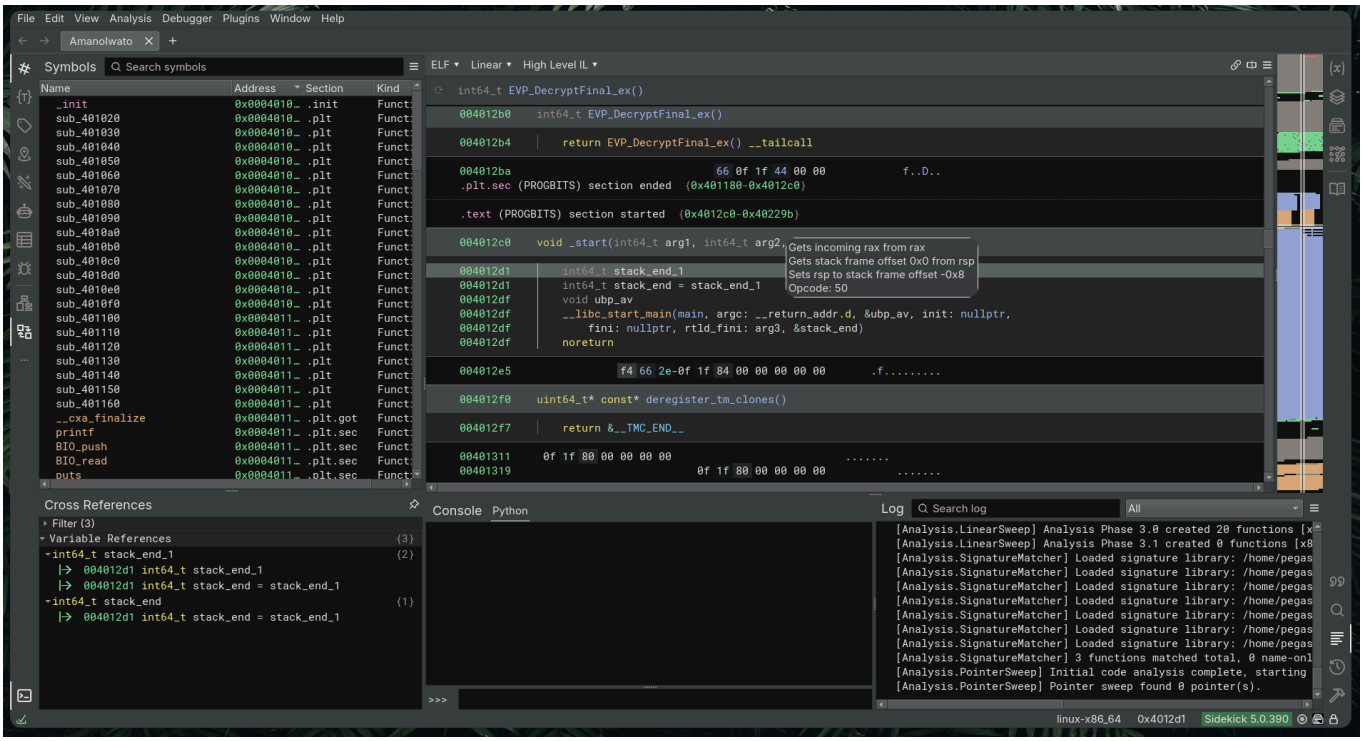
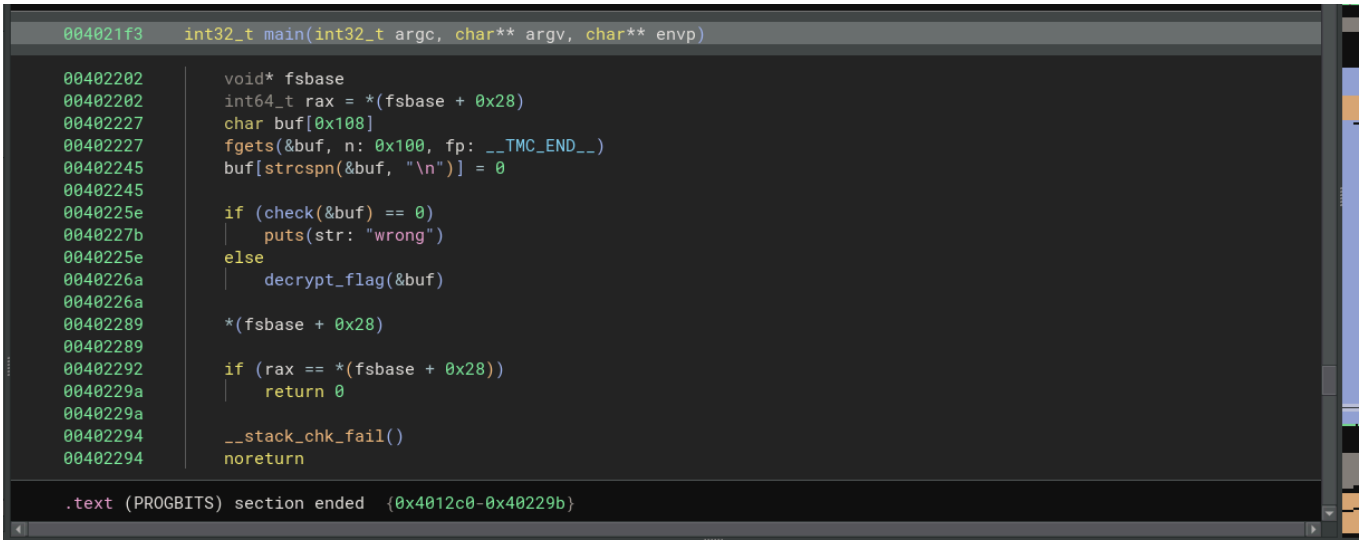
First open the binary in Binary Ninja



