**RE lab 02 - x86\_64 assembly**

**Lab files and setup**

In this lab, we will be working in the Linux environment only!

Ensure you have python 3 and pwntools:

*$ apt-get update*

*$ apt-get install python3-pycryptodome xinetd libffi-dev python3-wheel gcc gdb python3-setuptools python3-dev libssl-dev git libc6-dbg python3-pip make gcc-multilib socat*

*$ pip3 install pwntools*

Download files from [here](https://pwnthybytes.ro/unibuc_re/02-lab-files.zip) Password is infected

**Practical examples:**

[**01. Subtracting two numbers**](https://gcc.godbolt.org/z/cQ4py0)

* Observe the parameters in the call convention:
* rdi is the first parameter
* rsi is the second
* rax is the return value

[**02. Adding two numbers**](https://gcc.godbolt.org/z/oACb8t)

* Observe what happened: the compiler found a shorter way to do the operations

[**03. Adding three numbers**](https://gcc.godbolt.org/z/BtwJcQ)

* Observe the limitations of the previous optimization

[**04. Adding the first N numbers**](https://gcc.godbolt.org/z/uN9Gs2)

* Observe the initial TEST
* Which registers were allocated to i and sum ?

[**05. Accessing structure fields**](https://gcc.godbolt.org/z/4D--cF)

* Observe the pointer arithmetic
* Observe the first parameter to printf (string reference)

[**06. Singly linked list traversal**](https://gcc.godbolt.org/z/RqMaKS)

[**07. Simple division and compiler optimizations**](https://gcc.godbolt.org/z/9_nfLQ)

[**08. Simple crackme**](https://gcc.godbolt.org/z/pkX4B0)

**Task 1: Assembly analysis**

For each of the following, write an equivalent C line for each assembly instruction (if possible), then try to simplify until you understand what the code does.

At the end you can check gcc.godbolt.org to see if you get similar assembly generated from your C code (perfect matches might not be possible though).

You will need to **give each function a descriptive name** after you understand what it does.

**a) Write the equivalent in C for this ASM snippet (1p)**

myst1:

**test** **rdi**, **rdi**

**je** .L4

**mov** **edx**, 0

**mov** **eax**, 0

.L3:

**mov** **rcx**, **rdx**

**imul** **rcx**, **rdx**

**add** **rax**, **rcx**

**add** **rdx**, 1

**cmp** **rdi**, **rdx**

**jne** .L3

**ret**

.L4:

**mov** **rax**, **rdi**

**ret**

**b) Write the equivalent in C for this ASM snippet (1p)**

myst2:

**cmp** BYTE PTR [**rdi**], 0

**je** .L4

**mov** **eax**, 0

.L3:

**add** **rax**, 1

**cmp** BYTE PTR [**rdi**+**rax**], 0

**jne** .L3

**ret**

.L4:

**mov** **eax**, 0

**ret**

**c) Write the equivalent in C for this ASM snippet (1p)**

myst3:

**sub** **r8**, **r9**

**add** **r8**, **rdx**

**sub** **r8**, **rcx**

**lea** **rax**, [**r8**+**rdi**]

**sub** **rax**, **rsi**

**ret**

**d) Write the equivalent in C for this ASM snippet (2p)**

myst4:

push **rbp**

push **rbx**

sub **rsp**, 8

mov **rbx**, **rdi**

cmp **rdi**, 1

ja .L4

.L2:

mov **rax**, **rbx**

add **rsp**, 8

pop **rbx**

pop **rbp**

ret

.L4:

lea **rdi**, [**rdi**-1]

call myst4

mov **rbp**, **rax**

lea **rdi**, [**rbx**-2]

call myst4

lea **rbx**, [**rbp**+0+**rax**]

jmp .L2

**e) Write the equivalent in C for this ASM snippet (3p)**

myst5:

xor **eax**, **eax**

cmp **rdi**, 1

jbe .L1

cmp **rdi**, 3

jbe .L6

test dil, 1

je .L1

mov **ecx**, 2

jmp .L3

.L4:

mov **rax**, **rdi**

xor **edx**, **edx**

div **rcx**

test **rdx**, **rdx**

je .L8

.L3:

add **rcx**, 1

mov **rax**, **rcx**

imul **rax**, **rcx**

cmp **rax**, **rdi**

jbe .L4

.L6:

mov **eax**, 1

ret

.L8:

xor **eax**, **eax**

.L1:

ret

**Task 2: Assembly synthesis**

**Linux syscalls: get the time (3p)**

Starting from the template in the lab archive, write an assembly snippet to call the [sys\_time](https://linux.die.net/man/2/time) syscall (index 201) in order to obtain the current unix timestamp.

Since it is not immediately possible to call higher-level print functions such as printf, intercept the result using the tools from the previous class (strace or ltrace -S) in order to verify the results. Check against the Unix time at [Epoch Converter](https://www.epochconverter.com/)

Note: if the crash at the end bothers you, also call sys\_exit to close the process gracefully.

**Task 3: Compiler magic tricks**

**Un-optimize**

Write the equivalent in C/Python. Then, play around with it by feeding it various inputs to find out:

* what the initial code was **(3p)**
* how it works **(2p)**

bonus:

movabs **rdx**, 4880784091129696121

mov **rax**, **rdi**

mul **rdx**

sub **rdi**, **rdx**

shr **rdi**

lea **rax**, [**rdx**+**rdi**]

shr **rax**, 11

ret