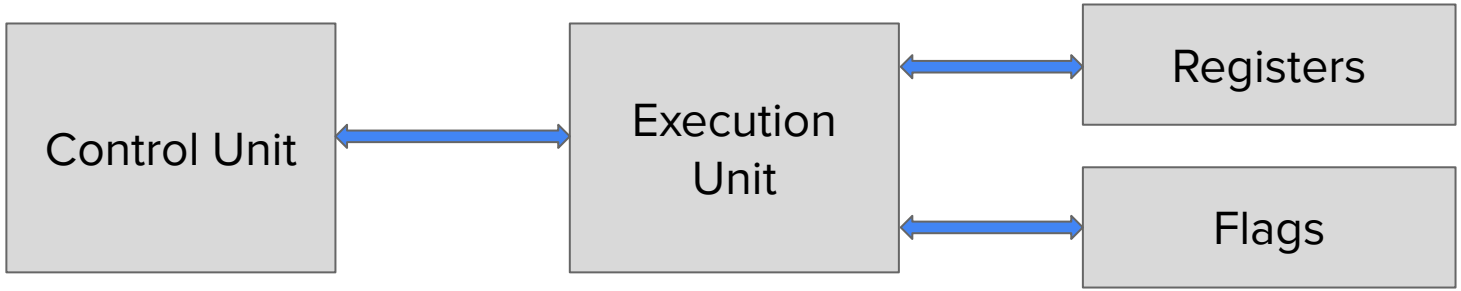
Buffer Overflow for OSCP

D3crypt

Please Subscribe :)

Basics First (reference: http://www.securitytube.net/)

Inside CPU



Control Unit - Retrieve / Decode instructions, Retrieve /Store data in memory

Execution Unit - Execution of instructions

Registers - Internal memory locations used as “variables”

Flags - Used to indicate various “events” when execution is happening

Basics First Contd... (reference: http://www.securitytube.net/)

General Purpose Registers:

EAX,EBX,ECX,EDX,ESI,EDI,ESP,EBP

Segment Registers: //informational

CS, DS, SS, ES, FS, GS

Instruction Pointer Register:

EIP

Control Registers: //informational

CR0, CR1, CR2, CR3, CR4

Basics First Contd... (reference: http://www.securitytube.net/)

General Purpose Register:

ESP - Stack Pointer Register: Always points to the top of the stack

EBP - Stack Data Pointer Register: Typically points to base of previous stack frame

Instruction Pointer Register:

EIP - The address of next instruction to execute is stored here

Let's talk about Memory

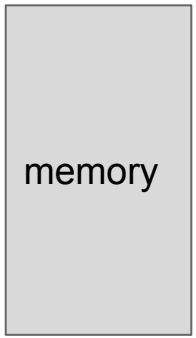
A program, when executed, will become a "process"

The process will get a **space allocated in memory** where it runs

The cool thing about this is that the process get a "**virtual**" memory space. The memory space will have a **start address** and **end address**

end address *0xFFFFFFFF*

{ address range is just for an example}



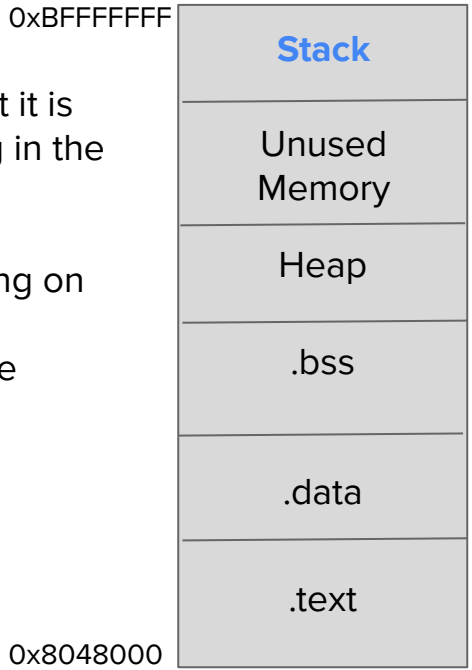
start address *0x00000000*

So what does 'virtual' mean?

Let's talk about Memory Contd...

