CS419/579
Cyber Attacks & Defense
Arbitrary Read/Write and Format String

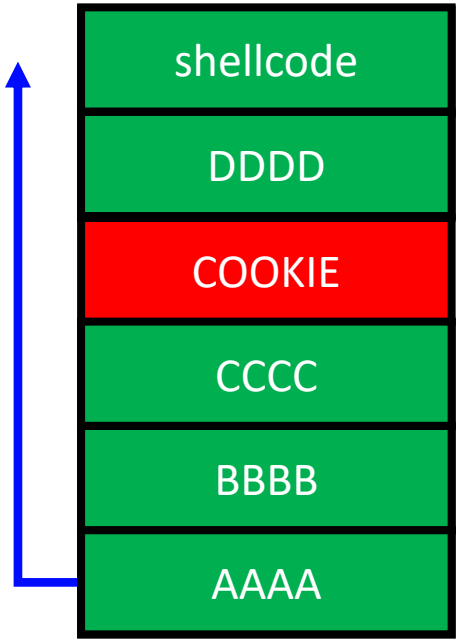
11/12/19

Oregon State University

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

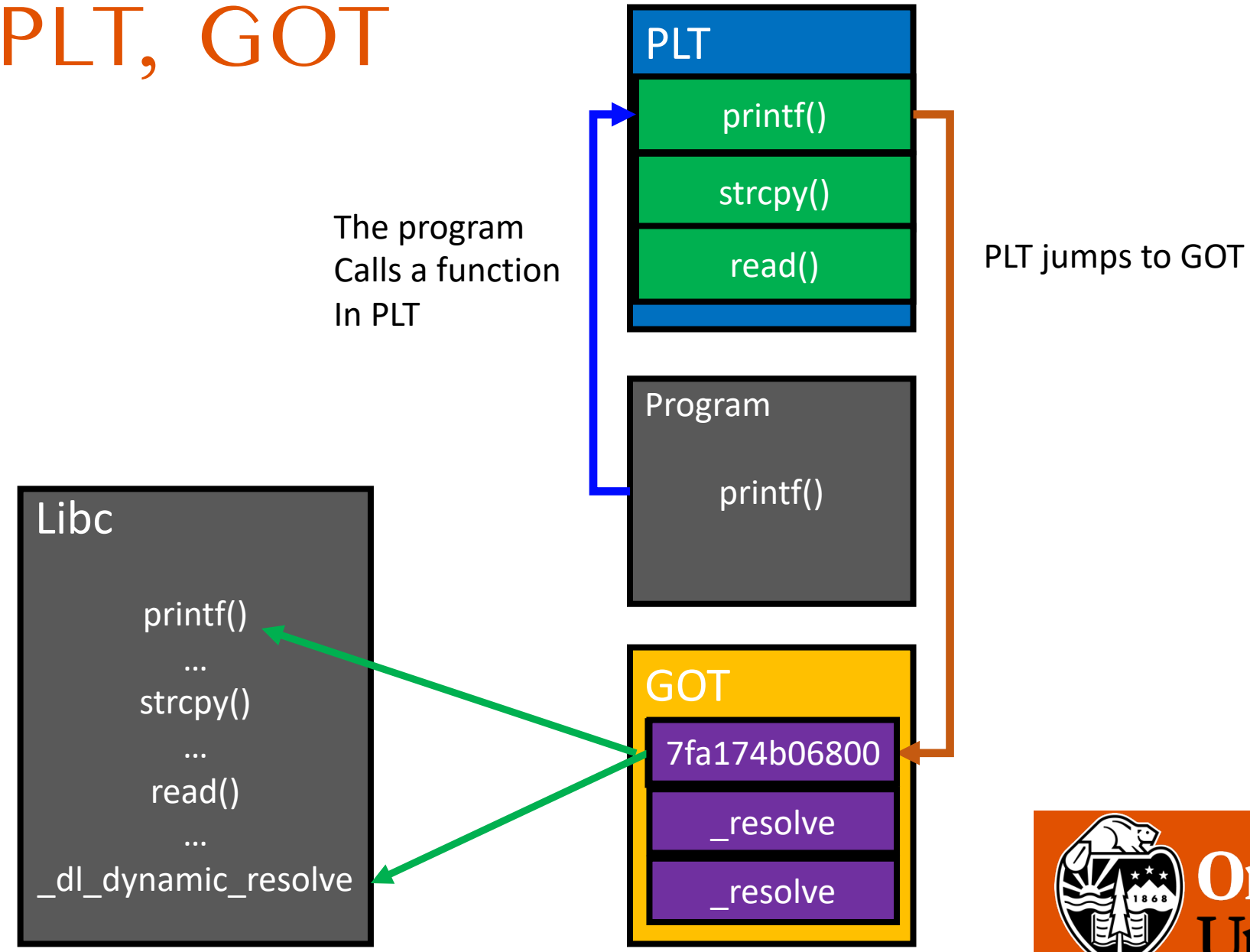


Control-flow Hijacking

- 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

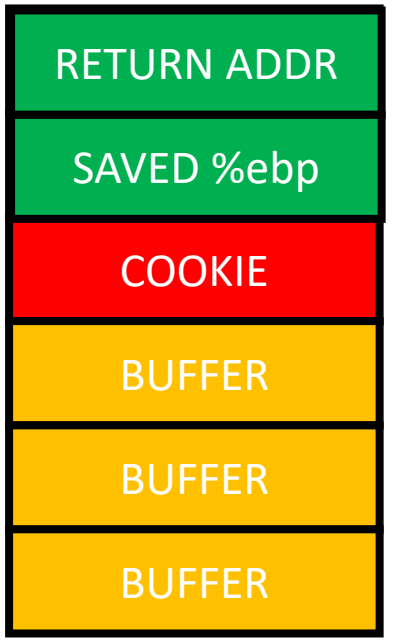


GOT stores addr of _dl_dynamic_resolve

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

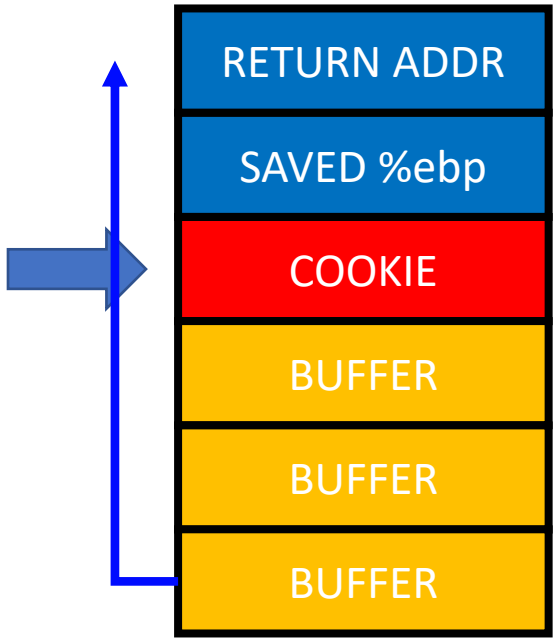
- Example:
  - Read 100 bytes from 0xffffd100 (somewhere in the stack)
  - Read 100 bytes from 0x8048500 (somewhere in the code section)