When you look at the main function, you can see that it simply calls a function called `check` and the program needs it to return 1 to give up the flag



If we take a look at the `decrypt_flag` function, we see that it's impossible to force it to spill out the flag as it uses the user input to decrypt the flag with an AES-128-CBC cipher. Therefore, we

need to understand and find what `check` needs to return 1.

```
00401f7c    int64_t decrypt_flag(char* arg1)

00401f92        void* fsbase
00401f92        int64_t rax = *(fsbase + 0x28)
00401fc7        int64_t var_298
00401fc7        SHA256(arg1, strlen(arg1), &var_298)
00401fda        int64_t var_2b8 = var_298
00401fe1        int64_t var_290
00401fe1        int64_t var_2b0 = var_290
00401ffa        int64_t var_288
00401ffa        int64_t var_2a8 = var_288
00402001        int64_t var_280
00402001        int64_t var_2a0 = var_280
0040201c        int64_t var_278
0040201c        __builtin_strcpy(dest: &var_278,
0040201c            src: "osqexW5JG4swc/e8b9m0OVMaeR4N3xqO8wjJ53LW6TGkkVe7x9ll2+cfdXZU9CwBXoktlJ497OwSwj9n71qemA==")
004020e8        void var_218
004020e8        int32_t rax_6 = base64_decode(&var_278, &var_218)
004020f3        int64_t rax_7 = EVP_CIPHER_CTX_new()
0040212a        EVP_DecryptInit_ex(rax_7, EVP_aes_128_cbc(), 0, &var_2b8, &var_2a8)
00402157        int32_t var_2cc
00402157        char var_118[0x108]
00402157        EVP_DecryptUpdate(rax_7, &var_118, &var_2cc, &var_218, zx.q(rax_6))
00402183        int32_t var_2c8
00402183        EVP_DecryptFinal_ex(rax_7, &var_118[sx.q(var_2cc)], &var_2c8)
00402198        var_118[sx.q(var_2c8 + var_2cc)] = 0
004021aa        puts(str: "Amano-Iwato has opened...")
004021c8        printf(format: "flag: %s\n", &var_118)
004021d7        EVP_CIPHER_CTX_free(rax_7)
004021d7
004021ea        if (rax == *(fsbase + 0x28))
004021f2            return rax - *(fsbase + 0x28)
004021f2
004021ec        __stack_chk_fail()
004021ec        noreturn
```