Virtual Memory Space

- Every process thinks that it is the only process running in the system
- Regardless of where the process is actually running on the physical memory, all processes will have same **virtual** memory space



Function arguments and local variables are stored in stack

Dynamic Memory

Uninitialized Data

Initialized Data

Actual program code (low level assembly instructions)

Let's talk about Stack

L Last

I in

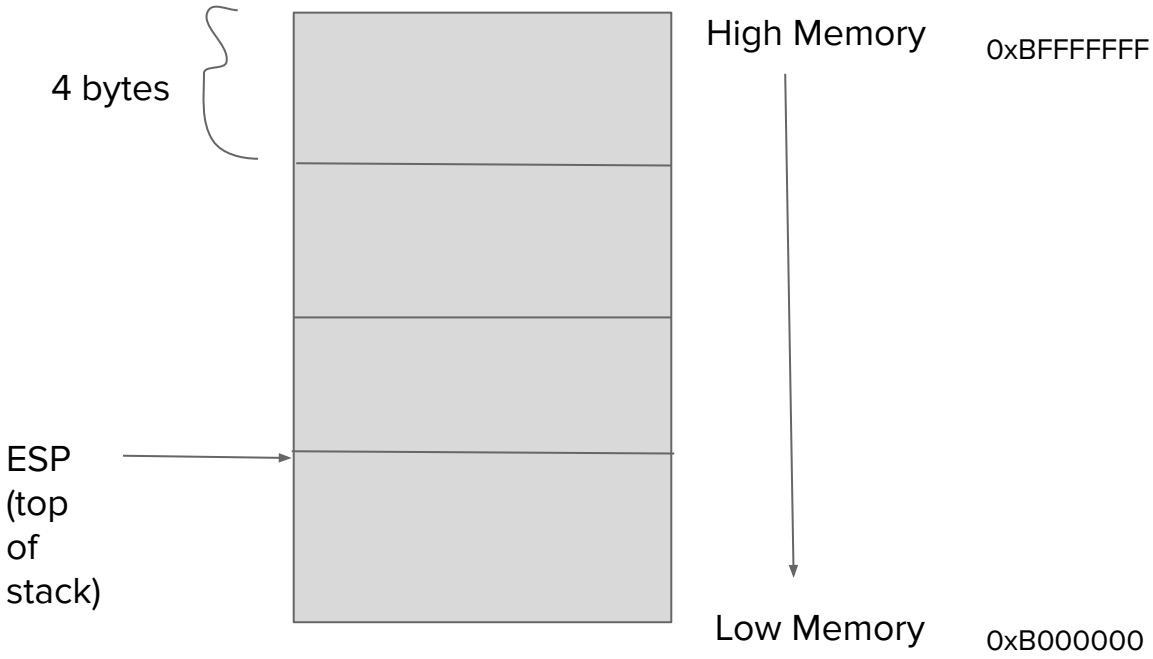
F first

O out

Operations:

PUSH

POP

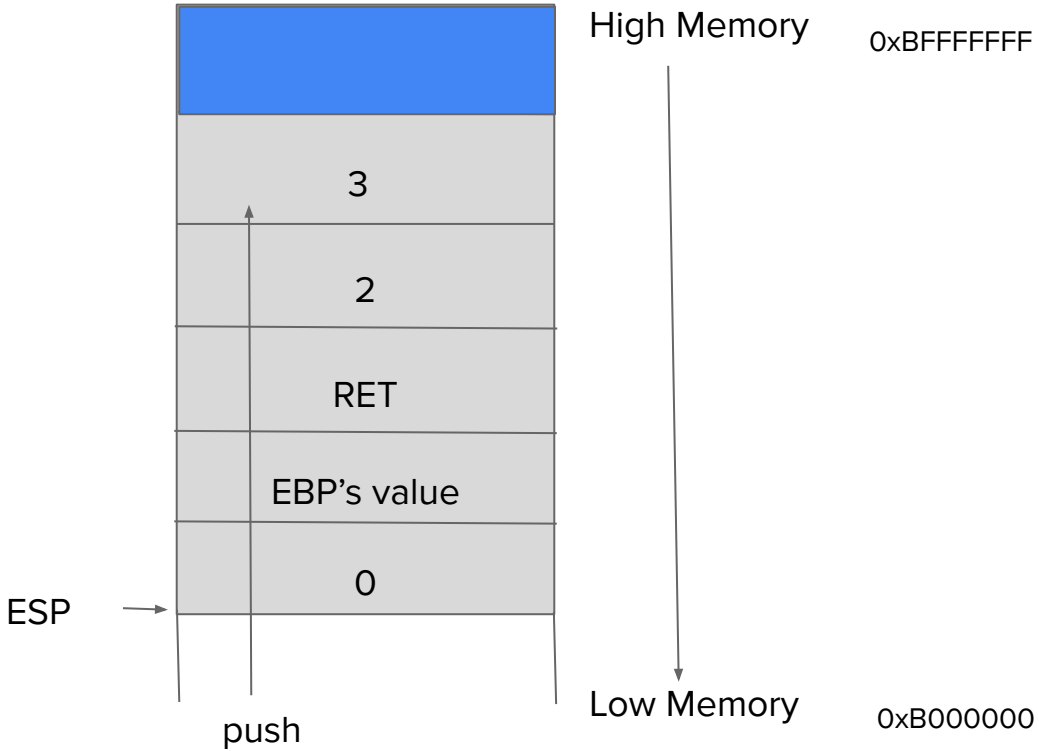


Let's talk about Stack Contd...

