Challenge name: FlagChecker
Category: Reverse Engineering
Difficulty: Medium
Description: Can you beat this FlagChecker?
Flag: `1337UP{tHr33_Zs_FTW!!}`

Writeup

```
→  intigriti_2023 file FlagChecker
FlagChecker: ELF 64-bit LSB pie executable, x86-64, version 1
 (SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-
64.so.2, BuildID[sha1]=c942dfa7bc3f00bb73a255dcb29e6c94d22ddb
f3, for GNU/Linux 3.2.0, with debug_info, not stripped
→  intigriti_2023
```

- It's a 64 bit executable file with **debug_info**

```
→  intigriti_2023 gdb -q FlagChecker
pwndbg: loaded 198 commands. Type pwndbg [filter] for a list.
pwndbg: created $rebase, $ida gdb functions (can be used with print/break)
Reading symbols from FlagChecker...
warning: Missing auto-load script at offset 0 in section .debug_gdb_scripts
of file /home/kali/CTFs/dev/VIT/rev/intigriti_2023/FlagChecker.
Use `info auto-load python-scripts [REGEXP]' to list them.
pwndbg> r
Starting program: /home/kali/CTFs/dev/VIT/rev/intigriti_2023/FlagChecker
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Enter the flag:
idk
thread 'main' panicked at 'index out of bounds: the len is 3 but the index is 18', test.rs:4:5
note: run with `RUST_BACKTRACE=1` environment variable to display a backtrace
[Inferior 1 (process 74063) exited with code 0145]
pwndbg>
```

- Out of bounds error indicates that, it fails because of our input length
- We can view the source code in gdb using `list` command (since it's compiled with debug info)

```
pwndbg> list 1
1       use std::io;
2
3       fn check_flag(flag: &str) -> bool {
4           flag.as_bytes()[18] as i32 * flag.as_bytes()[7] as i32 & flag.as_bytes()[12] as i32 ^ flag.as_bytes()[2] as i32 == 127 &&
5           flag.as_bytes()[1] as i32 % flag.as_bytes()[14] as i32 - flag.as_bytes()[21] as i32 % flag.as_bytes()[15] as i32 == 21 &&
6           flag.as_bytes()[10] as i32 + flag.as_bytes()[4] as i32 * flag.as_bytes()[11] as i32 - flag.as_bytes()[20] as i32 == 4353 &&
7           flag.as_bytes()[19] as i32 + flag.as_bytes()[12] as i32 * flag.as_bytes()[0] as i32 ^ flag.as_bytes()[16] as i32 == 4630 &&
8           flag.as_bytes()[9] as i32 ^ flag.as_bytes()[13] as i32 * flag.as_bytes()[8] as i32 & flag.as_bytes()[16] as i32 == 50 &&
9           flag.as_bytes()[3] as i32 * flag.as_bytes()[17] as i32 + flag.as_bytes()[5] as i32 + flag.as_bytes()[6] as i32 == 4823 &&
10          flag.as_bytes()[21] as i32 * flag.as_bytes()[2] as i32 ^ flag.as_bytes()[3] as i32 ^ flag.as_bytes()[19] as i32 == 6385 &&
pwndbg>
11          flag.as_bytes()[11] as i32 ^ flag.as_bytes()[20] as i32 * flag.as_bytes()[1] as i32 + flag.as_bytes()[6] as i32 == 1853 &&
12          flag.as_bytes()[7] as i32 + flag.as_bytes()[5] as i32 - flag.as_bytes()[18] as i32 & flag.as_bytes()[9] as i32 == 96 &&
13          flag.as_bytes()[12] as i32 * flag.as_bytes()[8] as i32 - flag.as_bytes()[10] as i32 + flag.as_bytes()[4] as i32 == 6874 &&
14          flag.as_bytes()[16] as i32 ^ flag.as_bytes()[17] as i32 * flag.as_bytes()[13] as i32 + flag.as_bytes()[14] as i32 == 7613 &&
15          flag.as_bytes()[0] as i32 * flag.as_bytes()[15] as i32 + flag.as_bytes()[3] as i32 == 4710 &&
16          flag.as_bytes()[13] as i32 + flag.as_bytes()[18] as i32 * flag.as_bytes()[2] as i32 & flag.as_bytes()[5] as i32 ^ flag.as_bytes()[10]
 as i32 == 51 &&
17          flag.as_bytes()[0] as i32 % flag.as_bytes()[12] as i32 - flag.as_bytes()[19] as i32 % flag.as_bytes()[7] as i32 == 16 &&
18          flag.as_bytes()[14] as i32 + flag.as_bytes()[21] as i32 * flag.as_bytes()[16] as i32 - flag.as_bytes()[8] as i32 == 8793 &&
19          flag.as_bytes()[3] as i32 + flag.as_bytes()[17] as i32 * flag.as_bytes()[9] as i32 ^ flag.as_bytes()[11] as i32 == 9644 &&
20          flag.as_bytes()[15] as i32 ^ flag.as_bytes()[4] as i32 * flag.as_bytes()[20] as i32 & flag.as_bytes()[1] as i32 == 110 &&
pwndbg>
21          flag.as_bytes()[6] as i32 * flag.as_bytes()[12] as i32 + flag.as_bytes()[19] as i32 + flag.as_bytes()[2] as i32 == 11769 &&
22          flag.as_bytes()[7] as i32 * flag.as_bytes()[5] as i32 ^ flag.as_bytes()[10] as i32 ^ flag.as_bytes()[0] as i32 == 9282 &&
23          flag.as_bytes()[21] as i32 ^ flag.as_bytes()[13] as i32 * flag.as_bytes()[15] as i32 + flag.as_bytes()[11] as i32 == 8676 &&
24          flag.as_bytes()[16] as i32 + flag.as_bytes()[20] as i32 - flag.as_bytes()[3] as i32 & flag.as_bytes()[9] as i32 == 48 &&
25          flag.as_bytes()[18] as i32 * flag.as_bytes()[1] as i32 - flag.as_bytes()[4] as i32 + flag.as_bytes()[14] as i32 == 4467 &&
26          flag.as_bytes()[8] as i32 ^ flag.as_bytes()[6] as i32 * flag.as_bytes()[17] as i32 + flag.as_bytes()[12] as i32 == 10483 &&
27          flag.as_bytes()[11] as i32 * flag.as_bytes()[2] as i32 + flag.as_bytes()[15] as i32 == 2696
28      }
29
30      fn main() {
pwndbg>
31          let mut flag = String::new();
32          println!("Enter the flag: ");
33          io::stdin().read_line(&mut flag).expect("Failed to read line");
34          let flag = flag.trim();
35
36          if check_flag(flag) {
37              println!("Correct flag");
38          } else {
```

