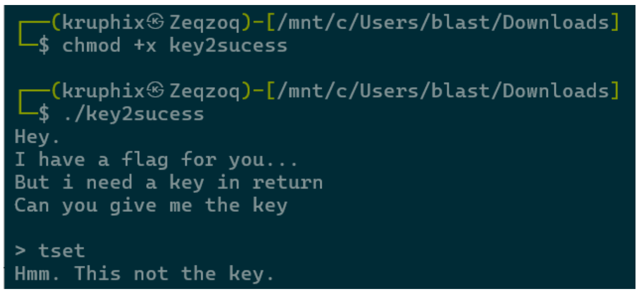
# Zeqzoq Learning Reverse Engineering with Kasimir123 writeup

https://github.com/Kasimir123/CTFWriteUps

# Reverse Engineering Noob to Pro

## Trivial

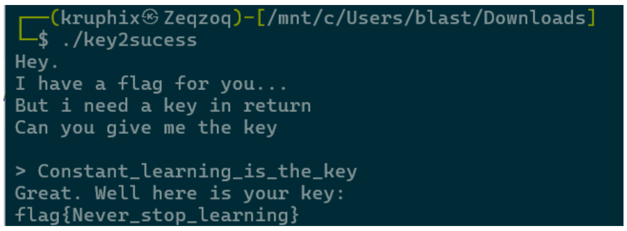
### Key2success



```
┌──(kruphix㉿Zeqzoq)-[/mnt/c/Users/blast/Downloads]
└─$ chmod +x key2sucess

┌──(kruphix㉿Zeqzoq)-[/mnt/c/Users/blast/Downloads]
└─$ ./key2sucess
Hey.
I have a flag for you...
But i need a key in return
Can you give me the key

> tset
Hmm. This not the key.
```

Found potential password in strings which will be the key.

```
┌──(kruphix㉿Zeqzoq)-[/mnt/c/Users/blast/Downloads]
└─$ strings key2sucess
...
u/UH
flag{NevH
er_stop_H
learningH
[]A\A]A^A_
Constant_learning_is_the_key
Hey.
I have a flag for you...
But i need a key in return
Can you give me the key
Great. Well here is your key:
Hmm. This not the key.
;*3$"
GCC: (Debian 9.3.0-18) 9.3.0
crtstuff.c
deregister_tm_clones
...
```



```
┌──(kruphix㉿Zeqzoq)-[/mnt/c/Users/blast/Downloads]
└─$ ./key2sucess
Hey.
I have a flag for you...
But i need a key in return
Can you give me the key

> Constant_learning_is_the_key
Great. Well here is your key:
flag{Never_stop_learning}
```

Flag: flag{Never_stop_learning}

## Unchallenging

```
┌──(kruphix㉿Zeqzoq)-[/mnt/c/Users/blast/Downloads]
└─$ strings unchallenging
...
What is the password?
op3n_se5ame
{Ar@b1an_night5}
Wrong!!
;*3$"
GCC: (Debian 9.3.0-18) 9.3.0
crtstuff.c
...
```

Flag: {Ar@b1an_night5}

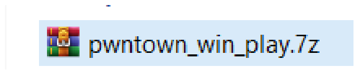
## Warm-up-rev

```
┌──(kruphix㉿Zeqzoq)-[/mnt/c/Users/blast/Downloads]
└─$ strings intro
...
you need patience to get the flag.
{steppingstone}
;*3$"
...
```

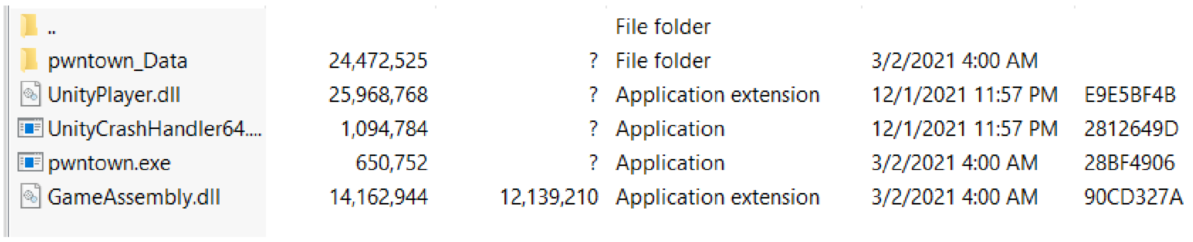
Flag: {steppingstone}

## Pwntown-1

Got a 7z archive

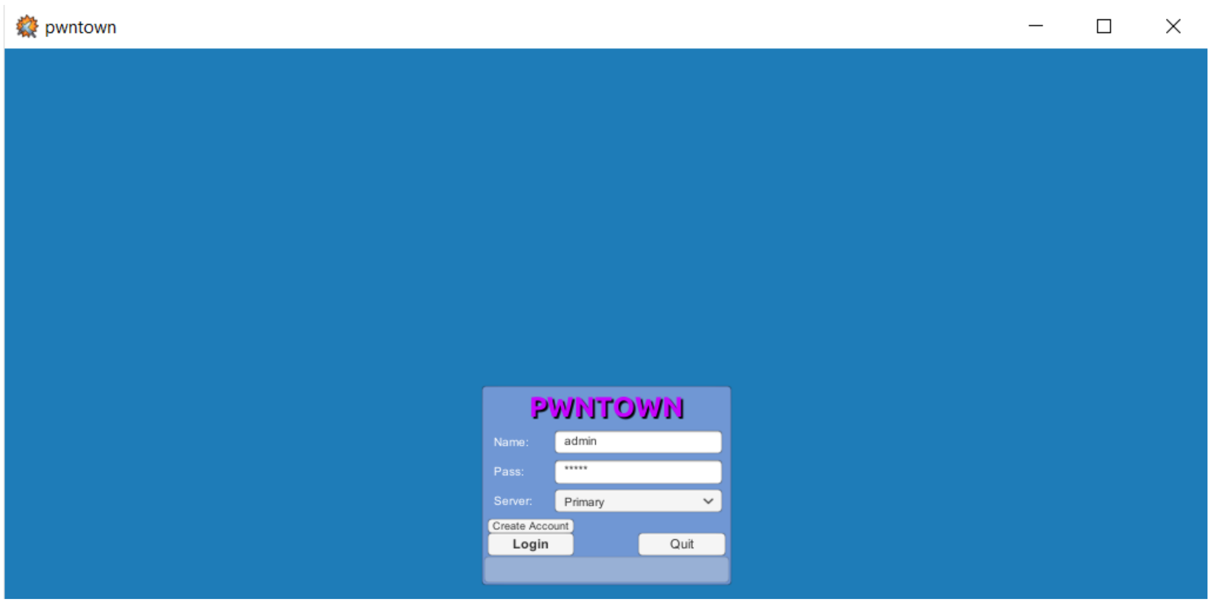

pwntown_win_play.7z

Which contain exe file

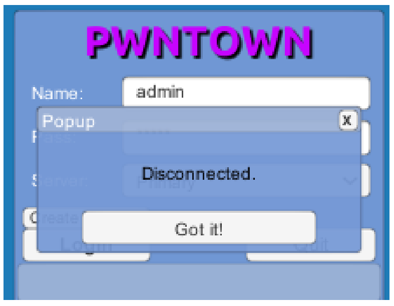


| | | | | | |
|---|---|---|---|---|---|
| .. | | | File folder | | |
| pwntown_Data | 24,472,525 | ? | File folder | 3/2/2021 4:00 AM | |
| UnityPlayer.dll | 25,968,768 | ? | Application extension | 12/1/2021 11:57 PM | E9E5BF4B |
| UnityCrashHandler64.... | 1,094,784 | ? | Application | 12/1/2021 11:57 PM | 2812649D |
| pwntown.exe | 650,752 | ? | Application | 3/2/2021 4:00 AM | 28BF4906 |
| GameAssembly.dll | 14,162,944 | 12,139,210 | Application extension | 3/2/2021 4:00 AM | 90CD327A |

Double click the pwntown.exe to start the game



Can't press login. Can't login to game nvm



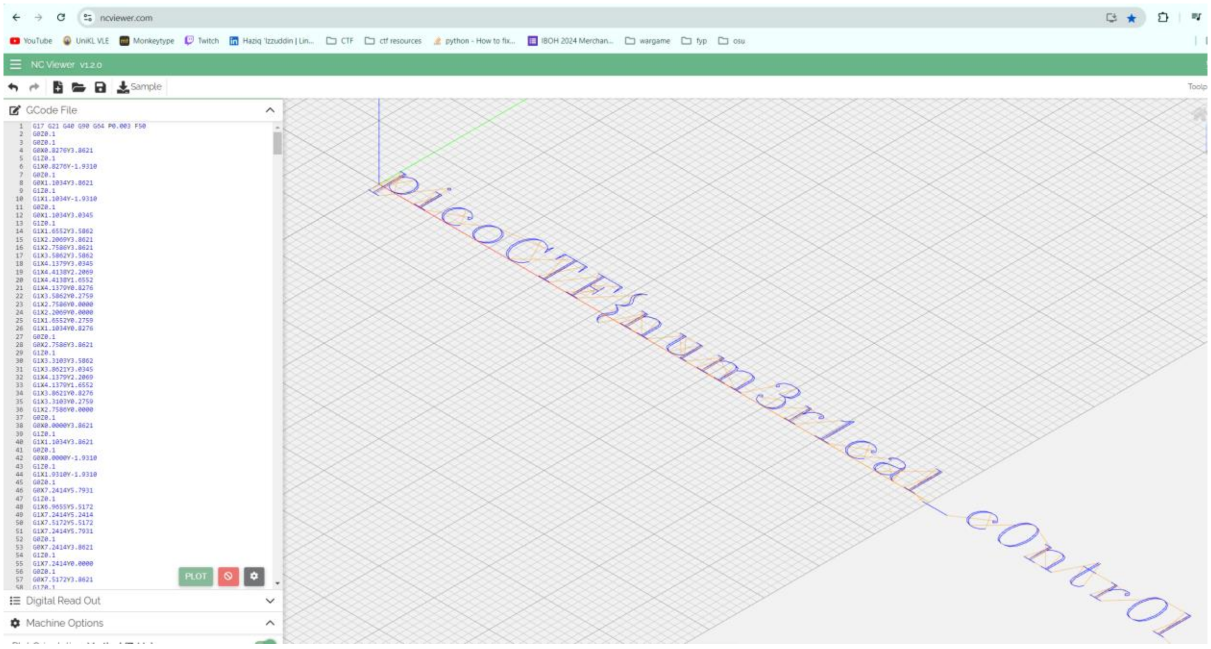
Disconnected.

## Speeds and feeds

Nc showing bunch of gcode

```
┌──(kruphix㉿Zeqzoq)-[/mnt/c/Users/blast/Downloads]
└─$ nc mercury.picoctf.net 33596
G17 G21 G40 G90 G64 P0.003 F50
G0Z0.1
G0Z0.1
G0X0.8276Y3.8621
G1Z0.1
G1X0.8276Y-1.9310
G0Z0.1
G0X1.1034Y3.8621
G1Z0.1
G1X1.1034Y-1.9310
G0Z0.1
G0X1.1034Y3.0345
G1Z0.1
...
```

Using this website. https://ncviewer.com/ copy and paste all the code and got flag

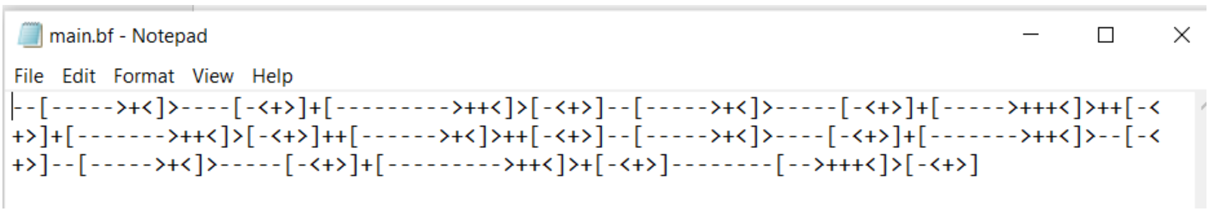


```
G17 G21 G40 G90 G64 P0.003 F50
G0Z0.1
G0Z0.1
G0X0.8276Y3.8621
G1Z0.1
G1X0.8276Y-1.9310
G0Z0.1
G0X1.1034Y3.8621
G1Z0.1
G1X1.1034Y-1.9310
G0Z0.1
G0X1.1034Y3.0345
G1Z0.1
G1X1.6552Y3.5862
G1X2.2069Y3.8621
G1X2.7586Y3.8621
G1X3.5862Y3.5862
G1X4.1379Y3.0345
G1X4.4138Y2.2069
G1X4.4138Y1.6552
G1X4.1379Y0.8276
G1X3.5862Y0.2759
G1X2.7586Y0.0000
G1X2.2069Y0.0000
G1X1.6552Y0.2759
G1X1.1034Y0.8276
G0Z0.1
G0X2.7586Y3.8621
G1Z0.1
G1X3.3103Y3.5862
G1X3.8621Y3.0345
G1X4.1379Y2.2069
G1X4.1379Y1.6552
G1X3.8621Y0.8276
G1X3.3103Y0.2759
G1X2.7586Y0.0000
G0Z0.1
G0X0.0000Y3.8621
G1Z0.1
G1X1.1034Y3.8621
G0Z0.1
G0X0.0000Y-1.9310
G1Z0.1
G1X1.9310Y-1.9310
G0Z0.1
G0X7.2414Y5.7931
G1Z0.1
G1X6.9655Y5.5172
G1X7.2414Y5.2414
G1X7.5172Y5.5172
G1X7.2414Y5.7931
G0Z0.1
G0X7.2414Y3.8621
G1Z0.1
G1X7.2414Y0.0000
G0Z0.1
G0X7.5172Y3.8621
```

Flag: picoCTF{num3r1cal_c0ntr0l_e7749028}

## BF means best friend, right?

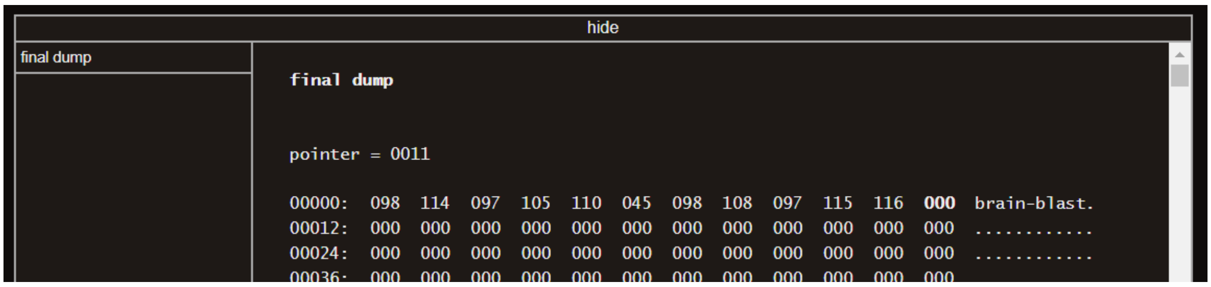
Got bf file which means brainfuck



```
--[----->+<]>----[-<+>]+[--------->++<]>[-<+>]--[----->+<]>-----[-<+>]+[----->+++<]>++[-<
+>]+[------->++<]>[-<+>]++[------>+<]>++[-<+>]--[----->+<]>----[-<+>]+[------->++<]>--[-<
+>]--[----->+<]>-----[-<+>]+[--------->++<]>+[-<+>]--------[-->+++<]>[-<+>]
```

Using this website https://copy.sh/brainfuck/ copy and paste and run

Then look at the view memory -> final dump



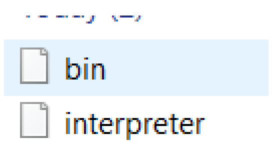
```
final dump


pointer = 0011

00000:  098  114  097  105  110  045  098  108  097  115  116  000  brain-blast.
00012:  000  000  000  000  000  000  000  000  000  000  000  000  ............
00024:  000  000  000  000  000  000  000  000  000  000  000  000  ............
00036:  000  000  000  000  000  000  000  000  000  000  000  000
```

And got flag

Flag: flag{brain-blast}

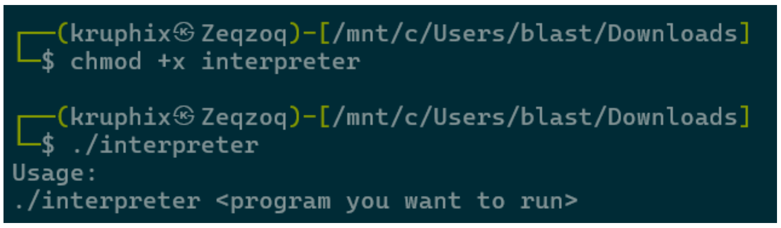
## Tiny Interpreter

Got two file



```
bin
interpreter
```

Tried to run interpreter



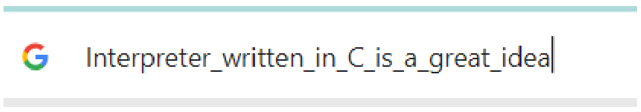
```
┌──(kruphix㉿Zeqzoq)-[/mnt/c/Users/blast/Downloads]
└─$ chmod +x interpreter

┌──(kruphix㉿Zeqzoq)-[/mnt/c/Users/blast/Downloads]
└─$ ./interpreter
Usage:
./interpreter <program you want to run>
```

Based on usage, run again like so

```
┌──(kruphix㉿Zeqzoq)-[/mnt/c/Users/blast/Downloads]
└─$ ./interpreter bin
I
n
t
e
r
p
r
e
t
e
r
_
w
r
i
t
t
e
n
_
i
n
_
C
_
i
s
_
a
_
g
r
e
a
t
_
i
d
e
```

```
a
```

Copy the strings and paste at browser. Browser will make it in one line. Make sure to copy again


Interpreter_written_in_C_is_a_great_idea

dctf{Interpreter_written_in_C_is_a_great_idea}

## Little Mountain

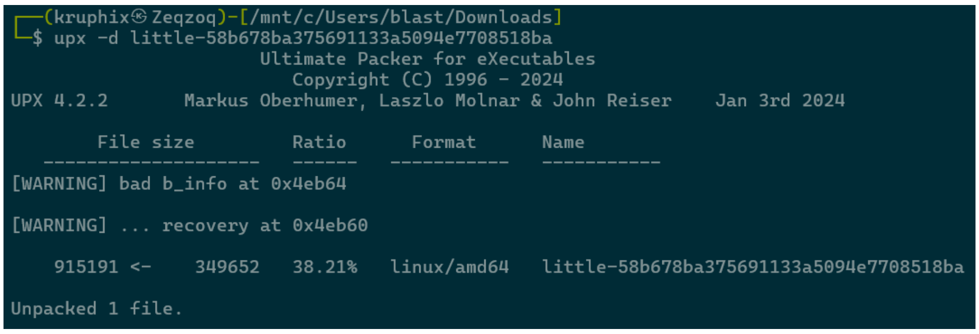
Got this file

little-58b678ba375691133a5094e7708518ba    24/9/2024 3:58 AM    File    342 KB

From strings we know its packed with upx.

```
┌──(kruphix㉿Zeqzoq)-[/mnt/c/Users/blast/Downloads]
└─$ strings little-58b678ba375691133a5094e7708518ba
UPX!
...
PROT_EXEC|PROT_WRITE failed.
$Info: This file is packed with the UPX executable packer http://upx.sf.net $
$Id: UPX 3.96 Copyright (C) 1996-2020 the UPX Team. All Rights Reserved. $
...
UPX!
UPX!
```

Use upx to unpacked



```
┌──(kruphix㉿Zeqzoq)-[/mnt/c/Users/blast/Downloads]
└─$ upx -d little-58b678ba375691133a5094e7708518ba
                       Ultimate Packer for eXecutables
                          Copyright (C) 1996 - 2024
UPX 4.2.2       Markus Oberhumer, Laszlo Molnar & John Reiser    Jan 3rd 2024

        File size         Ratio      Format      Name
   --------------------   ------   -----------   -----------
[WARNING] bad b_info at 0x4eb64

[WARNING] ... recovery at 0x4eb60

    915191 <-    349652   38.21%   linux/amd64   little-58b678ba375691133a5094e7708518ba

Unpacked 1 file.
```

Open IDA

The main function:

```
int __fastcall __noreturn main(int argc, const char **argv, const char **envp)
{
 int v3; // edx
 int v4; // ecx
 int v5; // r8d
 int v6; // r9d
 char v7; // [rsp+0h] [rbp-10h]
 int v8; // [rsp+Ch] [rbp-4h] BYREF

 setabuf(argc, argv, envp);
 while ( 1 )
 {
  puts("Option 0: Guess the number");
  puts("Option 1: Change the number");
  puts("Option 2: Exit");
  _isoc99_scanf((unsigned int)"%d", (unsigned int)&v8, v3, v4, v5, v6, v7);
  ((void (*)(void))funcs[v8])();
 }
}
```

The setabuf function:

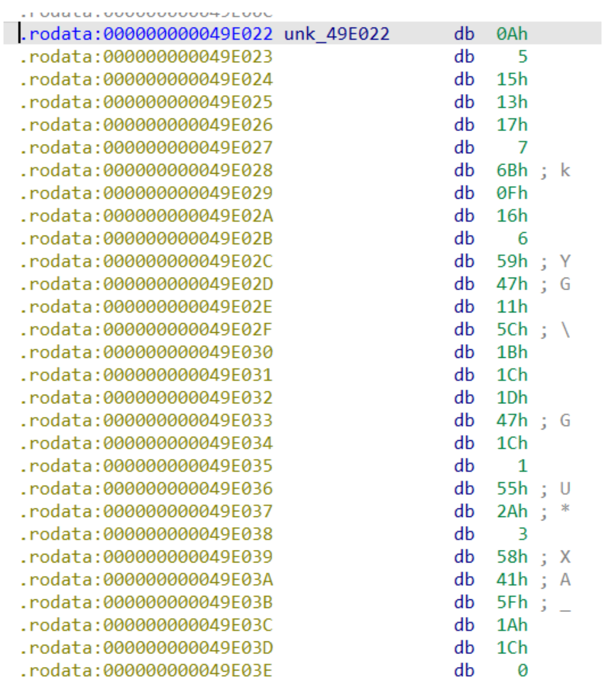
```
void (__fastcall __noreturn *setabuf())()
```

```
{
  void (__fastcall __noreturn *result)(); // rax

  srandom(1337LL);
  magic = random();
  funcs[0] = (__int64)a;
  qword_4CC348 = (__int64)b;
  qword_4CC350 = (__int64)c;
  result = d;
  qword_4CC358 = (__int64)d;
  return result;
}
```

Followed the last function pointer (the d) to:

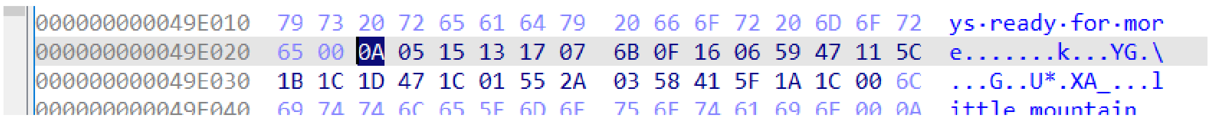
```
void __noreturn d()
{
  char v0; // [rsp+Fh] [rbp-21h] BYREF
  int v1; // [rsp+10h] [rbp-20h]
  int v2; // [rsp+14h] [rbp-1Ch]
  const char *v3; // [rsp+18h] [rbp-18h]
  _BYTE *v4; // [rsp+20h] [rbp-10h]
  int v5; // [rsp+28h] [rbp-8h]
  int i; // [rsp+2Ch] [rbp-4h]

  v4 = &unk_49E022;
  v3 = "little_mountain";
  v2 = j_strlen_ifunc(&unk_49E022);
  v1 = j_strlen_ifunc("little_mountain");
  v5 = 0;
  if ( modded == 20 )
  {
    for ( i = 0; i < v2; ++i )
    {
      if ( v5 == v1 )
        v5 = 0;
      v0 = v4[i] ^ v3[v5++];
      write(1LL, &v0, 1LL);
    }
    puts("\n");
  }
  exit(0LL);
}
```

When clicking &unk_49E022 we go to



```
.rodata:000000000049E022 unk_49E022      db  0Ah
.rodata:000000000049E023                 db    5
.rodata:000000000049E024                 db  15h
.rodata:000000000049E025                 db  13h
.rodata:000000000049E026                 db  17h
.rodata:000000000049E027                 db    7
.rodata:000000000049E028                 db  6Bh ; k
.rodata:000000000049E029                 db  0Fh
.rodata:000000000049E02A                 db  16h
.rodata:000000000049E02B                 db    6
.rodata:000000000049E02C                 db  59h ; Y
.rodata:000000000049E02D                 db  47h ; G
.rodata:000000000049E02E                 db  11h
.rodata:000000000049E02F                 db  5Ch ; \
.rodata:000000000049E030                 db  1Bh
.rodata:000000000049E031                 db  1Ch
.rodata:000000000049E032                 db  1Dh
.rodata:000000000049E033                 db  47h ; G
.rodata:000000000049E034                 db  1Ch
.rodata:000000000049E035                 db    1
.rodata:000000000049E036                 db  55h ; U
.rodata:000000000049E037                 db  2Ah ; *
.rodata:000000000049E038                 db    3
.rodata:000000000049E039                 db  58h ; X
.rodata:000000000049E03A                 db  41h ; A
.rodata:000000000049E03B                 db  5Fh ; _
.rodata:000000000049E03C                 db  1Ah
.rodata:000000000049E03D                 db  1Ch
.rodata:000000000049E03E                 db    0
```

But we can go to Hex View 1 and it will highlight it



```
000000000049E010  79 73 20 72 65 61 64 79  20 66 6F 72 20 6D 6F 72  ys·ready·for·mor
000000000049E020  65 00 0A 05 15 13 17 07  6B 0F 16 06 59 47 11 5C  e.......k...YG.\
000000000049E030  1B 1C 1D 47 1C 01 55 2A  03 58 41 5F 1A 1C 00 6C  ...G..U*.XA_...l
000000000049E040  69 74 74 6C 65 5F 6D 6F  75 6E 74 61 69 6E 00 0A  ittle mountain
```

Them Kasimir123 do this script to get flag

```
# initialize variables
v5 = 0
v3 = "little_mountain"
v4 = [0x0A, 0x05, 0x15, 0x13, 0x17, 0x07, 0x6B, 0x0F, 0x16, 0x06, 0x59, 0x47, 0x11, 0x5C, 0x1B, 0x1C, 0x1D, 0x47,
0x1C, 0x01, 0x55, 0x2A, 0x03, 0x58, 0x41, 0x5F, 0x1A, 0x1C]
v1 = len(v3)
v2 = len(v4)

# string to store flag
s = ""

# loop through v2
for i in range(0, v2):
  # reset counter if it reaches the end
        if v5 == v1:
                v5 = 0

  # do the xor
        v0 = v4[i] ^ ord(v3[v5])
        v5 += 1

  # add the character
        s += chr(v0)

# print flag
print(s)
```

This is ChatGPT script. Both got the expected output

```
# The two key arrays
v4 = [0x0A, 0x05, 0x15, 0x13, 0x17, 0x07, 0x6B, 0x0F, 0x16, 0x06,
     0x59, 0x47, 0x11, 0x5C, 0x1B, 0x1C, 0x1D, 0x47, 0x1C, 0x01,
     0x55, 0x2A, 0x03, 0x58, 0x41, 0x5F, 0x1A, 0x1C]

v3 = "little_mountain"

# XOR each byte of v4 with the corresponding byte in v3 (looping over v3)
flag = ""
v5 = 0
v1 = len(v3)  # Length of v3

for i in range(len(v4)):
   if v5 == v1:  # If we've reached the end of v3, loop back to the start
      v5 = 0
   # XOR the current byte of v4 with the current byte of v3
   xor_byte = v4[i] ^ ord(v3[v5])
   # Append the result to the flag
   flag += chr(xor_byte)
   v5 += 1

# Output the flag
print("Flag:", flag)
```

flag{b4bys73p5upt3hm0un741n}

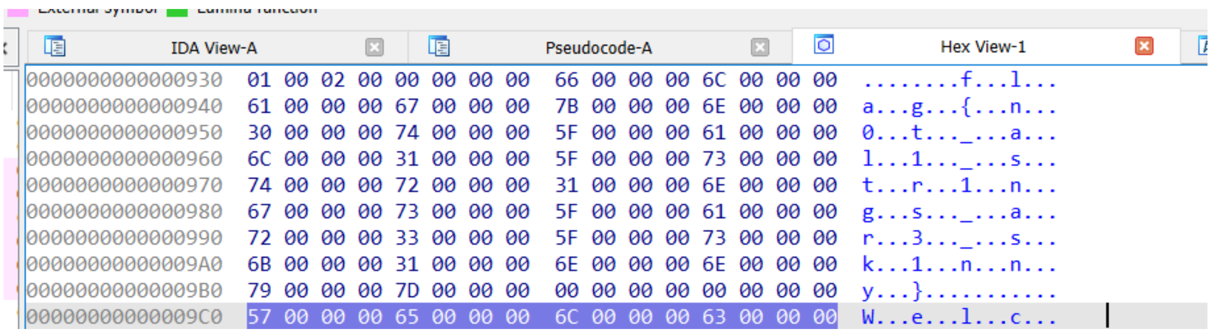
## wstrings

| | | | |
|---|---|---|---|
| wstrings | 24/9/2024 4:27 AM | File | 9 KB |

Open IDA. The main function:

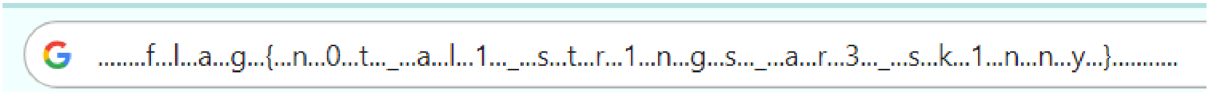
```
int __fastcall main(int argc, const char **argv, const char **envp)
{
  wchar_t ws[82]; // [rsp+0h] [rbp-150h] BYREF
  unsigned __int64 v5; // [rsp+148h] [rbp-8h]

  v5 = __readfsqword(0x28u);
  wprintf(U"Welcome to flag checker 1.0.\nGive me a flag> ", argv, envp);
  fgetws(ws, 80, stdin);
  if ( !wcscmp(flag, ws) )
    fputws(U"Correct!", stdout);
  return 0;
}
```

Open Hex view

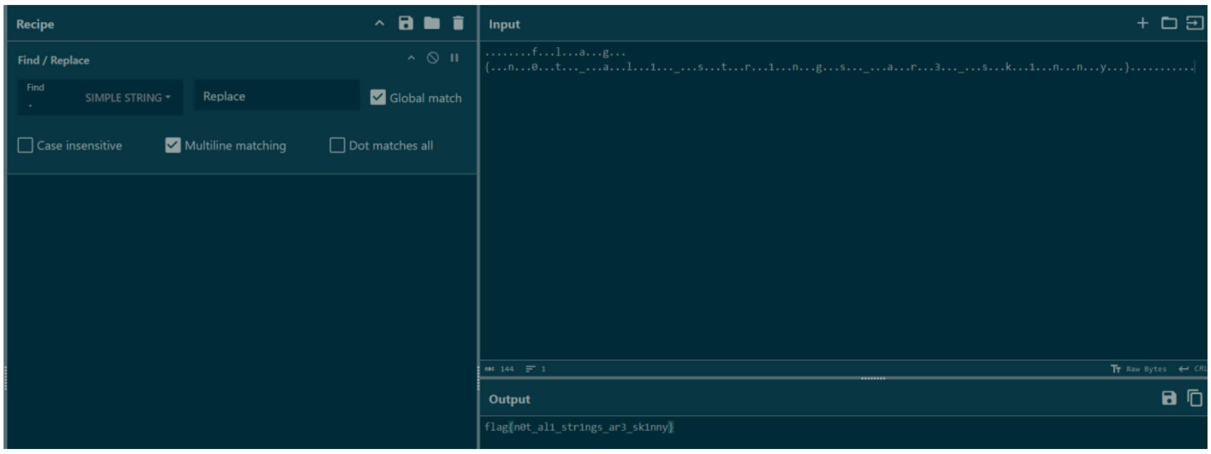


```
0000000000000930  01 00 02 00 00 00 00 00  66 00 00 00 6C 00 00 00  ........f...l...
0000000000000940  61 00 00 00 67 00 00 00  7B 00 00 00 6E 00 00 00  a...g...{...n...
0000000000000950  30 00 00 00 74 00 00 00  5F 00 00 00 61 00 00 00  0...t..._...a...
0000000000000960  6C 00 00 00 31 00 00 00  5F 00 00 00 73 00 00 00  l...1..._...s...
0000000000000970  74 00 00 00 72 00 00 00  31 00 00 00 6E 00 00 00  t...r...1...n...
0000000000000980  67 00 00 00 73 00 00 00  5F 00 00 00 61 00 00 00  g...s..._...a...
0000000000000990  72 00 00 00 33 00 00 00  5F 00 00 00 73 00 00 00  r...3..._...s...
00000000000009A0  6B 00 00 00 31 00 00 00  6E 00 00 00 6E 00 00 00  k...1...n...n...
00000000000009B0  79 00 00 00 7D 00 00 00  00 00 00 00 00 00 00 00  y...}...........
00000000000009C0  57 00 00 00 65 00 00 00  6C 00 00 00 63 00 00 00  W...e...l...c...
```

Got flag.



```
........f...l...a...g...{...n...0...t..._...a...l...1..._...s...t...r...1...n...g...s..._...a...r...3..._...s...k...1...n...n...y...}..........
```



Find: . SIMPLE STRING
Replace
[✓] Global match
[ ] Case insensitive
[✓] Multiline matching
[ ] Dot matches all

Input

```
........f...l...a...g...
{...n...0...t..._...a...l...1..._...s...t...r...1...n...g...s..._...a...r...3..._...s...k...1...n...n...y...}..........
```

Output

```
flag{n0t_al1_str1ngs_ar3_sk1nny}
```

flag{n0t_al1_str1ngs_ar3_sk1nny}

# Easy

## Thirsty Cow

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  __int64 v3; // rax
  __int64 v4; // rax
  __int64 v5; // rax
  __int64 v6; // rax
  __int64 v7; // rax
  __int64 v8; // rax
  __int64 v9; // rax
  __int64 v10; // rax
  char v12[36]; // [rsp+0h] [rbp-170h] BYREF
  char v13[9]; // [rsp+24h] [rbp-14Ch] BYREF
  char v14[3]; // [rsp+2Dh] [rbp-143h] BYREF
  char v15[4]; // [rsp+30h] [rbp-140h] BYREF
  char v16[10]; // [rsp+34h] [rbp-13Ch] BYREF
  char v17[11]; // [rsp+3Eh] [rbp-132h] BYREF
  char v18[4]; // [rsp+49h] [rbp-127h] BYREF
  char v19[3]; // [rsp+4Dh] [rbp-123h] BYREF
  char v20[4]; // [rsp+50h] [rbp-120h] BYREF
  char v21[4]; // [rsp+54h] [rbp-11Ch] BYREF
  char v22[4]; // [rsp+58h] [rbp-118h] BYREF
  char v23[4]; // [rsp+5Ch] [rbp-114h] BYREF
  char v24[268]; // [rsp+60h] [rbp-110h] BYREF

  strcpy(v23, "_si");
  strcpy(v22, "sin");
  strcpy(v21, "_5i");
  strcpy(v20, "rty");
  strcpy(v19, "th");
  strcpy(v18, "x_r");
  strcpy(&v17[7], "_th");
  strcpy(&v17[5], "x");
  strcpy(v17, "irty");
  strcpy(v16, v19);
  strcat(v16, v17);
  v3 = std::operator<<<std::char_traits<char>>(
      &std::cout,
      " The crow is thirsty and he needs your help to gather stones to fill the pot");
  v4 = std::ostream::operator<<(v3, &std::endl<char,std::char_traits<char>>);
  std::operator<<<std::char_traits<char>>(v4, " > ");
  std::operator>><char,std::char_traits<char>>(&std::cin, v24);
  strcat(v16, v23);
  strcat(v16, &v17[5]);
  if ( strcmp(v24, v16) )
  {
   v10 = std::operator<<<std::char_traits<char>>(&std::cout, " Not enough stones :( ");
   std::ostream::operator<<(v10, &std::endl<char,std::char_traits<char>>);
  }
  else
  {
   strcpy(v15, "Thi");
   strcpy(v14, "e_");
   strcpy(&v13[6], "ck");
   strcpy(&v13[4], "0");
   strcpy(v13, "p0t");
   strcpy(v12, v15);
   strcat(v12, v20);
   strcat(v12, v21);
```

```
    strcat(v12, v18);
    strcat(v12, &v13[4]);
    strcat(v12, &v13[6]);
    strcat(v12, v22);
    strcat(v12, &v17[7]);
    strcat(v12, v14);
    strcat(v12, v13);
    v5 = std::ostream::operator<<(&std::cout, &std::endl<char,std::char_traits<char>>);
    v6 = std::operator<<<std::char_traits<char>>(v5, "shadowCTF{");
    v7 = std::operator<<<std::char_traits<char>>(v6, v12);
    v8 = std::operator<<<std::char_traits<char>>(v7, "} ");
    v9 = std::ostream::operator<<(v8, &std::endl<char,std::char_traits<char>>);
    std::ostream::operator<<(v9, &std::endl<char,std::char_traits<char>>);
  }
  return 0;
}
```

V24 is out input. And it will be compared to v16 so v16 will be the flag. So we want to set breakpoint on if statement, look at the register and the flag should be there.

```
.text:00000000000012DB            call    _strcmp
.text:00000000000012E0            mov     [rbp+var_4], eax
.text:00000000000012E3            cmp     [rbp+var_4], 0
.text:00000000000012E7            jnz     loc_1499
```

In this snippet:

- call _strcmp calls the strcmp function, which returns the result in the eax register (or, as shown here, it is stored in [rbp+var_4]).
- cmp [rbp+var_4], 0 compares the result of the strcmp function with 0.
- jnz loc_1499 jumps to the failure message ("Not enough stones") if the comparison is not zero (jnz = "jump if not zero").

Based on above, we want the eax to be 0. We will set the breakpoint on .text:00000000000012E0.

We can use either gdb or radare2 in linux. I use r2 because got problem with gdb donno why.

```
┌──(kruphix㉿Zeqzoq)-[/mnt/c/Users/blast/Downloads]
└─$ r2 -d crow.out
...
[0x7fba812ea810]> aaa
...
[0x7fba812ea810]> pdf @main
...
```

Find the strcmp part in r2, which is 0x55f0e2e252e0

```
0x55f0e2e252db    e8a0fdffff    call sym.imp.strcmp    ; int strcmp(const char *s1, const char *s2)
0x55f0e2e252e0    8945fc        mov dword [var_4h], eax
0x55f0e2e252e3    837dfc00      cmp dword [var_4h], 0
0x55f0e2e252e7    0f85ac010000  jne 0x55f0e2e25499
```

Run, then set the breakpoints, change eax to 0 then continue

```
[0x7fba812ea810]> db 0x55f0e2e252e0
[0x7fba812ea810]> dc
 The crow is thirsty and he needs your help to gather stones to fill the pot
 > test
INFO: hit breakpoint at: 0x55f0e2e252e0
[0x55f0e2e252e0]> dr eax=0
0xfffffffd -> 0x00000000
[0x55f0e2e252e0]> dc

shadowCTF{Thirty_5ix_r0cksin_the_p0t}

(137) Process exited with status=0x0
```

shadowCTF{Thirty_5ix_r0cksin_the_p0t}

## Vault

When running the file, it prompt for password

```
┌──(kruphix㉿Zeqzoq)-[/mnt/c/Users/blast/Downloads]
└─$ chmod +x vault

┌──(kruphix㉿Zeqzoq)-[/mnt/c/Users/blast/Downloads]
└─$ ./vault
    .--------.
   / .------. \n  / /       \n  | |      | |
  | |      | |
 _| |________| |_
.' |_|       |_| '.
'._____ ____ _____.'
|     .'____'.     |
'.__.'.'    '.'.__.'
'.__  Shad0w |  __.'
|   '.'.____.'.'   |
'.____'.____.'____.'
'.________________.'secure vault
Enter our password:
donno
You Failed
```

Tried to use strings, but it got scambled.

```
┌──(kruphix㉿Zeqzoq)-[/mnt/c/Users/blast/Downloads]
└─$ strings vault
..
hackers_H
accef
trolf
no-cash_H
passwordH
sh@df
CongratsH
, your fH
lag is: H
{R3ver5eH
_chal1enH
ge_pwnedH
%16s
reversinH
You FailH
[]A\A]A^A_
    .--------.
   / .------. \n
  / /       \n
  | |      | |
 _| |________| |_
.' |_|       |_| '.
'._____ ____ _____.'
|     .'____'.     |
'.__.'.'    '.'.__.'
'.__  Shad0w |  __.'
|   '.'.____.'.'   |
'.____'.____.'____.'
'.________________.'
secure vault
Enter our password:
```

```
%16s
You Win
_rul3s
You Failed
…
```

Found the password (and the flag) in main function

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  char v4[13]; // [rsp+12h] [rbp-AEh] BYREF
  char v5[10]; // [rsp+1Fh] [rbp-A1h] BYREF
  char v6[5]; // [rsp+29h] [rbp-97h] BYREF
  char v7[2]; // [rsp+2Eh] [rbp-92h] BYREF
  char v8[50]; // [rsp+30h] [rbp-90h] BYREF
  char format[7]; // [rsp+69h] [rbp-57h] BYREF
  char s2[32]; // [rsp+70h] [rbp-50h] BYREF
  char v11[24]; // [rsp+90h] [rbp-30h] BYREF
  char s1[15]; // [rsp+ADh] [rbp-13h] BYREF
  int i; // [rsp+BCh] [rbp-4h]

  strcpy(s1, "hackers_access");
  for ( i = 10; i <= 19; ++i )
    ;
  strcpy(v11, "no-cash_password");
  strcpy(format, "sh@d0w");
  puts("    .--------.");
  printf("   / .------. \\n");
  printf("  / /        \\n");
  puts("  | |       | |");
  puts("  | |       | |");
  puts(" _| |________| |_");
  puts(".' |_|       |_| '.");
  puts("'._____ ____ _____.'");
  puts("|    .'____'.    |");
  puts("'.__.'   '.__.'");
  puts("'.__  Shad0w |  __.'");
  puts("|  '.____.'  |");
  puts("'.____'.____.'____.'");
  printf("'.________________.'");
  strcpy(v8, "Congrats, your flag is: {R3ver5e_chal1enge_pwned}");
  strcpy(v7, "{");
  strcpy(v6, "%16s");
  strcpy(v5, "reversing");
  strcpy(&v4[2], "You Failed");
  strcpy(v4, "}");
  puts("secure vault");
  puts("Enter our password:");
  __isoc99_scanf("%16s", s2);
  if ( !strcmp(s1, s2) )
  {
    puts("You Win");
    printf(format);
    printf(v7);
    printf(format);
    putchar(95);
    printf(v5);
    printf("_rul3s");
    printf(v4);
  }
  else
  {
    puts("You Failed");
  }
  return 0;
}
```

```
┌──(kruphix㉿Zeqzoq)-[/mnt/c/Users/blast/Downloads]
└─$ ./vault
    .--------.
   / .------. \n  / /      \n  | |      | |
  | |      | |
 _| |________| |_
.' |_|      |_| '.
'._____ ____ _____.'
|    .'____'.    |
'.__.'    '.__.'
'.__  Shad0w |  __.'
|   '.'____.'   |
'.____'.____'.____.'
'.________________.'secure vault
Enter our password:
hackers_access
You Win
sh@d0w{sh@d0w_reversing_rul3s}
```

sh@d0w{sh@d0w_reversing_rul3s}

## Little Baby Rev (takreti lagi)

Tried to run the file, but number 1-10 is wrong

```
┌──(kruphix㉿Zeqzoq)-[/mnt/c/Users/blast/Downloads]
└─$ ./warmup
I have a number from 1 to 10, what is it?
1
Try again
I have a number from 1 to 10, what is it?
2
Try again
I have a number from 1 to 10, what is it?
3
Try again
...
I have a number from 1 to 10, what is it?
^CTraceback (most recent call last)
/root/rev/warmup.nim(39) warmup
/root/Nim/lib/system/io.nim(491) readLine
/root/Nim/lib/system/io.nim(453) readLine
SIGINT: Interrupted by Ctrl-C.
```

Notice that when we ctrl c it said to run in warmup.nim file, but or file name is only warmup.