Operations:

- PUSH
- POP

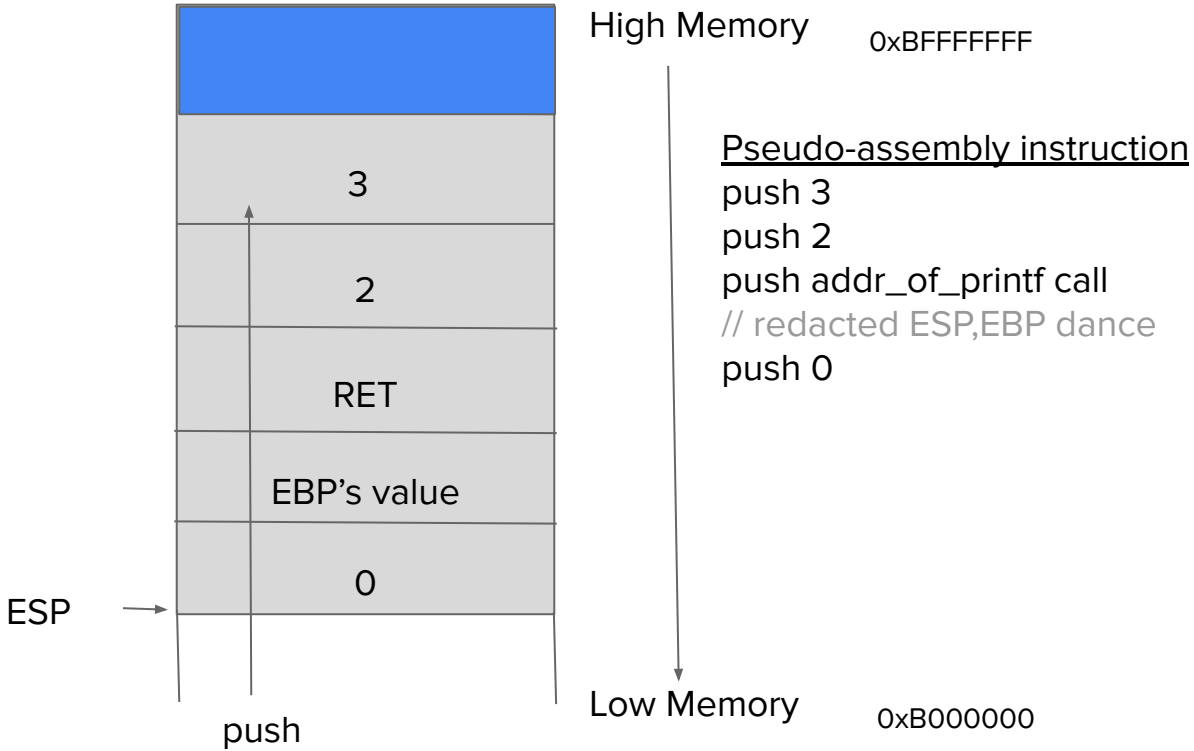
```
add(x,y)
{
int z=0; // local var
z=x+y;
return z;
}
add(2,3)
printf("done bye!");
```



Let's talk about Stack Contd...

Code Explained:

```
add(x,y)
{
int z=0; // local var
z=x+y;
return z;
}
add(2,3)
printf("done bye!");
```