- 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

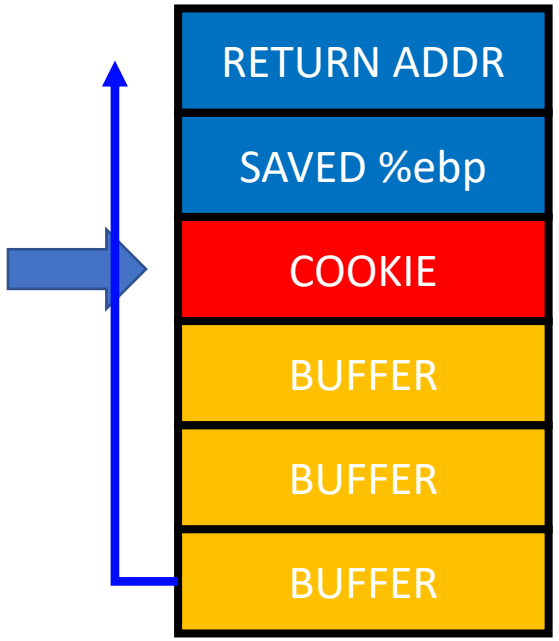
Example of Sequential Read

- 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] 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)
```

Attacking Global Offset Table

- Read the address in GOT
- Leak the address of puts()
- Calculate the address of execve() or system() from the offset

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!
