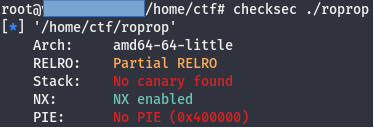
Writeup for Darkctf 2020’s “roprop” pwn challenge.

**Initial notes:**

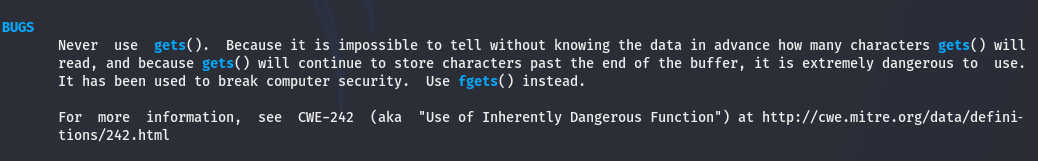
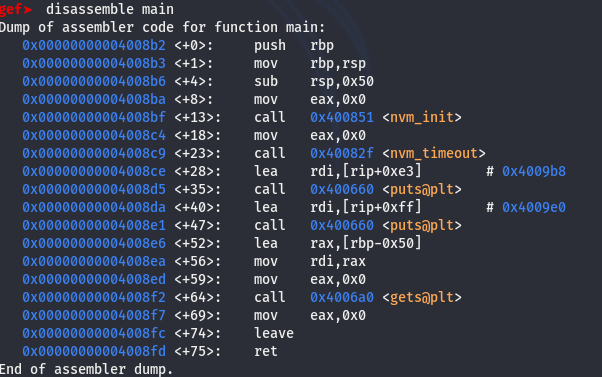
**useful tools:**

* pwntools library for python
* Gef add on for gdb
* Checksec

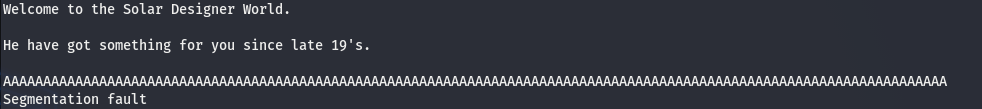
**Checksec:** This tool will show you the properties of an executable, here is what we get when we run it on this binary. You can see that the stack is non-executable, so shellcode won’t work. One solution is a ret2libc attack

**Finding the vulnerability:**

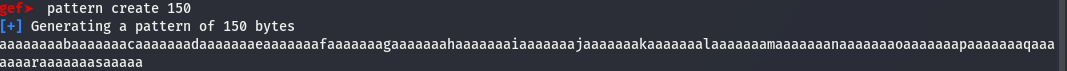
This section was pretty straightforward, it has a main function that calls gets(), which is vulnerable to a buffer overflow. Here is the disassembly for main in the binary, and part of the manpage for gets().



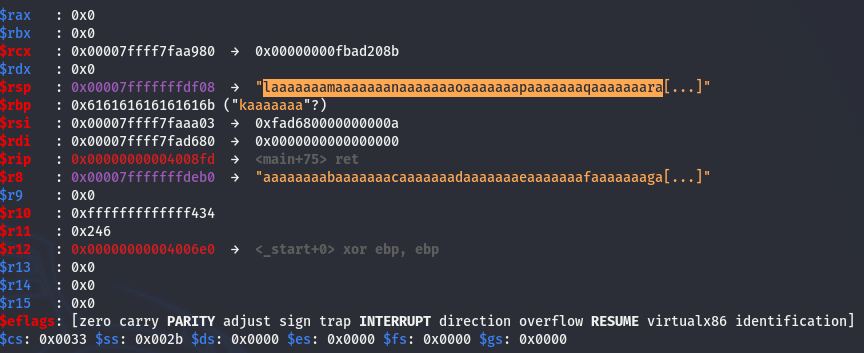
Going off of that information, I tried to get a segfault.

****

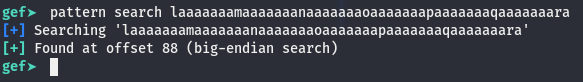
We still need to measure how much padding we need before we overwrite the return address, which can be done with Gef for GDB. If you type “pattern create 150” it will generate a 150 character pattern that you can use as input for your program.

****

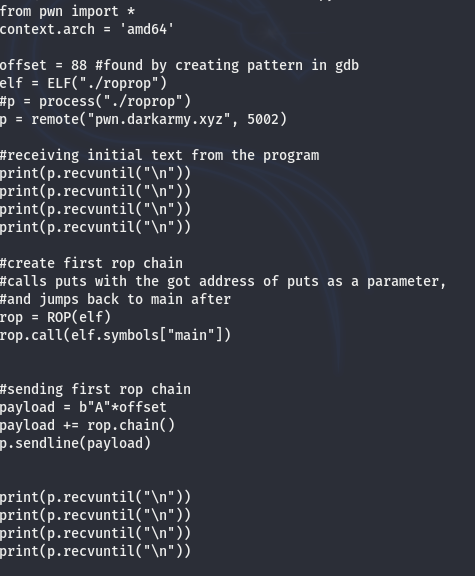
Now if we run the program, and copy paste this into the field when gets() is called, we can overwrite the saved return address. You can find where you overwrite the saved return address in the rsp register.

