**RE lab 05 - Stack frames 101**

**Lab files and setup**

Download the lab files from [here](https://pwnthybytes.ro/unibuc_re/05-lab-files.zip). The archive password is infected.

* For the Linux task, run:

*$ apt-get install gdb git*

*$ cd*

*$ git clone https://github.com/longld/peda*

*$ echo "source ~/peda/peda.py" >> ~/.gdbinit*

**pwntools documentation**

* Opening processes/sockets and programmatic communication. [Link](https://github.com/Gallopsled/pwntools-tutorial/blob/master/tubes.md)
* Packing, unpacking bytes, file I/O, etc [Link](https://github.com/Gallopsled/pwntools-tutorial/blob/master/utility.md#packing-and-unpacking-integers)

**extra gdb commands**

* Investigating qwords starting from an address

gdb-peda$ telescope $rsp 20

0000| 0x7fffffffde70 --> 0x401360 (<\_\_libc\_csu\_init>: push r15)

0008| 0x7fffffffde78 --> 0x7ffff7dea09b (<\_\_libc\_start\_main+235>: mov edi,eax)

0016| 0x7fffffffde80 --> 0x0

0024| 0x7fffffffde88 --> 0x7fffffffdf58 --> 0x7fffffffe2a2 ("/ctf/unibuc/curs\_re/curs\_"...)

0032| 0x7fffffffde90 --> 0x100040000

0040| 0x7fffffffde98 --> 0x401208 (<main>: push rbp)

0048| 0x7fffffffdea0 --> 0x0

0056| 0x7fffffffdea8 --> 0xe2f3fe41bf131724

0064| 0x7fffffffdeb0 --> 0x4010c0 (<\_start>: xor ebp,ebp)

0072| 0x7fffffffdeb8 --> 0x7fffffffdf50 --> 0x1

0080| 0x7fffffffdec0 --> 0x0

0088| 0x7fffffffdec8 --> 0x0

0096| 0x7fffffffded0 --> 0x1d0c013e24d31724

0104| 0x7fffffffded8 --> 0x1d0c117cd9751724

0112| 0x7fffffffdee0 --> 0x0

0120| 0x7fffffffdee8 --> 0x0

0128| 0x7fffffffdef0 --> 0x0

0136| 0x7fffffffdef8 --> 0x7fffffffdf68 --> 0x7fffffffe2d1 ("CLUTTER\_IM\_MODULE=xim")

0144| 0x7fffffffdf00 --> 0x7ffff7ffe190 --> 0x0

0152| 0x7fffffffdf08 --> 0x7ffff7fe44b6 (<\_dl\_init+118>: cmp ebx,0xffffffff)

**Task 1: stack-buffer overflow into data**

* Do an initial analysis of the binary in IDA. Find the buffer overflow vulnerability and calculate the required input length to overwrite the **pass\_len** variable **(1p)**
* Starting from the template, construct an input that overflows into the **pass\_len** variable and make the program print: **(1p)**

The correct password has length 12345

Unauthorized!

* Bypass the memcmp comparison by forcing its third parameter to be 0. **(1p)**
* Exploit the service running at 45.76.91.112 port 10051 **(1p)**

**Task 2: stack-buffer overflow into ret addr**

* Do an initial analysis of the binary in IDA. Find the buffer overflow vulnerability, look at the stack frame and calculate the required input length to overwrite the return address **(1p)**
* Starting from the template, construct an input that overflows into the return address and replaces it with the address of **do\_login\_success** **(1p)**
* Exploit the service running at 45.76.91.112 port 10052 **(1p)**

**Task 3: stack-buffer overflow protection**

* Starting from the template, construct an input that bypasses the stack protection but still overflows into the return address and replaces it with the address of **do\_login\_success** **(2p)**
* Exploit the service running at 45.76.91.112 port 10053 **(1p)**

**Task 4 (bonus): complex riddle**

* You have 10 tries to guess a random password. However the game is rigged both for and against you:
  + For: the randomness is determined by a seed that is provided at the beginning. Basically, you can determine what the password should be at any step.
  + Against: the game does not accept passwords with uppercase letters and the probability this happens is HIGH
* Find the vulnerability of this game, exploit it to get the following output: Task 4 solved! **(5p)**
* Exploit the service running at 45.76.91.112 port 10054 **(1p)**
  + No more than 5 connections are needed