- 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, whatevery 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

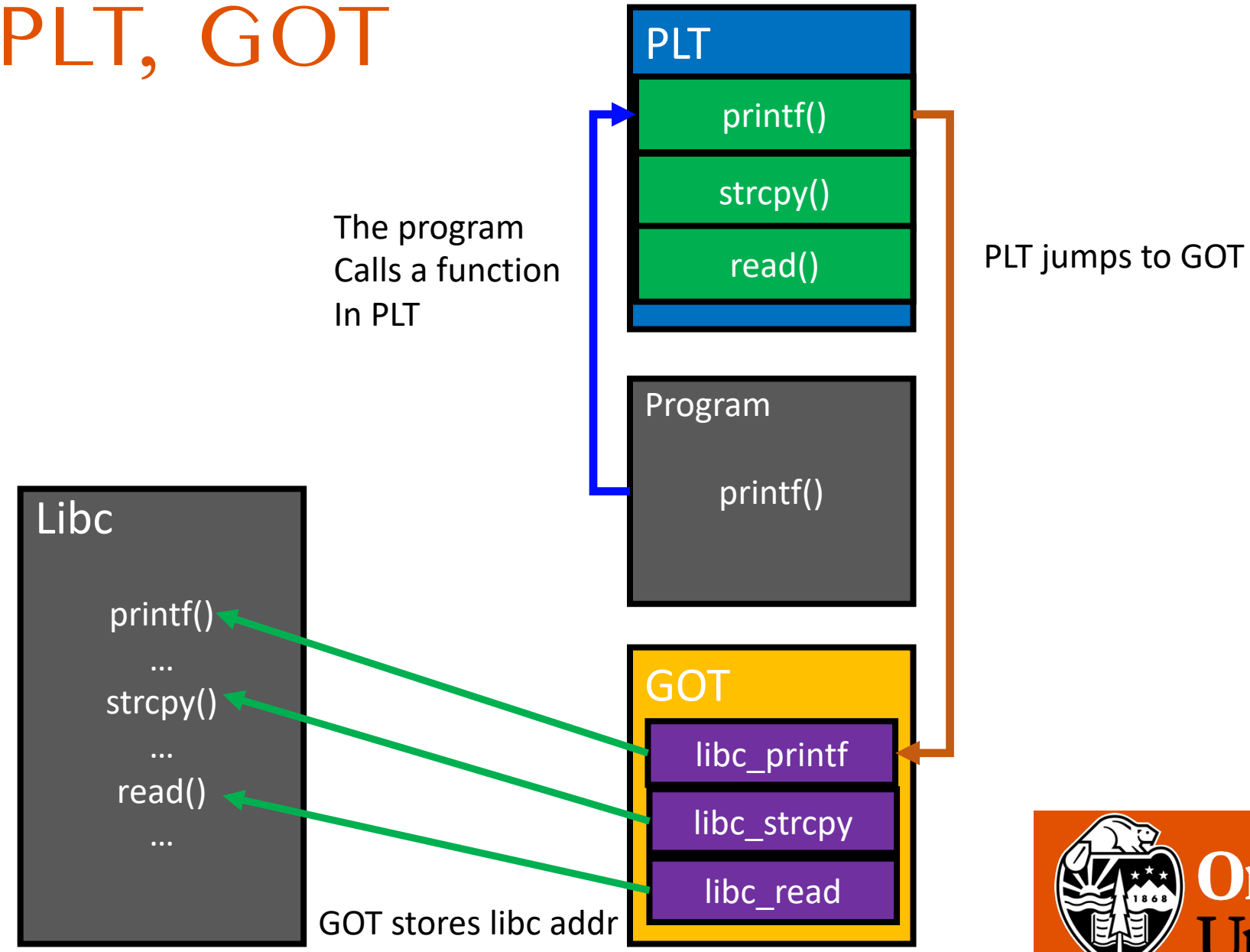
Attacking Global Offset Table

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

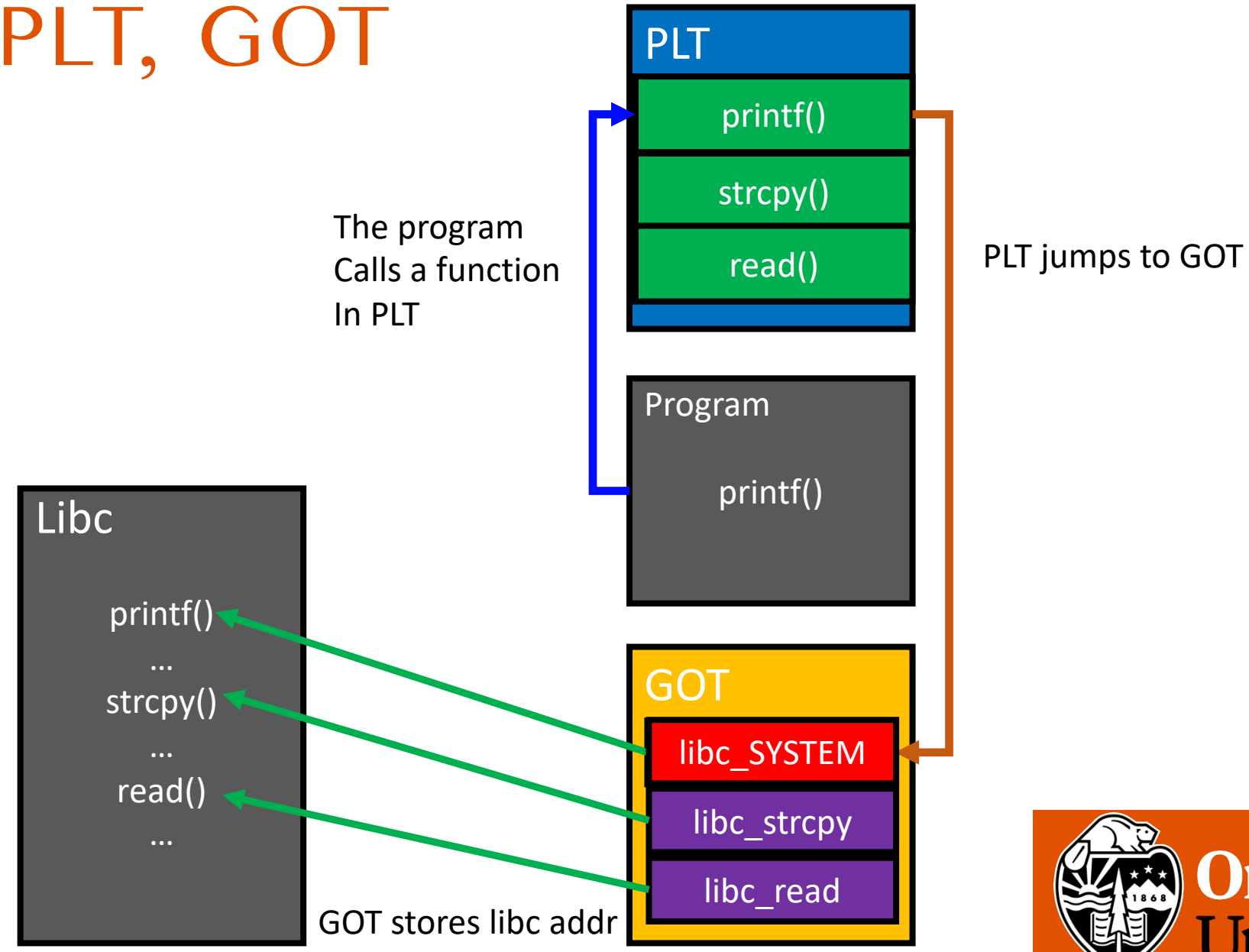
- Write that address system() to printf's GOT

- What will happen a program calls printf?

ELF, PLT, GOT



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”);`

One_Gadget

- Or, you may take a simpler approach

