with %esp

```
ebp = esp;
```

```
0x080485a0 <+0>:     push   %ebp
0x080485a1 <+1>:     mov    %esp,%ebp
0x080485a3 <+3>:     sub    $0x228,%esp
```

Function Prolog: Stack Setup



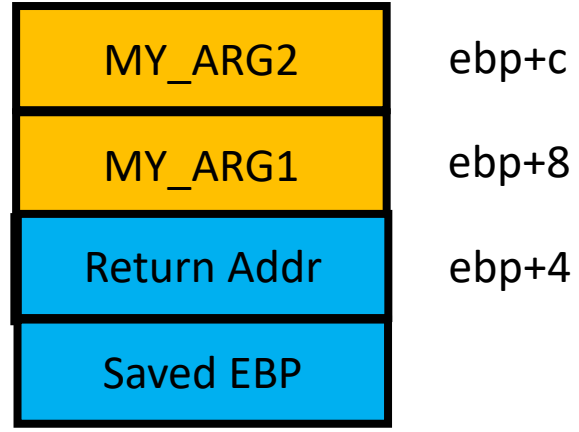
- CPU Stack (grows down)
  - Starts with %ebp
  - Ends with %esp

%ebp %esp

```
ebp = esp;
```

```
0x080485a0 <+0>:     push   %ebp
0x080485a1 <+1>:     mov    %esp,%ebp
0x080485a3 <+3>:     sub    $0x228,%esp
```

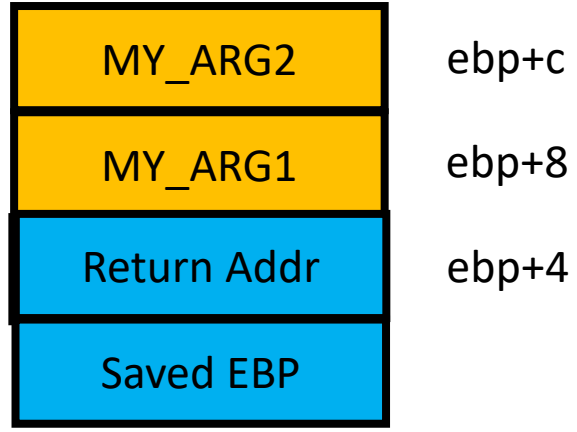
Function Prolog: Stack Setup



- CPU Stack (grows down)
  - Starts with %ebp
  - Ends with %esp

```
0x080485a0 <+0>:     push   %ebp
0x080485a1 <+1>:     mov    %esp,%ebp
0x080485a3 <+3>:     sub    $0x228,%esp
```

Function Prolog: Stack Setup



- CPU Stack (grows down)
  - Starts with %ebp
  - Ends with %esp

```
0x080485a0 <+0>:     push   %ebp
0x080485a1 <+1>:     mov    %esp,%ebp
0x080485a3 <+3>:     sub    $0x228,%esp
```

