**RE lab 07 - Mitigations and bypasses**

**Lab files and setup**

Download the lab files from [here](https://pwnthybytes.ro/unibuc_re/07-lab-files.zip). The archive password is infected.

* Install one\_gadget from [here](https://github.com/david942j/one_gadget)

**extra gdb commands**

* Investigating the protections of a binary

gdb-peda$ checksec

CANARY : disabled # No stack cookie

FORTIFY : disabled

NX : ENABLED # Code is not modifiable, data is not executable

PIE : ENABLED # The binary is position independent

RELRO : FULL # The GOT table cannot be modified

**Working with a PIE binary (when stripped)**

Unfortunately, gdb has some bugs when setting breakpoints for a stripped PIE binary:

$ gdb ./task3

Reading symbols from ./task3...(no debugging symbols found)...done.

gdb-peda$ start

No unwaited-for children left.

Aborted (core dumped)

$ gdb ./task3

Reading symbols from ./task3...(no debugging symbols found)...done.

gdb-peda$ b \*0x1338

Breakpoint 1 at 0x1338

gdb-peda$ run

Starting program: /ctf/unibuc/curs\_re/curs\_07/lab\_07/task\_03/task3

Warning:

Cannot insert breakpoint 1.

Cannot access memory at address 0x1338

The recommended way to set breakpoints is to disable ASLR system-wide:

echo 0 | sudo tee /proc/sys/kernel/randomize\_va\_space

Next, run the program once and get the base of the executable and set breakpoints relative to it:

gdb-peda$ vmmap

Start End Perm Name

0x0000555555554000 0x0000555555555000 r--p /ctf/unibuc/curs\_re/curs\_07/lab\_07/task\_03/task3

0x0000555555555000 0x0000555555556000 r-xp /ctf/unibuc/curs\_re/curs\_07/lab\_07/task\_03/task3

0x0000555555556000 0x0000555555557000 r--p /ctf/unibuc/curs\_re/curs\_07/lab\_07/task\_03/task3

0x0000555555557000 0x0000555555558000 r--p /ctf/unibuc/curs\_re/curs\_07/lab\_07/task\_03/task3

Note 0x0000555555555000, the .text section

gdb-peda$ b \*0x0000555555554000 + 0x1338

Breakpoint 2 at 0x555555555338

gdb-peda$ run

Starting program: /ctf/unibuc/curs\_re/curs\_07/lab\_07/task\_03/task3

....

Breakpoint 2, 0x0000555555555338 in ?? ()

gdb-peda$ p $rip

$1 = (void (\*)()) 0x555555555338

**Working with a PIE binary (when not stripped)**

Breakpoints for relative addresses do not work (as seen in gdb before starting the executable or in IDA):

$ gdb ./task01

Reading symbols from ./task01...

(No debugging symbols found in ./task01)

gdb-peda$ pdis main

Dump of assembler code for function main:

0x0000000000001253 <+0>: push rbp

0x0000000000001254 <+1>: mov rbp,rsp

0x0000000000001257 <+4>: mov eax,0x0

0x000000000000125c <+9>: call 0x11c0 <setup>

0x0000000000001261 <+14>: mov eax,0x0

0x0000000000001266 <+19>: call 0x11ef <vuln>

0x000000000000126b <+24>: mov eax,0x0

0x0000000000001270 <+29>: pop rbp

0x0000000000001271 <+30>: ret

End of assembler dump.

gdb-peda$ b \*0x1271

Breakpoint 1 at 0x1271

gdb-peda$ run

Starting program: task01

Warning:

Cannot insert breakpoint 1.

Cannot access memory at address 0x1271

Breakpoints relative to known symbols work even under ASLR

$ gdb -q ./task01

Reading symbols from ./task01...

(No debugging symbols found in ./task01)

gdb-peda$ pdis main

Dump of assembler code for function main:

0x0000000000001253 <+0>: push rbp

0x0000000000001254 <+1>: mov rbp,rsp

0x0000000000001257 <+4>: mov eax,0x0

0x000000000000125c <+9>: call 0x11c0 <setup>

0x0000000000001261 <+14>: mov eax,0x0

0x0000000000001266 <+19>: call 0x11ef <vuln>

0x000000000000126b <+24>: mov eax,0x0

0x0000000000001270 <+29>: pop rbp

0x0000000000001271 <+30>: ret

End of assembler dump.

gdb-peda$ set disable-randomization off

gdb-peda$ b \*main+30

Breakpoint 1 at 0x1271

gdb-peda$ run

Starting program: /work/unibuc/curs\_re/curs\_07/lab\_07/pack/task01

[============== Task 1 ==============]

What is your name?

1234

Hello there, 1234

!

