Random password generator 2

Luigi Dell'Eva, 04/05/2024.

Background

The challenge consists in analyzing an ELF binary file to find a way to retrieve the flag. When executed the file asks to enter a password and if correct it will print the flag otherwise it prints the correct password. The user has two attempts, then the program will exit. Additionally, the password changes each try.

Reverse engineering, is a process through which one attempts to understand how a process, system, or piece of software accomplishes a task with very little insight into exactly how it does so. This process is usually done to perform security analysis, extract sensitive information, remodel obsolete objects or to understand how a software works. [1]

More specifically, **static reverse engineering** is a method of analyzing software without executing it. This approach involves examining the code and structure of a software, which can include analyzing the source code if available, or examining the binary executable using disassemblers or decompilers. [2]

Usially people utilizes different techniques to to make reverse engineering more difficult. Some techniques include **stripping** which consists in *strip* all the symbols such as function names, global variable names, etc (to strip shared object symbols we have to statically link the shared libraries) or **anti-disassembly** which manipulates software such as Ghidra or IDA, by inserting bytes that exploit bugs or limitation in their functionalities. These bytes will not affect the program execuion but can mislead the disassemblers and others techniques such as **packing**, **virtual machine obfuscator**, etc. [3] However, in the case of this challenge on the binary file we do not have any kind of stripping or anti-disassembly techniques.

Vulnerability

Disassemblying the binary with Ghidra we can see that the program is composed by two principal functions: `main` and a function called `rand_pass`. Other functions are exernal and dynamically linked. In the main function the program asks the user a password (two try maximum) and each time it generats a random password by calling the `rand_pass` function. The password is generated by using as seed the the PID of the process (obtained by calling `getpid`) which is set at the beginning of the program (and never changes). The list of character that can be used to generate the password is restricted by this code `(char)iVar1 + (char)(iVar1 / 0x5e) * -0x5e + '!'` which allows only character between `!` and `~` (ASCII 0x21 to 0x7e). The password is then returned to the `main` and compared with the user input. If the password is correct the program will print the flag otherwise it will print the correct password.

The problem with this implementation is that the password is generated using a **pseudo-random number generator** (PRNG) which is seeded with a constant value (the PID of the process). In computer science PRNG algorithms are used to generate a sequence of numbers that approximate the properties of random numbers. The problem with PRNG is that they are **deterministic**, meaning that if the seed is known the **sequence of numbers generated can be predicted**. [4]

Solution

By looking at the binary file using `file bin` we can see that it has been compiled for 64-bit architecture. Knowing that the value of PIDs in 64-bit operating systems can be set up to 4194304 (if the OS used is 64-bit it can be checked with `cat /proc/sys/kernel/pid_max`) [5] and that the seed (which is the PID) does not change through the execution of the program, we can brute force the PID by retriving the first password and check which PID from 0 to 4194304 generates that password. Once we have the PID we can use it to generate the second password and once entered into the program we will get the flag. This can be done by using the following python script.

```python
from ctypes import CDLL
from pwn import *

conn = remote('cyberchallenge.disi.unitn.it', 50150)
#conn = process('./bin')

conn.sendlineafter(b':', b'rng')
conn.recvuntil(b'was: ')
gen_pwd = conn.recvline().decode().strip()

libc = CDLL("libc.so.6")
for i in range(0, 4194304):
    libc.srand(i)
    password = ''.join([chr(libc.rand() % 0x5e + ord('!')) for _ in range(0x10)])
    if (gen_pwd == password):
        #print(f"Found seed: {i}")
        break

password = ''.join([chr(libc.rand() % 0x5e + ord('!')) for _ in range(0x10)])
conn.sendlineafter(b':', password.encode())
result = conn.recvall(timeout=1)
print(result.decode())
```

References

[1] Reverse engineering: https://en.wikipedia.org/wiki/Reverse_engineering

[2] Static reverse engineering: https://cyberw1ng.medium.com/unveiling-the-hidden-an-introduction-to-the-fundamentals-of-reverse-engineering-karthikeyan-89b39ec4d0bb

[3] Ethical Hacking course slides

[4] Pseudo number generator: https://en.wikipedia.org/wiki/Pseudorandom_number_generator

[5] PID: https://medium.com/@gargi_sharma/pid-allocation-in-linux-kernel-dc0c78d14e77