Reverse Engineering
Part 1

Hiding in plain sight

Quentin FRATY, for WOCSA

WOCSA

Reverse engineering 101

- It’s the art of dismantling an object, a software, etc… to see how it works

- Use cases in IT:
    - **Interoperability**: creation of drivers
    - **Legacy system support**: when support is dropped by the vendor
    - **Debugging:** Sometimes, debugging with GDB is the only way to find why code does not work
    - **Security assessment:** Analysis of one’s own software to spot vulnerabilities
    - **Malware protection:** Reversing malwares to understand its behaviour

1. Static analysis
2. Dynamic analysis
3. Scripting the analysis
4. It’s your turn!

WOCSA

1

Static analysis

Other tools (not covered today)

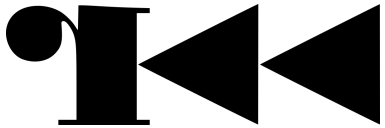
Static analysis (without executing the code)

JD-GUI

cutter

Binary Ninja ->

Radare2 ->



CLI tools without a logo:

- Objdump (information gathering)
- strings

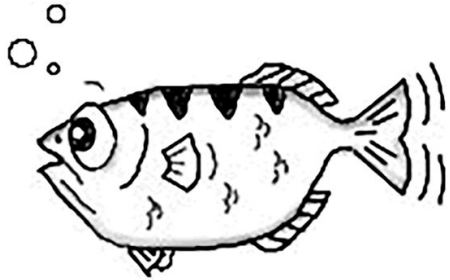
Static analysis goals

- Expose hidden information (unused functions, passwords, etc…)
- Understand the overall behaviour of the program
- Modify the program to achieve a certain goal
- Find if the program is statically linked or not (LD_PRELOAD)

2

Dynamic analysis

Dynamic analysis



GDB
The GNU Project Debugger

Use GDB-PEDA for a better experience!

CLI tools without a logo:

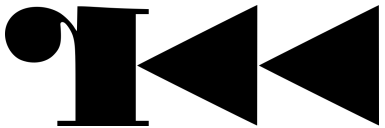
- Strace (system calls and signals tracer)
- Ltrace (library function calls tracer

Other tools (not covered today)

cutter

JD Eclipse plugin

Radare2 ->



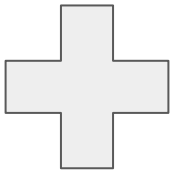
Dynamic analysis goals

- Bypass static obfuscation
- Follow specific variables values during execution
- Find values of function calls parameters
- Modify behaviour of the program on-the-fly

3

Scripting the analysis

Scripting the analysis



Scripting the analysis

- Why?
  - Bruteforce a value
  - Retrieve different values of some variables
  - Check if program behaviour is constant
- How?
  - Create a python script (example in the next slide)
  - Start GDB
  - Within GDB, run “source script.py”

Scripting the analysis

- Program is asking for a password
- In each loop, the script runs the program with a different password, and checks if the comparison with the expected password works
- Twist: in the case of this program, it checks letter by letter so we can bruteforce each letter individually

```
#!/usr/bin/env python3
# run this script through gdb by typing `source solver.py`
p = ['a']*43
chars = [i for i in range(0x21, 0x7f)] # assume the password is composed of printable chars
list_idx = 0
currchar = 0 # idx of the current evaluated char
nb_continue= 0
gdb.execute("set confirm off")
gdb.execute("set pagination off")
gdb.execute("file ./a.out")
gdb.execute("b *0x000055555555542b")
for i in range(10000):
    # set stdin to pass
    with open("pass", "w") as f:
        f.write("".join(p))
    gdb.execute("r < pass")
    for i in range(nb_continue):
        gdb.execute("c")
    c = gdb.parse_and_eval("$eflags")
    if "ZF" not in str(c):
        # the current letter is incorrect
        p[currchar] = chr(chars[list_idx])
        list_idx = (list_idx + 1) % len(chars)
    else:
        print("".join(p))
        currchar += 1
        list_idx = 0
        nb_continue = nb_continue + 1
    gdb.execute("k")
```

4

It’s your turn!

WOCSA

It’s your turn!

Wifi: Nothing Phone 2; Password: WocsaWorkshop
http://192.168.53.54:8080/archive.tar

Challenges:

- **edit_me**: Redacted!
- **execution_time**: Redacted!
- **sleepy_program**: Redacted!
- **java_dummy**: Redacted!
- **Incremental**: Redacted! script

WOCSA

Cheatsheet GDB

Ouvrir un programme: gdb ./binaire

Lancer le débogage depuis gdb: start

Aller à l’instruction suivante: stepi

Créer un point d’arrêt: break *adresse

Continuer jusqu’au prochain breakpoint ou jusqu’à la fin: continue

Regarder les adresses des instructions dans GDB: disas main

Pour ld_preload:

Compiler une librairie: gcc -shared fichier.c -o lib.so

L’utiliser: LD_PRELOAD=./lib.so fichier.bin