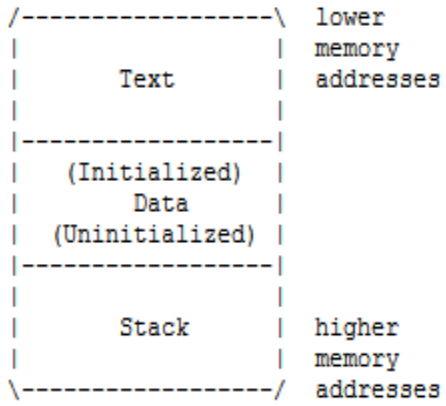
Basic Buffer Overflow

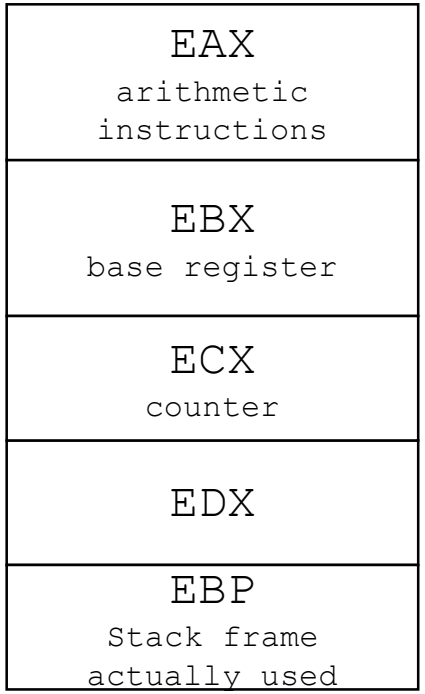
Memory management



↑ Lower addresses (0x08000000)
Shared libraries
.text
.bss
Heap (grows ↓)
Stack (grows ↑)
env pointer
Argc
↓ Higher addresses (0xbfffffff)

CPU registers

General purpose registers



CPU registers

General purpose

stack pointer (top of the stack)

instruction pointer

Buffers

- A buffer is defined as a limited, contiguously allocated set of memory. The most common buffer in C is an array
- Buffer overflows: no inherent bounds-checking exists on buffers
- The C language and its derivatives do not have a built-in function to ensure that data being copied into a buffer will not be larger than the buffer can hold.
- The program has to be explicitly coded to check for oversized input