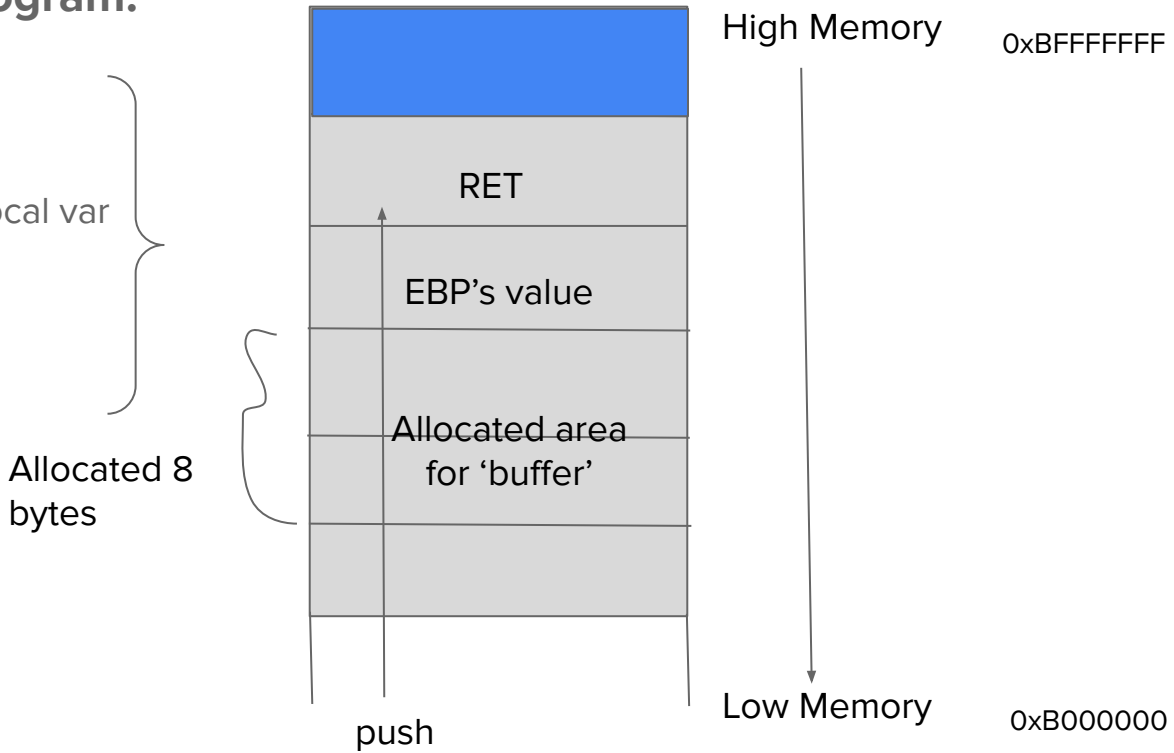


Pseudo-assembly instruction
push 3
push 2
push addr_of_printf call
// redacted ESP,EBP dance
push 0

How Buffer Overflow Works?

Vulnerable C Program:

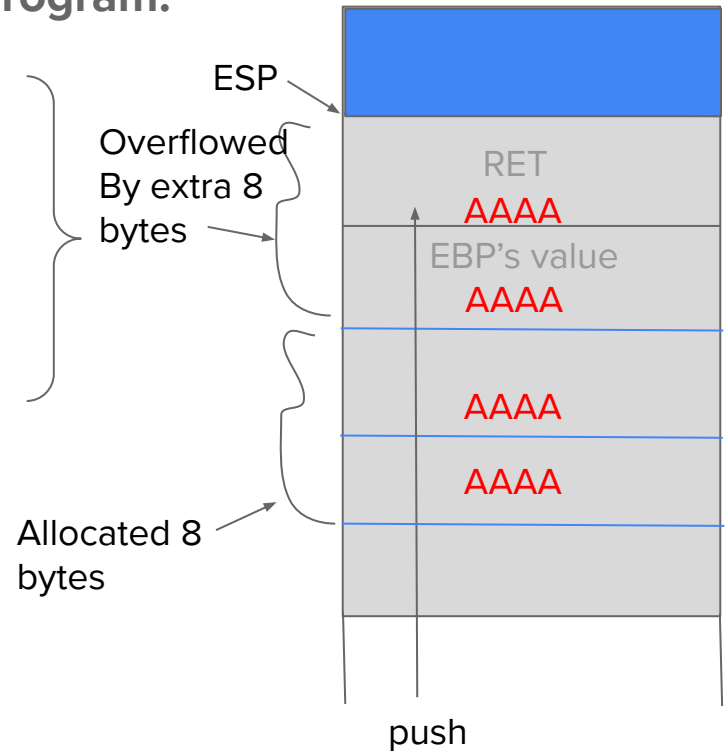
```
GetInput()
{
    char buffer[8]  // local var
    gets(buffer);
    puts(buffer);
}
GetInput()
```



Let's Buffer Overflow

Vulnerable C Program:

```
GetInput()
{
    char buffer[8]
    gets(buffer);
    puts(buffer);
}
GetInput()
```



High Memory 0xBFFFFFFF

Low Memory 0xB000000

Evil Input

AAAAAAAAAAAAAAAA

Program flow:

1. Hacker enter 16 A's
2. Program copies A's in the stack (during 'gets' execution) and overwrites our RET box
3. After executing GetInput() block, the program takes whatever in RET box and copies to EIP register
4. The program increments ESP pointer address so that it will point to the area after RET box which here is start of blue box

Demo

Questions? Please leave a comment !