# [pwn] ECSC2020 Romania — Write-up

Gabriel Pirjolescu May 11, 2020 · 6 min read

The past weekend I took part in a CTF organised for qualifying to the ECSC 2020 in the Romanian team. In this post, I will explain how I solved the two pwn challenges available. I have to mention that the remote server is Ubuntu 20.04, thus we need to use `libc-2.31`.

### PLT and GOT

First I would like to explain the concepts of PLT and GOT. The Global Offset Table (GOT) is made up of the `.got` and `.got.plt` sections. The `.got` segment holds the addresses of the functions in libc. The addresses in GOT are populated dynamically by the dynamic linker while the program is running. The `.got.plt` holds a reference to `.got` for each symbol.

The Procedure Linkage Table (PLT) contains only one section, `.plt`. For each function in an outside library referenced in our program, we have an entry in `.plt` which is actually a jump to the address in `.got.plt` for that function. Here I have found a very good explanation of how shared functions are linked. LiveOverflow also has a nice video where he explains the process.

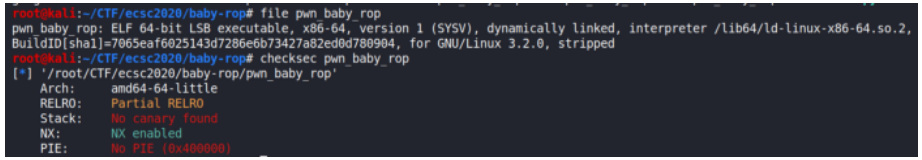
```
The first time a shared function is called, the GOT contains a
pointer back to the PLT, where the dynamic linker is called to find
```

```
the actual location of the function in question. The location found
is then written to the GOT. The second time a function is called, the
GOT contains the known location of the function. This is called “lazy
binding.” This is because it is unlikely that the location of the
shared function has changed and it saves some CPU cycles as well.
```

### pwn_baby_rop

This challenge is a return-oriented-programming one. However, the binary has been slightly modified so that it doesn't decompile nicely. Therefore, IDA was not of much help this time. The only security measures in place are:

- NX -> the stack is not executable
- Partial RELRO -> the non-PLT part of the GOT section is read-only
- ASLR is enabled on remote, meaning the base address of libc is randomised on the remote server




The active security measures

To understand the source code first I disassembled the binary in IDA, then I opened it in GDB and disassembled the range of addresses where I knew a function was residing. The functionality of `main` is the following:

- Sets the buffer mode for `stdin` and `stdout`
- Prints `Solve this challenge to prove your understanding to black magic.`
- Calls a function at `0x401176`

The main function disassembled

The function called by main is the one which contains the vulnerability. It doesn't do much except for the vulnerability itself which is a buffer overflow by using `gets with no bounds checking.`

The function called by main

To exploit the vulnerability we need to find out how far away from the start of our input string lies the address of main where the execution will

return to after the function ends. To do this we create a pattern and set a breakpoint at the `ret` instruction of the function. Consequently, we search for the data on the stack in the pattern. It looks like we need 264 bytes to overflow the return instruction pointer.