Now, as you can see, `check` is just a (long) list of constraints that one must pass to get coveted 1.

```
004013a9    int64_t check(char* arg1)

004013c9        if (strlen(arg1) != 0x20)
004013cb            return 0
004013cb
004013ef        if ((arg1[0x17] & arg1[0x13]) != 0x48)
004013f1            return 0
004013f1
00401415        if ((arg1[0xf] ^ arg1[0x18]) != 0x31)
00401417            return 0
00401417
0040143d        if (arg1[0x1e] - *arg1 != 0x11)
0040143f            return 0
0040143f
0040146a        if (arg1[0xc] * arg1[0x1f] != 0x90)
0040146c            return 0
0040146c
00401497        if (arg1[0xa] * arg1[0x1a] != 0)
00401499            return 0
00401499
004014c4        if (arg1[0xd] * arg1[0x1d] != 0x72)
004014c6            return 0
004014c6
004014ec        if (arg1[0xe] + arg1[0x11] != 0x7d)
004014ee            return 0
004014ee
00401518        if (arg1[0x1c] - arg1[0x1b] != 0xe8)
0040151a            return 0
0040151a
0040153e        if ((arg1[0x15] | arg1[1]) != 0x78)
00401540            return 0
00401540
00401564        if ((arg1[0x16] ^ arg1[2]) != 0x42)
00401566            return 0
00401566
```

...

```
00401d49
00401d6d    if ((arg1[0xb] & arg1[0xf]) != 0x41)
00401d6f        return 0
00401d6f
00401d9a    if (arg1[4] * arg1[9] != 0xb9)
00401d9c        return 0
00401d9c
00401dbe    if (*arg1 + arg1[0x12] != 0xbf)
00401dc0        return 0
00401dc0
00401deb    if (arg1[0x18] * arg1[0xc] != 0x90)
00401ded        return 0
00401ded
00401e11    if ((arg1[0xb] & arg1[0x18]) != 0x70)
00401e13        return 0
00401e13
00401e33    if ((*arg1 & arg1[0x1e]) != 0x68)
00401e35        return 0
00401e35
00401e60    if (arg1[0x14] * arg1[8] != 0xf4)
00401e62        return 0
00401e62
00401e83    if ((arg1[1] | arg1[0xe]) != 0x7d)
00401e85        return 0
00401e85
00401ea8    if (arg1[0x1d] + arg1[5] != 0x9d)
00401eaa        return 0
00401eaa
00401ed1    if (arg1[0xc] - arg1[9] == 7)
00401eda        return 1
00401eda
00401ed3    return 0
```

From now, we can see that we can solve this challenge with a solver. In this case, we will use Z3. The following code can be derived manually, or easily obtained by sending the content of `check` to an LLM, along with a convenient prompt.

```python
from z3 import *

s = [BitVec(f's{i}', 8) for i in range(32)]

solver = Solver()

def b(i):
    return s[i]

solver.add((b(19) & b(23)) == 72)
solver.add((b(24) ^ b(15)) == 49)
solver.add(b(30) - b(0) == 17)
solver.add(b(31) * b(12) == 0x90)
solver.add(b(26) * b(10) == 0)
solver.add(b(29) * b(13) == 114)
solver.add(b(17) + b(14) == 125)
solver.add(b(28) - b(27) == 0xE8)
solver.add((b(1) | b(21)) == 120)
solver.add((b(2) ^ b(22)) == 66)
solver.add((b(20) ^ b(3)) == 39)
solver.add((b(8) ^ b(9)) == 45)
```

```
solver.add(b(25) * b(18) == 0xC0)
solver.add((b(16) | b(7)) == 122)
solver.add((b(6) ^ b(11)) == 20)
solver.add(b(5) + b(4) == 0xB0)
solver.add((b(21) ^ b(28)) == 10)
solver.add(b(4) + b(23) == 0xB1)
solver.add(b(9) - b(19) == 0xF2)
solver.add((b(10) & b(1)) == 64)
solver.add((b(12) & b(0)) == 104)
solver.add(b(5) + b(18) == 0xAE)
solver.add((b(27) | b(24)) == 122)
solver.add((b(22) | b(30)) == 125)
solver.add((b(7) & b(13)) == 112)
solver.add(b(16) - b(17) == 2)
solver.add(b(8) * b(3) == 96)
solver.add(b(26) * b(6) == 104)
solver.add(b(11) - b(2) == 63)
solver.add((b(25) | b(29)) == 70)
solver.add(b(15) + b(14) == 120)
solver.add(b(20) + b(31) == 0xC9)
solver.add(b(22) * b(7) == 96)
solver.add((b(25) ^ b(16)) == 18)
solver.add(b(13) + b(12) == 0xDB)
solver.add(b(24) * b(14) == 114)
solver.add(b(6) - b(10) == 33)
solver.add(b(18) * b(23) == 0xE8)
solver.add((b(29) | b(0)) == 110)
solver.add(b(8) + b(15) == 0x97)
solver.add((b(30) | b(4)) == 121)
solver.add((b(1) & b(31)) == 80)
solver.add(b(28) - b(26) == 0xDA)
solver.add(b(11) + b(20) == 0xE4)
solver.add(b(21) - b(17) == 0xF8)
solver.add(b(5) - b(27) == 0xFD)
solver.add((b(9) ^ b(3)) == 41)
solver.add((b(2) ^ b(19)) == 89)
solver.add((b(23) & b(29)) == 64)
solver.add(b(3) + b(17) == 0x98)
solver.add(b(7) - b(28) == 54)
solver.add((b(31) | b(22)) == 126)
solver.add(b(27) + b(26) == 0xC2)
solver.add((b(16) | b(5)) == 87)
solver.add(b(25) * b(14) == 64)
solver.add(b(8) + b(10) == 0x8C)
solver.add((b(19) ^ b(6)) == 14)
solver.add(b(13) + b(20) == 0xE2)
```

```
solver.add((b(30) & b(21)) == 72)
solver.add(b(2) - b(1) == 0xC6)
solver.add((b(15) & b(11)) == 65)
solver.add(b(9) * b(4) == 0xB9)
solver.add(b(18) + b(0) == 0xBF)
solver.add(b(12) * b(24) == 0x90)
solver.add((b(24) & b(11)) == 112)
solver.add((b(30) & b(0)) == 104)
solver.add(b(8) * b(20) == 0xF4)
solver.add((b(14) | b(1)) == 125)

solver.add(b(5) + b(29) == 0x9D)
solver.add(b(12) - b(9) == 7)

for i in range(32):
    solver.add(b(i) >= 32, b(i) <= 126)

if solver.check() == sat:
    model = solver.model()
    result = ''.join(chr(model[b(i)].as_long()) for i in range(32))
    print("Found valid input:")
    print(result)
    print("Bytes:", [model[b(i)].as_long() for i in range(32)])
else:
    print("No solution found.")
```

Running this scripts gives the key `hp6HYWaxLa@uhs-KRPWooHtXz@hZBFyZ` which we can then supply to the program and get the flag:

```
$ ./AmanoIwato
hp6HYWaxLa@uhs-KRPWooHtXz@hZBFyZ
Amano-Iwato has opened...
flag: ICC{Ame-no-Tajikarao_threw_away_the_rock_609e6de7256999bc}
```