Cyber Attacks & Defense

Arbitrary Read/Write

Yeongjin Jang

Oregon State University

Schedules

Feb 20 **LEC 12:** Arbitrary read/write **TUT 28:** tutorial VIDEO PY **TUT 29:** tutorial PY PY PY	Feb 21	Feb 22 **LEC 13:** Format String Vulnerability **TUT 30:** tutorial PY PY PY **DUE:** Week 4
Feb 27 **LEC 14:** Final CTF challenges	Feb 28	Mar 1 **LEC 15:** Defense **DUE:** Week 5
Mar 6 **LEC 16:** ShadowStack, CFI, and other defenses	Mar 7	Mar 8 **LEC 17:** Movie Day -- Hackers **DUE:** Week 6
Mar 13 Final CTF (online, no class)	Mar 14	Mar 15 **LEC 18:** Award Ceremony **DUE:** Final CTF

Final CTF

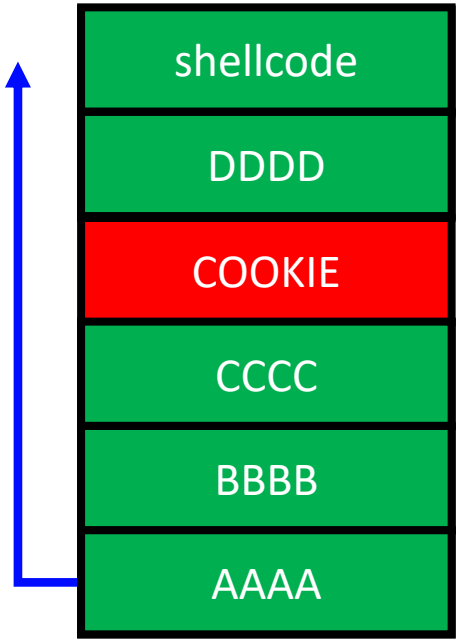
- 15+ challenges
- All extra credits

- Awards for 1st, 2nd, and 3rd place
  - Prizes are scrambled

Buffer Overflow and Control-flow Hijacking

- Buffer overflow: fill the buffer more than its size
  - Overwriting return address (saved %eip)
  - Overwriting frame pointer (saved %ebp)

- Control-flow Hijacking
  - Return to some other functions
  - Return to shellcode
  - Return to ROP gadgets