[----------------------------------registers-----------------------------------]

RAX: 0x0

RBX: 0x0

RCX: 0x7f1dc64856e0 (<\_\_write\_nocancel+7>: cmp rax,0xfffffffffffff001)

RDX: 0x7f1dc6754780 --> 0x0

RSI: 0x7fff987901e0 ("Hello there, 1234\n!\n")

RDI: 0x1

RBP: 0x55915b47c280 (<\_\_libc\_csu\_init>: push r15)

RSP: 0x7fff98792908 --> 0x7f1dc63af830 (<\_\_libc\_start\_main+240>: mov edi,eax)

RIP: 0x55915b47c271 (<main+30>: ret)

R8 : 0x7f1dc6979700 (0x00007f1dc6979700)

R9 : 0x14

R10: 0x5

R11: 0x246

R12: 0x55915b47c090 (<\_start>: xor ebp,ebp)

R13: 0x7fff987929e0 --> 0x1

R14: 0x0

R15: 0x0

EFLAGS: 0x202 (carry parity adjust zero sign trap INTERRUPT direction overflow)

[-------------------------------------code-------------------------------------]

0x55915b47c25c <main+9>: call 0x55915b47c1c0 <setup>

0x55915b47c261 <main+14>: mov eax,0x0

0x55915b47c266 <main+19>: call 0x55915b47c1ef <vuln>

0x55915b47c26b <main+24>: mov eax,0x0

0x55915b47c270 <main+29>: pop rbp

=> 0x55915b47c271 <main+30>: ret

0x55915b47c272: nop WORD PTR cs:[rax+rax\*1+0x0]

0x55915b47c27c: nop DWORD PTR [rax+0x0]

0x55915b47c280 <\_\_libc\_csu\_init>: push r15

0x55915b47c282 <\_\_libc\_csu\_init+2>: mov r15,rdx

0x55915b47c285 <\_\_libc\_csu\_init+5>: push r14

0x55915b47c287 <\_\_libc\_csu\_init+7>: mov r14,rsi

[------------------------------------stack-------------------------------------]

0000| 0x7fff98792908 --> 0x7f1dc63af830 (<\_\_libc\_start\_main+240>: mov edi,eax)

0008| 0x7fff98792910 --> 0x1

0016| 0x7fff98792918 --> 0x7fff987929e8 --> 0x7fff987933fc ("/work/unibuc/curs\_re/curs\_07/lab\_07/pack/task01")

0024| 0x7fff98792920 --> 0x1c697dca0

0032| 0x7fff98792928 --> 0x55915b47c253 (<main>: push rbp)

0040| 0x7fff98792930 --> 0x0

0048| 0x7fff98792938 --> 0x573f0e24e3051cf4

0056| 0x7fff98792940 --> 0x55915b47c090 (<\_start>: xor ebp,ebp)

0064| 0x7fff98792948 --> 0x7fff987929e0 --> 0x1

0072| 0x7fff98792950 --> 0x0

0080| 0x7fff98792958 --> 0x0

0088| 0x7fff98792960 --> 0x3e2885934251cf4

[------------------------------------------------------------------------------]

Legend: code, data, rodata, value

Breakpoint 1, 0x000055915b47c271 in main ()

gdb-peda$

**Tasks**

**Task 1**

* Identify the binary protections and the helper function (which spawns a shell) **(1p)**
* Is the binary stripped? What approach is needed for breakpoints?
* Set a breakpoint on the return address in the vulnerable function
  + run a couple of times with ASLR
  + for each run, observe the raw 8 bytes of the (overwriteable) return address
  + for each run, also observe the raw 8 bytes of the target (helper function) address
  + how many bytes differ between the (overwriteable) return address and the target (helper function) address **(2p)**
  + calculate the probability that a partial overwrite of the return address succeeds
* Exploit the vulnerability by doing a partial overwrite of the return address. Remote end: 45.76.91.112 10071 **(3p)**

**Task 2**

* Identify the binary protections **(1p)**
* Is the binary stripped? What approach is needed for breakpoints?
* Analyze the binary in IDA. What is the vulnerability present? What can it be used for? **(2p)**
* Use the one\_gadget tool to find a couple of offsets into libc for shell spawn
* Scan the GOT table to see which of those addresses differs the least (less bytes to overwrite => less failed tries) and calculate the probability here as well **(1p)**
  + Use the same approach as in task 1
* Exploit the vulnerability locally and remotely. Remote end: 45.76.91.112 10072 **(2p)**

**Task 3:**

* Identify the binary protections and the vulnerability **(1p)**
* Is the binary stripped? What approach is needed for breakpoints?
* Read the stack cookie and the libc address (return address for main) using the format string vulnerability
* Exploit the vulnerability locally and remotely. Remote end: 45.76.91.112 10073 **(3p)**