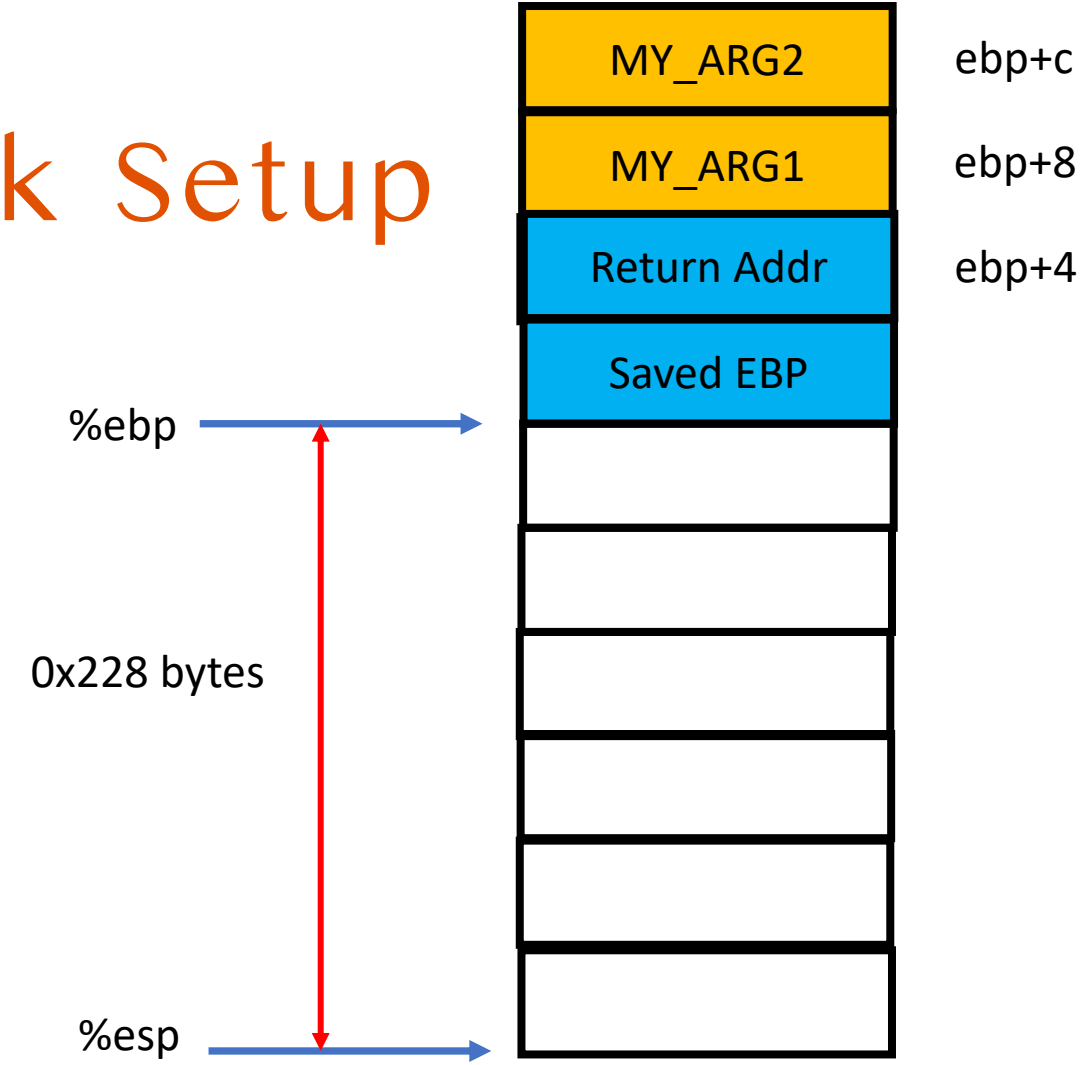
esp = esp – 0x228

Function Prolog: Stack Setup

- CPU Stack (grows down)
  - Starts with %ebp
  - Ends with %esp

```
0x080485a0 <+0>:     push   %ebp
0x080485a1 <+1>:     mov    %esp,%ebp
0x080485a3 <+3>:     sub    $0x228,%esp
```

esp = esp - 0x228

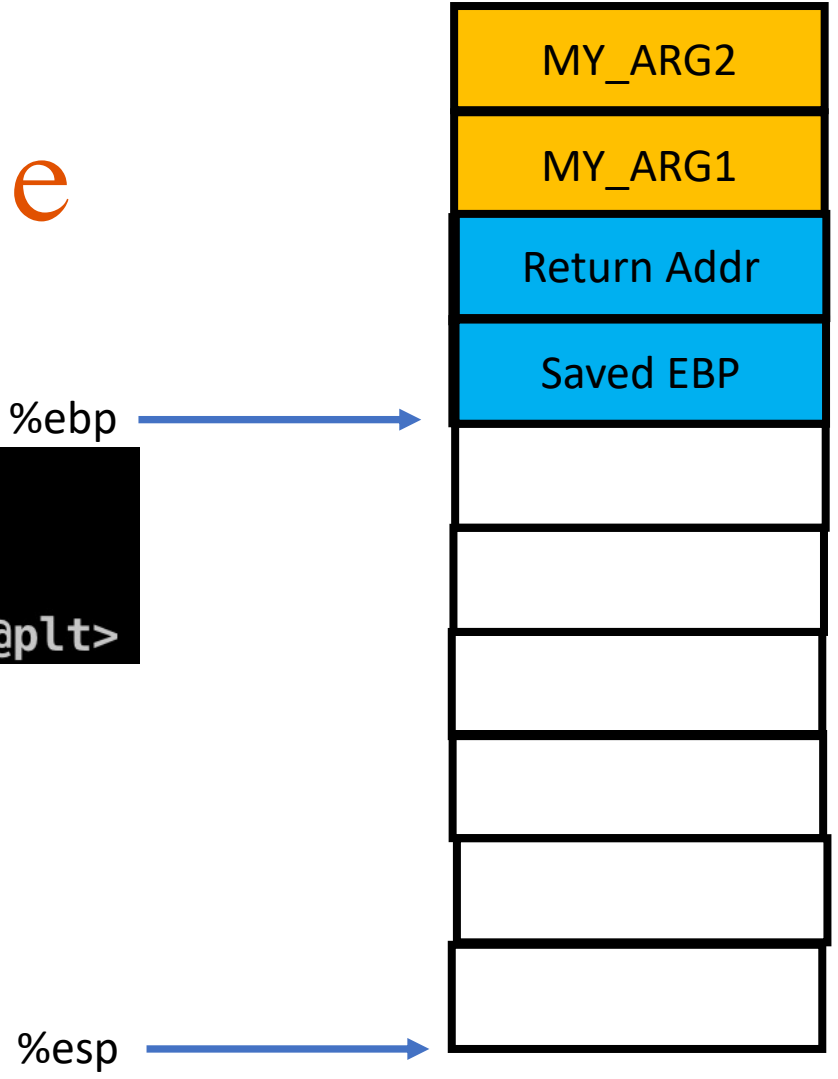


This function can use this area freely for storing local variables..

Stack Usage in Example

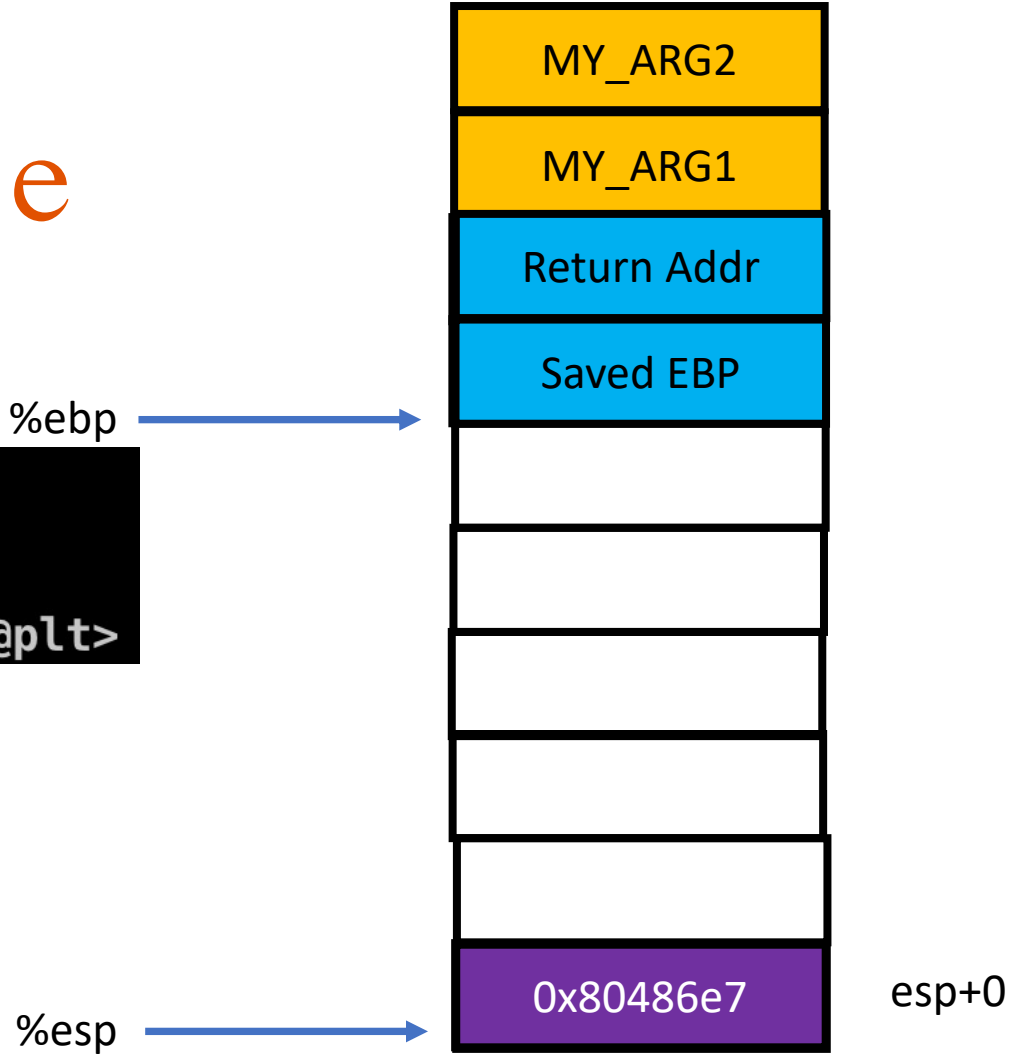
```
0x080485a9 <+9>:     lea    0x80486e7,%eax
0x080485af <+15>:    movl   $0x0,-0x4(%ebp)
0x080485b6 <+22>:    mov    %eax,(%esp)
0x080485b9 <+25>:    call   0x80483c0 <printf@plt>
```