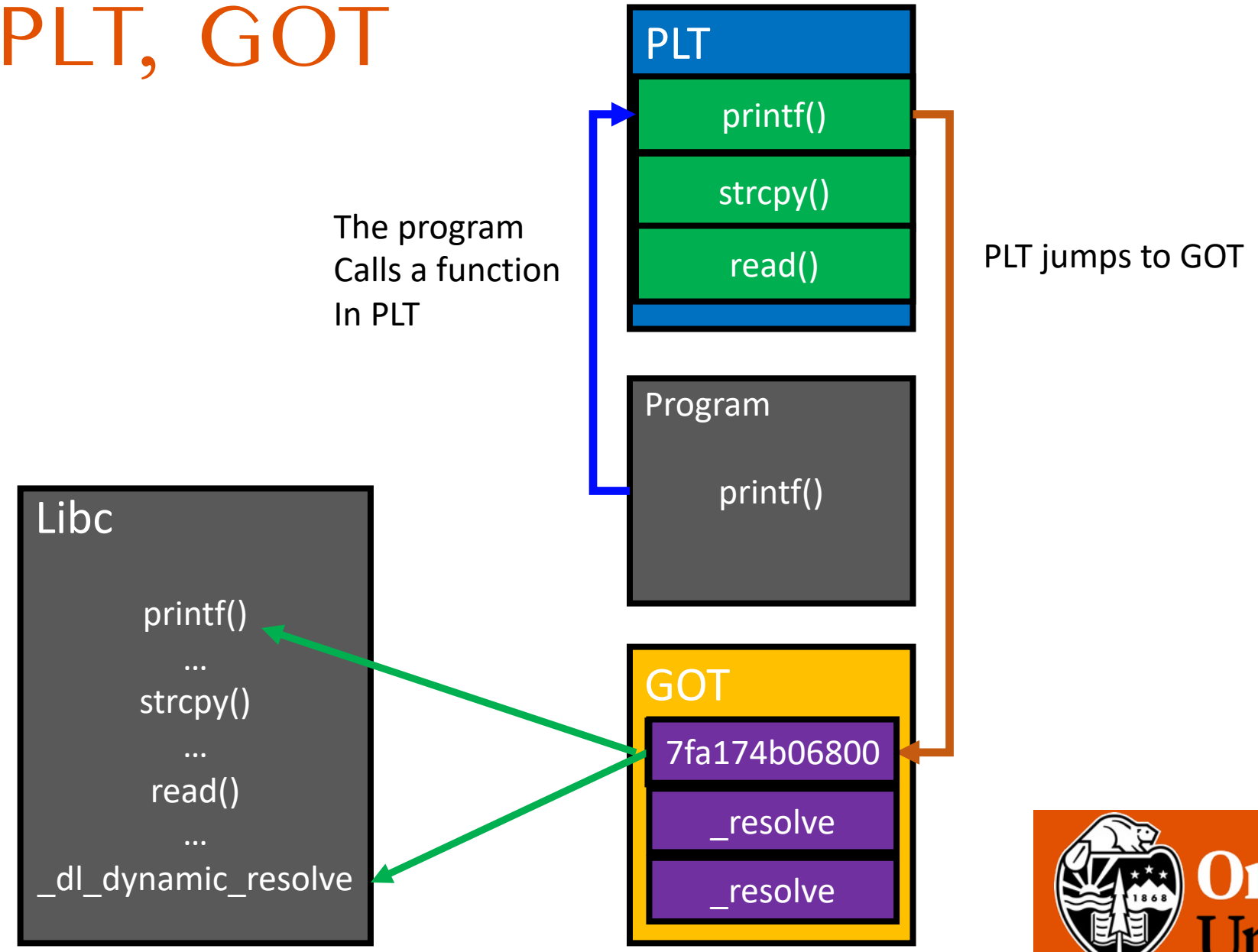


Control-flow Hijacking

- Control-flow Hijacking
  - Return/jump/call to some other functions
  - Return/jump/call to shellcode
  - Return/jump/call to ROP gadgets

- Attack targets
  - Return address
  - Global offset table (GOT)
  - Function pointer
  - Jump table

ELF, PLT, GOT

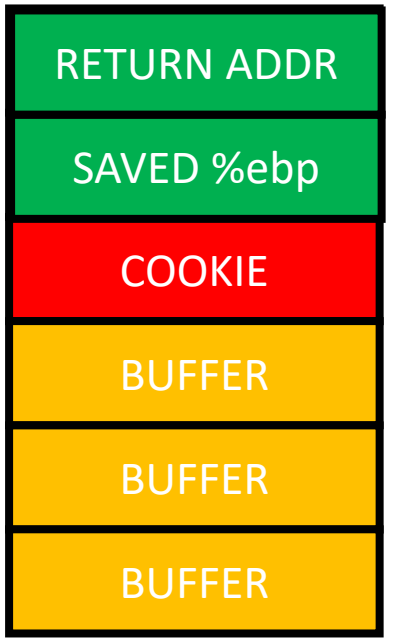


GOT stores addr of _dl_dynamic_resolve

Oregon State University

What is the Requirement for CFH?

- Is buffer overflow the only way to achieve control-flow hijacking?
  - Overwriting return address (saved %eip)
  - Overwriting frame pointer (saved %ebp)



- What we need is the capability to write on certain addr
  - Arbitrary write!