```
blue9057@blue9057-vm-ctf2 /home/labs/week6/3-aw-2 $ ldd aw-2
        linux-vdso.so.1 =>  (0x00007ffd743f1000)
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f14f91b8000)
        /lib64/ld-linux-x86-64.so.2 (0x00007f14f9582000)
blue9057@blue9057-vm-ctf2 /home/labs/week6/3-aw-2 $ one_gadget /lib/x86_64-linux-gnu/libc.so.6
0x45216 execve("/bin/sh", rsp+0x30, environ)
constraints:
  rax == NULL

0x4526a execve("/bin/sh", rsp+0x30, environ)
constraints:
  [rsp+0x30] == NULL

0xf02a4 execve("/bin/sh", rsp+0x50, environ)
constraints:
  [rsp+0x50] == NULL

0xf1147 execve("/bin/sh", rsp+0x70, environ)
constraints:
  [rsp+0x70] == NULL
```

Why one_gadget exists?

- system(command)
  - fork() – parent waits
  - Child – execve(“/bin/sh”, [“sh”, “-c”, command, NULL], environ)

- When we jump to the start of system()
  - Expects 1st arg is the command (either rdi or 1st arg on the stack)
  - Run execve(“/bin/sh”, [“sh”, “-c”, command, NULL], environ)
- When we jump in the middle of system()
  - Skip the part that set up arguments
  - Call execve(“/bin/sh”, rsp+0x30, environ);
  - What if rsp+0x30 == 0?

Why one_gadget exists?

- When we jump in the middle of system()
  - Skip the part that set up arguments
  - Call execve(“/bin/sh”, rsp+0x30, environ);
  - What if rsp+0x30 == 0?
  - What if rax == 0?
  - What if rsp+0x50 == 0?
  - What if rsp+0x70 == 0?
- Use it carefully…