- lea 0x80486e7, %eax
  - eax = 0x80486e7
- Mov %eax, (%esp)
  - esp[0] = %eax = 0x80486e7



Stack Usage in Example

```
0x080485a9 <+9>:     lea    0x80486e7,%eax
0x080485af <+15>:    movl   $0x0,-0x4(%ebp)
0x080485b6 <+22>:    mov    %eax,(%esp)
0x080485b9 <+25>:    call   0x80483c0 <printf@plt>
```



- lea 0x80486e7, %eax
  - eax = 0x80486e7
- Mov %eax, (%esp)
  - esp[0] = %eax = 0x80486e7
- Non-negative indexing over %esp
  - Function arguments to callee
- printf(0x80486e7)

What's in 0x80486e7?

- Use gdb to trace the execution

```
└─$ gdb level0
pwndbg: loaded 181 commands. Type pwndbg [filter] for a list.
pwndbg: created $rebase, $ida gdb functions (can be used with print/break)
Reading symbols from level0...(no debugging symbols found)...done.
pwndbg>
```

```
0x080485a9 <+9>:     lea    0x80486e7,%eax
0x080485af <+15>:    movl   $0x0,-0x4(%ebp)
0x080485b6 <+22>:    mov    %eax,(%esp)
0x080485b9 <+25>:    call   0x80483c0 <printf@plt>
```

disass main

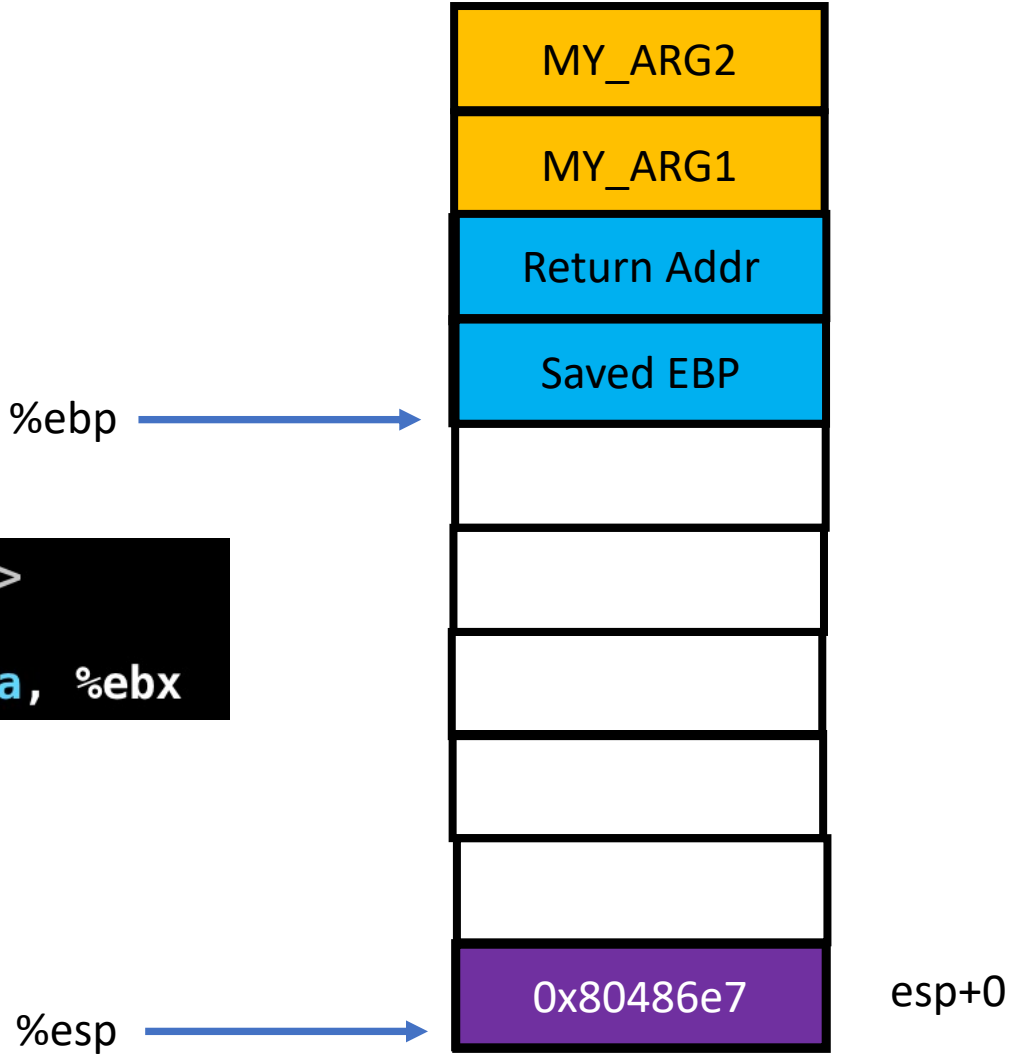
```
pwndbg> b *main+25
Breakpoint 1 at 0x80485b9
pwndbg> r
```

Set a breakpoint and run!