- Today we will learn arbitrary read/write

Attack Primitives

- Arbitrary Read
  - Read from any address A, any number N of bytes
  - The attacker must know the address A

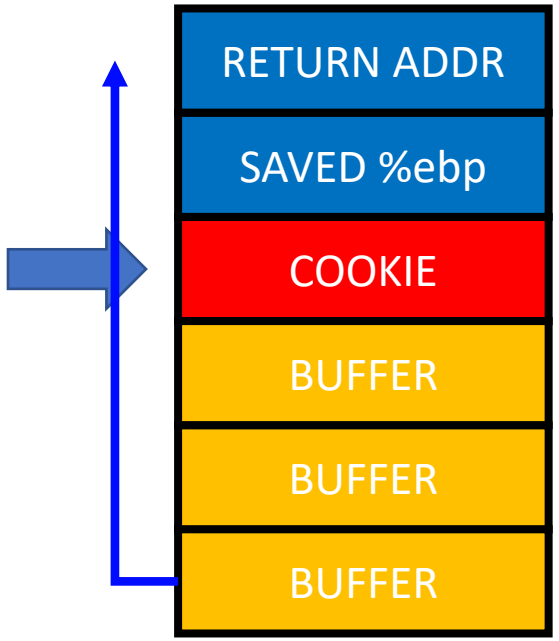
- Example:
  - Read 100 bytes from 0xffffd100 (somewhere in the stack)
  - Read 100 bytes from 0x8048500 (somewhere in the code section)

- Can this break:
  - Stack-Cookie? 
  - ASLR? 
  - DEP? 

Breaking Stack-Cookie

- The cookie value is on the stack

```
printf("How many bytes of your name do you want to print?\n");
scanf("%d", &i);
printf("Hello ");
write(1, buf, i);
printf("!\n");
```



- Sequential read can break stack-cookie easily

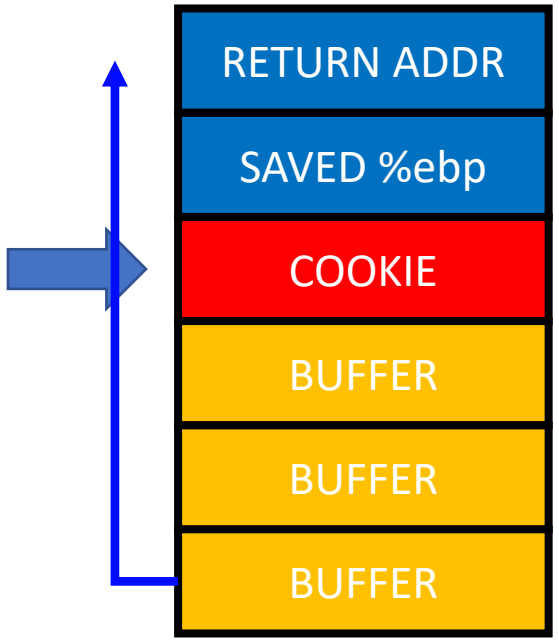
Example of Sequential Read

- ASLR-3
  - Leak addresses via sequential leaks from one point of the stack
  - We do not know ‘A’, but can specify ‘N’
  - Sometimes this is more powerful than arbitrary read

```
$ ./aslr-3
Your buffer? I don't wanna let you know my address!
Please type your name:
asdf
How many bytes of your name do you want to print?
500
asdf
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible] Hello !
```

Breaking Stack-Cookie

- The cookie value is on the stack
- Arbitrary Read
  - Read where?
  - The address that the cookie is stored!
- Then launch buffer overflow attack
- You should know the address of the stack..