The Stack

LIFO data structure

Last in First Out

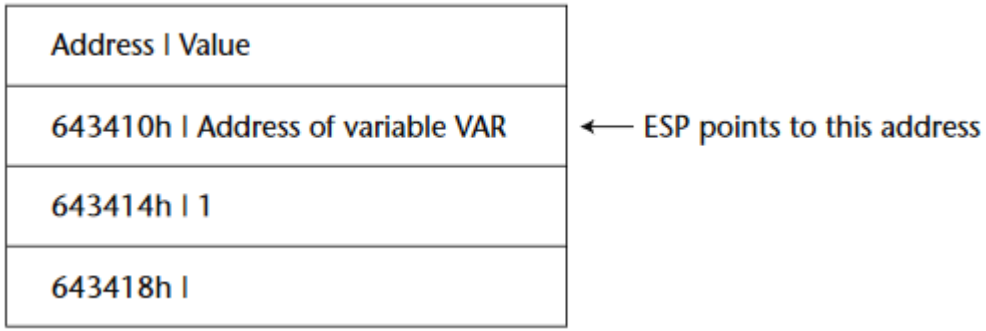
Stack-specific instructions

PUSH and POP

use ESP to know where the stack is in memory

The Stack

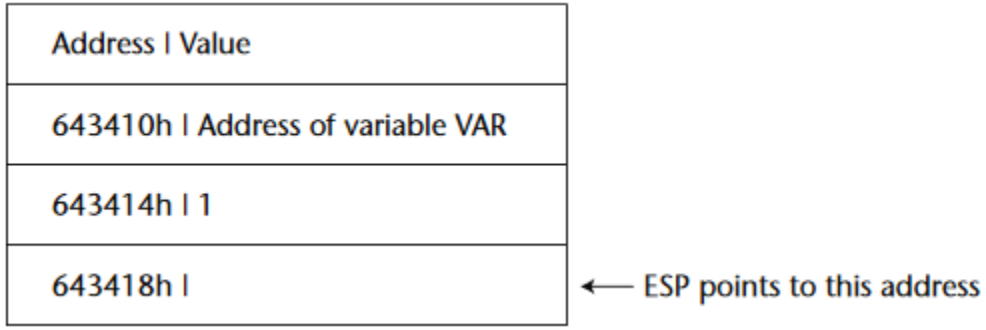
```
push 1
push addr var
```



The ESP register will point to the top of the stack, address 643410h

The Stack

```
pop eax
pop ebx
```

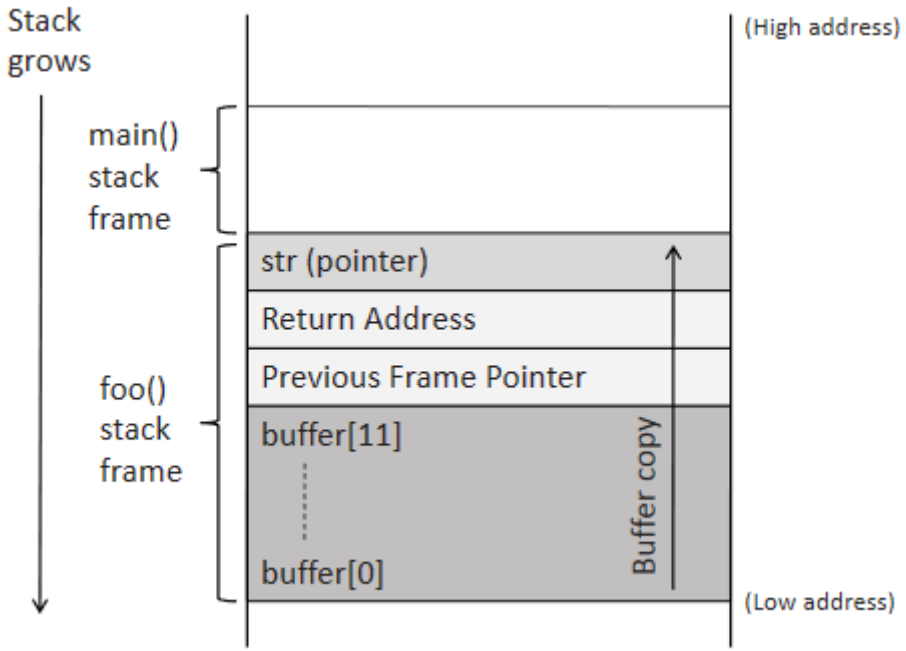


Buffer Overflow

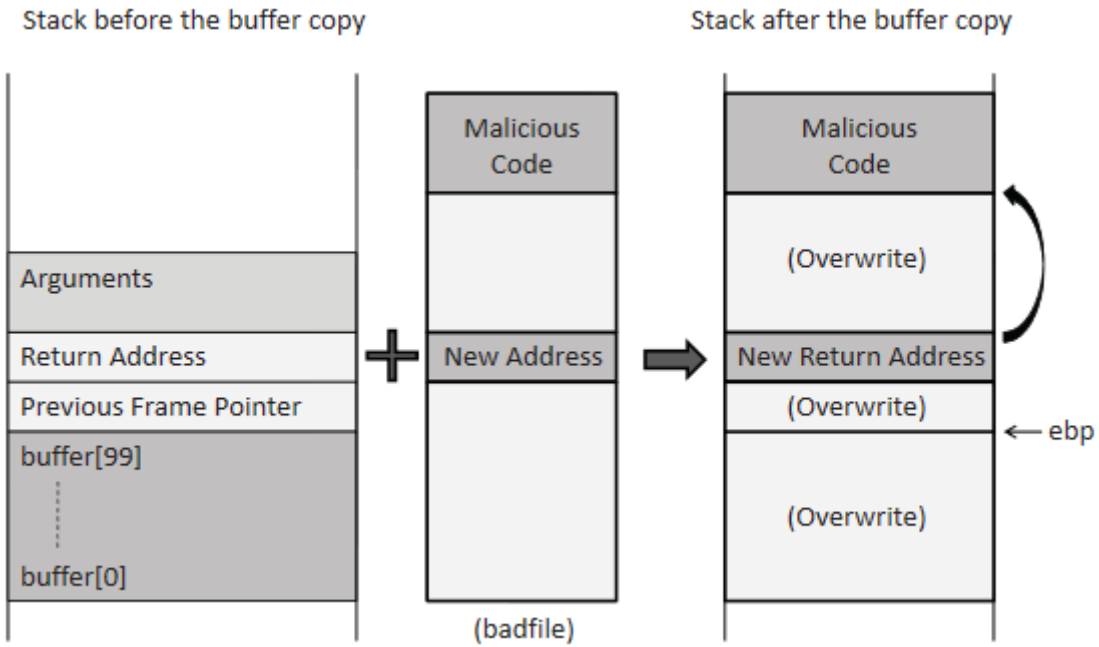
```
#include <string.h>
void foo(char *str){
        char buffer[12];
        /*The following statement will result in buffer overflow*/
        strcpy(buffer, str);
}
int main(){
        char *str = "This is definitely longer than 12";
        foo(str);
        return 1;
}
```

The local array buffer[]in foo()has 12 bytes of memory. The foo()function uses strcpy()to copy the string from str to buffer[]. The strcpy()function does not stop until it sees a zero (a number zero,'\0') in the source string. Since the source string is longer than 12 bytes, strcpy()will overwrite some portion of the stack above the buffer. This is called buffer overflow.

Buffer Overflow



Buffer Overflow



Buffer Overflow