The main function:

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  cmdLine = (__int64)argv;
  cmdCount = argc;
  gEnv = (__int64)envp;
  NimMain();
  return nim_program_result;
}
```

But this is not what we are looking for because when we run the file, it put us in a loop. So we want to search for any looping code. Then found in main()-> NimMain() -> NimMainInner() -> NimMainModule()

```
__int64 NimMainModule()
{
  __int64 v0; // rax
  __int64 Line__IfmAdseskhTUnfEYpOo5fA; // rax
  char v3[8]; // [rsp+0h] [rbp-50h] BYREF
  const char *v4; // [rsp+8h] [rbp-48h]
  __int64 v5; // [rsp+10h] [rbp-40h]
  const char *v6; // [rsp+18h] [rbp-38h]
  __int16 v7; // [rsp+20h] [rbp-30h]
  __int64 v8; // [rsp+38h] [rbp-18h] BYREF
  __int64 v9; // [rsp+40h] [rbp-10h] BYREF
  __int64 v10; // [rsp+48h] [rbp-8h] BYREF

  nimRegisterGlobalMarker(TM__ijE9cayl8YPnol3rizbiT0g_2);
  nimRegisterGlobalMarker(TM__ijE9cayl8YPnol3rizbiT0g_6);
  v4 = "warmup";
  v6 = "/root/rev/warmup.nim";
  v5 = 0LL;
  v7 = 0;
  nimFrame_1(v3);
```

```
 v5 = 36LL;
 v6 = "/root/rev/warmup.nim";
 v0 = decodeStr__R9b5IlyQjG2mcdkp9a67LGTQ("!", 1025579140LL);
 asgnRef_1(&answer__2bKjAtEJJ5cp19bpmXcStjQ, v0);
 v5 = 37LL;
 v6 = "/root/rev/warmup.nim";
 while ( 1 )
 {
  v5 = 38LL;
  v6 = "/root/rev/warmup.nim";
  nimZeroMem_0(&v10, 8LL);
  v10 = decodeStr__R9b5IlyQjG2mcdkp9a67LGTQ(&TM__ijE9cayl8YPnol3rizbiT0g_5, 1058641868LL);
  echoBinSafe(&v10, 1LL);
  asgnRef_1(&guess__62AlRyOQv9cCViqvgl14ssA, 0LL);
  v5 = 39LL;
  v6 = "/root/rev/warmup.nim";
  Line__IfmAdseskhTUnfEYpOo5fA = readLine__IfmAdseskhTUnfEYpOo5fA(stdin);
  asgnRef_1(&guess__62AlRyOQv9cCViqvgl14ssA, Line__IfmAdseskhTUnfEYpOo5fA);
  v5 = 41LL;
  v6 = "/root/rev/warmup.nim";
  if ( (unsigned __int8)eqStrings(guess__62AlRyOQv9cCViqvgl14ssA, answer__2bKjAtEJJ5cp19bpmXcStjQ) )
   break;
  v5 = 42LL;
  v6 = "/root/rev/warmup.nim";
  nimZeroMem_0(&v9, 8LL);
  v9 = decodeStr__R9b5IlyQjG2mcdkp9a67LGTQ(&TM__ijE9cayl8YPnol3rizbiT0g_7, 558495268LL);
  echoBinSafe(&v9, 1LL);
 }
 v5 = 44LL;
 v6 = "/root/rev/warmup.nim";
 nimZeroMem_0(&v8, 8LL);
 v8 = decodeStr__R9b5IlyQjG2mcdkp9a67LGTQ(&TM__ijE9cayl8YPnol3rizbiT0g_8, 191789740LL);
 echoBinSafe(&v8, 1LL);
 v5 = 50LL;
 v6 = "/root/rev/warmup.nim";
 return popFrame_1();
}
```

## Rocca Pia

The main function:

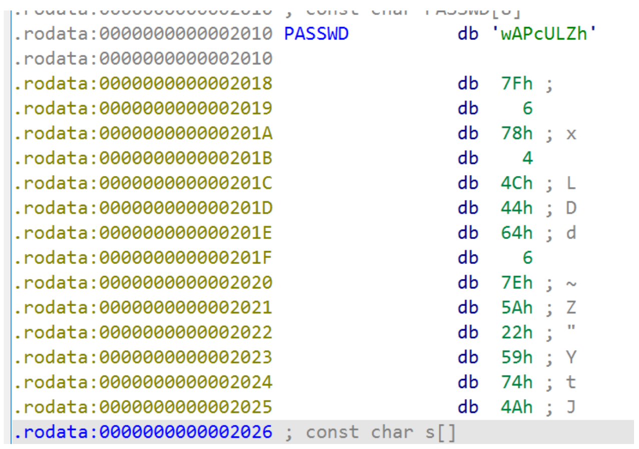
```
int __cdecl main(int argc, const char **argv, const char **envp)
{
 if ( argc == 2 )
 {
  if ( (unsigned int)transform(argv[1], argv, envp) )
   puts("Nice try");
  else
   puts("Nice flag");
  return 0;
 }
 else
 {
  printf("Usage: %s <password>\n", *argv);
  return 1;
 }
}
```

Go read the transform function

```
int __fastcall transform(__int64 a1)
{
 int i; // [rsp+14h] [rbp-1Ch]
 char *s2; // [rsp+18h] [rbp-18h]

 for ( i = 0; i < strlen(PASSWD); ++i )
 {
  if ( (i & 1) != 0 )
   s2[i] = *(_BYTE *)(i + a1) ^ 0x37;
  else
   s2[i] = *(_BYTE *)(i + a1) ^ 0x13;
 }
 return strncmp(PASSWD, s2, 0x16uLL);
}
```

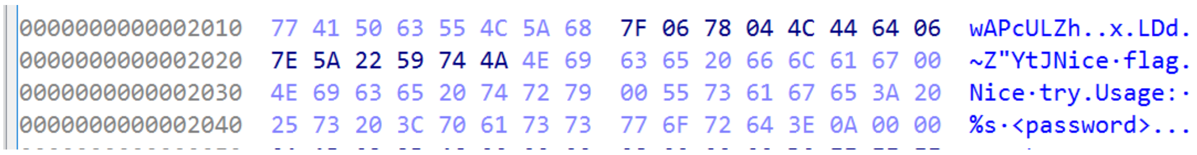
- The function transform XORs each byte of the input string (a1) with either 0x37 (for odd indices) or 0x13 (for even indices).
- It then compares the result (s2) with the known password (PASSWD), which is provided in hex.
- The password length is 22 bytes, so we need to reverse the XOR process to obtain the correct input that would pass the check.

Found PASSWD



```
.rodata:0000000000002010 PASSWD          db 'wAPcULZh'
.rodata:0000000000002010
.rodata:0000000000002018                 db  7Fh ;
.rodata:0000000000002019                 db    6
.rodata:000000000000201A                 db  78h ; x
.rodata:000000000000201B                 db    4
.rodata:000000000000201C                 db  4Ch ; L
.rodata:000000000000201D                 db  44h ; D
.rodata:000000000000201E                 db  64h ; d
.rodata:000000000000201F                 db    6
.rodata:0000000000002020                 db  7Eh ; ~
.rodata:0000000000002021                 db  5Ah ; Z
.rodata:0000000000002022                 db  22h ; "
.rodata:0000000000002023                 db  59h ; Y
.rodata:0000000000002024                 db  74h ; t
.rodata:0000000000002025                 db  4Ah ; J
.rodata:0000000000002026 ; const char s[]
```

Want to turn to char or hex. Can do it manually or look at hex view (if want to turn wAPcULZh to hex)



```
0000000000002010  77 41 50 63 55 4C 5A 68  7F 06 78 04 4C 44 64 06  wAPcULZh..x.LDd.
0000000000002020  7E 5A 22 59 74 4A 4E 69  63 65 20 66 6C 61 67 00  ~Z"YtJNice·flag.
0000000000002030  4E 69 63 65 20 74 72 79  00 55 73 61 67 65 3A 20  Nice·try.Usage:·
0000000000002040  25 73 20 3C 70 61 73 73  77 6F 72 64 3E 0A 00 00  %s·<password>...
```

Which is 77 41 50 63 55 4C 5A 68 7F 06 78 04 4C 44 64 06 7E 5A 22 59 74 4A

Then create the script to get the flag. What to reverse?

The XOR operations can be reversed by applying the same XOR value to each byte of the password (PASSWD), but with the same alternating pattern (0x13 for even indices, 0x37 for odd indices).

For each byte in PASSWD:

If the index is even, the byte was XORed with 0x13, so you can reverse it by XORing it again with 0x13.

If the index is odd, the byte was XORed with 0x37, so you can reverse it by XORing it again with 0x37.

```
# Given PASSWD in hex
passwd_hex = [0x77, 0x41, 0x50, 0x63, 0x55, 0x4C, 0x5A, 0x68,
          0x7F, 0x06, 0x78, 0x04, 0x4C, 0x44, 0x64, 0x06,
          0x7E, 0x5A, 0x22, 0x59, 0x74, 0x4A]

# Reversing the transformation
def reverse_transform(passwd_hex):
  flag = []
  for i, byte in enumerate(passwd_hex):
    if i % 2 == 0:
      # Even index -> XOR with 0x13
      flag_byte = byte ^ 0x13
    else:
      # Odd index -> XOR with 0x37
      flag_byte = byte ^ 0x37
    flag.append(chr(flag_byte))
  return ''.join(flag)

# Get the flag
flag = reverse_transform(passwd_hex)
print("Flag:", flag)
```

Explanation of the Script:

We define the password in hexadecimal (passwd_hex), which is the value of PASSWD in the code.

The reverse_transform function loops through each byte of PASSWD:

For even indices (i % 2 == 0), it XORs the byte with 0x13 (to reverse the XOR).

For odd indices (i % 2 != 0), it XORs the byte with 0x37.

After XORing, the byte is converted back to its character representation using chr().

The result is concatenated into a string and printed as the flag.

dvCTF{I_l1k3_sw1mm1ng}

## crackme-py

Got a python file

```
┌──(kruphix㉿Zeqzoq)-[/mnt/c/Users/blast/Downloads]
└─$ cat crackme.py
# Hiding this really important number in an obscure piece of code is brilliant!
# AND it's encrypted!
# We want our biggest client to know his information is safe with us.
bezos_cc_secret = "A:4@r%uL`M-^M0c0AbcM-MFE02fh3e4a5N"

# Reference alphabet
alphabet = "!\"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ"+ \
            "[\\]^_`abcdefghijklmnopqrstuvwxyz{|}~"


def decode_secret(secret):
    """ROT47 decode
    NOTE: encode and decode are the same operation in the ROT cipher family.
    """
    # Encryption key
    rotate_const = 47

    # Storage for decoded secret
    decoded = ""

    # decode loop
    for c in secret:
        index = alphabet.find(c)
        original_index = (index + rotate_const) % len(alphabet)
        decoded = decoded + alphabet[original_index]

    print(decoded)

def choose_greatest():
    """Echo the largest of the two numbers given by the user to the program

    Warning: this function was written quickly and needs proper error handling
    """

    user_value_1 = input("What's your first number? ")
    user_value_2 = input("What's your second number? ")
    greatest_value = user_value_1 # need a value to return if 1 & 2 are equal

    if user_value_1 > user_value_2:
        greatest_value = user_value_1
    elif user_value_1 < user_value_2:
        greatest_value = user_value_2

    print( "The number with largest positive magnitude is "
        + str(greatest_value) )

choose_greatest()
```

The python code already has a function that decode the encoded message. So just change choose_greatest() to decode_secret(bezos_cc_secret) (because bezos_cc_secret is the secret here) and run to got flag.

picoCTF{1|\/|_4_p34|\|ut_a79b6c2d}

## Hurry Up! Wait!

This file has many function and we do not need to run it so its better to use ghidra than IDA. But I already have IDA open but the solving should be similar.

The main function:

```
__int64 __fastcall main(int a1, char **a2, char **a3)
{
  char v4[8]; // [rsp+28h] [rbp-8h] BYREF

  gnat_argc = a1;
  gnat_argv = (__int64)a2;
  gnat_envp = (__int64)a3;
  __gnat_initialize(v4);
  sub_1D7C();
  sub_298A();
  sub_1D52();
  __gnat_finalize();
  return (unsigned int)gnat_exit_status;
}
```

All of it is obfuscated so we really don't understand. Tried to look into sub_1D7C() but I don't understand but in sub_298A() got something suspicious.

```
__int64 sub_298A()
{
  ada__calendar__delays__delay_for(1000000000000000LL);
  sub_2616();
  sub_24AA();
  sub_2372();
  sub_25E2();
  sub_2852();
  sub_2886();
  sub_28BA();
  sub_2922();
  sub_23A6();
  sub_2136();
  sub_2206();
  sub_230A();
  sub_2206();
  sub_257A();
  sub_28EE();
  sub_240E();
  sub_26E6();
  sub_2782();
  sub_28EE();
  sub_22A2();
  sub_226E();
  sub_23DA();
  sub_2206();
  sub_230A();
  sub_233E();
  sub_2136();
  return sub_2956();
}
```

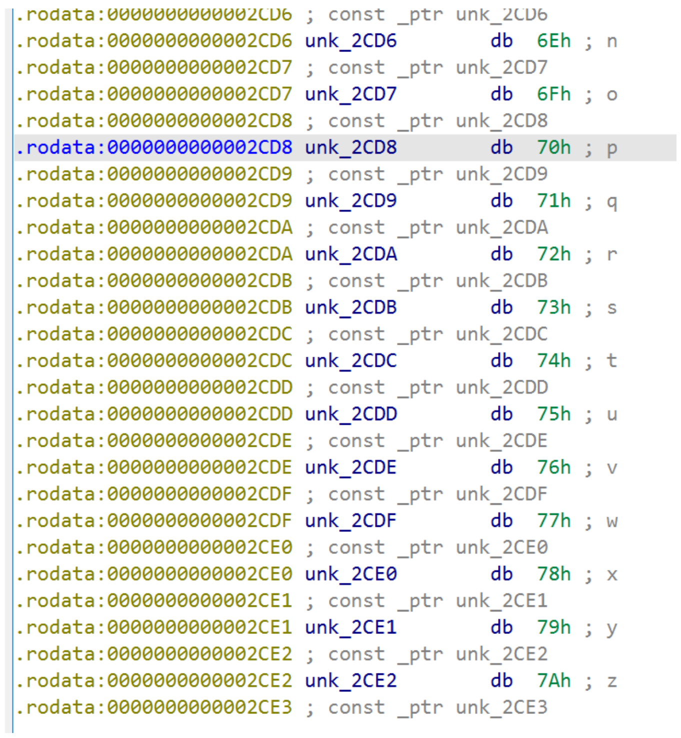
It looks like the function collect some data and return it. Maybe the flag?

```
__int64 sub_2616()
```

```
{
  return ada__text_io__put__4((_ptr *)&unk_2CD8);
}
```

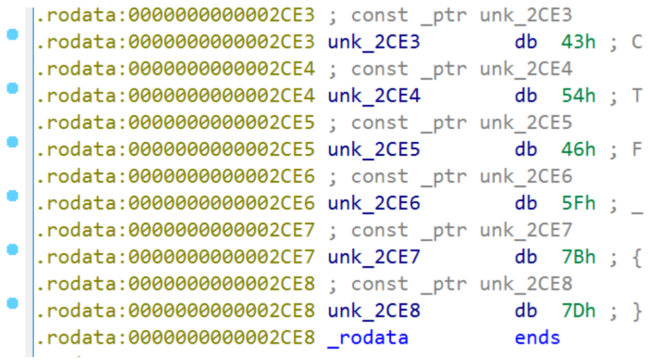
We don't need to look int ada__text_io__put__4() function. We are interested in thee data it collect. So we look into &unk_2CD8.

It brought me to the text view.



```
.rodata:0000000000002CD6 ; const _ptr unk_2CD6
.rodata:0000000000002CD6 unk_2CD6        db  6Eh ; n
.rodata:0000000000002CD7 ; const _ptr unk_2CD7
.rodata:0000000000002CD7 unk_2CD7        db  6Fh ; o
.rodata:0000000000002CD8 ; const _ptr unk_2CD8
.rodata:0000000000002CD8 unk_2CD8        db  70h ; p
.rodata:0000000000002CD9 ; const _ptr unk_2CD9
.rodata:0000000000002CD9 unk_2CD9        db  71h ; q
.rodata:0000000000002CDA ; const _ptr unk_2CDA
.rodata:0000000000002CDA unk_2CDA        db  72h ; r
.rodata:0000000000002CDB ; const _ptr unk_2CDB
.rodata:0000000000002CDB unk_2CDB        db  73h ; s
.rodata:0000000000002CDC ; const _ptr unk_2CDC
.rodata:0000000000002CDC unk_2CDC        db  74h ; t
.rodata:0000000000002CDD ; const _ptr unk_2CDD
.rodata:0000000000002CDD unk_2CDD        db  75h ; u
.rodata:0000000000002CDE ; const _ptr unk_2CDE
.rodata:0000000000002CDE unk_2CDE        db  76h ; v
.rodata:0000000000002CDF ; const _ptr unk_2CDF
.rodata:0000000000002CDF unk_2CDF        db  77h ; w
.rodata:0000000000002CE0 ; const _ptr unk_2CE0
.rodata:0000000000002CE0 unk_2CE0        db  78h ; x
.rodata:0000000000002CE1 ; const _ptr unk_2CE1
.rodata:0000000000002CE1 unk_2CE1        db  79h ; y
.rodata:0000000000002CE2 ; const _ptr unk_2CE2
.rodata:0000000000002CE2 unk_2CE2        db  7Ah ; z
.rodata:0000000000002CE3 ; const _ptr unk_2CE3
```

It map the unk_2C** to characters. I scroll a bit and found flag characters.



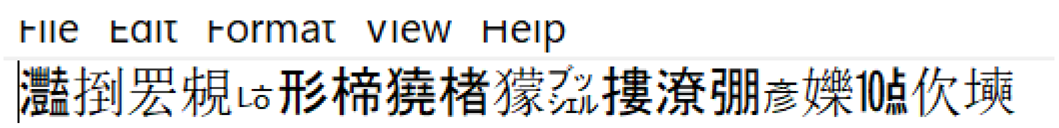
```
.rodata:0000000000002CE3 ; const _ptr unk_2CE3
.rodata:0000000000002CE3 unk_2CE3        db  43h ; C
.rodata:0000000000002CE4 ; const _ptr unk_2CE4
.rodata:0000000000002CE4 unk_2CE4        db  54h ; T
.rodata:0000000000002CE5 ; const _ptr unk_2CE5
.rodata:0000000000002CE5 unk_2CE5        db  46h ; F
.rodata:0000000000002CE6 ; const _ptr unk_2CE6
.rodata:0000000000002CE6 unk_2CE6        db  5Fh ; _
.rodata:0000000000002CE7 ; const _ptr unk_2CE7
.rodata:0000000000002CE7 unk_2CE7        db  7Bh ; {
.rodata:0000000000002CE8 ; const _ptr unk_2CE8
.rodata:0000000000002CE8 unk_2CE8        db  7Dh ; }
.rodata:0000000000002CE8 _rodata         ends
```

So now it look promising as we saw C T F _ { } so we know this is the flag. Just need to make the IDA script oor Ghidra script to extract the flag but we can just manually extract it.

picoCTF{d15a5m_ftw_87e5ab1}

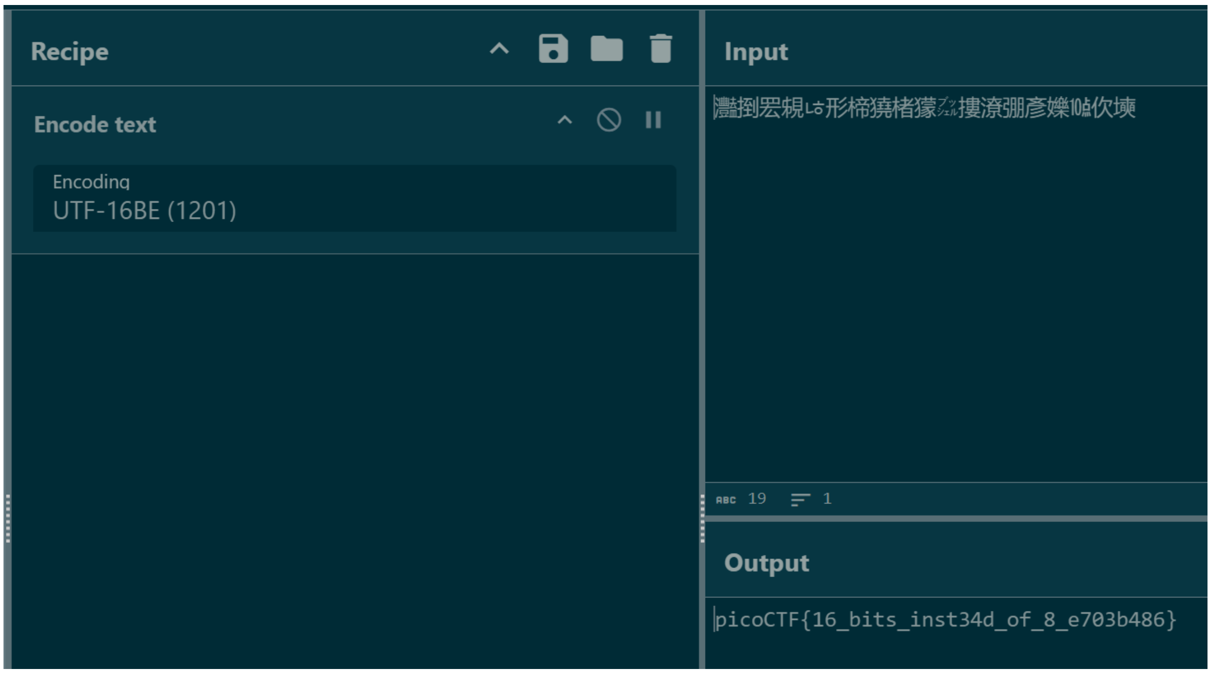
## Transformation

Got Chinese encrypted text



灩捯䍔䙻ㄶ形楴獟楮獴㌴摟潦弸彥㜰㍢㐸㙽

Went into cyberchef and use magic and got flag



Encode text

Encoding
UTF-16BE (1201)

灩捯䍔䙻ㄶ形楴獟楮獴㌴摟潦弸彥㜰㍢㐸㙽

```
picoCTF{16_bits_inst34d_of_8_e703b486}
```

picoCTF{16_bits_inst34d_of_8_e703b486}

## FREE FLAGS

When running the file:

```
┌──(kruphix㉿Zeqzoq)-[/mnt/c/Users/blast/Downloads]
└─$ ./free_flags
Congratulations! You are the 1000th CTFer!!! Fill out this short survey to get FREE FLAGS!!!
What number am I thinking of???
14
Wrong >:((((
```

The main function:

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  int v5; // [rsp+24h] [rbp-11Ch] BYREF
  int v6; // [rsp+28h] [rbp-118h] BYREF
  int v7; // [rsp+2Ch] [rbp-114h] BYREF
  char s[264]; // [rsp+30h] [rbp-110h] BYREF
  unsigned __int64 v9; // [rsp+138h] [rbp-8h]

  v9 = __readfsqword(0x28u);
  puts("Congratulations! You are the 1000th CTFer!!! Fill out this short survey to get FREE FLAGS!!!");
  puts("What number am I thinking of???");
  __isoc99_scanf("%d", &v7);
  if ( v7 == 31337 )
  {
    puts("What two numbers am I thinking of???");
    __isoc99_scanf("%d %d", &v6, &v5);
    if ( v5 + v6 == 1142
      && v5 * v6 == 302937
      && (puts("What animal am I thinking of???"),
        __isoc99_scanf(" %256s", s),
        s[strcspn(s, "\n")] = 0,
        !strcmp(s, "banana")) )
    {
      puts("Wow!!! Now I can sell your information to the Russian government!!!");
      puts("Oh yeah, here's the FREE FLAG:");
      print_flag();
      return 0;
    }
    else
    {
      puts("Wrong >:((((");
      return 1;
    }
  }
  else
  {
    puts("Wrong >:((((");
    return 1;
  }
}
```

The main function already give us the answers we need.

First input: 31337

Second input: two number x and y, when x + y =  1142 and xy = 302937

Third input: banana

```
┌──(kruphix㉿Zeqzoq)-[/mnt/c/Users/blast/Downloads]
└─$ ./free_flags
Congratulations! You are the 1000th CTFer!!! Fill out this short survey to get FREE FLAGS!!!
What number am I thinking of???
31337
What two numbers am I thinking of???
723 419
What animal am I thinking of???
banana
Wow!!! Now I can sell your information to the Russian government!!!
Oh yeah, here's the FREE FLAG:
Could not find the flag file.
Segmentation fault
```

Flag: on their server not on the file.

## GaussBot (takreti sama dengan little baby rev)

Main function:

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  unsigned int len; // [esp+0h] [ebp-10h]

  len = getpagesize();
  mprotect((char *)&code - (unsigned int)&code % len, len, 6);
  ((void (*)(void))code)();
  return 0;
}
```

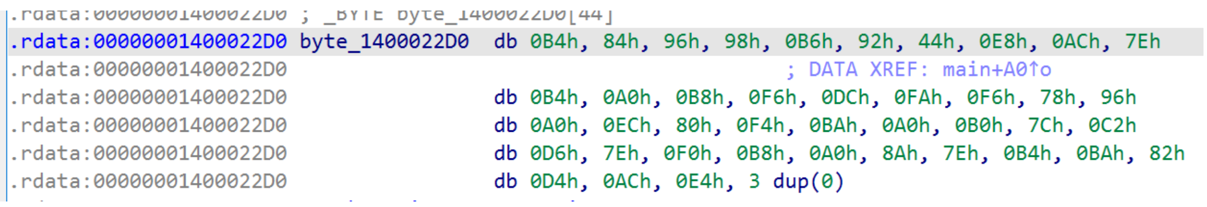
## Painting Windows

The main function:

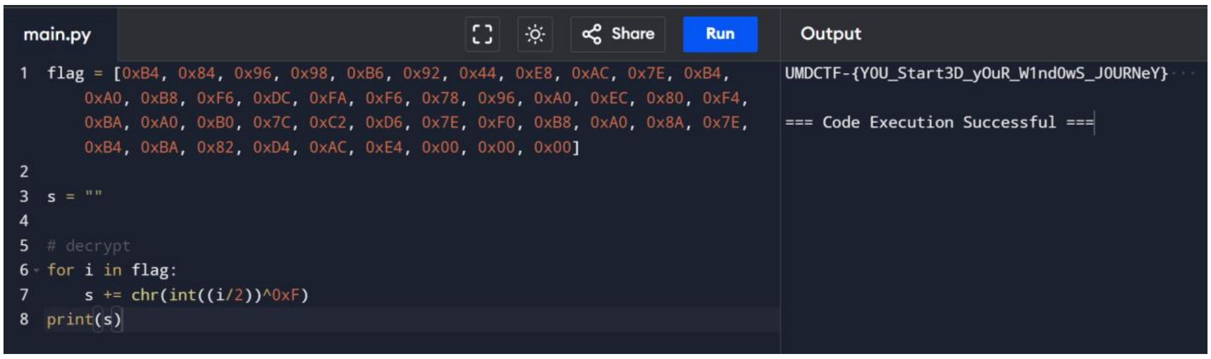
```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  __int64 v4; // r8
  int v5; // eax
  __int64 v6; // rdx
  __int64 v7; // r9
  __int64 v8; // rax
  int v9; // er8
  const char *v10; // rcx
  char String[512]; // [rsp+20h] [rbp-218h] BYREF

  if ( IsDebuggerPresent() )
  {
    printf_5("That is not allowed!\n");
    return 1;
  }
  else
  {
    printf_5("What is the password?\n");
    scanf("%s", String, v4);
    v5 = strnlen(String, 0x100ui64);
    v6 = 0i64;
    v7 = v5;
    if ( v5 <= 0 )
    {
      v9 = 0;
    }
    else
    {
      v8 = 0i64;
      do
      {
        String[v8 + 256] = 2 * (String[v8] ^ 0xF);
        ++v8;
      }
      while ( v8 < v7 );
      v9 = 0;
      do
      {
        if ( String[v6 + 256] != byte_1400022D0[v6] )
          v9 = 1;
        ++v6;
      }
      while ( v6 < v7 ;
    }
    v10 = "Failed to unlock the Windows\n";
    if ( !v9 )
      v10 = "Successfully unlocked the Windows!\n";
    printf_5(v10);
    return 0;
  }
}
```

After some explaining from ChatGPT, to get the flag I need to byte_1400022D0 with 0xF

Found byte_1400022D0



```
.rdata:00000001400022D0 ; _BYTE byte_1400022D0[44]
.rdata:00000001400022D0 byte_1400022D0  db 0B4h, 84h, 96h, 98h, 0B6h, 92h, 44h, 0E8h, 0ACh, 7Eh
.rdata:00000001400022D0                                         ; DATA XREF: main+A0↑o
.rdata:00000001400022D0                 db 0B4h, 0A0h, 0B8h, 0F6h, 0DCh, 0FAh, 0F6h, 78h, 96h
.rdata:00000001400022D0                 db 0A0h, 0ECh, 80h, 0F4h, 0BAh, 0A0h, 0B0h, 7Ch, 0C2h
.rdata:00000001400022D0                 db 0D6h, 7Eh, 0F0h, 0B8h, 0A0h, 8Ah, 7Eh, 0B4h, 0BAh, 82h
.rdata:00000001400022D0                 db 0D4h, 0ACh, 0E4h, 3 dup(0)
```



```python
flag = [0xB4, 0x84, 0x96, 0x98, 0xB6, 0x92, 0x44, 0xE8, 0xAC, 0x7E, 0xB4,
    0xA0, 0xB8, 0xF6, 0xDC, 0xFA, 0xF6, 0x78, 0x96, 0xA0, 0xEC, 0x80, 0xF4,
    0xBA, 0xA0, 0xB0, 0x7C, 0xC2, 0xD6, 0x7E, 0xF0, 0xB8, 0xA0, 0x8A, 0x7E,
    0xB4, 0xBA, 0x82, 0xD4, 0xAC, 0xE4, 0x00, 0x00, 0x00]

s = ""

# decrypt
for i in flag:
    s += chr(int((i/2))^0xF)
print(s)
```

```
UMDCTF-{Y0U_Start3D_yOuR_W1nd0wS_J0URNeY}

=== Code Execution Successful ===
```

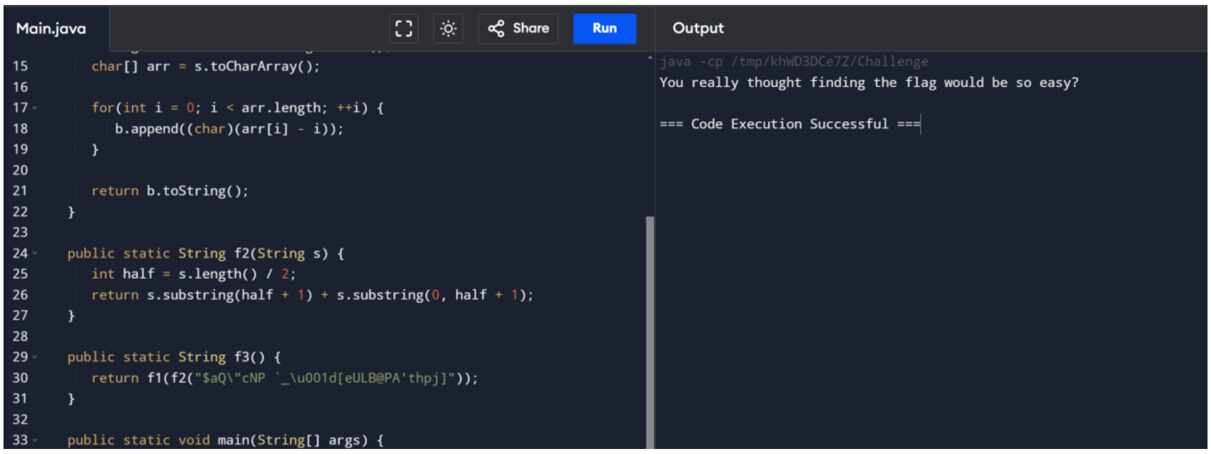
UMDCTF-{Y0U_Start3D_yOuR_W1nd0wS_J0URNeY}

## Starbucks

Got a class file. Use https://www.decompiler.com/ to decompile. Then got a single java file

```java
 public class Challenge {
   public static String f1(String s) {
     StringBuilder b = new StringBuilder();
     char[] arr = s.toCharArray();

     for(int i = 0; i < arr.length; ++i) {
       b.append((char)(arr[i] + i));
     }

     return b.toString();
   }

   public static String f1_rev(String s) {
     StringBuilder b = new StringBuilder();
     char[] arr = s.toCharArray();

     for(int i = 0; i < arr.length; ++i) {
       b.append((char)(arr[i] - i));
     }

     return b.toString();
   }

   public static String f2(String s) {
     int half = s.length() / 2;
     return s.substring(half + 1) + s.substring(0, half + 1);
   }

   public static String f3() {
     return f1(f2("$aQ\"cNP `_\u001d[eULB@PA'thpj]"));
   }

   public static void main(String[] args) {
     System.out.println("You really thought finding the flag would be so easy?");
   }
 }
```

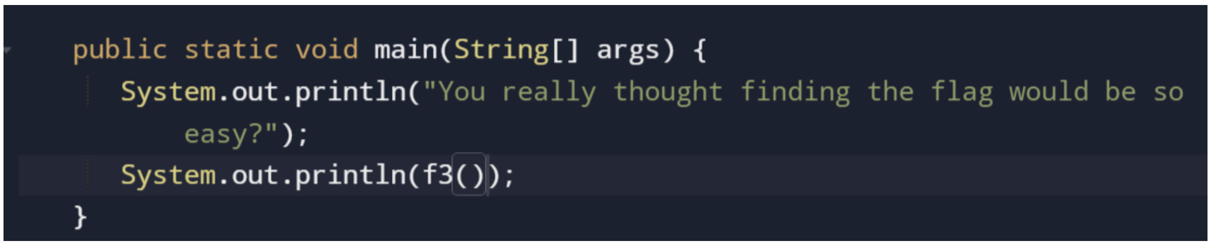
When running the file I don't get the flag



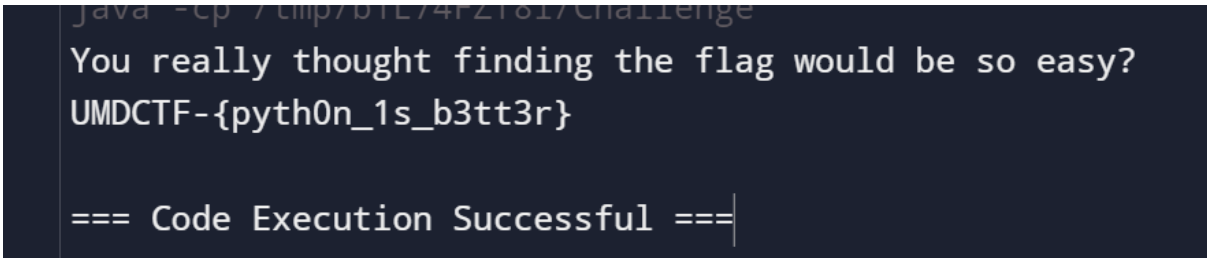
```java
        char[] arr = s.toCharArray();

        for(int i = 0; i < arr.length; ++i) {
            b.append((char)(arr[i] - i));
        }

        return b.toString();
    }

    public static String f2(String s) {
        int half = s.length() / 2;
        return s.substring(half + 1) + s.substring(0, half + 1);
    }

    public static String f3() {
        return f1(f2("$aQ\"cNP `_\u001d[eULB@PA'thpj]"));
    }

    public static void main(String[] args) {
```

```
java -cp /tmp/khWD3DCe7Z/Challenge
You really thought finding the flag would be so easy?

=== Code Execution Successful ===
```

Non of the code in main called the other function. There are f1(), f2() and f3(). Since f3() got f1() and f2() called in it, I called the f3() in main.



```java
public static void main(String[] args) {
    System.out.println("You really thought finding the flag would be so
        easy?");
    System.out.println(f3());
}
```



```
java -cp /tmp/b1L74FZT81/Challenge
You really thought finding the flag would be so easy?
UMDCTF-{pyth0n_1s_b3tt3r}

=== Code Execution Successful ===
```

UMDCTF-{pyth0n_1s_b3tt3r}

## Justintime

When running the file:

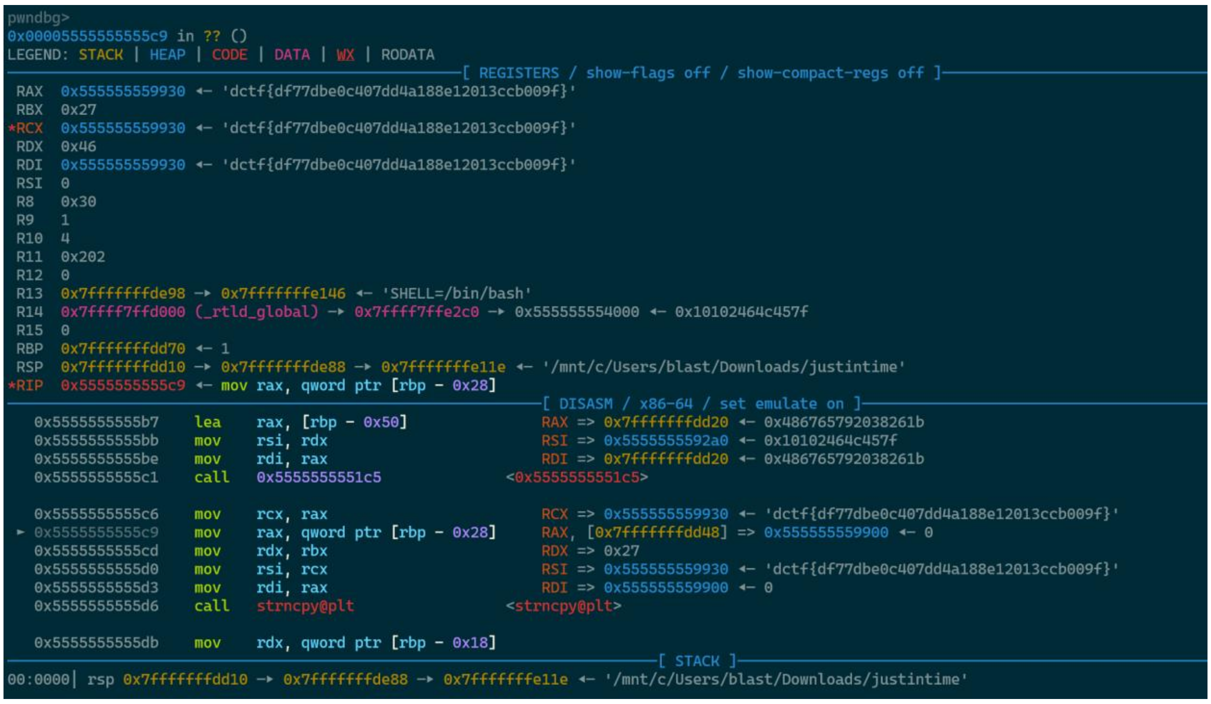
```
┌──(kruphix㉿Zeqzoq)-[/mnt/c/Users/blast/Downloads]
└─$ ./justintime
Decryption finished.
```

The main function:

```
__int64 __fastcall main(int a1, char **a2, char **a3)
{
  const char *v3; // rax
  const char *v4; // rax
  char src[40]; // [rsp+10h] [rbp-50h] BYREF
  char *v7; // [rsp+38h] [rbp-28h]
  char *v8; // [rsp+40h] [rbp-20h]
  char *dest; // [rsp+48h] [rbp-18h]

  dest = (char *)malloc(8uLL);
  v3 = (const char *)sub_126A(*a2);
  strncpy(dest, v3, 8uLL);
  strcpy(src, "\x1B&8 yegHr($g1bKu{\"f5`N}t#331Nv/%`11F#1");
  v8 = (char *)malloc(0x27uLL);
  strncpy(v8, src, 0x27uLL);
  sub_1372(v8);
  puts("Decryption finished.");
  v7 = (char *)malloc(0x27uLL);
  v4 = (const char *)sub_11C5(src, dest);
  strncpy(v7, v4, 0x27uLL);
  v7 = (char *)sub_11C5(v7, dest);
  sub_1460(v7);
  free(v8);
  free(v7);
  free(dest);
  return 0LL;
}
```

It seems that the file do the decryption process for us. We can do the decryption manually or do the script. But we can just intercept the decrypted text dynamically.

Set breakpoint at strncpy and step one by one using “ni”. Then found flag.

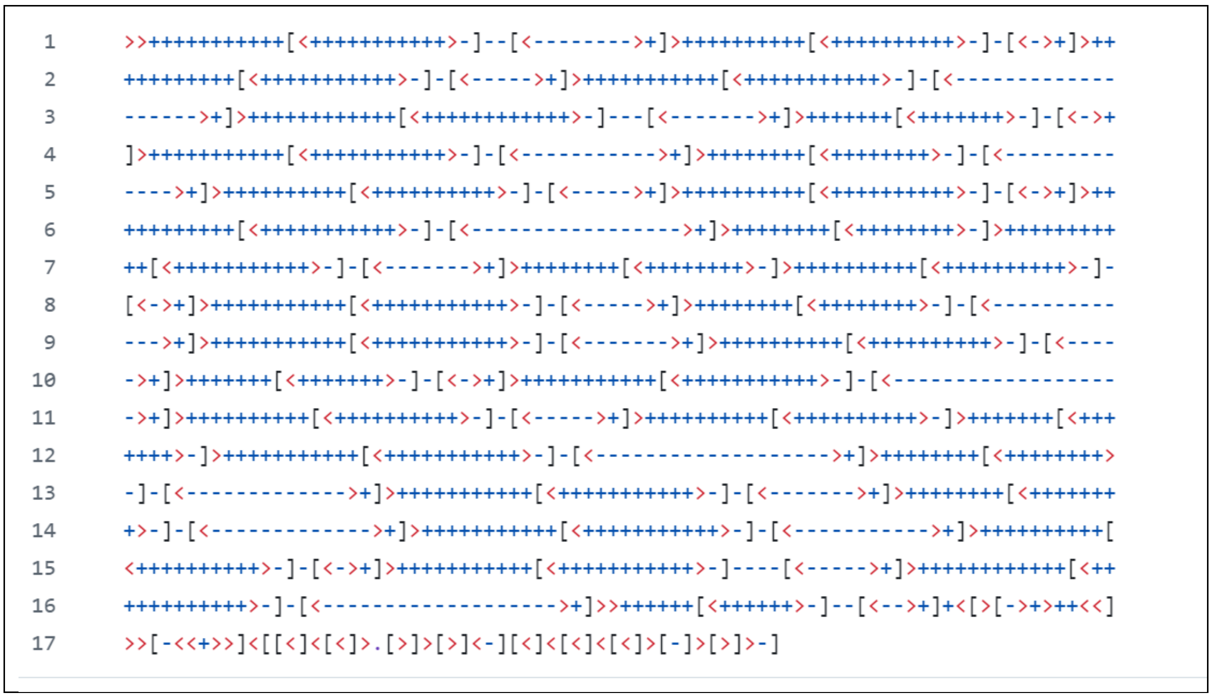


```
pwndbg>
0x00005555555555c9 in ?? ()
LEGEND: STACK | HEAP | CODE | DATA | WX | RODATA
─────────────────────────────────[ REGISTERS / show-flags off / show-compact-regs off ]──────────
 RAX  0x555555559930 ◂— 'dctf{df77dbe0c407dd4a188e12013ccb009f}'
 RBX  0x27
*RCX  0x555555559930 ◂— 'dctf{df77dbe0c407dd4a188e12013ccb009f}'
 RDX  0x46
 RDI  0x555555559930 ◂— 'dctf{df77dbe0c407dd4a188e12013ccb009f}'
 RSI  0
 R8   0x30
 R9   1
 R10  4
 R11  0x202
 R12  0
 R13  0x7fffffffde98 —▸ 0x7fffffffe146 ◂— 'SHELL=/bin/bash'
 R14  0x7ffff7ffd000 (_rtld_global) —▸ 0x7ffff7ffe2c0 —▸ 0x555555554000 ◂— 0x10102464c457f
 R15  0
 RBP  0x7fffffffdd70 ◂— 1
 RSP  0x7fffffffdd10 —▸ 0x7fffffffde88 —▸ 0x7fffffffe11e ◂— '/mnt/c/Users/blast/Downloads/justintime'
*RIP  0x5555555555c9 ◂— mov rax, qword ptr [rbp - 0x28]
────────────────────────────────────────[ DISASM / x86-64 / set emulate on ]──────────
   0x5555555555b7    lea    rax, [rbp - 0x50]           RAX => 0x7fffffffdd20 ◂— 0x486765792038261b
   0x5555555555bb    mov    rsi, rdx                    RSI => 0x5555555592a0 ◂— 0x10102464c457f
   0x5555555555be    mov    rdi, rax                    RDI => 0x7fffffffdd20 ◂— 0x486765792038261b
   0x5555555555c1    call   0x5555555551c5          <0x5555555551c5>

   0x5555555555c6    mov    rcx, rax                    RCX => 0x555555559930 ◂— 'dctf{df77dbe0c407dd4a188e12013ccb009f}'
 ► 0x5555555555c9    mov    rax, qword ptr [rbp - 0x28]     RAX, [0x7fffffffdd48] => 0x555555559900 ◂— 0
   0x5555555555cd    mov    rdx, rbx                    RDX => 0x27
   0x5555555555d0    mov    rsi, rcx                    RSI => 0x555555559930 ◂— 'dctf{df77dbe0c407dd4a188e12013ccb009f}'
   0x5555555555d3    mov    rdi, rax                    RDI => 0x555555559900 ◂— 0
   0x5555555555d6    call   strncpy@plt             <strncpy@plt>

   0x5555555555db    mov    rdx, qword ptr [rbp - 0x18]
──────────────────────────────────────────────────[ STACK ]──────────
00:0000│ rsp 0x7fffffffdd10 —▸ 0x7fffffffde88 —▸ 0x7fffffffe11e ◂— '/mnt/c/Users/blast/Downloads/justintime'
```

dctf{df77dbe0c407dd4a188e12013ccb009f}

## Fewer Thoughts, Head Empty

Got bf code

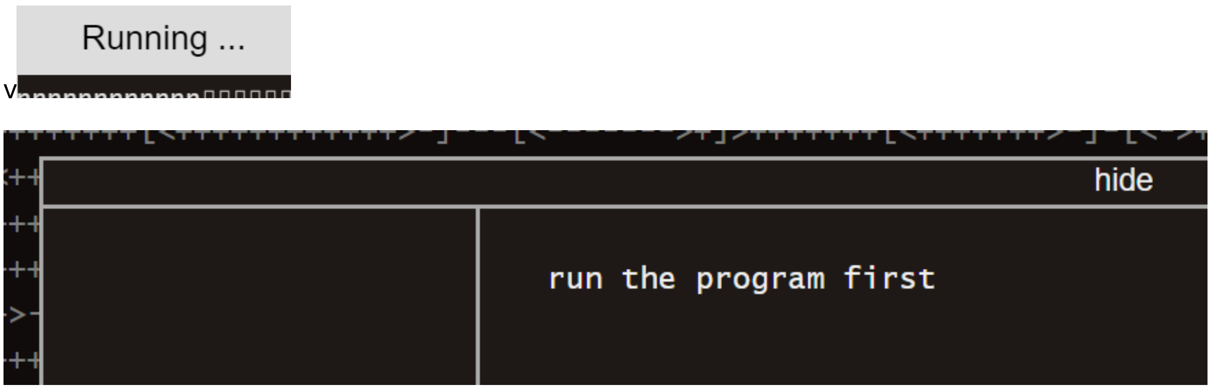


```
>>+++++++++++[<+++++++++++>-]--[<-------->+]>++++++++++[<++++++++++>-]-[<->+]>++
+++++++++[<+++++++++++>-]-[<----->+]>+++++++++++[<+++++++++++>-]-[<-------------
------>+]>++++++++++++[<++++++++++++>-]---[<------->+]>+++++++[<+++++++>-]-[<->+
]>+++++++++++[<+++++++++++>-]-[<----------->+]>++++++++[<++++++++>-]-[<---------
---->+]>++++++++++[<++++++++++>-]-[<----->+]>++++++++++[<++++++++++>-]-[<->+]>++
+++++++++[<+++++++++++>-]-[<----------------->+]>++++++++[<++++++++>-]>+++++++++
++[<+++++++++++>-]-[<------->+]>++++++++[<++++++++>-]>++++++++++[<++++++++++>-]-
[<->+]>+++++++++++[<+++++++++++>-]-[<----->+]>++++++++[<++++++++>-]-[<----------
--->+]>+++++++++++[<+++++++++++>-]-[<------->+]>++++++++++[<++++++++++>-]-[<----
->+]>+++++++[<+++++++>-]-[<->+]>+++++++++++[<+++++++++++>-]-[<------------------
->+]>++++++++++[<++++++++++>-]-[<----->+]>++++++++++[<++++++++++>-]>+++++++[<+++
++++>-]>+++++++++++[<+++++++++++>-]-[<------------------->+]>++++++++[<++++++++>
-]-[<------------->+]>+++++++++++[<+++++++++++>-]-[<------->+]>++++++++[<+++++++
+>-]-[<------------->+]>+++++++++++[<+++++++++++>-]-[<----------->+]>++++++++++[
<++++++++++>-]-[<->+]>+++++++++++[<+++++++++++>-]----[<----->+]>++++++++++++[<++
++++++++>-]-[<------------------->+]>>++++++[<++++++>-]--[<-->+]+<[>[->+>++<<]
>>[-<<+>>]<[[<]<[<]>.[>]>[>]<-][<]<[<]<[<]>[-]>[>]>-]
```

Then run the code to get:

```
iccttttffffffff{{{{{{{{{{{{{{{{000000000000000000000000000000000nnnnnnnnnnnnnnnnnnnnn
nnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnÿ
```

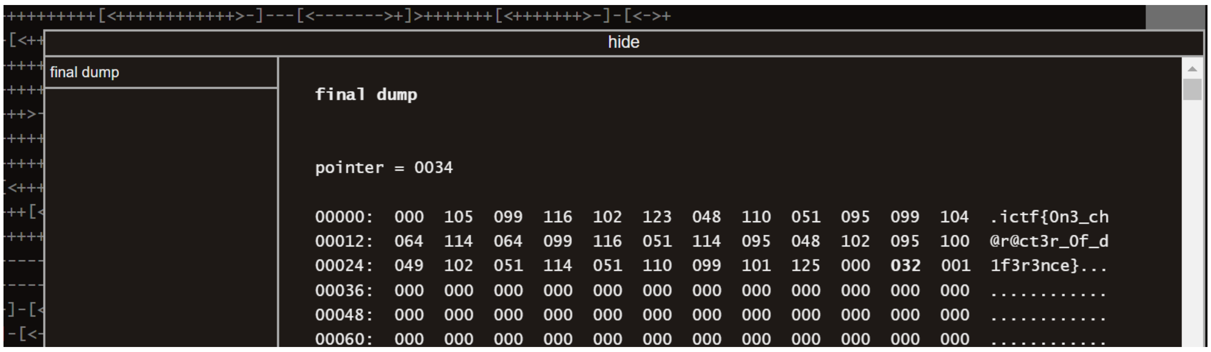
If we want to view the memory it said to run the code first but we already ran it and it is running.



```
run the program first
```

Need to remove [>[->+>++<<]>>[-<<+>>]<[[<]<[<]>.[>]>[>]<-][<]<[<]<[<]>[-]>[>]>-] to exit the program properly.

Then found flag in memory



```
+++++++++[<++++++++++++>-]---[<------->+]>+++++++[<+++++++>-]-[<->+
```

hide

final dump

```
final dump


pointer = 0034

00000:  000  105  099  116  102  123  048  110  051  095  099  104   .ictf{0n3_ch
00012:  064  114  064  099  116  051  114  095  048  102  095  100   @r@ct3r_0f_d
00024:  049  102  051  114  051  110  099  101  125  000  032  001   1f3r3nce}...
00036:  000  000  000  000  000  000  000  000  000  000  000  000   ............
00048:  000  000  000  000  000  000  000  000  000  000  000  000   ............
00060:  000  000  000  000  000  000  000  000  000  000  000  000   ............
```

ictf{0n3_ch@r@ct3r_0f_d1f3r3nce}

## Normal

Got a txt file which said how they compile the vvp file, and we got the v file. So need to decompile the v file to vvp.

Turn to vvp

```
┌──(kruphix㉿Zeqzoq)-[/mnt/c/Users/blast/Downloads]
└─$ iverilog -o normal.vvp -s main normal.v
```

Tried to run but it immediately exit

```
┌──(kruphix㉿Zeqzoq)-[/mnt/c/Users/blast/Downloads]
└─$ vvp normal.vvp
Incorrect flag...
normal.v:31: $finish called at 10 (1s)
```

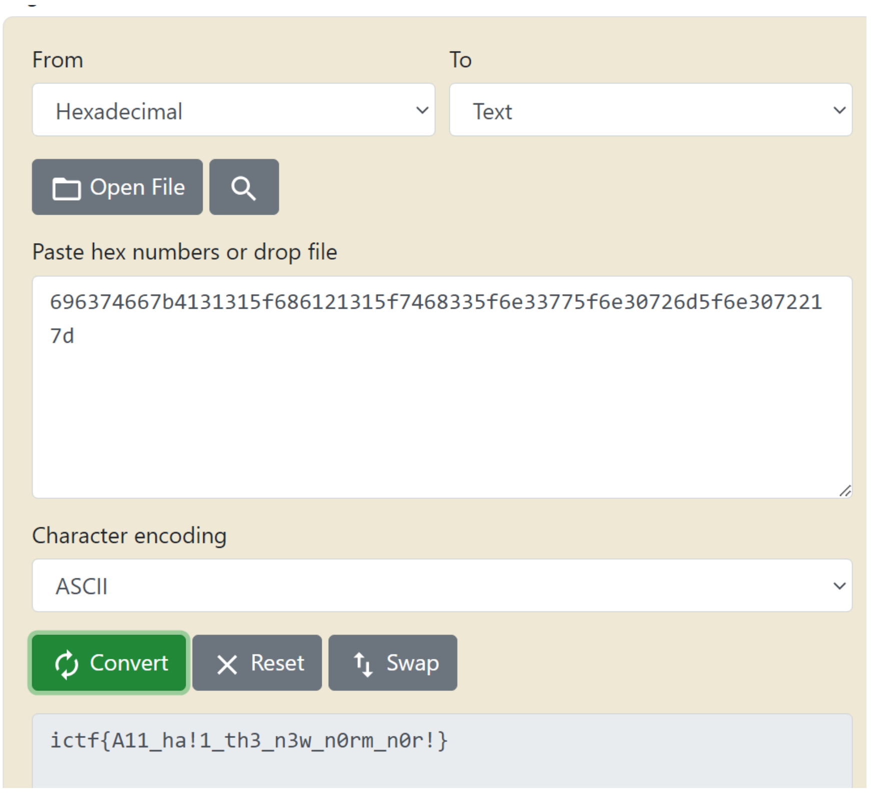
Using z3 library, I prompt chatgpt to make the solve script and got this

```
from z3 import *

# Define 256-bit variables for the input (flag) and intermediate wires
flag = BitVec('flag', 256)

# Constants from the Verilog code
c1 = BitVecVal(0x44940e8301e14fb33ba0da63cd5d2739ad079d571d9f5b987a1c3db2b60c92a3, 256)
c2 = BitVecVal(0xd208851a855f817d9b3744bd03fdacae61a70c9b953fca57f78e9d2379814c21, 256)

# Define the NOR gate logic in Z3
def nor_gate(x, y):
  return ~(x | y)

# Implement the Verilog logic
w1 = nor_gate(flag, c1)
w2 = nor_gate(flag, w1)
w3 = nor_gate(c1, w1)
w4 = nor_gate(w2, w3)
w5 = nor_gate(w4, w4)
w6 = nor_gate(w5, c2)
w7 = nor_gate(w5, w6)
w8 = nor_gate(c2, w6)
out = nor_gate(w7, w8)

# Define the solver
solver = Solver()

# Add the condition that the output (wrong) should be zero (correct flag)
solver.add(out == 0)

# Check if a solution exists
if solver.check() == sat:
  model = solver.model()
  correct_flag = model[flag].as_long()
  # Convert the solution to a hex string (flag)
  print(f"Correct flag (hex): {hex(correct_flag)}")
else:
  print("No solution found.")
```

Then got output:

```
┌──(kruphix㉿Zeqzoq)-[/mnt/c/Users/blast/Downloads]
└─$ python3 solve.py
Correct flag (hex): 0x696374667b4131315f686121315f7468335f6e33775f6e30726d5f6e3072217d
```

Then turn to ascii



From: Hexadecimal
To: Text

```
696374667b4131315f686121315f7468335f6e33775f6e30726d5f6e307221
7d
```

Character encoding: ASCII

```
ictf{A11_ha!1_th3_n3w_n0rm_n0r!}
```

ictf{A11_ha!1_th3_n3w_n0rm_n0r!}

*don't need to turn to vvp file actually.

## Stings

When running the file, it prompt for password.

The main function:

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  int i; // [rsp+Ch] [rbp-1214h]
  char v5[256]; // [rsp+10h] [rbp-1210h] BYREF
  _DWORD v6[10]; // [rsp+110h] [rbp-1110h] BYREF
  __int64 v7; // [rsp+138h] [rbp-10E8h]
  char v8[208]; // [rsp+140h] [rbp-10E0h] BYREF
  char s[1964]; // [rsp+210h] [rbp-1010h] BYREF
  int v10; // [rsp+9BCh] [rbp-864h]
  _BYTE v11[2128]; // [rsp+9C0h] [rbp-860h] BYREF
  unsigned __int64 v12; // [rsp+1218h] [rbp-8h]

  v12 = __readfsqword(0x28u);
  strcpy(
    s,
    "    %        %\n"
    "      %        %\n"
    "        %        %\n"
    "          %       %\n"
    "           %       %\n"
    "            %       %              :::\n"
    "             %       %            ::::::\n"
    "           %%%%%%  %%%%%%%%%%         ::::::::\n"
    "         %%%%%ZZZZ%%%%%%   %%%ZZZZ    ::::::::::       ::::::\n"
    "        %%%ZZZZZ%%%%%%%%%%%%%%%ZZZZZZ  :::::::::::   :::::::::::::::::\n"
    "        ZZZ%ZZZ%%%%%%%%%%%%%%%%%ZZZZZZZ::::::::::***:::::::::::::::::::::\n"
    "      ZZZ%ZZZZZZ%%%%%%%%%%%%%%%%ZZZZZZZZZ::::::***::::::::::::::::::::::::\n"
    "     ZZZ%ZZZZZZZZZZ%%%%%%%%%%%ZZZZZZ%ZZZZ:::***::::::::::::::::::::::\n"
    "    ZZ%ZZZZZZZZZZZZZZZZZZZZZZZZZ%%%%% %ZZZ:**:::::::::::::::::::::::\n"
    "   ZZ%ZZZZZZZZZZZZZZZZZZZZ%%%%% | | %ZZZ *::::::::::::::::::::\n"
    "   Z%ZZZZZZZZZZZZZZZZZ%%%%%%%%%%%%%%%%ZZZ::::::::::::::::::::::::::::\n"
    "    ZZZZZZZZZZZ%%%%%ZZZZZZZZZZZZZZZZZZ%%%%:::ZZZZ:::::::::::::::::::\n"
    "      ZZZZ%%%%%ZZZZZZZZZZZZZZZZZZZ%%%%%ZZZ%%ZZZ%ZZ%%*:::::::::::\n"
    "        ZZZZZZZZZZZZZZZZZZZ%%%%%%%%%%ZZZZZZZZZZ%ZZ%:::*:::::::\n"
    "        *:::%%%%%%%%%%%%%%%%%%%%%%%%%ZZZZZZZZZZ%%%*::::*::::\n"
    "      *:::::::%%%%%%%%%%%%%%%%%%%%%%%%%%ZZZZZ%%     *:::Z\n"
    "     **:ZZZZ:::%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%ZZ     ZZZZZ\n"
    "    *:ZZZZZZZ     %%%%%%%%%%%%%%%%%%%%%%%%%ZZZZ   ZZZZZZZ\n"
    "   *::::ZZZZZZ      %%%%%%%%%%%%%%%%%%%ZZZZZZZ     ZZZ\n"
    "    *::ZZZZZZ        Z%%%%%%%%%%%%%%ZZZZZZZ%%\n"
    "      ZZZZ           ZZZZZZZZZZZZZZZZZ%%%%%\n"
    "                  %%%ZZZZZZZZZZZZ%%%%%%%%%\n"
    "                  Z%%%%%%%%%%%%%%%%%%%%%%%%%\n"
    "                  ZZ%%%%%%%%%%%%%%%%%%%%%%%\n"
    "                  %ZZZZZZZZZZZZZZZZZZZZ\n"
    "                  %%ZZZZZZZZZZZZZZZZZZ\n"
    "                   %%%%%%%%%%%%%%%%%%\n"
    "                    %%%%%%%%%%%%%%\n"
    "                     %%%%%%%%%%\n"
    "                      ZZZZ\n"
    "                      ZZZ\n"
    "                     ZZ\n"
    "                    Z\n");
  v10 = 0;
  memset(v11, 0, sizeof(v11));
  strcpy((char *)v6, "jdug|tus2oht`5s4ou`i2ee4o`28c32b7:~");
  v6[9] = 0;
  v7 = 0LL;
  memset(v8, 0, sizeof(v8));
  puts(s);
  puts("Welcome to the beehive.");
  puts("Enter the password, or you'll get stung!");
  __isoc99_scanf("%50s", v5);
  for ( i = 0; i <= 34; ++i )
  {
    if ( v5[i] != *((char *)v6 + i) - 1 )
    {
      puts("I'm disappointed. *stings you*");
      return -1;
    }
  }
  puts("Congrats! The password is the flag.");
  return 0;
}
```

Our input here is v5, it then compared to v6 subtract 1.

V6 is jdug|tus2oht`5s4ou`i2ee4o`28c32b7:~ so we need to subtract 1 each to get v5 which is the password.

```
┌──(kruphix㉿Zeqzoq)-[/mnt/c/Users/blast/Downloads]
└─$ python3
Python 3.11.9 (main, Apr 10 2024, 13:16:36) [GCC 13.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> print(''.join([chr(ord(i) - 1) for i in "jdug|tus2oht`5s4ou`i2ee4o`28c32b7:~"]))
ictf{str1ngs_4r3nt_h1dd3n_17b21a69}
```

ictf{str1ngs_4r3nt_h1dd3n_17b21a69}

## Dotty

Here we got an exe file. When running file on the file we got .net assembly.

```
┌──(kruphix㉿Zeqzoq)-[/mnt/c/Users/blast/Downloads]
└─$ file Dotty.exe
Dotty.exe: PE32 executable (console) Intel 80386 Mono/.Net assembly, for MS Windows, 3 sections
```

So opened it with dnSpy.

Program @02000002 function:

```
using System;
using System.Collections.Generic;
using System.Linq;

namespace Dotty
{
  // Token: 0x02000002 RID: 2
  internal class Program
  {
    // Token: 0x06000002 RID: 2 RVA: 0x00002058 File Offset: 0x00000258
    private static string Dotter(string phrase)
    {
      return string.Join("|", from char c in phrase
      select Program.mapper[char.ToUpper(c)]);
    }

    // Token: 0x06000003 RID: 3 RVA: 0x0000208C File Offset: 0x0000028C
    private static void Main(string[] args)
    {
      Console.Write("Please enter your secret to encode: ");
      string phrase = Console.ReadLine();
      string text = Program.Dotter(phrase);
      if (text == Check.check)
      {
        Console.WriteLine("That's the right secret!");
      }
      else
      {
        Console.WriteLine(text);
      }
    }

    // Token: 0x04000001 RID: 1
    private static Dictionary<char, string> mapper = new Dictionary<char, string>
    {
      {
        ' ',
        "/"
      },
      {
        'A',
        ".-"
      },
      {
        'B',
        "-..."
      },
      {
        'C',
      "-.-."
    },
    {
      'D',
      "-.."
    },
    {
      'E',
      "."
    },
    {
      'F',
      "..-."
    },
    {
      'G',
      "--."
    },
    {
      'H',
      "...."
    },
    {
      'I',
      ".."
    },
    {
      'J',
      ".---"
    },
    {
      'K',
      "-.-"
    },
    {
      'L',
      ".-.."
    },
    {
      'M',
      "--"
    },
    {
      'N',
      "-."
    },
    {
      'O',
      "---"
    },
    {
      'P',
      ".--."
    },
    {
      'Q',
      "--.-"
    },
    {
      'R',
      ".-."
    },
    {
            'S',
            "..."
        },
        {
            'T',
            "-"
        },
        {
            'U',
            "..-"
        },
        {
            'V',
            "...-"
        },
        {
            'W',
            ".--"
        },
        {
            'X',
            "-..-"
        },
        {
            'Y',
            "-.--"
        },
        {
            'Z',
            "--.."
        },
        {
            '1',
            ".----"
        },
        {
            '2',
            "..---"
        },
        {
            '3',
            "...--"
        },
        {
            '4',
            "....-"
        },
        {
            '5',
            "....."
        },
        {
            '6',
            "-...."
        },
        {
            '7',
            "--..."
        },
        {
            '8',
            "---.."
        },
            {
                '9',
                "----."
            },
            {
                '0',
                "-----"
            }
        };
    }
}
```

The Check @02000003 function:

```
using System;

namespace Dotty
{
    // Token: 0x02000003 RID: 3
    internal class Check
    {
        // Token: 0x04000003 RID: 3
        public static string check = "-|....|.|/|..-.|.-..|.-|--.|/|..|...|/|---|.---|--.-|-..-|.|-.--|...--|..-|--|--..|.....|.--|..|--|.-..|.|.-
..|.....|....-|-|.-|.....|-.-|--...|---|.-|--..|-|--.|..---|..---|--...|--.|-...|--..|..-.|-....|-.|.-..|--.-|.--.|.|--...|-|-....|.--.|--..|--...|.-..|.....|-|--
.|-.-.|-.|-..|-...|--|--|...--|-..|.-|-.|.-..|.....|/|-...|.-|...|.|...--|..---";
    }
}
```

From here we can see its mapped. A Is .- and so on. So we need to map strings check.

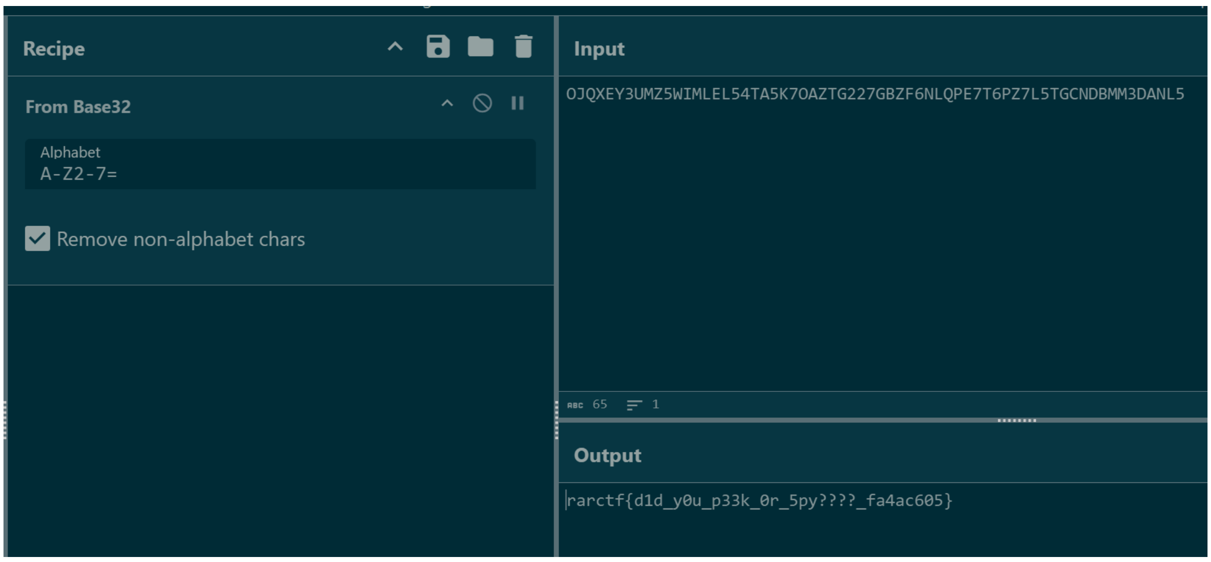
Simple script:

```
# flag string
flag = "-|....|.|/|..-.|.-..|.-|--.|/|..|...|/|---|.---|--.-|-..-|.|-.--|...--|..-|--|--..|.....|.--|..|--|.-..|.|.-..|.....|....-|-|.-|.....|-.-|--...|---|.-|--
..|-|--.|..---|..---|--...|--.|-...|--..|..-.|-....|-.|.-..|--.-|.--.|.|--...|-|-....|.--.|--..|--...|.-..|.....|-|--.|-.-.|-.|-..|-...|--|--|...--|-..|.-|-.|.-
..|.....|/|-...|.-|...|.|...--|..---"

# mapped dictionary
mapped = [[ ' ', "/" ], [ 'A', ".-" ], [ 'B', "-..." ], [ 'C', "-.-." ], [ 'D', "-.." ], [ 'E', "." ], [ 'F', "..-." ], [ 'G', "--." ], [ 'H', "...." ], [ 'I', ".." ], [ 'J',
".---" ], [ 'K', "-.-" ], [ 'L', ".-.." ], [ 'M', "--" ], [ 'N', "-." ], [ 'O', "---" ], [ 'P', ".--." ], [ 'Q', "--.-" ], [ 'R', ".-." ], [ 'S', "..." ], [ 'T', "-" ], [
'U', "..-" ], [ 'V', "...-" ], [ 'W', ".--" ], [ 'X', "-..-" ], [ 'Y', "-.--" ], [ 'Z', "--.." ], [ '1', ".----" ], [ '2', "..---" ], [ '3', "...--" ], [ '4', "....-" ], [
'5', "....." ], [ '6', "-...." ], [ '7', "--..." ], [ '8', "---.." ], [ '9', "----." ], [ '0', "-----" ] ]

# split the flag the way the program does it
flag = flag.split("|")

s = ""

# go through and convert the flag
for i in flag:
        for j in mapped:
                if i == j[1]:
                        s += j[0]
                        break



print(s)
```

And we got the output

THE FLAG IS
OJQXEY3UMZ5WIMLEL54TA5K7OAZTG227GBZF6NLQPE7T6PZ7L5TGCNDBMM3DANL5 BASE32

So went to cyberchef too decrypt



Alphabet
A-Z2-7=

```
OJQXEY3UMZ5WIMLEL54TA5K7OAZTG227GBZF6NLQPE7T6PZ7L5TGCNDBMM3DANL5
```

```
rarctf{d1d_y0u_p33k_0r_5py????_fa4ac605}
```

rarctf{d1d_y0u_p33k_0r_5py????_fa4ac605}

## verybabyrev(nak explain balik)

The file prompt for password

```
┌──(kruphix㉿Zeqzoq)-[/mnt/c/Users/blast/Downloads]
└─$ ./verybabyrev
Enter your flag: lfsfd
Nope!
```

The main function:

```
int __cdecl __noreturn main(int argc, const char **argv, const char **envp)
{
  __int64 s1[12]; // [rsp+0h] [rbp-100h] BYREF
  char v4; // [rsp+60h] [rbp-A0h]
  char s[140]; // [rsp+70h] [rbp-90h] BYREF
  int v6; // [rsp+FCh] [rbp-4h]

  setvbuf(stdout, 0LL, 2, 0LL);
  memset(s, 0, 0x80uLL);
  s1[0] = 0x45481D1217111313LL;
  s1[1] = 0x95F422C260B4145LL;
  s1[2] = 0x541B56563D6C5F0BLL;
  s1[3] = 0x585C0B3C2945415FLL;
  s1[4] = 0x402A6C54095D5F00LL;
  s1[5] = 0x4B5F4248276A0606LL;
  s1[6] = 0x6C5E5D432C2D4256LL;
  s1[7] = 0x6B315E434707412DLL;
  s1[8] = 0x5E54491C6E3B0A5ALL;
  s1[9] = 0x2828475E05342B1ALL;
  s1[10] = 0x60450073B26111FLL;
  s1[11] = 0xA774803050B0D04LL;
  v4 = 0;
  printf("Enter your flag: ");
  fgets(s, 128, stdin);
  v6 = 0;
  if ( s[0] != 114 )
  {
   puts("Nope!");
   exit(0);
  }
  while ( v6 <= 126 )
  {
   s[v6] ^= s[v6 + 1];
   ++v6;
  }
  if ( !memcmp(s1, s, 0x61uLL) )
  {
   puts("Correct!");
   exit(1);
  }
  puts("Nope!");
  exit(0);
}
```

The encrypted flag is s1. As we can see, s1 is absolutely long because we need to combine all the s1 variables. But notice that when we ran file command it show LSB. which means the file is using **little-endian** byte ordering. This is expected for a program compiled for **x86-64** architecture, as it's little-endian by default.

```
┌──(kruphix㉿Zeqzoq)-[/mnt/c/Users/blast/Downloads]
└─$ file verybabyrev
verybabyrev: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux 3.2.0, BuildID[sha1]=276af40b2393e3013daafb8acfc000ab3e0d1ab8, not stripped
```

So when extracting s1, for 0x45481D1217111313LL, instead of it become 0x45, 0x48, ..., 0x13, 0x13, it will be 0x13, 0x13, ..., 0x48, 0x45.

So s1 will be

```
0x13, 0x13, 0x11, 0x17, 0x12, 0x1D, 0x48, 0x45, 0x45, 0x41, 0x0B, 0x26, 0x2C, 0x42, 0x5F, 0x09, 0x0B, 0x5F, 0x6C, 0x3D, 0x56, 0x56, 0x1B, 0x54, 0x5F, 0x41, 0x45, 0x29, 0x3C, 0x0B, 0x5C, 0x58, 0x00, 0x5F, 0x5D, 0x09, 0x54, 0x6C, 0x2A, 0x40, 0x06, 0x06, 0x6A, 0x27, 0x48, 0x42, 0x5F, 0x4B, 0x56, 0x42, 0x2D, 0x2C, 0x43, 0x5D, 0x5E, 0x6C, 0x2D, 0x41, 0x07, 0x47, 0x43, 0x5E, 0x31, 0x6B, 0x5A, 0x0A, 0x3B, 0x6E, 0x1C, 0x49, 0x54, 0x5E, 0x1A, 0x2B, 0x34, 0x05, 0x5E, 0x47, 0x28, 0x28, 0x1F, 0x11, 0x26, 0x3B, 0x07, 0x50, 0x04, 0x06, 0x04, 0x0D, 0x0B, 0x05, 0x03, 0x48, 0x77, 0x0A
```

The "encryption" was simply using the current character and xor'ing it with the next one. I knew the start of the flag would be rarctf{ so here is the script to solve

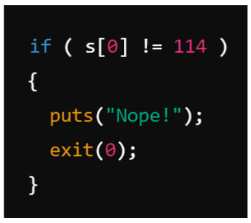
```
# encrypted flag
enc = [0x13, 0x13, 0x11, 0x17, 0x12, 0x1D, 0x48, 0x45, 0x45, 0x41, 0x0B, 0x26, 0x2C, 0x42, 0x5F, 0x09, 0x0B, 0x5F, 0x6C, 0x3D, 0x56, 0x56, 0x1B, 0x54, 0x5F, 0x41, 0x45, 0x29, 0x3C, 0x0B, 0x5C, 0x58, 0x00, 0x5F, 0x5D, 0x09, 0x54, 0x6C, 0x2A, 0x40, 0x06, 0x06, 0x6A, 0x27, 0x48, 0x42, 0x5F, 0x4B, 0x56, 0x42, 0x2D, 0x2C, 0x43, 0x5D, 0x5E, 0x6C, 0x2D, 0x41, 0x07, 0x47, 0x43, 0x5E, 0x31, 0x6B, 0x5A, 0x0A, 0x3B, 0x6E, 0x1C, 0x49, 0x54, 0x5E, 0x1A, 0x2B, 0x34, 0x05, 0x5E, 0x47, 0x28, 0x28, 0x1F, 0x11, 0x26, 0x3B, 0x07, 0x50, 0x04, 0x06, 0x04, 0x0D, 0x0B, 0x05, 0x03, 0x48, 0x77, 0x0A]

# input flag, with known parts
s = "rarctf{"

# loop through the length of the encrypted bytes from length of known
for i in range(len(s)-1, len(enc) - 1):

  # try the range of characters
	for j in range(32, 127):

    # if the character gives us the encrypted bytes
		if ord(s[i]) ^ j == enc[i]:

      # add the byte to the string and break the loop
			s += chr(j)
			break

# print the flag
print(s)
```

rarctf{3v3ry_s1ngl3_b4by-r3v_ch4ll3ng3_u535_x0r-f0r_s0m3_r34s0n_4nd_1-d0nt_kn0w_why_dc37158365}

//but why do we need the first part of the flag?

The first few characters of the flag are important for multiple reasons:

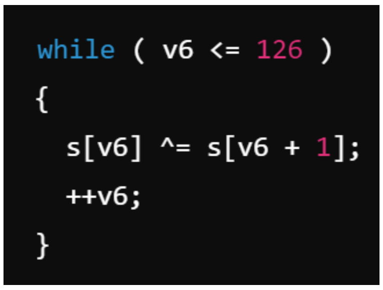
1. **Initial Check on s[0]:** In the code, the first character of the input string (s[0]) is compared against the ASCII value of 114 ('r'):



```
if ( s[0] != 114 )
{
  puts("Nope!");
  exit(0);
}
```

This means the flag must start with the character 'r'. If the first character is incorrect, the program will immediately reject the flag.

2. **XOR Operation in the Loop:** After the initial check, the program applies an XOR operation across the entire flag string:



```
while ( v6 <= 126 )
{
  s[v6] ^= s[v6 + 1];
  ++v6;
}
```

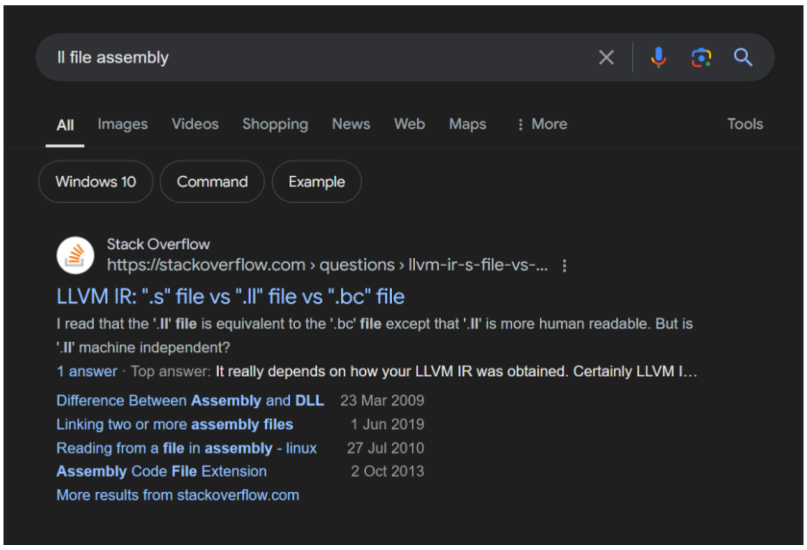
This XOR chain depends on the initial characters of the flag, so the starting characters will influence the entire flag during this transformation. Knowing the correct first character ('r') ensures that the XOR chain is computed correctly.

**Why the First Character is Important:**
The XOR operation depends on the values of previous characters. Knowing the first character ('r') is essential because it starts the XOR chain, and any mistake in the initial value would propagate through the entire flag, leading to incorrect results.

# Easy – Medium

## Better than ASM

Got ll file. When searching for what is ll file we get to know about llvm



ll file assembly

All Images Videos Shopping News Web Maps More Tools

Windows 10 Command Example

Stack Overflow
https://stackoverflow.com › questions › llvm-ir-s-file-vs-...

LLVM IR: ".s" file vs ".ll" file vs ".bc" file

I read that the '.ll' file is equivalent to the '.bc' file except that '.ll' is more human readable. But is '.ll' machine independent?

1 answer · Top answer: It really depends on how your LLVM IR was obtained. Certainly LLVM I...

| | |
|---|---|
| Difference Between Assembly and DLL | 23 Mar 2009 |
| Linking two or more assembly files | 1 Jun 2019 |
| Reading from a file in assembly - linux | 27 Jul 2010 |
| Assembly Code File Extension | 2 Oct 2013 |

More results from stackoverflow.com

We can compile the ll file to run it

```
┌──(kruphix㉿Zeqzoq)-[/mnt/c/Users/blast/Downloads]
└─$ clang task.ll -mllvm -W -g -W1,-pie -o task.out
warning: unknown warning option '-W1,-pie' [-Wunknown-warning-option]
warning: overriding the module target triple with x86_64-pc-linux-gnu [-Woverride-module]
2 warnings generated.
```

And now we can run the file

```
┌──(kruphix㉿Zeqzoq)-[/mnt/c/Users/blast/Downloads]
└─$ ./task.out
Only the chosen one will know what the flag is!
Are you the chosen one?
flag: donno

😠😡😠😡😠😡 You are not the chosen one! 😡😠😡😠😡😠
```

Now can use decompiler. The main function:

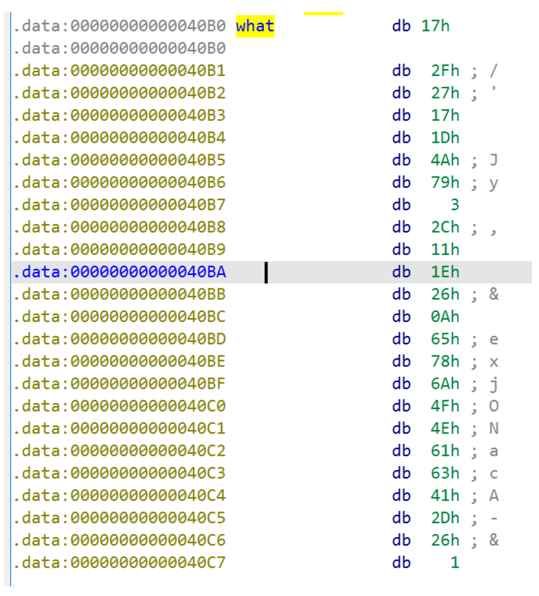
```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  char v4; // [rsp+Ch] [rbp-7Ch]
  char v5; // [rsp+24h] [rbp-64h]
  size_t v6; // [rsp+30h] [rbp-58h]
  int j; // [rsp+38h] [rbp-50h]
  int i; // [rsp+3Ch] [rbp-4Ch]
  char s[68]; // [rsp+40h] [rbp-48h] BYREF
  int v10; // [rsp+84h] [rbp-4h]

  v10 = 0;
  printf("Only the chosen one will know what the flag is!\n");
  printf("Are you the chosen one?\n");
  printf("flag: ");
  __isoc99_scanf("%64s", s);
  v6 = strlen(s);
  if ( v6 == strlen(&what) )
  {
    if ( (unsigned int)check(s) )
    {
      for ( i = 0; i < strlen(s); ++i )
      {
        v5 = s[i];
        s[i] = secret[i % strlen(secret)] ^ v5;
      }
    }
    else
    {
      for ( j = 0; j < strlen(s); ++j )
      {
        v4 = flag[j];
        s[j] = secret[j % strlen(secret)] ^ v4;
      }
    }
    printf(format, s);
    return 0;
  }
  else
  {
    printf(asc_205A);
    return 1;
  }
}
```

The program reads in a string (max 64 chars), if it is the same length as what it then checks the string and if it passes it xor's the string with secret, if not i

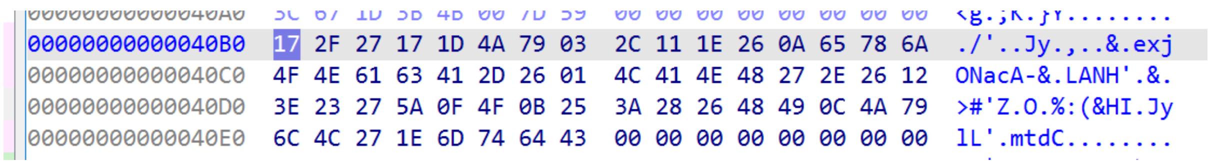
t xor's some string (flag) with secret.

There are some variables that we need to find, which is, what, secret, format, asc_205A and flag.

Here we can learn something new, before we extract the variable into hex by looking at the assembly or the hex view.



```
.data:00000000000040B0 what            db 17h
.data:00000000000040B0
.data:00000000000040B1                 db  2Fh ; /
.data:00000000000040B2                 db  27h ; '
.data:00000000000040B3                 db  17h
.data:00000000000040B4                 db  1Dh
.data:00000000000040B5                 db  4Ah ; J
.data:00000000000040B6                 db  79h ; y
.data:00000000000040B7                 db    3
.data:00000000000040B8                 db  2Ch ; ,
.data:00000000000040B9                 db  11h
.data:00000000000040BA                 db  1Eh
.data:00000000000040BB                 db  26h ; &
.data:00000000000040BC                 db  0Ah
.data:00000000000040BD                 db  65h ; e
.data:00000000000040BE                 db  78h ; x
.data:00000000000040BF                 db  6Ah ; j
.data:00000000000040C0                 db  4Fh ; O
.data:00000000000040C1                 db  4Eh ; N
.data:00000000000040C2                 db  61h ; a
.data:00000000000040C3                 db  63h ; c
.data:00000000000040C4                 db  41h ; A
.data:00000000000040C5                 db  2Dh ; -
.data:00000000000040C6                 db  26h ; &
.data:00000000000040C7                 db    1
```



```
00000000000040A0  3C 67 1D 3B 4B 00 7D 59  00 00 00 00 00 00 00 00  <g.;K.}Y........
00000000000040B0  17 2F 27 17 1D 4A 79 03  2C 11 1E 26 0A 65 78 6A  ./'..Jy.,..&.exj
00000000000040C0  4F 4E 61 63 41 2D 26 01  4C 41 4E 48 27 2E 26 12  ONacA-&.LANH'.&.
00000000000040D0  3E 23 27 5A 0F 4F 0B 25  3A 28 26 48 49 0C 4A 79  >#'Z.O.%:(&HI.Jy
00000000000040E0  6C 4C 27 1E 6D 74 64 43  00 00 00 00 00 00 00 00  lL'.mtdC........
```

So our "what" will be 0x17 0x2F 0x27 0x17 … . But in the assembly we can see 2F is already mapped to "/", 27 mapped to" '", 4A mapped to "J" etc, so instead of extracting int hex, we can extract into bytes(I think?)

what = b"\x17/'\x17\x1DJy\x03,\x11\x1E&\x0AexjONacA-&\x01LANH'.&\x12>#'Z\x0FO\x0B%:(&HI\x0CJylL'\x1EmtdC\x00\x00\x00\x00\x00\x00\x00\x00"

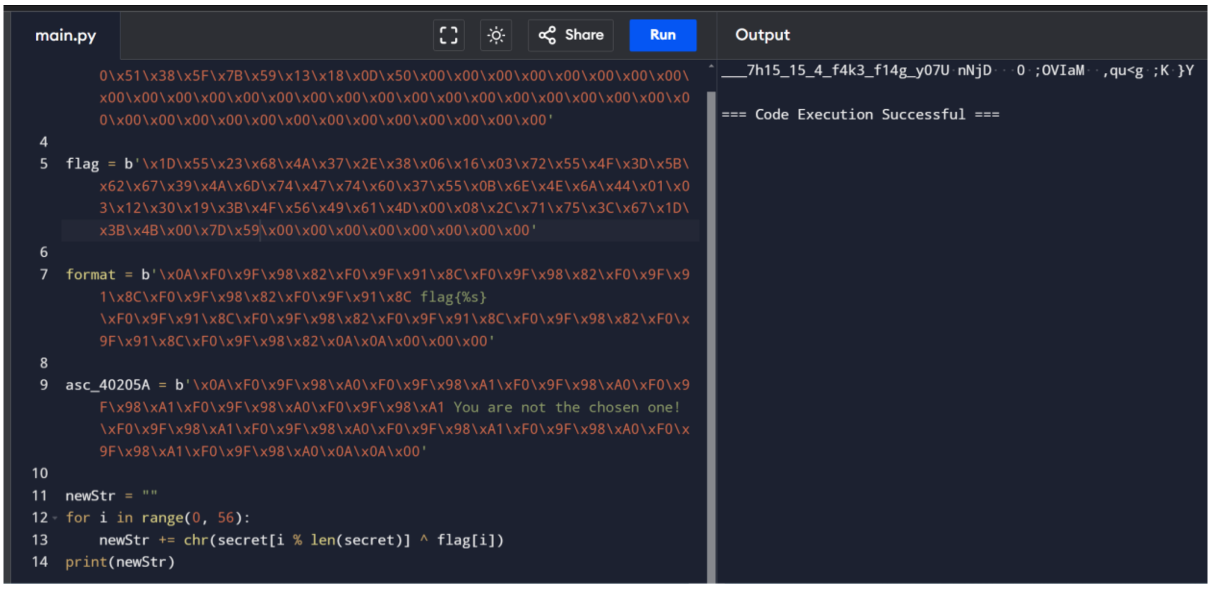
secret = b'B\x0A|_\x22\x06\x1Bg7#\x5CF\x0A)\x090Q8_{Y\x13\x18\x0DP\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00'

flag = b'\x1DU#hJ7.8\x06\x16\x03rUO=[bg9JmtGt` 7U\x0BnNjD\x01\x03\x120\x19;OVIaM\x00\x08,qu<g\x1D;K\x00}Y\x00\x00\x00\x00\x00\x00\x00\x00'

format = b'\x0A\xF0\x9F\x98\x82\xF0\x9F\x91\x8C\xF0\x9F\x98\x82\xF0\x9F\x91\x8C\xF0\x9F\x98\x82\xF0\x9F\x91\x8C flag{%s} \xF0\x9F\x91\x8C\xF0\x9F\x98\x82\xF0\x9F\x91\x8C\xF0\x9F\x98\x82\xF0\x9F\x91\x8C\xF0\x9F\x98\x82\x0A\x0A\x00\x00\x00'

asc_40205A = b'\x0A\xF0\x9F\x98\xA0\xF0\x9F\x98\xA1\xF0\x9F\x98\xA0\xF0\x9F\x98\xA1\xF0\x9F\x98\xA0\xF0\x9F\x98\xA1 You are not the chosen one! \xF0\x9F\x98\xA1\xF0\x9F\x98\xA0\xF0\x9F\x98\xA1\xF0\x9F\x98\xA0\xF0\x9F\x98\xA1\xF0\x9F\x98\xA0\x0A\x0A\x00'

asc_40205A was obviously the string that was printed when I put in a random value, however I noticed that the format string included the flag{} component of the flag meaning that I would not simply be able to solve the flag by assuming the first 5 characters and the last character since the wrapper was added later. I means that the encrypted flag does not contain flag{}.



```
0\x51\x38\x5F\x7B\x59\x13\x18\x0D\x50\x00\x00\x00\x00\x00\x00\x00\x00\
x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x0
0\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00'

flag = b'\x1D\x55\x23\x68\x4A\x37\x2E\x38\x06\x16\x03\x72\x55\x4F\x3D\x5B\
x62\x67\x39\x4A\x6D\x74\x47\x74\x60\x37\x55\x0B\x6E\x4E\x6A\x44\x01\x0
3\x12\x30\x19\x3B\x4F\x56\x49\x61\x4D\x00\x08\x2C\x71\x75\x3C\x67\x1D\
x3B\x4B\x00\x7D\x59\x00\x00\x00\x00\x00\x00\x00\x00'

format = b'\x0A\xF0\x9F\x98\x82\xF0\x9F\x91\x8C\xF0\x9F\x98\x82\xF0\x9F\x9
1\x8C\xF0\x9F\x98\x82\xF0\x9F\x91\x8C flag{%s}
\xF0\x9F\x91\x8C\xF0\x9F\x98\x82\xF0\x9F\x91\x8C\xF0\x9F\x98\x82\xF0\x
9F\x91\x8C\xF0\x9F\x98\x82\x0A\x0A\x00\x00\x00'

asc_40205A = b'\x0A\xF0\x9F\x98\xA0\xF0\x9F\x98\xA1\xF0\x9F\x98\xA0\xF0\x9
F\x98\xA1\xF0\x9F\x98\xA0\xF0\x9F\x98\xA1 You are not the chosen one!
\xF0\x9F\x98\xA1\xF0\x9F\x98\xA0\xF0\x9F\x98\xA1\xF0\x9F\x98\xA0\xF0\x
9F\x98\xA1\xF0\x9F\x98\xA0\x0A\x0A\x00'

newStr = ""
for i in range(0, 56):
    newStr += chr(secret[i % len(secret)] ^ flag[i])
print(newStr)
```

```
___7h15_15_4_f4k3_f14g_y07U nNjD   0 ;OVIaM  ,qu<g ;K }Y

=== Code Execution Successful ===
```

But even if I take the script from the writeup, and I even change the byte value to hex to really confirm that is the variabl, I could not get the expected result.