Breaking ASLR

- Arbitrary Read
  - Read anywhere if you know the address…
- Read where?
  - Non-PIE: Code section is fixed 0x8048000 ~ 0x804a000
    - 0x401000 ~ and 0x601000 in 64 bit
  - Stack – random
  - Library – random
  - Heap – random

- Code/data sections are the only fixed locations

Breaking ASLR + DEP

- GOT stores address of libc functions
  - printf, puts, read, etc.

```
Relocation section '.rela.plt' at offset 0x560 contains 11 entries:
  Offset          Info           Type           Sym. Value    Sym. Name + Addend
000000601018  000100000007 R_X86_64_JUMP_SLO 0000000000000000 puts@GLIBC_2.2.5 + 0
000000601020  000200000007 R_X86_64_JUMP_SLO 0000000000000000 printf@GLIBC_2.2.5 + 0
000000601028  000300000007 R_X86_64_JUMP_SLO 0000000000000000 read@GLIBC_2.2.5 + 0
000000601030  000400000007 R_X86_64_JUMP_SLO 0000000000000000 __libc_start_main@GLIBC_
000000601038  000500000007 R_X86_64_JUMP_SLO 0000000000000000 fgets@GLIBC_2.2.5 + 0
000000601040  000700000007 R_X86_64_JUMP_SLO 0000000000000000 prctl@GLIBC_2.2.5 + 0
000000601048  000800000007 R_X86_64_JUMP_SLO 0000000000000000 fflush@GLIBC_2.2.5 + 0
000000601050  000900000007 R_X86_64_JUMP_SLO 0000000000000000 __isoc99_sscanf@GLIBC_2.
000000601058  000a00000007 R_X86_64_JUMP_SLO 0000000000000000 getegid@GLIBC_2.2.5 + 0
000000601060  000b00000007 R_X86_64_JUMP_SLO 0000000000000000 setregid@GLIBC_2.2.5 + 0
000000601068  000c00000007 R_X86_64_JUMP_SLO 0000000000000000 fwrite@GLIBC_2.2.5 + 0
```

Breaking ASLR + DEP

- Read a value from GOT
  - Can get the address of puts()
- Can you calculate the address of execve() or system() from puts?
- Yes
  - Load libc and find offsets, diff, and calculate

```
blue9057@blue9057-vm-ctf2 /home/labs/week6/arbitrary-read-1 $ ldd arbitrary-read-1
        linux-vdso.so.1 =>  (0x00007ffc2f3fa000)
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f58954e3000)
        /lib64/ld-linux-x86-64.so.2 (0x00007f58958ad000)
```

Challenges – Week6

- sr-1
  - The program will perform a sequential read from the stack for you
  - Read addresses from the leak and exploit the vulnerability!
  - All protection enabled: ASLR + DEP + Stack-cookie + PIE
  - Steps: break stack cookie first, then kill ASLR on libc, call execv!
- ar-2
  - You may perform arbitrary read
    - Can specify where to read and how many bytes to read
  - Read GOT of some functions to get the address of that function in libc
  - Call system, execve, whatever you want!

Attack Primitives

- Arbitrary Write
  - Write on any address, any number of bytes
  - The attacker must know the address
- Example:
  - Write 100 bytes of data to 0xffffd100 (somewhere in the stack)
  - Write 100 bytes of data to 0x8048500 (somewhere in the code section)
- Can achieve control-flow hijacking
  - Overwrite return address
  - Overwrite frame pointer
  - Overwriting GOT