```
blue9057@blue9057-vm-ctf2 /home/labs/week6/3-aw-2 $ ldd aw-2
        linux-vdso.so.1 =>  (0x00007ffd743f1000)
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f14f91b8000)
        /lib64/ld-linux-x86-64.so.2 (0x00007f14f9582000)
blue9057@blue9057-vm-ctf2 /home/labs/week6/3-aw-2 $ one_gadget /lib/x86_64-linux-gnu/libc.so.6
0x45216 execve("/bin/sh", rsp+0x30, environ)
constraints:
  rax == NULL

0x4526a execve("/bin/sh", rsp+0x30, environ)
constraints:
  [rsp+0x30] == NULL

0xf02a4 execve("/bin/sh", rsp+0x50, environ)
constraints:
  [rsp+0x50] == NULL

0xf1147 execve("/bin/sh", rsp+0x70, environ)
constraints:
  [rsp+0x70] == NULL
```

In ASLR-2

```
18 void check_function() {
19   printf("Does these leak some?: %p %p %p %p %p %p %p %p %p %p %p %p %p %p %p\n");
20 }
21
```

- What kind of information this will print???
- 15 values of %p (print hexadecimal number as 0x????????, an addr)
- No arguments…

Format String Vulnerability

- Format String
  - printf(“%d %x %s %p %n\n”, 1, 2, “asdf”, 3, &i);

- The vulnerability
  - char buf[512];
  - printf(“%s”, buf);
  - **printf(buf);**
- If you can control a format string, you may inject arbitrary directives
  - %d %x %p %s %n etc.

Format String Vulnerability

- Format String
  - printf(“%d %x %s %p %n\n”, 1, 2, “asdf”, 3, &i);

- Can be exploited as:
  - Arbitrary read
  - Arbitrary write

The Format String

- Usage
  - printf(“%d %x %s”, 0, 65, “asdf”)
    - -> variable number of arguments
  - This will print 0 (decimal), 41 (hexadecimal), and “asdf”
- % parameters
  - % is a special character in the Format String
  - % seeks for an argument (corresponding to its order…)

Format String Parameters

- %d
  - Expects an integer value as its argument and print a decimal number
- %x
  - Expects an integer value as its argument and print a hexadecimal number
  - 8048000
- %p
  - Expects an integer value as its argument and print a hexadecimal number
  - 0x8048000 (It's pretty!)
- %s
  - Expects an address to a string (char *) and print it as a string

Format String Syntax

- %1$08d
- %[argument_position]$[length][parameter]

- Meaning
  - Print an integer as a decimal value
  - Justify its length to length
  - Get the value from n-th argument

  - Print 8-length decimal integer, with the value at the 1st argument (padded with 0)

Format String Parameters

- %d – Integer decimal %x – Integer hexadecimal %s – String

- printf("%2$08d", 15, 13, 14, "asdf");
  - 00000013
- printf("0x%3$08x", 15, 13, 14, "asdf");
  - 0x0000000d
- printf("%3$20s", 15, 13, 14, "asdf");
- printf("%4$20s", 15, 13, 14, "asdf");
-                asdf

fs-read-1-(32|64)

- Exploit a format string vulnerability to leak pointers from stack
- Guess the random value correctly to get the shell!

```
blue9057@blue9057-vm-ctf2 /home/labs/week6/6-fs-read-2-32 $ ./fs-read-2-32
Please type your name first:
%x
Hello ff932a98

Can you guess the random?
asdf
Wrong, your random was 0x1652bb51 but you typed 0x0000000a
```

fs-read-2-(32|64)

- Exploit a format string vulnerability to leak pointers from stack
- You have limited size for your format string as your input
- Use directives like
  - $100%x
- To skip some uninterested parts of the stack

```
/*
 * On the stack, these variables will be placed in backward:
 * [ret addr]
 * [saved ebp]            <-- %ebp points here
 * [other...]
 * [random - 4 bytes]
 * [name - 64 bytes]
 * [buf - 512 bytes]
 * [other - 4 bytes]
 * [arg space]
 * [arg space]
 * [arg space]            <-- %esp points here...
 *
 * check the disassembly to get a more accurate information
 */
```

Assignment: Week-6

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

- **Due: 11/21 14:00pm**