Intro to Reverse Engineering

What does Reverse Engineering mean?

It means **analyzing** for the purpose to **understand** how it works.

This process helps us **finding bugs and vulnerabilities** that we can exploit to find what we need, sometimes even directly the flag.

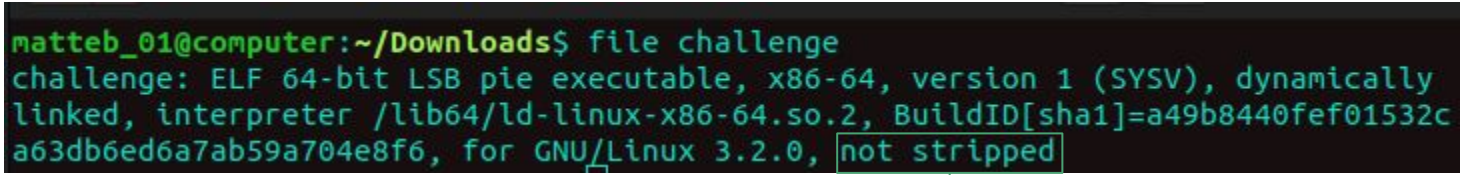
Reverse engineering might allow us to find **useful data** (for example passwords, DLC keys, seeds, etc.) or **unintended behaviours**. In some cases we can even **tamper** some parts of the code.

(like buffing our speed in a **videogame** or even implementing fly hacks as we'll see in the next lesson)

Information gathering

The **first step** of reverse engineering is gathering information about what kind of file we have and what it does. The **file** command helps us knowing more about that.

usage: **file [filename]**



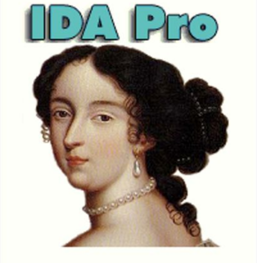
```
matteb_01@computer:~/Downloads$ file challenge
challenge: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dynamically
linked, interpreter /lib64/ld-linux-x86-64.so.2, BuildID[sha1]=a49b8440fef01532c
a63db6ed6a7ab59a704e8f6, for GNU/Linux 3.2.0, not stripped
```

this tells us that the file maintains symbols, so functions have their original name

Decompiling the executable

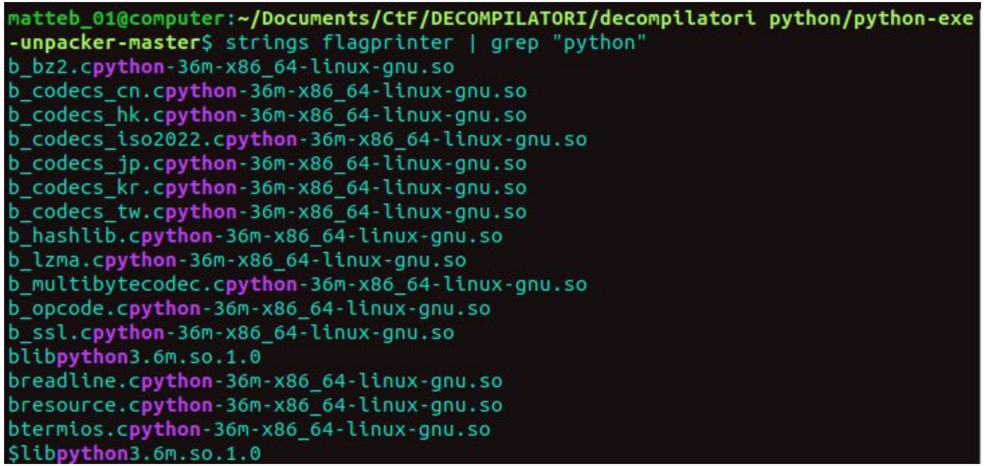
Decompiling means trying to obtain a **higher-level** version (the language the original program was written in) of the source code for a better understanding. Most of the time the code we obtain this way is **obfuscated**, nonetheless it is useful to have a **more comprehensible** version than just assembly.

There are various decompilers we might use to analyze our executable, like **Ghidra**, **IDA** or **Binary Ninja.**



Decompiling python

Yes, also python can be **compiled** (and obviously decompiled).
Usually the easiest method to spot a python executable is to **grep** the word “python” in the string command output



```
matteb_01@computer:~/Documents/CtF/DECOMPILATORI/decompilatori python/python-exe
-unpacker-master$ strings flagprinter | grep "python"
b_bz2.cpython-36m-x86_64-linux-gnu.so
b_codecs_cn.cpython-36m-x86_64-linux-gnu.so
b_codecs_hk.cpython-36m-x86_64-linux-gnu.so
b_codecs_iso2022.cpython-36m-x86_64-linux-gnu.so
b_codecs_jp.cpython-36m-x86_64-linux-gnu.so
b_codecs_kr.cpython-36m-x86_64-linux-gnu.so
b_codecs_tw.cpython-36m-x86_64-linux-gnu.so
b_hashlib.cpython-36m-x86_64-linux-gnu.so
b_lzma.cpython-36m-x86_64-linux-gnu.so
b_multibytecodec.cpython-36m-x86_64-linux-gnu.so
b_opcode.cpython-36m-x86_64-linux-gnu.so
b_ssl.cpython-36m-x86_64-linux-gnu.so
blibpython3.6m.so.1.0
breadline.cpython-36m-x86_64-linux-gnu.so
bresource.cpython-36m-x86_64-linux-gnu.so
btermios.cpython-36m-x86_64-linux-gnu.so
$libpython3.6m.so.1.0
```

Decompiling python

To reverse engineer a python program firstly you need to **extract** the .pyc files from the executable.

these are some extractors that might work

https://github.com/extremecoders-re/pyinstxtractor

https://github.com/WithSecureLabs/python-exe-unpacker

after extracting, **uncompyle6** can give you the .py file

https://pypi.org/project/uncompyle6/

NOTE: if the executable is an older version of python you might need to modify the **magic numbers** to make it work

NOW SOME CHALLENGES

About the next workshops

THANKS FOR YOUR ATTENTION