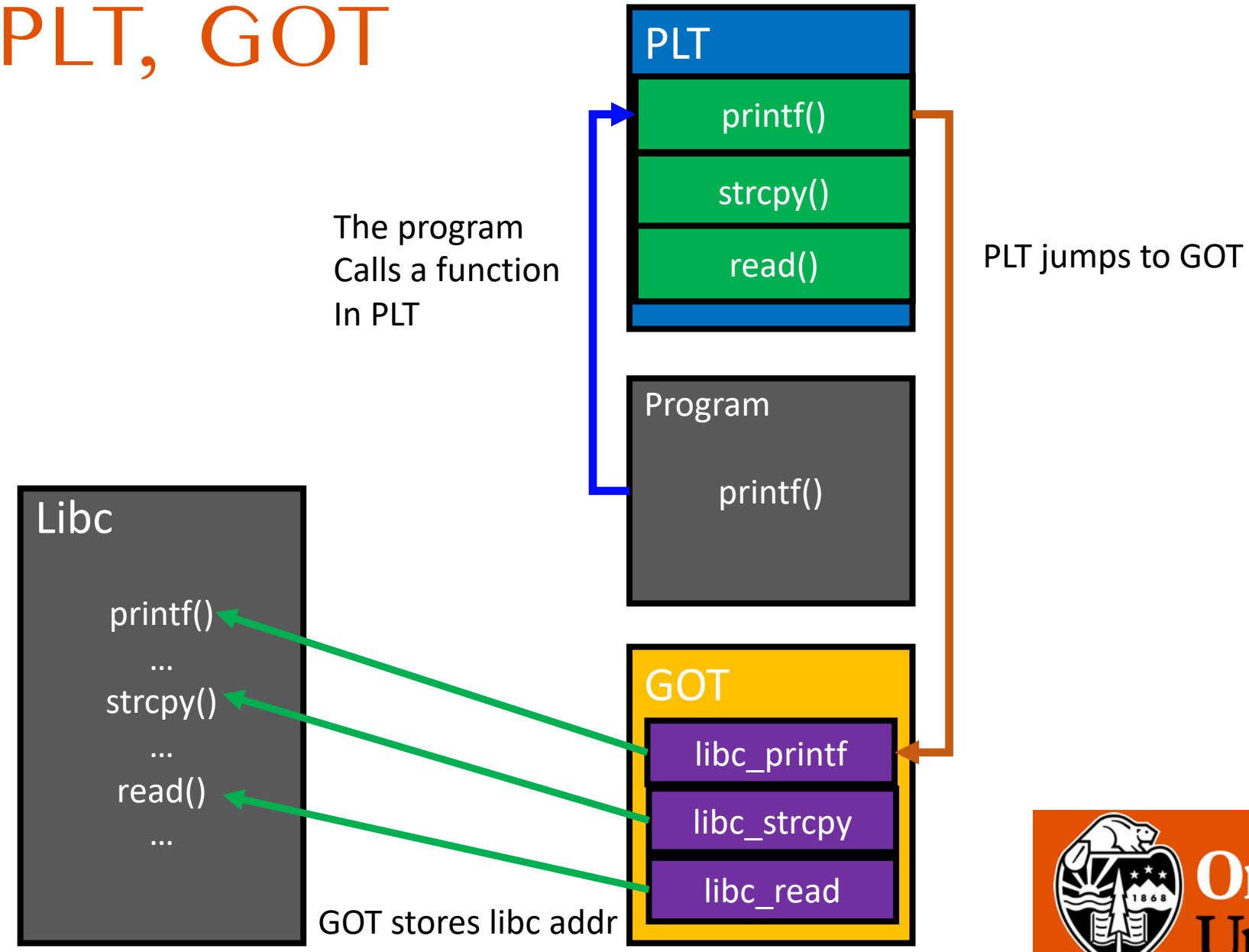
Attacking Global Offset Table

- Read the address in GOT
- Leak the address of printf()
- Calculate the address of system() from the offset

