

Attack 1: Leak Information from Stack

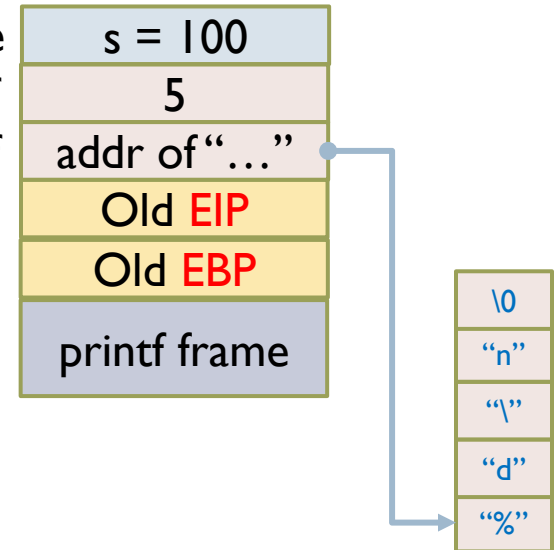
Correct usage of `printf`

- 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;
}
```

Local variable
arg1 of printf
arg0 of printf



Attack 1: Leak Information from Stack

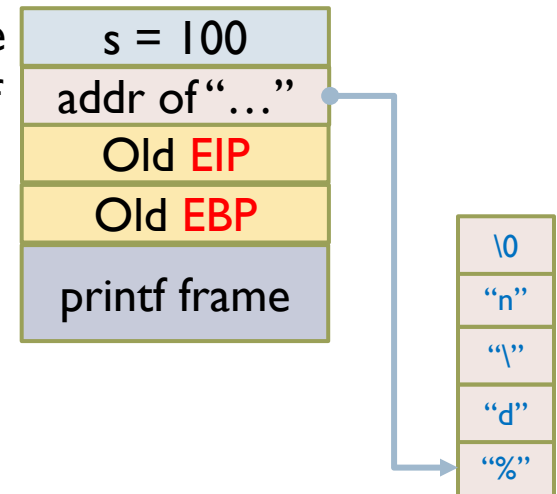
Incorrect usage of `printf`

- ▶ 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;
}
```

Local variable
arg0 of printf



Attack 2: Crash the Program

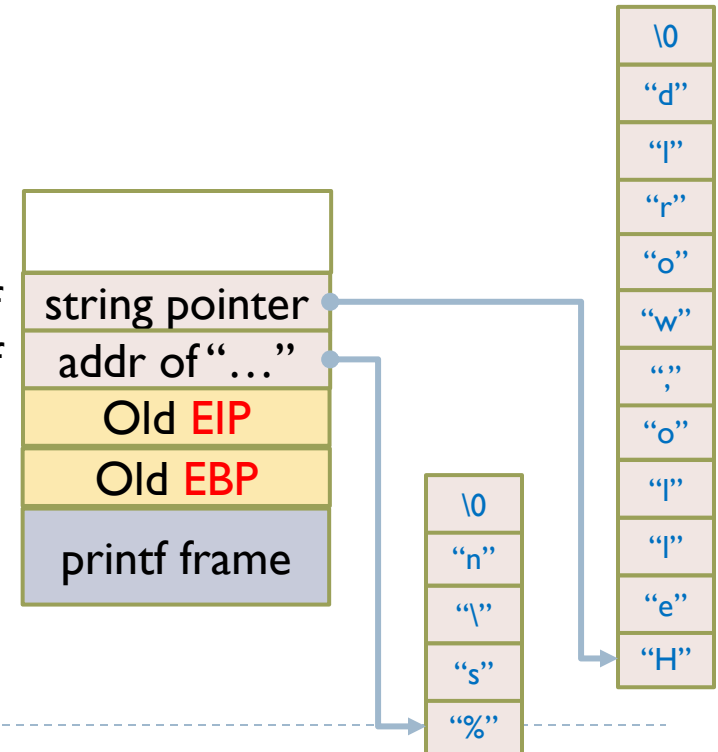
Correct usage of `printf`

- ▶ For format specifier `%s`, a pointer of a string is pushed into the stack as the corresponding function parameter

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

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

arg1 of printf
arg0 of printf



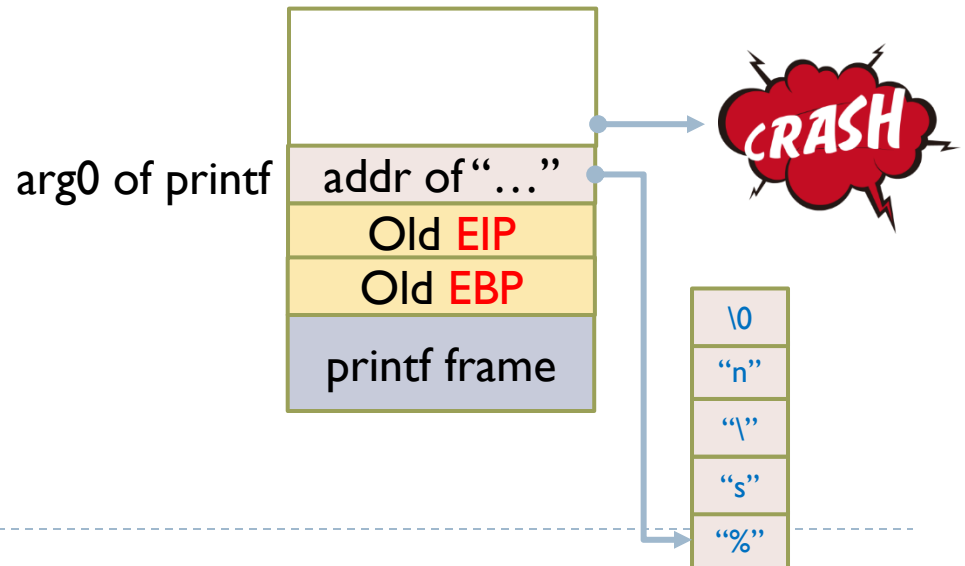
Attack 2: Crash the Program

Incorrect usage of `printf`

- ▶ The stack does not realize an argument is missing, and will retrieve other stack values as addresses, and access data there.
- ▶ This address can be invalidated, and then the program will crash
 - No physical address is assigned to this address
 - Protected memory, e.g., kernel.
- ▶ Can include more `%s` to increase the crash probability

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

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



Attack 3: Modify the Memory

Correct usage of `printf`

- ▶ For format specifier `%n`, a pointer of a signed integer is pushed into the stack as the corresponding function parameter.
- ▶ Store the number of characters written so far into that integer

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

int main(int argc, char **argv){
    int *x = (int *)malloc(sizeof(int));
    printf("abcdefg%n\n",x);
    return 0;
}
```

arg1 of printf
arg0 of printf