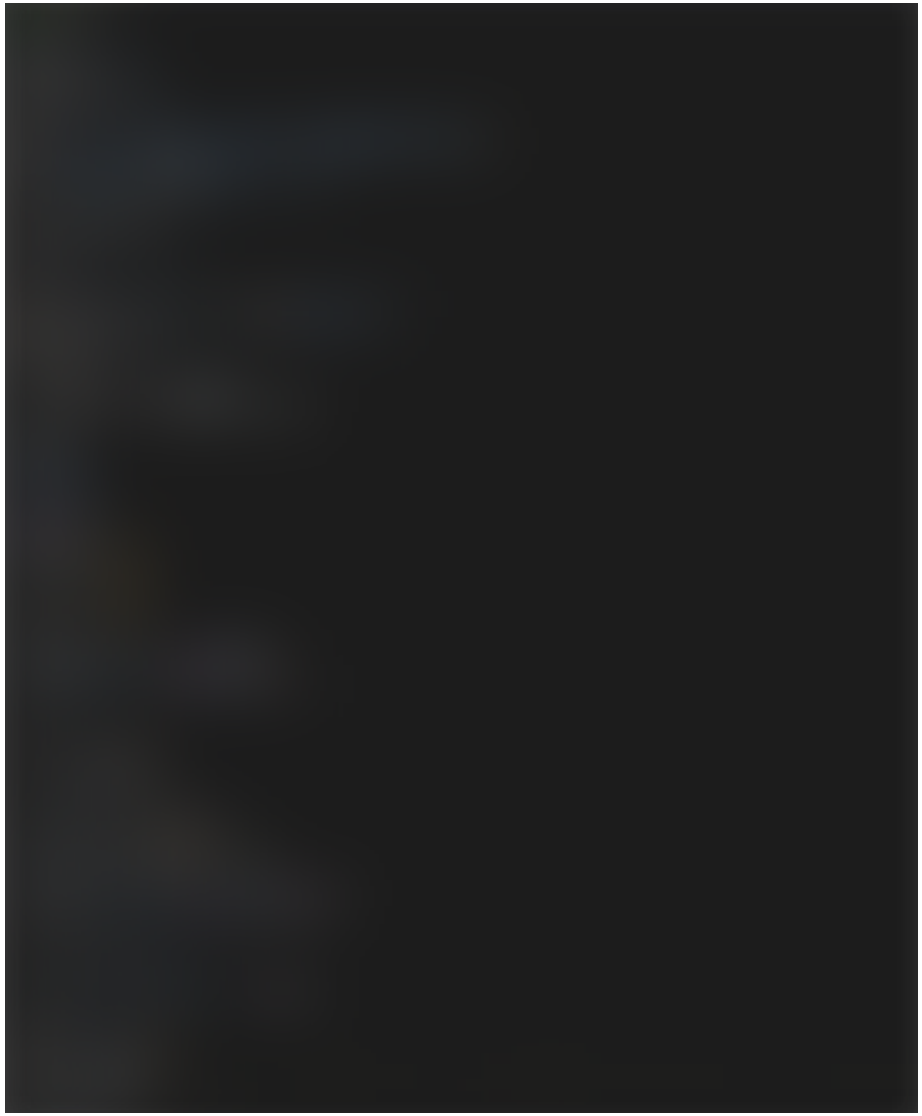
Finding out the pattern offset

First, we need to leak an address from libc so we can calculate the address of `system`. To do this we need to:

- Find the addresses of a `pop_rdi` gadget, `puts@got`, `puts@plt` and `main@plt`
- Pop `puts@got` which points to the address of puts in libc into the `rdi` register
- Print the `puts@got` address by calling `puts@plt`
- In addition, we need to call `main@plt` again so the execution doesn't stop

By using the leaked `puts` address now we need to compute the base address of `libc`, find a `/bin/sh` string and the address of `system` and

exploit the buffer overflow again.



The exploit script

**pwn_baby_fmt**

This challenge was a bit more complicated than the previous one. It has the following security protections enabled:

- NX -> stack not executable
- Full RELRO -> the GOT section is read-only
- PIE -> the base address of the binary is randomised
- ASLR is enabled on the remote system -> the address of libc is randomised on the server

It contains four important functions:

- `malicious`
- `primary`
- `check_random`
- `list_contains`

The malicious function runs `execve` with `/bin/cat` and `flag` as parameters, basically outputting the flag.

