printf in C

printf: print a format string to the standard output (screen).

- Format string: a string with special format specifiers (escape sequences prefixed with `%')
- **printf** can take more than one argument. The first argument is the format string; the rest consist of values to be substituted for the format specifiers.

Examples.

- `printf("Hello, World");`

  ```
  Hello, World
  ```

- `printf("Year %d", 2014);`

  ```
  Year 2014
  ```

- `printf("The value of pi: %f", 3.14);`

  ```
  The value of pi: 3.140000
  ```

- `printf("The first character in %s is %c", "abc", 'a');`

  ```
  The first character in abc is a
  ```

Format String

Format	Output	Example
d *or* i	Signed decimal integer	392
u	Unsigned decimal integer	7235
o	Unsigned octal	610
x	Unsigned hexadecimal integer	7fa
X	Unsigned hexadecimal integer (uppercase)	7FA
f	Decimal floating point, lowercase	392.65
F	Decimal floating point, uppercase	392.65
e	Scientific notation (mantissa/exponent), lowercase	3.9265e+2
E	Scientific notation (mantissa/exponent), uppercase	3.9265E+2
g	Use the shortest representation: %e or %f	392.65
G	Use the shortest representation: %E or %F	392.65
a	Hexadecimal floating point, lowercase	-0xc.90fep-2
A	Hexadecimal floating point, uppercase	-0XC.90FEP-2
c	Character	a
s	String of characters	sample
p	Pointer address	B8000000
n	Nothing printed. The corresponding argument must be a pointer to a signed int. The number of characters written so far is stored in the pointed location.	

Security Vulnerability in Format String

Escape sequences are essentially instructions.

- `printf` has no idea how many arguments it actually receives.
- It infers the number of arguments from the format string: number of arguments should match number of escape sequences in the format string.
- What if there is a mismatch?

A vulnerable program

- Users control both escape sequences and arguments in `user_input`.
- An attacker can deliberately cause mismatch between them.

```
#include <stdio.h>
#include <string.h>

int main(int argc, char* argv[]) {
  char user_input[100];
  scanf("%s", user_input);
  printf(user_input);
}
```

What potential consequences could this mismatch cause?

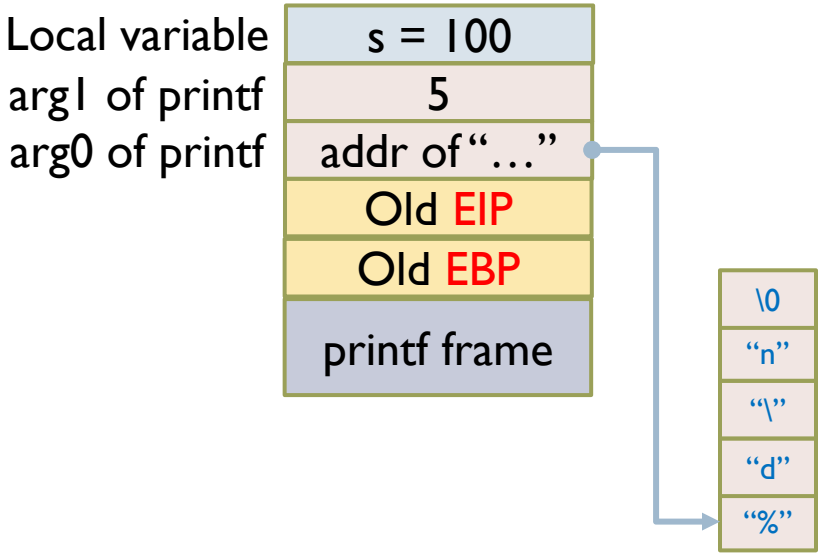
Attack 1: Leak Information from Stack

Correct usage of **printf**

- Two arguments are pushed into the stack as function parameter

```
#include <stdio.h>
#include <string.h>

int main(int argc, char **argv){
    int s = 100;
    printf("%d\n", 5);
    return 0;
}
```



Attack 1: Leak Information from Stack

Incorrect usage of `printf`

- The stack does not realize an argument is missing, and will retrieve the local variable as the argument to print out.
- Data that do not belong to the user are thus leaked to the attacker.
- The attacker can print out any types of data, including integer (**%d**), floating point (**%f**), string (**%s**), address (**%p**)…

```
#include <stdio.h>
#include <string.h>

int main(int argc, char **argv){
    int s = 100;
    printf("%d\n");
    return 0;
}
```