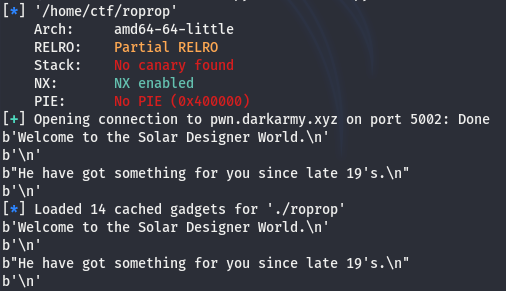


Now we need to search for the offset, which can also be done easily through gef, just type “pattern search text\_to\_find”, here is an example.



So now we have our offset, and we can overwrite the saved return address. To prove this, I will have it jump back to main. Here Is my exploit script so far, and the output from the exploit script.

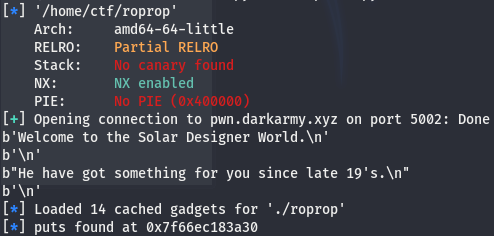




You can see that the welcome messages are printed twice. We have jumped back to the start of main. But calling main doesn’t really give us anything important, we want to call system(“/bin/sh”) and get a shell on the remote host. System() is not called in this program, so there isn’t a plt or got entry for it, so we need to find the offset of system in the version of libc being used, as well as the base address of libc. After that, the address of system can just be calculated as:

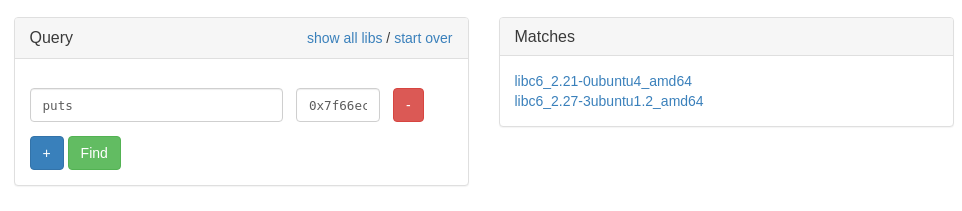
system = libc\_base\_address + system\_offset

**Finding libc version used by the binary:**

 We need to call puts(puts\_got\_address). This will give us the actual address of puts on the remote host, and we can use that to find the version of libc being used, and calculate the base address for libc.

We put the address into a database search on [https://libc.blukat.me](https://libc.blukat.me/), and after trying both, I found that

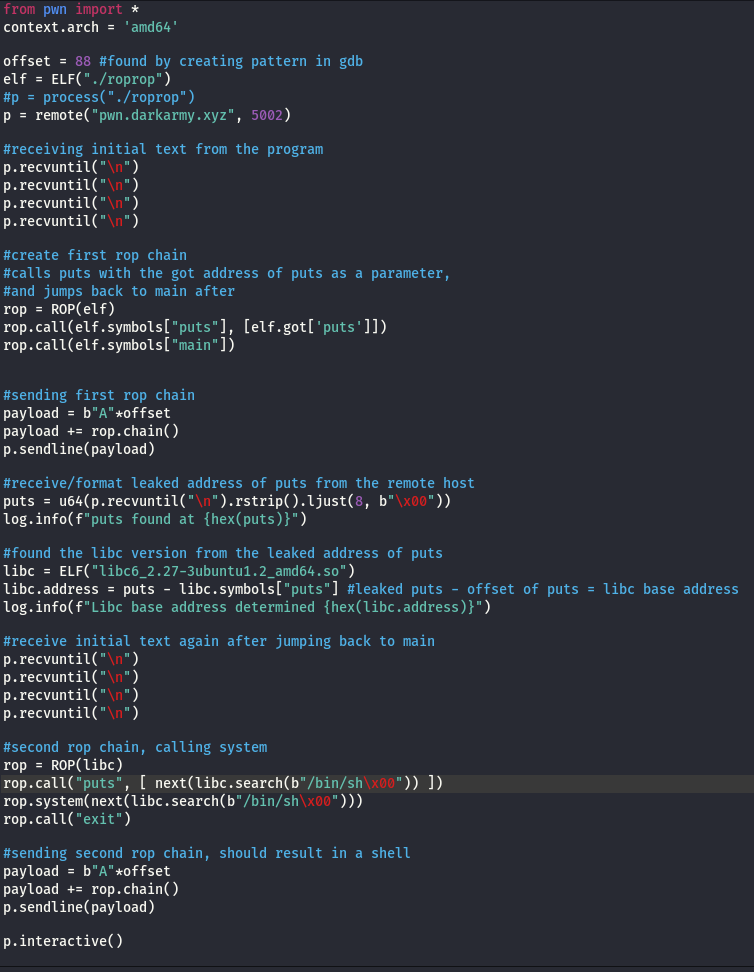
libc6\_2.27-3ubuntu1.2\_amd64 was the right libc version.

**Writing our exploit**

So now we use pwntools to load symbols from this version of libc, and we use this to make our call to system(). Thankfully pwntools makes this process a lot easier. We still have the initial call to leak the address of puts, we just need to add a few things:

1. Load the libc file into pwntools using libc = ELF(“libc\_file\_name”)

2. Use that to calculate the base: libc.address = leaked\_puts – libc.symbols[“puts”]

Lastly, we need to make a second ROP chain to call system(“/bin/sh”). Here is the final script.

Running the python script:

