

**this is the part where ravi
writes the flag on the board**

this slide is here so we don't forget

announcements:

— — —

SQUARECTF we're going hard see
#squarectf in discord

<https://2018.squarectf.com/>

CYPHERCON

APRIL 11/12, 2019 MILWAUKEE

i'm getting a discount code for
\$-25, do not buy tickets yet

buy the digital badge it's super
cool

Format String Vulnerabilities

When output becomes input

Goal: Leak & Modify Stack

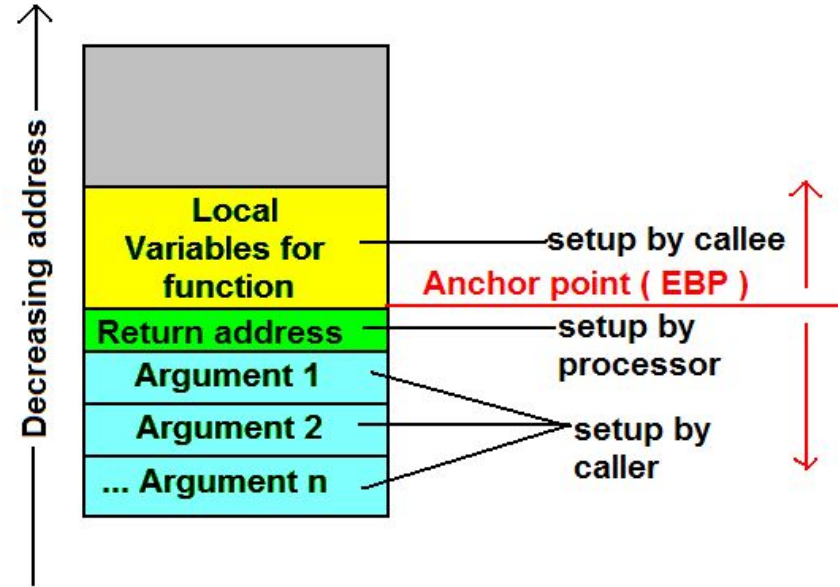
Leak & Modify

Leak: Local variables, arguments

Modify: Program execution (vars, or return addresses)

Review: The Stack

- Last in, first out
- Local variables
- Linkage



Printf's Perspective

- Doesn't know what arguments will be
- Expects a "format string" to tell it what to do

Example:

```
printf("%d %d %d", 1, 2, 3);
```

1 2 3

Format String "%d %d %d"
<u>Printf argument 1</u>
<u>Printf argument 2</u>
<u>Printf argument 3</u>
<u>Local vars from calling</u> function
...
Linkage (Return address contained here!)
Previous stack frame

Format String Examples

```
printf("Hello World!");
```

```
printf("This is a newline: \n");
```

```
printf("This is an integer argument: %d", 5);
```

```
printf("This is an integer in hex: %x", 5);
```

```
printf("This is a character arg: %c", 'a');
```

```
printf("This is a string arg: %s", "This is a string");
```


Format String Syntax

`%d`: Print the next thing on the stack as an integer

`%x`: Print the next thing on the stack as hex

`%c`: Print the next thing on the stack as an ASCII character

`%s`: Print the next thing as a string

(Note: for `%s` the string's starting address must be passed, not the entire string)

Exploit

- What's an argument? What is just on the stack?
- Same format string, different call, very different output:

```
printf ("%d %d %d", 1);
```

```
1 [1st local var] [2nd local var]
```

Format String "%d %d %d"
<u>Printf</u> argument 1
<u>Printf</u> argument 2
<u>Printf</u> argument 3
Local <u>vars</u> from calling function
...
Linkage (Return address contained here!)
Previous stack frame

Typical Exploit

1. Find a call to `printf` using user input as format string
2. Create malicious format string to leak or modify data
3. Profit?

sigpwny.com
Challenge 1:
Leak a stack variable

Challenge 1

Vulnerable call:

```
printf(input_buffer);
```

%x gives us what looks like an address... Possibly a string?

Format String
Local variable 1 (flag)
Remaining Local variables
...

sigpwny.com

Challenge 2:

Leak a variable far down the stack

Challenge 2

Vulnerable call:

```
printf(input_buffer);
```

The flag is further down the stack... Use multiple format specifiers to see more?

Format String
Local variable 1
Local variable 2
Local variable 3
...
Flag!
...

What if the buffer is too small?

`%(number)$x` will print the “number”-th argument.

Example:

```
printf(“%3$s”, “arg 1”, “arg 2”, “arg 3”);
```

Will print “arg 3,” since “arg 3” is the 3rd argument.

What if the buffer is too small?

`%(number)$x` will print the “number”-th **thing on the stack**.

Example:

```
printf(“%4$x”, 0x1, 0x2, 0x3);
```

Will print whatever was pushed on the stack prior to printf.

sigpwny.com

Challenge 3:

Leak a variable with a small format string

Challenge 3

The variable we want is at an offset of 11 from the format string.

If we could do %x 11 times, we would get the flag.

How can we effectively do %x more times using only 6 characters?

Format String
Local variable 1
Local variable 2
Local variable 3
...
Flag!
...

Modifying Contents

— — —

`%n`: Writes the number of characters output to an address passed on the stack.

Example:

```
int x;
```

```
printf("12345%n", &x);
```

Stores '5' in x, since 5 characters were written.

Modifying Contents

— — —

`%n` can be used to overwrite any address on the stack with new information!

sigpwny.com
Challenge 4:
Modify a variable

Challenge 4

- `dont_modify_me`'s address is on the stack and can be viewed using `%6$x`
- Using `%n` instead of `%x` will overwrite `dont_modify_me`'s value with the number of bytes written