It should be ___7h15_15_4_f4k3_f14g_y0u_w1ll_f41l_1f_y0u_subm17_17___ but I couldn't get it so no need to go further cause I don't know how.

## Ransomware

We got flag.enc and task.pyc. Uncompile the pyc file with uncompyle

```
┌──(kruphix㉿Zeqzoq)-[/mnt/c/Users/blast/Downloads]
└─$ uncompyle6 task.pyc
# uncompyle6 version 3.9.2
# Python bytecode version base 3.8.0 (3413)
# Decompiled from: Python 3.11.9 (main, Apr 10 2024, 13:16:36) [GCC 13.2.0]
# Embedded file name: task.py
# Compiled at: 2021-01-14 22:13:24
# Size of source mod 2**32: 420 bytes
(lambda data, key, iv: if len(data) != 0:
(lambda key, iv, data, AES: open("flag.enc", "wb").write(AES.new(key, AES.MODE_CBC, iv).encrypt(lambda x: x + 
b'\x00' * (16 - len(x) % 16)(data))))(data[key[:key + 16]], data[iv[:iv + 16]], open("flag.png", "rb").read(), 
__import__("Crypto.Cipher.AES").Cipher.AES) # Avoid dead code: lambda fn: 
__import__("os").remove(fn)("task.py"))(__import__("requests").get("https://ctf.bamboofox.tw/rules").text.encode()
, 99, 153)
# okay decompiling task.pyc
```

In the python code, we can see flag.png so I guess we need to reverse flag.enc to turn into flag.png. The code also send request to rules page. As I do not have the rules page, so I just take from the writeup. The writeup did the request in Postman and the response was:

```
<!DOCTYPE html>
<html>

<head>
	<title>BambooFox CTF</title>
	<meta charset="utf-8">
	<meta name="viewport" content="width=device-width, initial-scale=1.0">
	<link rel="shortcut icon"
```

```
                        href="/files/626f05557db4b8f323a06e0dfc7676d8/favicon-32x32-
a56b8e05e1d057431bef7fd212f394a18049e895a4db003909e9448478b8167d.png"
                        type="image/x-icon">
            <link rel="stylesheet" href="/themes/core/static/css/fonts.min.css?d=aa35138e">
            <link rel="stylesheet" href="/themes/core/static/css/main.min.css?d=aa35138e">
            <link rel="stylesheet" href="/themes/core/static/css/core.min.css?d=aa35138e">


...
```

[Writeup sentence] Looking at the python I saw that this was getting passed as data to the lambda function, I then opened up a python interpreter and saved the request response as data so that I could test the rest of the code. I then noticed that 99 was being passed as key and 153 was being passed as iv so I set up the python accordingly:

```
Python 3.8.6
[GCC 10.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> data = __import__('requests').get('https://ctf.bamboofox.tw/rules').text.encode()
>>> key = 99
>>> iv = 153
```

[writeup sentence] I then checked to see the two strings generated in the lambda function:

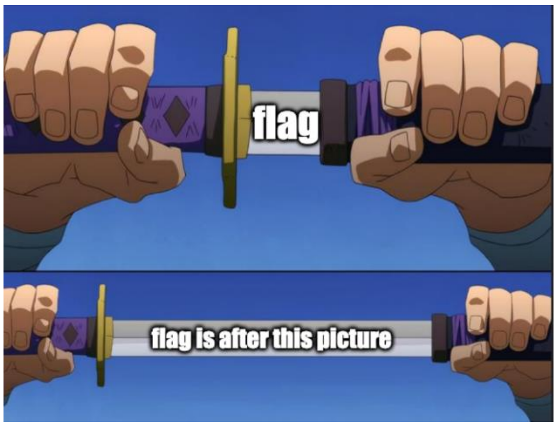
```
>>> data[key:key+16]
b'ewport" content='
>>> data[iv:iv+16]
b'">\n\t<link rel="s'
```

[writeup sentence] Both of these values were getting passed to another lambda and were then being used in order to create an AES (Advanced Encryption System) object which is from python's crypto module. After some research and refreshing my knowledge on AES I saw that as long as I had the key and iv I would be able to decrypt anything encrypted with AES. At this point I just put everything in a script and read in the flag.enc file and wrote the decrypted bytes into flag.png. Which resulted in the following code:

```
# Gets the data from the webpage
data = __import__('requests').get('https://ctf.bamboofox.tw/rules').text.encode()

key = 99

iv = 153

# Creates the AES object
AES = __import__('Crypto.Cipher.AES').Cipher.AES

# gets the real key from data
key = data[key:key+16]

# gets the real iv from data
iv = data[iv:iv+16]

# opens flag.enc so we can read bytes
ofile = open('flag.enc', 'rb')

# creates the decryption tool
decrypt = AES.new(key, AES.MODE_CBC, iv)
```

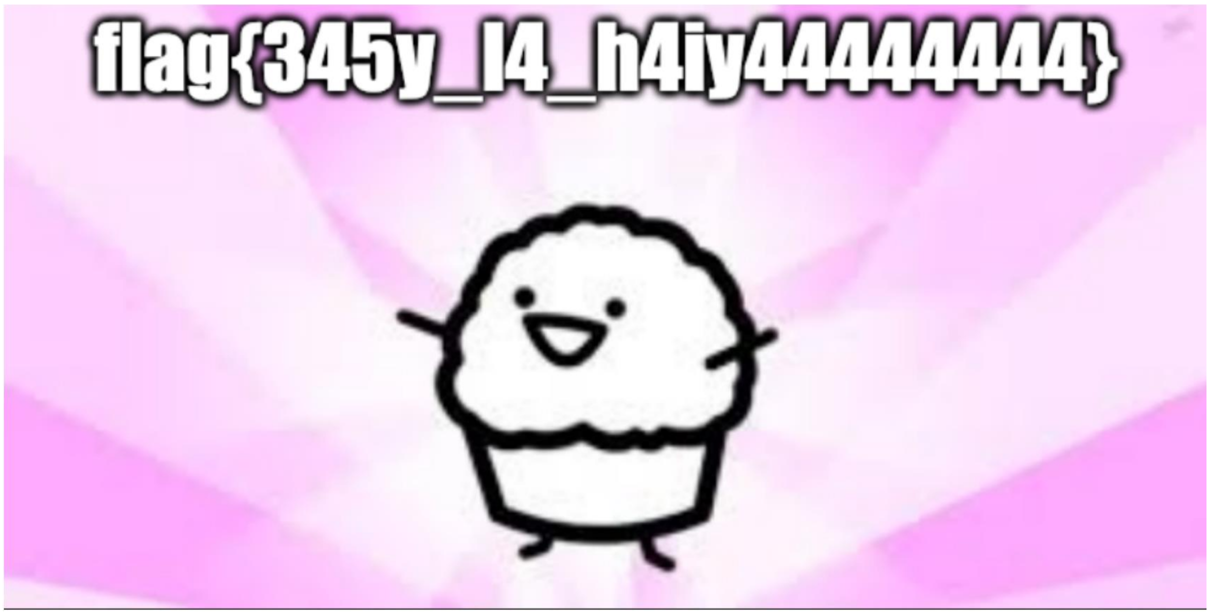
```
# opens the flag.png file so we can append bytes to it
nfile = open('flag.png', 'ab')

while True:
        # read 16 bytes at a time
        chunk = ofile.read(16)
        if len(chunk) == 0:
                break
        # decrypt the chunk and write it to the file
        nfile.write(decrypt.decrypt(chunk))
```

[writeup sentence] This python reads in the flag.enc file and decrypts it using the standard that was used to encrypt it in the original python code, the key and iv are pulled from the rules page of the CTF. Once I had finished the python script I ran it and ended up with this photo:



Then use binwalk on the png

```
$ binwalk ./flag.png

DECIMAL     HEXADECIMAL    DESCRIPTION
--------------------------------------------------------------------------------
0           0x0            PNG image, 980 x 746, 8-bit/color RGBA, non-interlaced
41          0x29           Zlib compressed data, default compression
808562      0xC5672        PNG image, 980 x 492, 8-bit/color RGBA, non-interlaced
808603      0xC569B        Zlib compressed data, default compression

$ binwalk -D=".*" ../flag.png
```


flag{345y_l4_h4iy44444444}

## Ware

When running strings on the file, it show UPX. So need to unpack with UPX

```
┌──(kruphix㉿Zeqzoq)-[/mnt/c/Users/blast/Downloads]
└─$ upx -d skidw4re
                Ultimate Packer for eXecutables
                  Copyright (C) 1996 - 2024
UPX 4.2.2      Markus Oberhumer, Laszlo Molnar & John Reiser   Jan 3rd 2024

        File size         Ratio      Format      Name
   --------------------   ------   -----------   -----------
   2052814 <-    706824   34.43%   linux/i386    skidw4re
```

When running the file:

```
┌──(kruphix㉿Zeqzoq)-[/mnt/c/Users/blast/Downloads]
└─$ ./skidw4re
This is the only message--------> ae385c6f1dd72132b2afcd4c25b9d35e0000000000000000
32 The message has been encrypted and written
```

Then use decompiler. The main function (too long):

```
void __cdecl main_main()
{
  ...
  main_Encryptfinal();
  *(_DWORD *)name = "encryptedmessage.txt";
  *(_DWORD *)&name[4] = 20;
  *(retval_80A6B20 *)&name[8] = os_Create(*(string *)name);
  v0 = *(os_File **)&name[8];
  err = *(error_0 *)&name[12];
  if ( *(_DWORD *)&name[12] )
  {
    v16 = 0;
    v17 = 0;
    if ( name == (_BYTE *)-48 )
      v16 = *(_DWORD *)&name[8];
    a.len = 1;
    a.cap = 1;
    ...
```

Went to look into main_Encryptfinal():

```
void __golang main_Encryptfinal()
{
  int v0; // eax
  uintptr v1; // eax
  interface_{} *array; // ebx
  _BYTE buf[28]; // [esp+0h] [ebp-84h] BYREF
  uint8 v4[32]; // [esp+1Ch] [ebp-68h] BYREF
  string pt; // [esp+3Ch] [ebp-48h]
  string key; // [esp+44h] [ebp-40h]
  string c; // [esp+4Ch] [ebp-38h]
  int v8; // [esp+54h] [ebp-30h]
  uintptr v9; // [esp+58h] [ebp-2Ch]
  int v10[2]; // [esp+5Ch] [ebp-28h] BYREF
  __int64 elem; // [esp+64h] [ebp-20h] BYREF
  __interface_{} a; // [esp+6Ch] [ebp-18h]
  int v13; // [esp+78h] [ebp-Ch]
  int v14; // [esp+7Ch] [ebp-8h]
  int v15; // [esp+80h] [ebp-4h] BYREF

  while ( (unsigned int)&v15 <= *(_DWORD *)(*(_DWORD *)(__readgsdword(0) - 4) + 8) )
    runtime_morestack_noctxt();
  pt.str = (uint8 *)"321174068998067 98980909";
  pt.len = 24;
  key.str = (uint8 *)"thisis32bitlongpassphraseimusing";
  *(_DWORD *)&buf[4] = "thisis32bitlongpassphraseimusing";
  key.len = 32;
  *(_DWORD *)&buf[8] = 32;
  *(__uint8 *)&buf[12] = runtime_stringtoslicebyte((uint8 (*)[32])v4, *(string *)&buf[4]);
  v13 = *(_DWORD *)&buf[12];
  *(_DWORD *)buf = *(_DWORD *)&buf[12];
  v14 = *(_DWORD *)&buf[16];
  *(_DWORD *)&buf[4] = *(_DWORD *)&buf[16];
  v15 = *(_DWORD *)&buf[20];
  *(_DWORD *)&buf[8] = *(_DWORD *)&buf[20];
  *(string *)&buf[20] = main_EncryptAES(*(__uint8 *)buf, pt);
  *(_DWORD *)&buf[4] = "This is the only message--------> ";
  *(_DWORD *)&buf[8] = 34;
  c = *(string *)&buf[20];
  *(_QWORD *)&buf[12] = *(_QWORD *)&buf[20];
  *(string *)&buf[20] = runtime_concatstring2(0, *(string (*)[2])&buf[4]);
  elem = *(_QWORD *)&buf[20];
  v10[0] = 0;
  v10[1] = 0;
  if ( buf == (_BYTE *)-92 )
    v10[0] = v0;
  a.len = 1;
  a.cap = 1;
  a.array = (interface_{} *)v10;
  runtime_convT2E((runtime__type_0 *)&stru_80F1F00, &elem, 0, *(runtime_eface_0 *)&buf[12]);
  v1 = *(_DWORD *)&buf[16];
  array = a.array;
  v8 = *(_DWORD *)&buf[12];
  a.array->_type = *(runtime__type_0 **)&buf[12];
  v9 = v1;
  if ( runtime_writeBarrier.enabled )
    runtime_writebarrierptr((uintptr *)&array->data, v1);
  else
    array->data = (void *)v1;
  fmt_Println(a, *(__int32 *)&buf[12], *(error_0 *)&buf[16]);
}
```

Here I noticed pt.str, since this was an encryption function I assumed pt stood for plain text, and the string matched the format the comment specified. The flag is

flag{32117406899806798980909}

## That's not crypto

We got checker.pyc. Bro this  a got like infinite number it takes like 15 pages so I just shortened it.

```
┌──(kruphix㉿Zeqzoq)-[/mnt/c/Users/blast/Downloads]
└─$ uncompyle6 checker.pyc
# uncompyle6 version 3.9.2
# Python bytecode version base 3.6 (3379)
# Decompiled from: Python 3.11.9 (main, Apr 10 2024, 13:16:36) [GCC 13.2.0]
# Embedded file name: checker.py
# Compiled at: 2021-01-30 23:41:40
# Size of source mod 2**32: 50109 bytes
from random import randint

def make_correct_array(s):
   from itertools import accumulate
   s = map(ord, s)
   s = accumulate(s)
   return [x * 69684751861829721459380039L for x in s]



def validate(a, xs):

   def poly(a, x):
      value = 0
      for ai in a:
         value *= x
         value += ai

      return value

   if len(a) != len(xs) + 1:
      return False
   else:
      for x in xs:
         value = poly(a, x)
         if value != 24196561:
            return False

      return True


if __name__ == "__main__":
   a = [1,
    -12036995612853156936286011036665L,
    70761097437137270936102167471287642036368358112612482078746420L,
    ...
    -
5587968083237306693195562096357832958348856200599199263740943377047831597170742714118363331
3452366519632038994934749477211189439167568620732596642583429996070406789358739943845558879
8256988677645323754039263287748208073986571463630315463528160656994318592529984130595009567
4512809966584000485313646221616954939163391472376548176842527697548931179097004421271969914
1950704222099324590163876778015516961878900287615622518902500543507497759966268832428491435
6331507363920217213649363265471629867298901042969782051702048331205489077043036293034833278
4451163949566660711426477882271152671165156885285036604399420394948135856011432574989596330
9460074720188424525321350221926995419621367736096982542278182384286274841887904108012587297
6097659234935939843831969071868329828198543934025101026210448471222980313395095525993255690
5220508512676888594007984236624024607124732938040311186947498957946575382475390782128841581
4837474250987144940583902612212267522458840062934016911150404288590075020587828603337942639
0866888430718628034165311726393903801759108001438550475452538760373040235963885593125407487
2878254972485538588994332813477762743909662550706683824097724659631763857519042863166397164
8982828592897863585535597074710503438803835031944735253826108788010971855260332980529686656
7042793649019459534848470724032360639799964633748905196867853384109201940701530483598626465
5504956790481598657539988749137879250002679708910831365499376856886175700227170381255365720
9817969355712664677608476840150892219635344560964752084747817741512138023811117859722715362
7832257799928795216823511402005832836435952524795942215581962882377976384250201837108076812
047801647103999999999995081L]
   a = [ai * 4919 for ai in a]
   flag_str = input("flag: ").strip()
   flag = make_correct_array(flag_str)
   if validate(a, flag):
      print("Yes, this is the flag!")
```

```
        print(flag_str)
    else:
        print("Incorrect, sorry. :(")

# okay decompiling checker.pyc
```

[writeup sentence] I ran a few tests and found out that a had a length of 58. Looking into the validate function I saw that this meant that the flag would need to be 57 characters long.

I then decided that my best course of action would be to simply use the python code to semi-brute force the correct values:

```
from random import randint

def make_correct_array(s):
    from itertools import accumulate
    s = map(ord, s)
    s = accumulate(s)
    return [x * 69684751861829721459380039 for x in s]


def validate(a, xs):

    def poly(a, x):
        value = 0
        for ai in a:
            value *= x
            value += ai

        return value

    for x in xs:
        value = poly(a, x)
        if value != 24196561:
            return False

    return True


if __name__ == '__main__':
    a = [...]
    a = [ai * 4919 for ai in a]

    flag_str = "justCTF{"

    while len(flag_str) < 57:
        i = 32
        while i < 127:
            print(flag_str + chr(i))
            flag = make_correct_array(flag_str + chr(i))
            if (validate(a, flag)):
                print("correct:     " + chr(i))
                flag_str += chr(i)
            i += 1

    print(flag_str)
```

[writeup sentence] I had done some testing so I knew that first part of the flag was justCTF{ which was the flag format. I then modified the validate function so that it no longer cared about the length of the input and instead only checked if it was correct.

I then went through for the length of the flag and added each possible char to the end of the string and "hashed" the string using their equation. I then checked each combination until I got the correct character, at which point I went to the next one.

The program then spit out the following flag:

justCTF{this_is_very_simple_flag_afer_so_big_polynomails}

## Solver

Got Elf file but its stripped

```
┌──(kruphix㉿Zeqzoq)-[/mnt/c/Users/blast/Downloads]
└─$ file crackme
crackme: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-
x86-64.so.2, BuildID[sha1]=6c8f4137ce00dd9571d3b551dc81d8d4354e4d91, for GNU/Linux 3.2.0, stripped
```

But can found the main function:

```
__int64 __fastcall main(int a1, char **a2, char **a3)
{
  int v3; // eax
  char buf[264]; // [rsp+0h] [rbp-110h] BYREF
  ssize_t v6; // [rsp+108h] [rbp-8h]

  printf("Enter key: ");
  v6 = read(0, buf, 0xFFuLL);
  buf[v6 - 1] = 0;
  if ( (unsigned int)sub_1200(buf, v6 - 1) )
  {
    printf("Congrats here is your flag: ");
    v3 = open("/flag", 0);
    sendfile(1, v3, 0LL, 0x100uLL);
  }
  else
  {
    puts("Invalid key");
  }
  return 0LL;
}
```

Sub_1200 function:

```
_BOOL8 __fastcall sub_1200(char *a1, unsigned __int64 a2)
{
  int i; // [rsp+1Ch] [rbp-4h]

  for ( i = 0; a2 > i; ++i )
  {
    if ( a1[i] <= 47 || a1[i] > 122 )
      return 0LL;
  }
  if ( *a1 + a1[3] != 100 )
```

```
    return 0LL;
  if ( a1[1] + a1[18] != 214 )
    return 0LL;
  if ( a1[2] + a1[4] != 178 )
    return 0LL;
  if ( ((unsigned __int8)a1[5] ^ (unsigned __int8)a1[6]) != 76 )
    return 0LL;
  if ( a1[8] - a1[7] != 17 )
    return 0LL;
  if ( a1[10] - a1[9] != 59 )
    return 0LL;
  if ( a1[12] + a1[11] - a1[13] != 69 )
    return 0LL;
  if ( a1[15] + a1[14] - a1[16] != 31 )
    return 0LL;
  if ( a1[16] + a1[17] - a1[18] == 88 )
    return ((unsigned __int8)(a1[20] ^ a1[19]) ^ (unsigned __int8)a1[21]) == 69;
  return 0LL;
}
```

We need 22 characters long that match with the condition. The script to get the phrase:

```
from z3 import *

# Creates solver
s = Solver()

# Creates an array of variables to solve for
flag = [BitVec(f"flag_{i}", 8) for i in range(0, 22)]

# checks that variables are in range
for i in range(0, 22):
  s.add(flag[i] >= 48)
  s.add(flag[i] <= 122)

# adds all the checks
s.add(flag[0] + flag[3] == 100)
s.add(flag[1] + flag[18] == 214)
s.add(flag[2] + flag[4] == 178)
s.add(flag[5] ^ flag[6] == 76)
s.add(flag[8] - flag[7] == 17)
s.add(flag[10] - flag[9] == 59)
s.add(flag[12] + flag[11] - flag[13] == 69)
s.add(flag[15] + flag[14] - flag[16] == 31)
s.add(flag[16] + flag[17] - flag[18] == 88)
s.add((flag[20] ^ flag[19]) ^ flag[21] == 69)

# print if we were able to solve or not
print(s.check())

# gets the variables
m = s.model()

# initializes an empty dictionary
t = {}

# parses the model to dictionary
for a in str(m)[1:-1].split(','):
  t[a.split('=')[0].strip()] = a.split('=')[1].strip()

# creates the string from the variables
s =""
```

```
for i in [BitVec(f"flag_{i}", 8) for i in range(0, 22)]:
   s += chr(int(t[str(i)]))

# prints the string
print(s)
```

Then got this

```
┌──(kruphix㉿Zeqzoq)-[/mnt/c/Users/blast/Downloads]
└─$ python3 solver.py
sat
4tx0:x4Rc=x0p[O@pJbO0:
```

```
nc 157.230.33.195 4444

Enter key: 4tx0:x4Rc=x0p[O@pJbO0:
Congrats here is your flag: Trollcat{z3_b4by}
```