- Here we can see there are a plenty of comparison checks with multiple operations
- We can't solve it manually, so we are going to use z3 theorem solver to satisfy these conditions

z3 script

```
from z3 import *

s = Solver()

flag = []
for i in range(22):
    flag.append(BitVec(f"f_{i}", 8))
```

- Our flag length is 22, so let's create 22 bit vectors
- Now extract all the condition from the output of gdb

```
s.add(flag[18] * flag[7] & flag[12] ^ flag[2] == 127)
s.add(flag[1] % flag[14] - flag[21] % flag[15] == 21)
s.add(flag[10] + flag[4] * flag[11] - flag[20] == 4353)
s.add(flag[19] + flag[12] * flag[0] ^ flag[16] == 4630)
s.add(flag[9] ^ flag[13] * flag[8] & flag[16] == 50)
s.add(flag[3] * flag[17] + flag[5] + flag[6] == 4823)
s.add(flag[21] * flag[2] ^ flag[3] ^ flag[19] == 6385)
s.add(flag[11] ^ flag[20] * flag[1] + flag[6] == 1853)
s.add(flag[7] + flag[5] - flag[18] & flag[9] == 96)
```

```
s.add(flag[12] * flag[8] - flag[10] + flag[4] == 6874)
s.add(flag[16] ^ flag[17] * flag[13] + flag[14] == 7613)
s.add(flag[0] * flag[15] + flag[3] == 4710)
s.add(flag[13] + flag[18] * flag[2] & flag[5] ^ flag[10] == 51)
s.add(flag[0] % flag[12] - flag[19] % flag[7] == 16)
s.add(flag[14] + flag[21] * flag[16] - flag[8] == 8793)
s.add(flag[3] + flag[17] * flag[9] ^ flag[11] == 9644)
s.add(flag[15] ^ flag[4] * flag[20] & flag[1] == 110)
s.add(flag[6] * flag[12] + flag[19] + flag[2] == 11769)
s.add(flag[7] * flag[5] ^ flag[10] ^ flag[0] == 9282)
s.add(flag[21] ^ flag[13] * flag[15] + flag[11] == 8676)
s.add(flag[16] + flag[20] - flag[3] & flag[9] == 48)
s.add(flag[18] * flag[1] - flag[4] + flag[14] == 4467)
s.add(flag[8] ^ flag[6] * flag[17] + flag[12] == 10483)
s.add(flag[11] * flag[2] + flag[15] == 2696)
```

- and add it to the solver carefully

```
for i in range(22):
    s.add(flag[i] >= 33)
    s.add(flag[i] <= 127)
```

- Flag needs to be in the ascii range, append that condition also

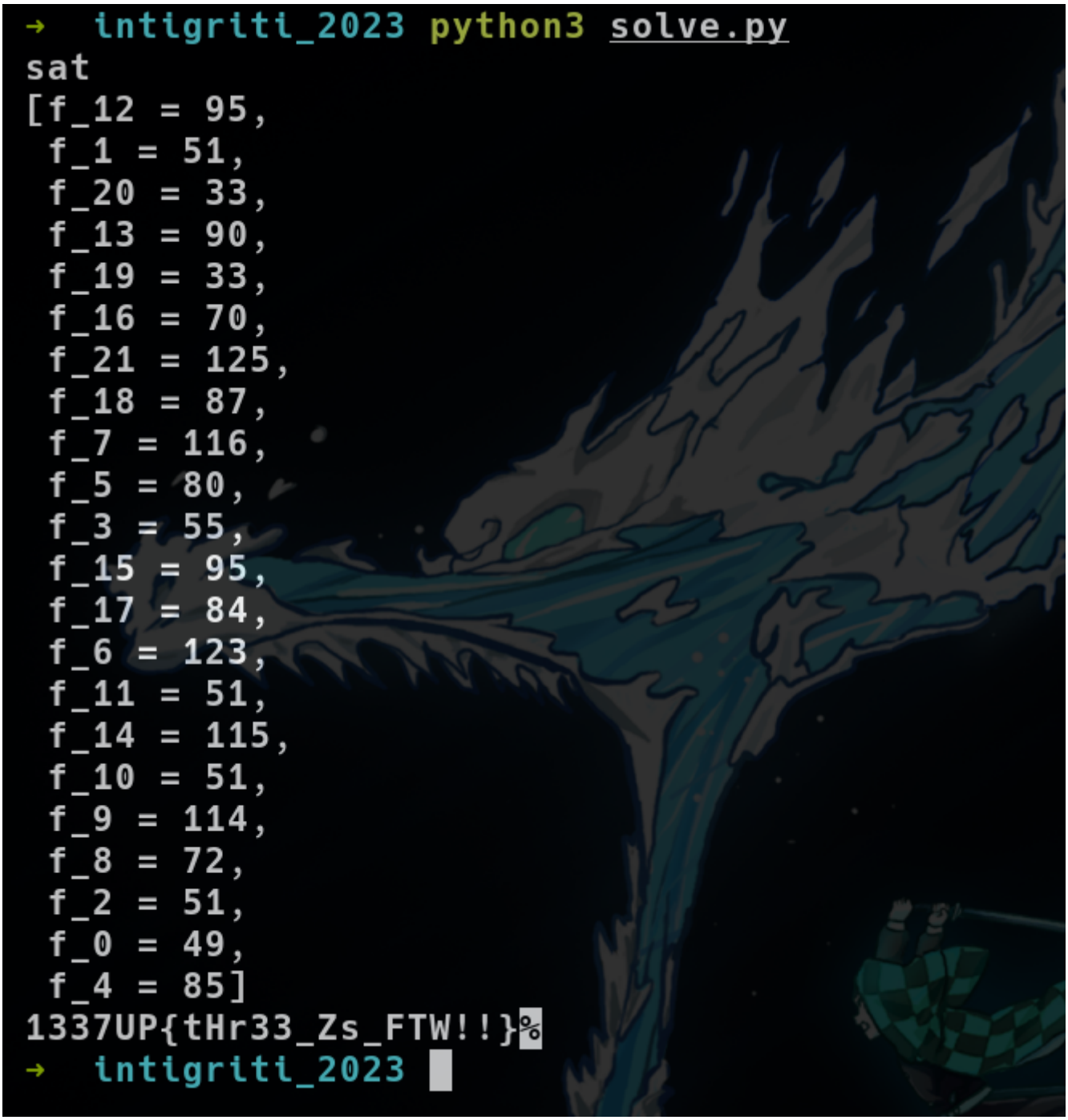
```
print(s.check())
flag_arr = s.model()
print(flag_arr)
```

- Now we can check all those conditions and print the correct values

- All conditions are satisfied

```
for i in range(0,22):
        print(chr(flag_arr.eval(eval(f'flag[{i}]')).as_long()),end='')
```

- We can print those ascii values into text using this



- That's all we got our flag