What's in 0x80486e7?

- GDB stopped at the break point

```
► 0x80485b9 <main+25>    calll  printf@plt <0x80483c0>
        format: 0x80486e7 ◂— "What's the password?\n"
        vararg: 0xf7ff0116 (malloc+6) ◂— addl   $0xceea, %ebx
```



- printf(0x80486e7)
  - printf("What's the password?\n");
- Examine string in gdb

```
pwndbg> x/s 0x80486e7
0x80486e7:      "What's the password?\n"
```

Pwndbg: shows func arguments

- Shows arguments when calling a function

```
► 0x80485b9 <main+25>    calll  printf@plt <0x80483c0>
        format: 0x80486e7 ◂— "What's the password?\n"
        vararg: 0xf7ff0116 (malloc+6) ◂— addl   $0xceea, %ebx
```

- So it's
  - printf("What's the password?\n");

```
14 int main() {
15     char buf[512];
16     printf("What's the password?\n");
17     scanf("%s", buf);
18     if(strcmp(buf, "PaSSw0Rd") == 0) {
19         printf("Correct!\n");
20         get_a_shell();
21     }
22     else {
23         printf("Wrong password!!\n");
24     }
25 }
```

Fast Route: call

```c
int main() {
    char buf[512];
    printf("What's the password?\n");
    scanf("%s", buf);
    if(strcmp(buf, "PaSSw0Rd") == 0) {
        printf("Correct!\n");
        get_a_shell();
    }
    else {
        printf("Wrong password!!\n");
    }
}
```

printf()

scanf()

strcmp()

printf()

get_a_shell()

printf()

Analyze call first to get an idea of what kind of functions it calls!

```
pwndbg> disass main
Dump of assembler code for function main:
   0x080485a0 <+0>:     push   %ebp
   0x080485a1 <+1>:     mov    %esp,%ebp
   0x080485a3 <+3>:     sub    $0x228,%esp
   0x080485a9 <+9>:     lea    0x80486e7,%eax
   0x080485af <+15>:    movl   $0x0,-0x4(%ebp)
   0x080485b6 <+22>:    mov    %eax,(%esp)
   0x080485b9 <+25>:    call   0x80483c0 <printf@plt>
   0x080485be <+30>:    lea    0x80486fd,%ecx
   0x080485c4 <+36>:    lea    -0x204(%ebp),%edx
   0x080485ca <+42>:    mov    %ecx,(%esp)
   0x080485cd <+45>:    mov    %edx,0x4(%esp)
   0x080485d1 <+49>:    mov    %eax,-0x208(%ebp)
   0x080485d7 <+55>:    call   0x8048410 <__isoc99_scanf@plt>
   0x080485dc <+60>:    lea    -0x204(%ebp),%ecx
   0x080485e2 <+66>:    mov    %esp,%edx
   0x080485e4 <+68>:    mov    %ecx,(%edx)
   0x080485e6 <+70>:    movl   $0x8048700,0x4(%edx)
   0x080485ed <+77>:    mov    %eax,-0x20c(%ebp)
   0x080485f3 <+83>:    call   0x80483b0 <strcmp@plt>
   0x080485f8 <+88>:    cmp    $0x0,%eax
   0x080485fb <+91>:    jne    0x804861f <main+127>
   0x08048601 <+97>:    lea    0x8048709,%eax
   0x08048607 <+103>:   mov    %eax,(%esp)
   0x0804860a <+106>:   call   0x80483c0 <printf@plt>
   0x0804860f <+111>:   mov    %eax,-0x210(%ebp)
   0x08048615 <+117>:   call   0x8048530 <get_a_shell>
   0x0804861a <+122>:   jmp    0x8048633 <main+147>
   0x0804861f <+127>:   lea    0x8048713,%eax
   0x08048625 <+133>:   mov    %eax,(%esp)
   0x08048628 <+136>:   call   0x80483c0 <printf@plt>
   0x0804862d <+141>:   mov    %eax,-0x214(%ebp)
   0x08048633 <+147>:   mov    -0x4(%ebp),%eax
   0x08048636 <+150>:   add    $0x228,%esp
   0x0804863c <+156>:   pop    %ebp
   0x0804863d <+157>:   ret
End of assembler dump.
```