The functionality of `list_contains` is:

- it takes a string as an argument and checks if it is part of another string `bucurest iasi cluj timisoar brasov constant`



The functionality of `check_random` is:

- It takes an integer as an argument
- It checks if the argument is equal to a number stored in the `.bss` section
- If it's not equal the execution ends
- Otherwise it prints `Ok`



The functionality of `primary` is:

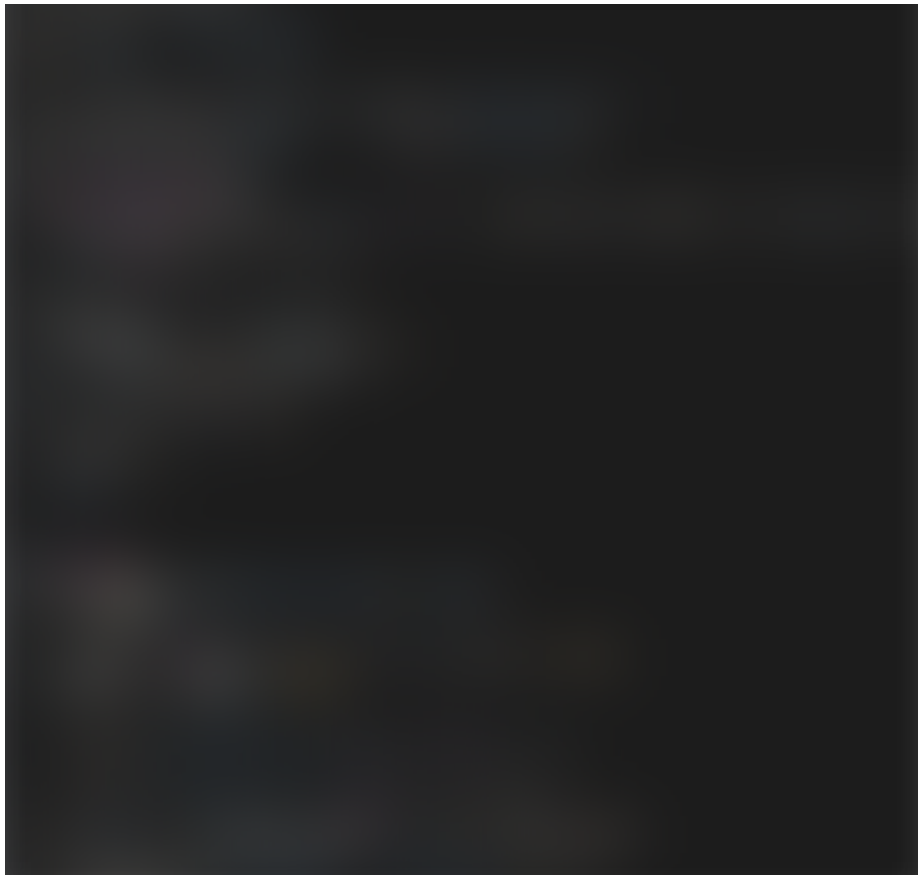
- initializes a seed with `time(0)`
- initializes a pseudo-random number generator with the seed
- generates a random number and stores it in `.bss`
- prints What's your town?

- Reads an input string of 7 characters and appends a null-terminating character to it
- Passes the input to `list_contains` to check if it's a substring of the string mentioned earlier
- If it is it prints `Hello, %s`
- Otherwise it prints `Hello stranger. What town is this?` and the string (here we have a format string vulnerability)
- Prints `Can you say hi in Chalcatongo?`
- Reads an input using `gets` (here we have a buffer overflow vulnerability)
- The random number is passed to the `check_canary` function
- The function returns

The exploitation strategy is the following:

- Leak an address on the stack using the format-string vulnerability (in this case the 10th address from the input string was leaked)
- Compute the address of the malicious function by subtracting an offset from the leak
- Write a program in C that generates a random number in the same way plus a time delay to account for the network delay (in our case it was 9 second after a little brute-forcing)
- Use the buffer overflow together with the generated random number and the computed address

The C program used to generate the random number



The exploit script

**Contact**

If you would like to get in touch you can contact me on one of the following platforms:

- Twitter
- LinkedIn

**Sources**

- https://www.redhat.com/en/blog/hardening-elf-binaries-using-relocation-read-only-relro