- Write that address system() to printf's GOT

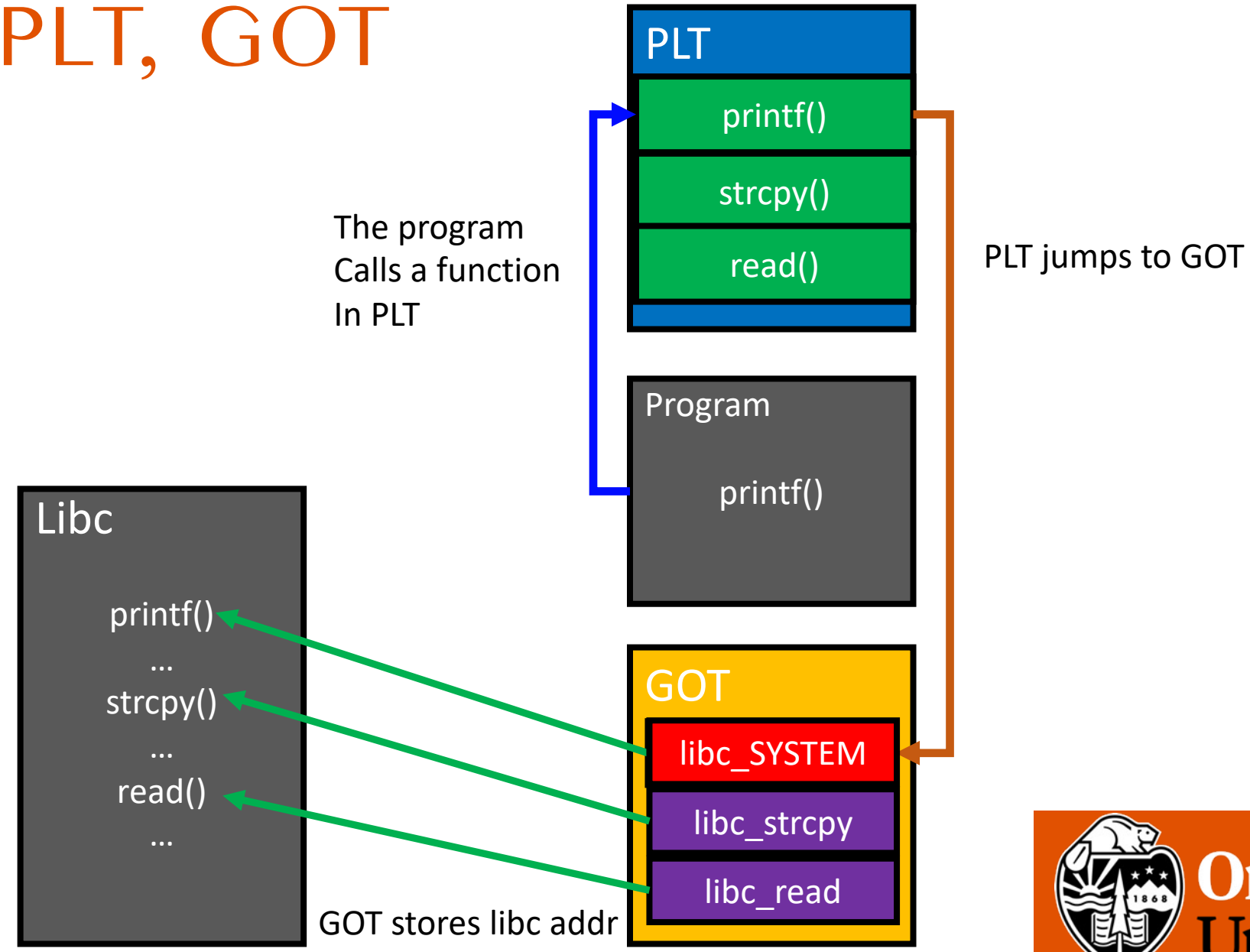
- What will happen a program calls printf?

ELF, PLT, GOT



Oregon State University

ELF, PLT, GOT



Attacks with Arbitrary Write

- GOT address is known for non-PIE

- Choose one function that is called in the program

```
    printf("Writing %lu bytes to %p\n", read_bytes, ptr);
}
```

- Overwrite the GOT of printf to system()
  - What will happen?
  - system("Writing %lu bytes to %p\n") -> fails…

Arbitrary-Write-1

- Change the GOT of printf to to ‘please_`execute_me()`’ via arbitrary write capability!

- Printf after the buffer overflow vulnerability will run that function for you.

Arbitrary-Write-2

- Program contains both arbitrary-read and arbitrary-write vulnerability.
- Exploit arbitrary read to get the address of libc function
  - Then calculate the address of system()
- Exploit arbitrary write to overwrite the GOT of printf to system()
  - `printf("Writing %lu bytes to %p\n”);`
  - `system(“Writing %lu bytes to %p\n”);`

Assignment: Week-6

- Please solve challenges in the `/home/labs/week6` directory
  - Hosted in vm-ctf2

- **Due: 03/08 11:59pm**