Local Buffer Variable

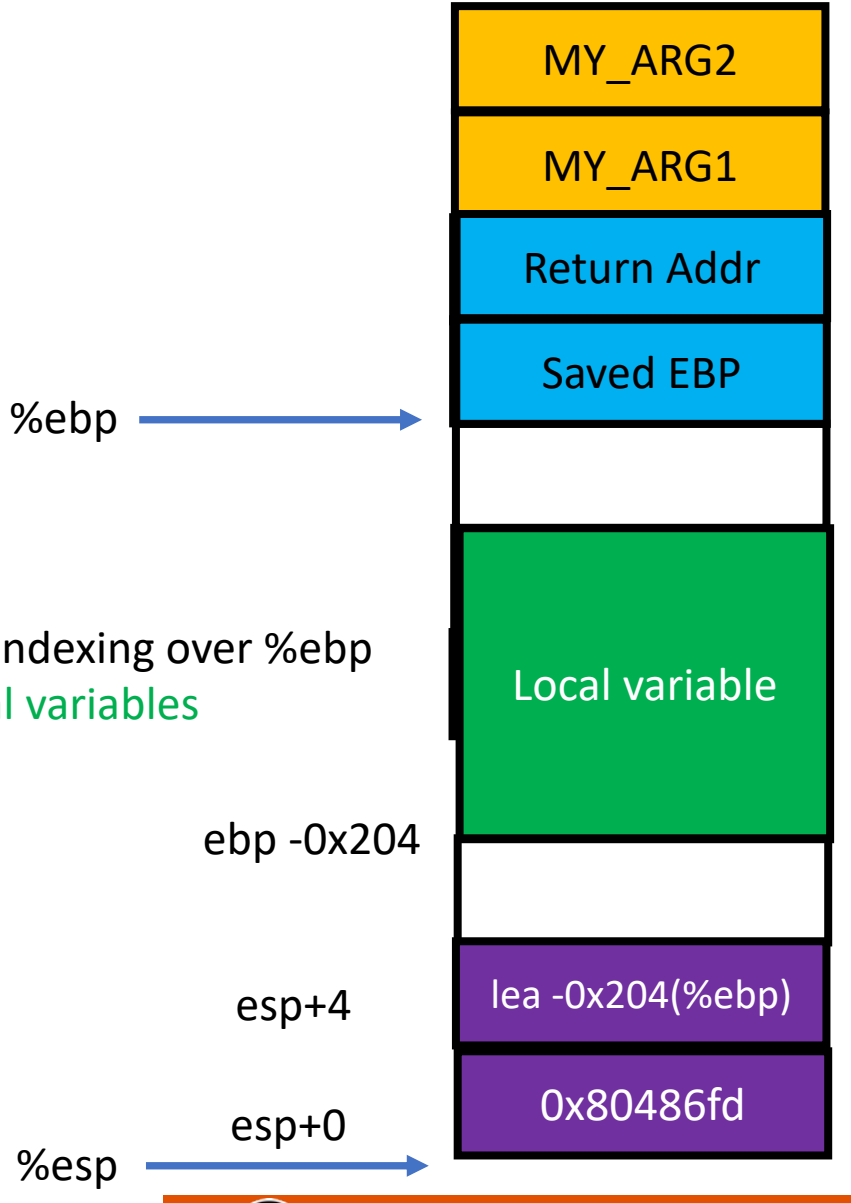
```
0x80485be <main+30>    leal   0x80486fd, %ecx
0x80485c4 <main+36>    leal   -0x204(%ebp), %edx
0x80485ca <main+42>    movl   %ecx, 0(%esp)
0x80485cd <main+45>    movl   %edx, 4(%esp)
0x80485d1 <main+49>    movl   %eax, -0x208(%ebp)
0x80485d7 <main+55>    calll  __isoc99_scanf@plt <0x8048410>
```

- leal 0x80486fd, %ecx
  - `ecx = 0x80486fd;`
- leal -0x204 (%ebp), %edx
  - `edx = &ebp[-129];` (-0x204/4 == -129)
- movl %ecx, 0(%esp)
  - `esp[0] = ecx;`
- Movl %edx, 4(%esp)
  - `esp[1] = edx;`

Local Buffer Variable

```
0x80485be <main+30>    leal   0x80486fd, %ecx
0x80485c4 <main+36>    leal   -0x204(%ebp), %edx
0x80485ca <main+42>    movl   %ecx, 0(%esp)
0x80485cd <main+45>    movl   %edx, 4(%esp)
0x80485d1 <main+49>    movl   %eax, -0x208(%ebp)
0x80485d7 <main+55>    calll  __isoc99_scanf@plt <0x8048410>
```

- leal 0x80486fd, %ecx
  - `ecx = 0x80486fd;`
- leal -0x204 (%ebp), %edx
  - `edx = &ebp[129];` (0x204/4 == 129)
- movl %ecx, 0(%esp)
  - `esp[0] = ecx;`
- Movl %edx, 4(%esp)
  - `esp[1] = edx;`



scanf()?

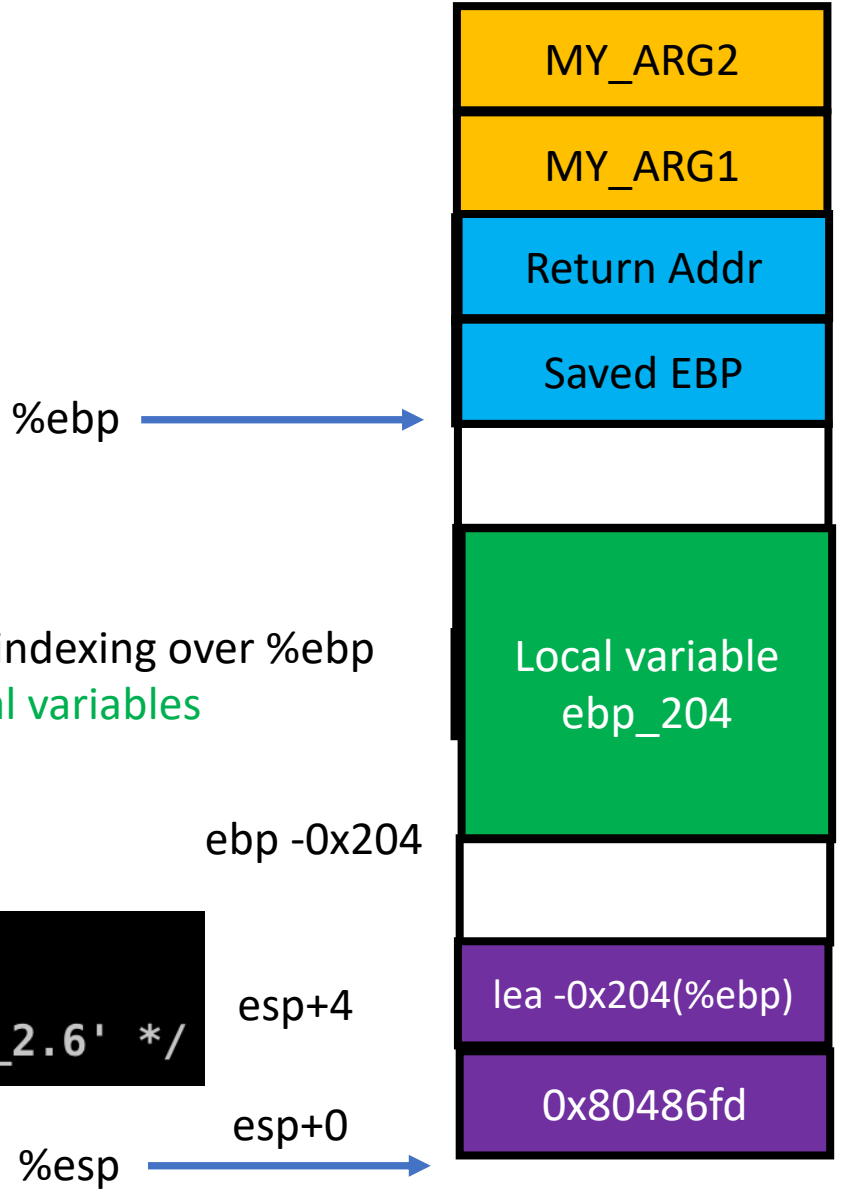
```
0x80485be <main+30>    leal   0x80486fd, %ecx
0x80485c4 <main+36>    leal   -0x204(%ebp), %edx
0x80485ca <main+42>    movl   %ecx, 0(%esp)
0x80485cd <main+45>    movl   %edx, 4(%esp)
0x80485d1 <main+49>    movl   %eax, -0x208(%ebp)
0x80485d7 <main+55>    calll  __isoc99_scanf@plt <0x8048410>
```

- `scanf(0x80486fd, &ebp_204);`

```
pwndbg> b *main+55
Breakpoint 2 at 0x80485d7
pwndbg> c
```

```
► 0x80485d7 <main+55>     calll  __isoc99_scanf@plt <0x8048410>
        format: 0x80486fd ◂— 0x50007325 /* '%s' */
        vararg: 0xffffcd94 —▸ 0xf7f5a398 ◂— decl   %esp /* 'LINUX_2.6' */
```

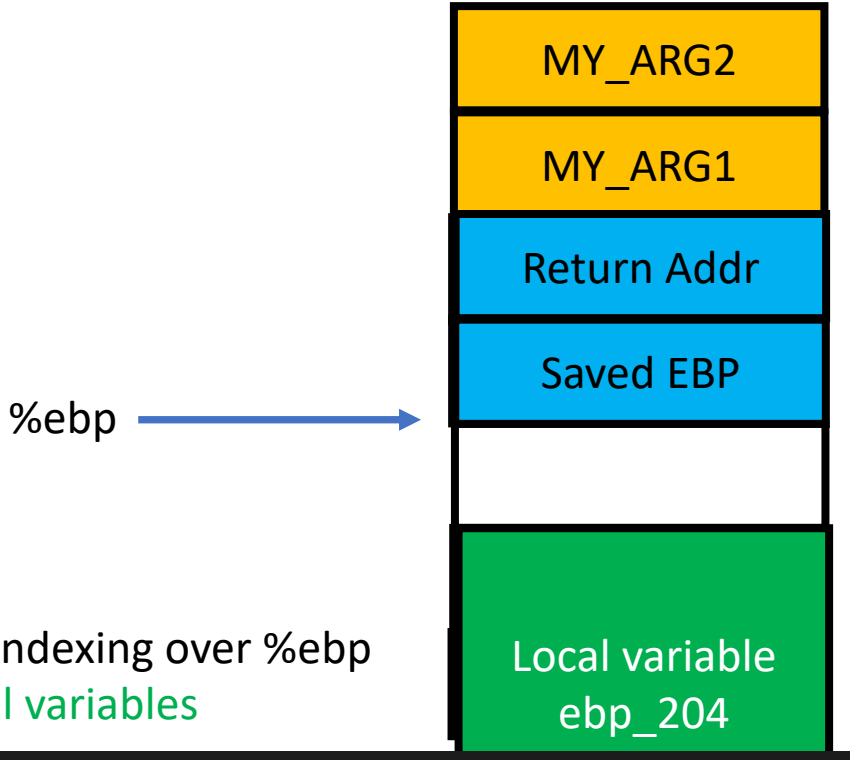
- `scanf("%s", &ebp_204);`



Oregon State University

scanf()?

```
0x80485be <main+30>    leal   0x80486fd, %ecx
0x80485c4 <main+36>    leal   -0x204(%ebp), %edx
0x80485ca <main+42>    movl   %ecx, 0(%esp)
0x80485cd <main+45>    movl   %edx, 4(%esp)
0x80485d1 <main+49>    movl   %eax, -0x208(%ebp)
0x80485d7 <main+55>    calll  __isoc99_scanf@plt <0x8048410>
```



- `scanf(0x80485af, &ebp_204);`

Negative indexing over %ebp

Local variables

```
pwndbg> b *main+55
Breakpoint 2 at 0x80485d7
pwndbg> c
```

```
► 0x80485d7 <main+55>     calll  __isoc99_scanf@plt
        format: 0x80486fd ◂— 0x50007325 /* '%s' */
        vararg: 0xffffcd94 —▸ 0xf7f5a398 ◂— decl   %
```

```
14 int main() {
15     char buf[512];
16     printf("What's the password?\n");
17     scanf("%s", buf);
18     if(strcmp(buf, "PaSSw0Rd") == 0) {
19         printf("Correct!\n");
20         get_a_shell();
21     }
22     else {
23         printf("Wrong password!!\n");
24     }
25 }
```

- `scanf("%s", &ebp_204);`

strcmp()

```
0x80485dc <main+60>     leal   -0x204(%ebp), %ecx
0x80485e2 <main+66>     movl   %esp, %edx
0x80485e4 <main+68>     movl   %ecx, 0(%edx)
0x80485e6 <main+70>     movl   $0x8048700, 4(%edx)
0x80485ed <main+77>     movl   %eax, -0x20c(%ebp)
0x80485f3 <main+83>     calll  strcmp@plt <0x80483b0>
```

- leal -0x204(%ebp) , %ecx
  - `ecx = &ebp_204;`
- mov %esp, %edx
  - `edx = esp;` (now edx stores same address as %esp)
- movl %ecx, 0(%edx)
  - We can rewrite this as movl %ecx, 0(%esp) because edx == esp
  - `esp[0] = %ecx;`
- movl $0x8048700, 4(%edx)
  - `esp[1] = 0x8048700;` (because edx == esp)
- strcmp(&ebp_204, 0x8048700)
  - strcmp(&ebp_204, “PaSSw0Rd”);

```
pwndbg> x/s 0x8048700
0x8048700:      "PaSSw0Rd"
```

strcmp()

- Let's see that on gdb (it waits for our input bcz it call scanf("%s",...))

```
pwndbg> b *main+83
Breakpoint 3 at 0x80485f3
pwndbg> c
Continuing.
asdfasdf
```

```
► 0x80485f3 <main+83>     calll  strcmp@plt <0x80483b0>
        s1: 0xffffcd94 ◂— 'asdfasdf'
        s2: 0x8048700 ◂— 'PaSSw0Rd'
```

- strcmp(&ebp_208, "PaSSw0Rd");

What's Next?

```
0x080485f8 <+88>:    cmp    $0x0,%eax
0x080485fb <+91>:    jne    0x804861f <main+127>
0x08048601 <+97>:    lea    0x8048709,%eax
0x08048607 <+103>:   mov    %eax,(%esp)
0x0804860a <+106>:   call   0x80483c0 <printf@plt>
0x0804860f <+111>:   mov    %eax,-0x210(%ebp)
0x08048615 <+117>:   call   0x8048530 <get_a_shell>
0x0804861a <+122>:   jmp    0x8048633 <main+147>
0x0804861f <+127>:   lea    0x8048713,%eax
0x08048625 <+133>:   mov    %eax,(%esp)
0x08048628 <+136>:   call   0x80483c0 <printf@plt>
```

- cmp $0x0, %eax
  - eax will store the return value of a function call
  - What was the last function call?
  - strcmp(&ebp_208, "PaSSw0Rd");
  - strcmp() returns 0 if two strings are equal

Handling an If Condition

- cmp $0x0, %eax
  - eax will store the return value of a function call
  - What was the last function call?
  - eax = strcmp(&ebp_208, “PaSSw0Rd”);
  - strcmp() returns 0 if two strings are equal

```
0x080485f8 <+88>:    cmp    $0x0,%eax
0x080485fb <+91>:    jne    0x804861f <main+127>
```

if (eax != 0)

```
0x0804861f <+127>:   lea    0x8048713,%eax
0x08048625 <+133>:   mov    %eax,(%esp)
0x08048628 <+136>:   call   0x80483c0 <printf@plt>
```

else

```
0x08048601 <+97>:    lea    0x8048709,%eax
0x08048607 <+103>:   mov    %eax,(%esp)
0x0804860a <+106>:   call   0x80483c0 <printf@plt>
0x0804860f <+111>:   mov    %eax,-0x210(%ebp)
0x08048615 <+117>:   call   0x8048530 <get_a_shell>
0x0804861a <+122>:   jmp    0x8048633 <main+147>
```

Handling an If Condition

- if (eax != 0) {

```
0x0804861f <+127>:   lea    0x8048713,%eax
0x08048625 <+133>:   mov    %eax,(%esp)
0x08048628 <+136>:   call   0x80483c0 <printf@plt>
```

  - printf(0x8048713);

```
pwndbg> x/s 0x8048713
0x8048713:      "Wrong password!!\n"
```

- }
- else {
  - printf(0x8048709);
  - get_a_shell();

```
0x08048601 <+97>:    lea    0x8048709,%eax
0x08048607 <+103>:   mov    %eax,(%esp)
0x0804860a <+106>:   call   0x80483c0 <printf@plt>
0x0804860f <+111>:   mov    %eax,-0x210(%ebp)
0x08048615 <+117>:   call   0x8048530 <get_a_shell>
0x0804861a <+122>:   jmp    0x8048633 <main+147>
```

```
pwndbg> x/s 0x8048709
0x8048709:      "Correct!\n"
```

- }

Let's Combine All the Code

- printf("What's the password?\n");
- scanf("%s", &ebp_208);
- if (strcmp(&ebp_208, "PaSSw0Rd") != 0) {
  - printf("Wrong Password!!\n");
- }
- else {
  - printf("Correct!\n");
  - get_a_shell();
- }

```
14 int main() {
15     char buf[512];
16     printf("What's the password?\n");
17     scanf("%s", buf);
18     if(strcmp(buf, "PaSSw0Rd") == 0) {
19         printf("Correct!\n");
20         get_a_shell();
21     }
22     else {
23         printf("Wrong password!!\n");
24     }
25 }
```

Basic Instructions

- push %ebp
  - Store the value of ebp at the top of the stack (esp gets -4)
  - mov %ebp, (%esp)
  - sub $4, %esp

- pop %ebp
  - Move the value at the stack top to ebp (esp gets +4)
  - mov (%esp), %ebp
  - add $4, %esp

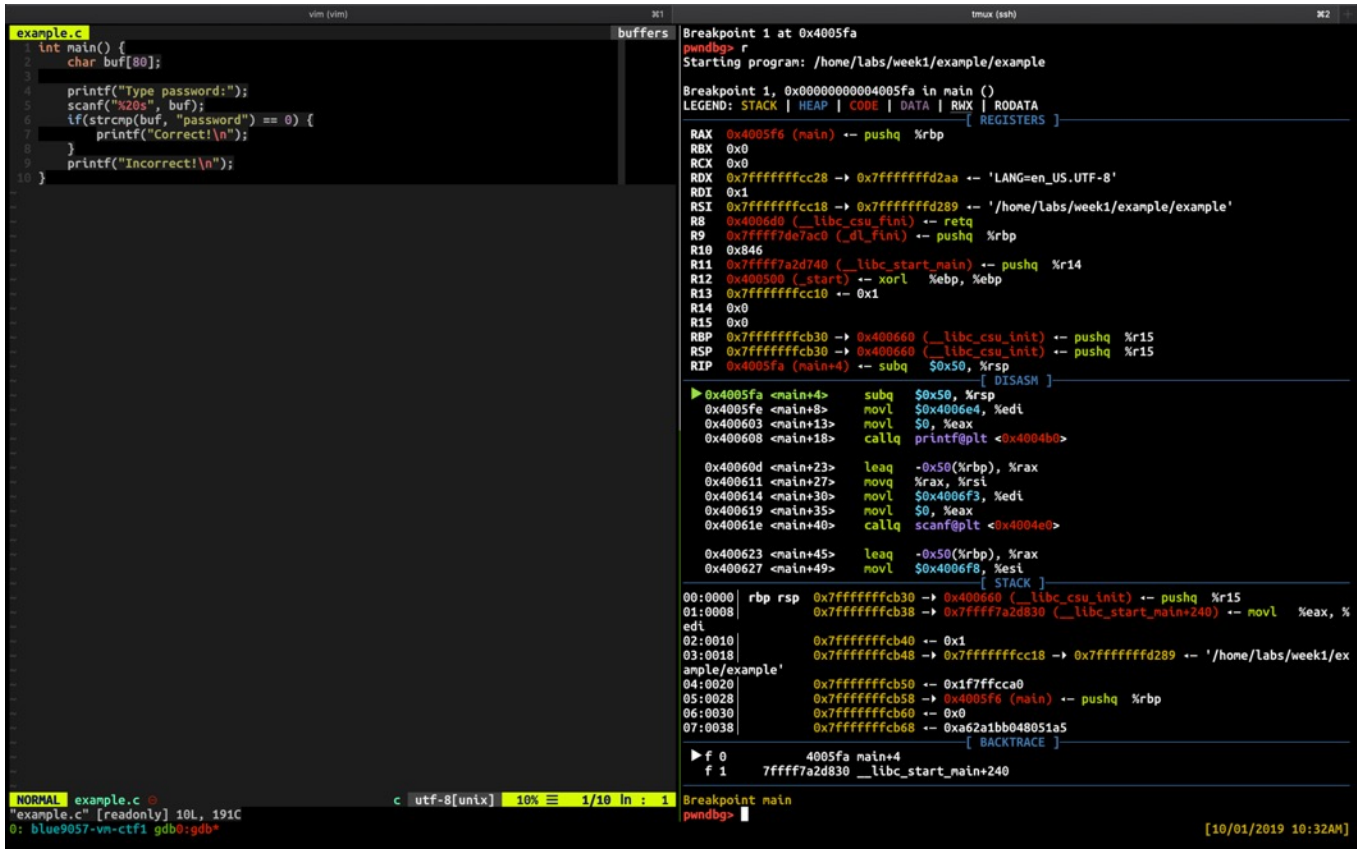
Variables (only existing in our mind..)

- Local variables – accessed by a negative index over %ebp register
  - mov -0x144(%ebp), %eax
  - mov %eax, -0x208(%ebp)
  - mov %al, -0x210(%ebp, %ecx, 1)
- Global variables – accessed by an address
  - lea 0x8048440, %ebx; mov (%ebx), %eax
  - mov %al, 0x8048440(,%ecx, 1)
- Stack arguments – accessed by a non-negative index over %esp
  - mov %eax, (%esp) <- index 0 over %esp, 1st argument
  - mov %eax, 0x4(%esp) <- index 4 over %esp, 2nd argument
  - mov %eax, 0x8(%esp) <- index 8 over %esp, 3rd argument
  - …

Debugging with Example

- /home/labs/week1/example/example

- Use tmux to split the window
  - Left: vim
  - Right: gdb



TMUX tips

- ` + c : create a new window
- ` + [0-9] (a number) : select the window with that number
- ` + -> (right arrow) : move to the right pane
- ` + <- (left arrow): move to the left pane
- ` + % : split the pane vertically
- ` + “ : split the pane horizontally

- TMUX CHEAT SHEETS: https://tmuxcheatsheet.com/
- Use backtick for CTRL-B in the default setting in vm-ctf1.

GDB

- b main
  - Set a break point at the main function
  - Put a function name to set a break point
- r
  - Run the program up to the break points
- ni
  - Execute next instruction, does not follow function calls
- si
  - Execute next instruction, follows function calls

GDB

Gdb tutorial 1: http://beej.us/guide/bggdb/
Gdb tutorial 2: https://www.exploit-db.com/papers/13205
Commands: https://visualgdb.com/gdbreference/commands/
(Laura, thank you!)

- x/x $eax
  - Get the value of eax register

```
Breakpoint main
pwndbg> x/x $eax
0xf7fb2dbc:     0xffffbf3c
```

- x/40wx $esp
  - Get the 40 integer values stored at the address pointed by esp

```
pwndbg> x/40wx $esp
0xffffcd70:     0x080486fd      0xffffcd94      0xf7ffd000      0xf7ff0179
0xffffcd80:     0x00000070      0xf7ff0116      0xf7fe933d      0xf7fe1f60
0xffffcd90:     0x00000015      0xf7f5a398      0xf7ffd53c      0xf7fe4017
0xffffcda0:     0xf7ffc000      0x00001000      0x00000001      0x03ae75f6
0xffffcdb0:     0xf7ffdad0      0xf7fd3790      0xf7fe1e39      0xf7fd7128
0xffffcdc0:     0x00000007      0xf7ffdc08      0x6e43a318      0xf7fe263d
0xffffcdd0:     0x00000000      0x00000000      0xf7fd71a0      0x00000007
0xffffcde0:     0xf7fd71c0      0xf7ffdc08      0xffffce3c      0xffffce38
0xffffcdf0:     0x00000001      0x00000000      0xf7ffd000      0xf7f5a3a2
0xffffce00:     0x6e43a318      0xf7fe1f60      0xf7e122e5      0x080482e5
```

Oregon State University

Tutorials and Slides

- https://cand.unexploitable.systems/cal.html

- Today's tutorial:
  - https://cand.unexploitable.systems/l/lec01/tut-level5.txt
- At LEC X, you can check links to Slides(PPTX and PDF)
- At TUT X, you can check a link to tutorial TXT

Assignment Due

- 4/13 02:00 pm (before class starts!)

Apr 13
LEC 5: Frame-pointer attack
DUE: Week 1

Oregon State University