Trollcat{z3_b4by}

## Keygenme Py

Got python file

```
username_trial = "PRITCHARD"
bUsername_trial = b"PRITCHARD"

key_part_static1_trial = "picoCTF{1n_7h3_|<3y_of_"
key_part_dynamic1_trial = "xxxxxxxx"
key_part_static2_trial = "}"
...
def check_key(key, username_trial):

   global key_full_template_trial

   if len(key) != len(key_full_template_trial):
      return False
   else:
      # Check static base key part --v
      i = 0
      for c in key_part_static1_trial:
         if key[i] != c:
            return False

         i += 1

      # TODO : test performance on toolbox container
      # Check dynamic part --v
      if key[i] != hashlib.sha256(username_trial).hexdigest()[4]:
         return False
      else:
         i += 1

      if key[i] != hashlib.sha256(username_trial).hexdigest()[5]:
         return False
      else:
         i += 1

      if key[i] != hashlib.sha256(username_trial).hexdigest()[3]:
        return False
    else:
        i += 1

    if key[i] != hashlib.sha256(username_trial).hexdigest()[6]:
        return False
    else:
        i += 1

    if key[i] != hashlib.sha256(username_trial).hexdigest()[2]:
        return False
    else:
        i += 1

    if key[i] != hashlib.sha256(username_trial).hexdigest()[7]:
        return False
    else:
        i += 1

    if key[i] != hashlib.sha256(username_trial).hexdigest()[1]:
        return False
    else:
        i += 1

    if key[i] != hashlib.sha256(username_trial).hexdigest()[8]:
        return False



    return True
```

Reverse the script

```
# import hashlib
import hashlib

# username
username_trial = "PRITCHARD"

# known flag
flag = "picoCTF{1n_7h3_|<3y_of_"

# decrypt flag
flag += hashlib.sha256(username_trial.encode()).hexdigest()[4]
flag += hashlib.sha256(username_trial.encode()).hexdigest()[5]
flag += hashlib.sha256(username_trial.encode()).hexdigest()[3]
flag += hashlib.sha256(username_trial.encode()).hexdigest()[6]
flag += hashlib.sha256(username_trial.encode()).hexdigest()[2]
flag += hashlib.sha256(username_trial.encode()).hexdigest()[7]
flag += hashlib.sha256(username_trial.encode()).hexdigest()[1]
flag += hashlib.sha256(username_trial.encode()).hexdigest()[8]

flag += '}'

# print flag
print(flag)
```

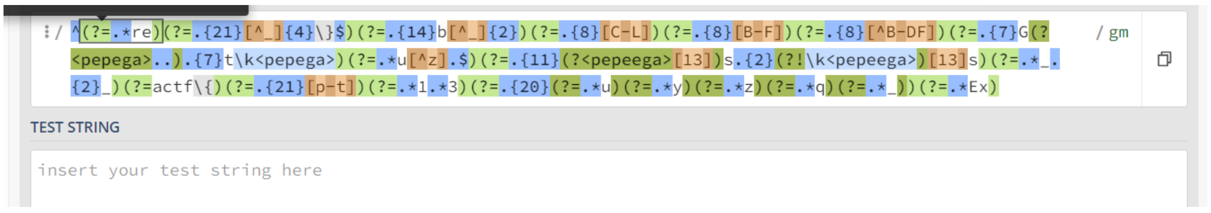
picoCTF{1n_7h3_|<3y_of_54ef6292}

## Revex

This is the question

^(?=.*re)(?=.{21}[^_]{4}\}$)(?=.{14}b[^_]{2})(?=.{8}[C-L])(?=.{8}[B-F])(?=.{8}[^B-DF])(?=.{7}G(?<pepega>..).{7}t\k<pepega>)(?=.*u[^z].$)(?=.{11}(?<pepeega>[13])s.{2}(?!\k<pepeega>)[13]s)(?=.*_.{2}_)(?=actf\{)(?=.{21}[p-t])(?=.*1.*3)(?=.{20}(?=.*u)(?=.*y)(?=.*z)(?=.*q)(?=.*_))(?=.*Ex)

This one can use https://regex101.com/ to debug. Putting all of it doesn't seem to do anything.



```
/ ^(?=.*re)(?=.{21}[^_]{4}\}$)(?=.{14}b[^_]{2})(?=.{8}[C-L])(?=.{8}[B-F])(?=.{8}[^B-DF])(?=.{7}G(?
<pepega>..).{7}t\k<pepega>)(?=.*u[^z].$)(?=.{11}(?<pepeega>[13])s.{2}(?!\k<pepeega>)[13]s)(?=.*_.
{2}_)(?=actf\{)(?=.{21}[p-t])(?=.*1.*3)(?=.{20}(?=.*u)(?=.*y)(?=.*z)(?=.*q)(?=.*_))(?=.*Ex)   / gm
```

TEST STRING

insert your test string here

- (?=.*re)

at some point in the string 're' will be present

- (?=.{21}[^_]{4}\}$)

after 21 characters there will be 4 characters that are not '_'

- (?=.{14}b[^_]{2})

after 14 characters there will be the character 'b'

after b there will be two characters that are not '_'

- (?=.{8}[C-L])

after 8 characters the character will be between C-L

- (?=.{8}[B-F])

after 8 characters the character will be between B-F

- (?=.{8}[^B-DF])

after 8 characters the character will not be B-D or F

this means the character will be 'E'

- (?=.{7}G(?<pepega>..).{7}t\k<pepega>)

after 7 characters we will have 'G'

then reads 2 characters and makes a copy in the group

then after 7 characters we will have 't'

then we will have the two characters we copied earlier

- (?=.*u[^z].$)

at some point we will have the character 'u' the character after u is not z

- (?=.{11}(?<pepeega>[13])s.{2}(?!\k<pepeega>)[13]s)

after 11 characters we will either have 1 or 3

this will be followed by 's'

then 2 characters

then either 1 or 3

then 's'

- (?=.*_.{2}_)

at some point there will be '_'

followed by two characters

then another '_'

- (?=actf\{)

the start is 'actf{'

- (?=.{21}[p-t])

after 21 characters we have a character in range p-t

- (?=.*1.*3)

at some point we have '1'

then at another point after we have '3'

- (?=.{20}(?=.*u)(?=.*y)(?=.*z)(?=.*q)(?=.*_))(?=.*Ex)

after 20 characters we have, in no particular order:

'u'

'y'

'z'

'q'

'_'

'Ex'

So got this

actf{reGEx_1s_b3stEx_qzuy}

## Backdoor

Got file named bd

```
┌──(zeqzoq㉿DESKTOP-TVA03PG)-[/mnt/c/Users/hzqzz/Downloads/tmp/htb]
```

```
└$ file bd
bd: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux 3.2.0, BuildID[sha1]=1da3a1d77c7109ce6444919f4a15e7e6c63d02fa, stripped
```

Running strings.

```
...
b_asyncio.cpython-38-x86_64-linux-gnu.so
b_bz2.cpython-38-x86_64-linux-gnu.so
b_codecs_cn.cpython-38-x86_64-linux-gnu.so
b_codecs_hk.cpython-38-x86_64-linux-gnu.so
b_codecs_iso2022.cpython-38-x86_64-linux-gnu.so
b_codecs_jp.cpython-38-x86_64-linux-gnu.so
b_codecs_kr.cpython-38-x86_64-linux-gnu.so
b_codecs_tw.cpython-38-x86_64-linux-gnu.so
b_contextvars.cpython-38-x86_64-linux-gnu.so
b_ctypes.cpython-38-x86_64-linux-gnu.so
b_decimal.cpython-38-x86_64-linux-gnu.so
b_hashlib.cpython-38-x86_64-linux-gnu.so
b_lzma.cpython-38-x86_64-linux-gnu.so
b_multibytecodec.cpython-38-x86_64-linux-gnu.so
b_multiprocessing.cpython-38-x86_64-linux-gnu.so
b_opcode.cpython-38-x86_64-linux-gnu.so
b_posixshmem.cpython-38-x86_64-linux-gnu.so
b_queue.cpython-38-x86_64-linux-gnu.so
b_ssl.cpython-38-x86_64-linux-gnu.so
blibbz2.so.1.0
blibcrypto.so.1.1
blibexpat.so.1
blibffi.so.6
bliblzma.so.5
blibmpdec.so.2
blibpython3.8.so.1.0
blibreadline.so.7
blibssl.so.1.1
blibtinfo.so.5
blibz.so.1
bmmap.cpython-38-x86_64-linux-gnu.so
breadline.cpython-38-x86_64-linux-gnu.so
bresource.cpython-38-x86_64-linux-gnu.so
btermios.cpython-38-x86_64-linux-gnu.so
xbase_library.zip
xinclude/python3.8/pyconfig.h
xlib/python3.8/config-3.8-x86_64-linux-gnu/Makefile
zPYZ-00.pyz
&libpython3.8.so.1.0
...
pydata
```

From here we know that this file is a python .pyc compiled with pyinstaller. Can confirm it with running binwalk and seeing zip archive containing pyinstaller.

## NET_DOT

Got .net file

```
┌──(zeqzoq㉿DESKTOP-TVA03PG)-[/mnt/c/Users/hzqzz/Downloads/tmp]
└─$ file win.dll
win.dll: PE32 executable (console) Intel 80386 Mono/.Net assembly, for MS Windows, 3 sections
```

Can open with dnspy. But I found decompiler.com

```
using System;

namespace win;

internal class Program
{
	private static int sum_all(string password)
	{
		int num = 0;
		foreach (char c in password)
		{
			num += c;
		}
		return num;
	}

	private static int check(int[] values)
	{
		int[] array = new int[26]
		{
			2410, 2404, 2430, 2408, 2391, 2381, 2333, 2396, 2369, 2332,
			2398, 2422, 2332, 2397, 2416, 2370, 2393, 2304, 2393, 2333,
			2416, 2376, 2371, 2305, 2377, 2391
		};
		int result = 0;
		for (int i = 0; i < array.Length; i++)
		{
			if (array[i] == values[i])
			{
				result = 1;
				continue;
			}
			result = 0;
			break;
		}
		return result;
	}

	private static void Main(string[] args)
	{
		Console.WriteLine("Hello there mate \nJust enter the flag to check : ");
		string text = Console.ReadLine();
		int[] array = new int[26];
		if (text.Length != 26)
		{
			Console.WriteLine("Input length error");
			Console.ReadLine();
			return;
		}
		for (int i = 0; i < text.Length; i++)
		{
```

```
                    array[i] = text[i];
                }
                int[] array2 = new int[26];
                for (int j = 0; j < 26; j++)
                {
                    array2[j] = (array[j] - (j % 2 * 2 + j % 3)) ^ sum_all(text);
                }
                int num = check(array2);
                if (num == 1)
                {
                    Console.WriteLine("Your flag : " + text);
                    Console.ReadLine();
                }
                else
                {
                    Console.WriteLine("try harder");
                    Console.ReadLine();
                }
        }
}
```

The flag start with GLUG{. Turn it into ascii and sum and got 71+76+85+71+123=2349. So here we use the known plaintext attack

```
# key in code
key = [2410, 2404, 2430, 2408, 2391, 2381, 2333, 2396, 2369, 2332, 2398, 2422, 2332, 2397, 2416, 2370, 2393, 2304, 2393, 2333, 2416, 2376, 2371, 2305, 2377, 2391]

# string to store result
s = ""

# loop through the 26 characters specified
for i in range(0, 26):

   # do the needed math
        s += chr((key[i]^2349)+((i%2)*2 + (i%3)))

# print
print(s)
```

GLUG{d0tn3t_1s_qu1t3_go0d}

## Numberical Computing

Got this file

```
┌──(zeqzoq㉿DESKTOP-TVA03PG)-[/mnt/c/Users/hzqzz/Downloads/tmp]
└─$ file try
try: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, BuildID[sha1]=f0e002ae3f9717bf0b1cdc8e90705dcd868e65b0, for GNU/Linux 3.2.0, with debug info, not stripped
```

```
└─$ ./try
 Enter the flag :
donno
 Wrong
```

The main function only got question()

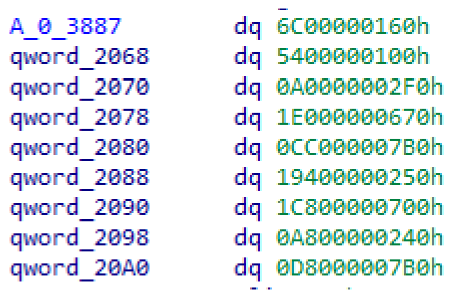
```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  _gfortran_set_args((unsigned int)argc, argv);
  _gfortran_set_options(7LL, &options_5_3914);
  question();
  return 0;
}
```

question()

```
void __cdecl question()
{
  int v0; // edx
  int v1; // [rsp+0h] [rbp-350h] BYREF
  int v2; // [rsp+4h] [rbp-34Ch]
  const char *v3; // [rsp+8h] [rbp-348h]
  int v4; // [rsp+10h] [rbp-340h]
  integer(kind_4) num2[18]; // [rsp+210h] [rbp-140h]
  integer(kind_4) num1[18]; // [rsp+260h] [rbp-F0h]
  integer(kind_4) num[18]; // [rsp+2B0h] [rbp-A0h]
  char inp[18]; // [rsp+300h] [rbp-50h] BYREF
  char enc[18]; // [rsp+320h] [rbp-30h] BYREF
  char c2[1]; // [rsp+33Eh] [rbp-12h]
  char c1[1]; // [rsp+33Fh] [rbp-11h]
  integer(kind_4) x2; // [rsp+340h] [rbp-10h]
  integer(kind_4) x1; // [rsp+344h] [rbp-Ch]
  integer(kind_4) n; // [rsp+348h] [rbp-8h]
  integer(kind_4) f; // [rsp+34Ch] [rbp-4h]

  f = 0;
  qmemcpy(enc, "QWERTYUIOPASDFGHJK", sizeof(enc));
  *(_QWORD *)num = A_0_3887;
  *(_QWORD *)&num[2] = 0x5400000100LL;
  *(_QWORD *)&num[4] = 0xA0000002F0LL;
  *(_QWORD *)&num[6] = 0x1E000000670LL;
  *(_QWORD *)&num[8] = 0xCC000007B0LL;
  *(_QWORD *)&num[10] = 0x19400000250LL;
  *(_QWORD *)&num[12] = 0x1C800000700LL;
  *(_QWORD *)&num[14] = 0xA800000240LL;
  *(_QWORD *)&num[16] = 0xD8000007B0LL;
  v3 = "/home/abhi/try2/main.f90";
  v4 = 14;
  v1 = 128;
  v2 = 6;
  _gfortran_st_write(&v1);
  _gfortran_transfer_character_write(&v1, "Enter the flag : you got itWrong", 17LL);
  _gfortran_st_write_done(&v1);
  v3 = "/home/abhi/try2/main.f90";
  v4 = 15;
  v1 = 128;
  v2 = 5;
  _gfortran_st_read(&v1);
  _gfortran_transfer_character(&v1, inp, 18LL);
  _gfortran_st_read_done(&v1);
  for ( n = 1; n <= 18; ++n )
  {
    c1[0] = enc[n - 1];
    c2[0] = inp[n - 1];
    x1 = (unsigned __int8)c1[0];
    x2 = (unsigned __int8)c2[0];
    num2[n - 1] = (unsigned __int8)c2[0] ^ (unsigned __int8)c1[0];
  }
  for ( n = 1; n <= 18; ++n )
  {
    if ( (n - 1) % 2 == 1 )
      v0 = 4 * num2[n - 1];
    else
      v0 = 16 * num2[n - 1];
    num1[n - 1] = v0;
  }
  for ( n = 1; n <= 18; ++n )
  {
    if ( num1[n - 1] == num[n - 1] )
      ++f;
  }
  v3 = "/home/abhi/try2/main.f90";
  if ( f == 18 )
  {
    v4 = 39;
    v1 = 128;
    v2 = 6;
    _gfortran_st_write(&v1);
    _gfortran_transfer_character_write(&v1, "you got itWrong", 10LL);
  }
  else
  {
    v4 = 41;
    v1 = 128;
    v2 = 6;
    _gfortran_st_write(&v1);
    _gfortran_transfer_character_write(&v1, "Wrong", 5LL);
  }
  _gfortran_st_write_done(&v1);
}
```

```
*(_QWORD *)num = A_0_3887;
  *(_QWORD *)&num[2] = 0x5400000100LL;
  *(_QWORD *)&num[4] = 0xA0000002F0LL;
  *(_QWORD *)&num[6] = 0x1E000000670LL;
  *(_QWORD *)&num[8] = 0xCC000007B0LL;
  *(_QWORD *)&num[10] = 0x19400000250LL;
  *(_QWORD *)&num[12] = 0x1C800000700LL;
  *(_QWORD *)&num[14] = 0xA800000240LL;
  *(_QWORD *)&num[16] = 0xD8000007B0LL;
```

For this QWORD part, Kasimir change these 64 bit value to 32 bit value. From this:



```
A_0_3887         dq 6C00000160h
qword_2068       dq 5400000100h
qword_2070       dq 0A0000002F0h
qword_2078       dq 1E000000670h
qword_2080       dq 0CC000007B0h
qword_2088       dq 19400000250h
qword_2090       dq 1C800000700h
qword_2098       dq 0A800000240h
qword_20A0       dq 0D8000007B0h
```

To this:

0x00000160, 0x0000006C, 0x00000100, 0x00000054, 0x000002F0, 0x000000A0, 0x00000670, 0x000001E0, 0x000007B0, 0x000000CC, 0x00000250, 0x00000194, 0x00000700, 0x000001C8, 0x00000240, 0x000000A8, 0x000007B0, 0x000000D8

Then he change QWERTYUIOPASDFGHJK ascii to hex:

0x51, 0x57, 0x45, 0x52, 0x54, 0x59, 0x55, 0x49, 0x4F, 0x50, 0x41, 0x53, 0x44, 0x46, 0x47, 0x48, 0x4A, 0x4B

Then make the solve script

```
# num 1 array
num = [0x00000160, 0x0000006C, 0x00000100, 0x00000054, 0x000002F0, 0x000000A0, 0x00000670,
0x000001E0, 0x000007B0, 0x000000CC, 0x00000250, 0x00000194, 0x00000700, 0x000001C8, 0x00000240,
0x000000A8, 0x000007B0, 0x000000D8]

# c1 array
c1 = [0x51, 0x57, 0x45, 0x52, 0x54, 0x59, 0x55, 0x49, 0x4F, 0x50, 0x41, 0x53, 0x44, 0x46, 0x47, 0x48, 0x4A, 0x4B]

# string for results
s = ""

# loop through 18 times for the flag
for i in range(0, 18):
  # try each combination of characters for each character
          for j in range(32, 128):
    # set x
                    x = 0

    # do the if/else that is done in the code
                    if i % 2 == 1:
      # do the xor
                              x = 4 * (c1[i] ^ j)
                    else:
      # do the xor
                              x = 16 * (c1[i] ^ j)

    # if it matches then add and break
                    if x == num[i]:
                              s += chr(j)
                              print(s)
      break
```

```
┌─(zeqzoq㉿DESKTOP-TVA03PG)-[/mnt/c/Users/hzqzz/Downloads/tmp]
```

```
└─$ python3 exploit.py
G
GL
GLU
GLUG
GLUG{
GLUG{q
GLUG{q2
GLUG{q21
GLUG{q214
GLUG{q214c
GLUG{q214cd
GLUG{q214cd6
GLUG{q214cd64
GLUG{q214cd644
GLUG{q214cd644c
GLUG{q214cd644cb
GLUG{q214cd644cb1
GLUG{q214cd644cb1}
```

## WarGames

Got pyc file. Use uncompyle6 or java decompiler

```
menu = [
 'HELP GAMES', 'LIST GAMES', 'PLAY <game>']
game = ["FALKEN'S MAZE", 'TIC TAC TOE ', 'GLOBAL THERMONUCLEAR WAR']

def validateLaunchcode(launchcode):
  if len(launchcode[::-2]) != 12 or len(launchcode[15:]) != 9:
    print(denied)
    return       return False
  clen = len(launchcode)
  l1 = launchcode[:8]
  cc = []
  for i in range(0, len(l1), 2):
    q = []
    q.append(ord(l1[i]))
    q.append(ord(l1[i + 1]))
    cc.append(q)
  else:
    enc = []
    for i in range(len(cc)):
      val1 = cc[i][0] << 1
      val1 ^= 69
      val2 = cc[i][1] << 2
      val2 ^= 10
      enc.append(val1)
      enc.append(val2)
    else:
      correct = [
       159, 218, 153, 214, 45, 206, 153, 374]
      if enc != correct:
        print('ACCESS DENIED ok')
        return            return False
      l2 = launchcode[8:16]
      key = 'PEACEOUT'
      res = []
      [res.append(ord(key[i]) - ord(l2[i])) if i & 1 == 1 else res.append(ord(key[i]) + ord(l2[i])) for i in range(len(l2))]
      ok = [
       192, 18, 117, -32, 120, -16, 173, -2]
      if ok != res:
        print('ACCESS DENIED')
        return              return False
      l3 = launchcode[int(2 * clen / 3):]
      KEY = "There's no way to win"
      I = 7
      KARMA = [
       123, 47, 86, 28, 74, 50, 32, 114]
      MISSILE = []
      for x in l3:
        MISSILE.append((ord(x) + I ^ ord(KEY[I])) % 255)
        I = (I + 1) % len(KEY)
      else:
        if KARMA == MISSILE:
          print(okk)
          exit()
```

This split the key into 3 different components and checked each, this also told me that the game I wanted to get was 'GLOBAL THERMONUCLEAR WAR'.

```
# first encrypted key
correct = [159, 218, 153, 214, 45, 206, 153, 374]

# string to store result
s = ""

# decrypt the first key
for i in range(0, len(correct), 2):
        val1 = correct[i]^69
        val2 = correct[i+1]^10
        s += (chr(val1 >> 1))
        s += (chr(val2 >> 2))

# print what we have so far
print(s)

# second encrypted key
ok = [192, 18, 117, -32, 120, -16, 173, -2]
key = 'PEACEOUT'

# decrypt the second key
for i in range(0, len(ok)):
  if i & 1 == 1:
    s += chr(ord(key[i])-ok[i])
  else:
    s += chr(-ord(key[i])+ok[i])

# print what we have so far
print(s)

# third encrypted key
KARMA = [123, 47, 86, 28, 74, 50, 32, 114]
KEY = "There's no way to win"
I = 7

# decrypt third key
for i in range(0, len(KARMA)):
  for j in range(32, 127):
    if ((j + I ^ ord(KEY[I])) % 255) == KARMA[i]:
      s += chr(j)
  I = (I+1)%len(KEY)

# print all
print(s)
```

```
m4n741n_
m4n741n_p34c3_XV
m4n741n_p34c3_XVT9022GLD
```

When entered into the "game" we got:

```
GLUG{15_7h15_r34l_0r_15_17_g4m3??}
```

## Function pointer fun

nc ctf2021.hackpack.club 10998

Main()

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
 int result;
 bool changed;
 int i;
 int (*fp)(void);
 char seed[5];
 unsigned __int64 v8;

 v8 = __readfsqword(0x28u);
 setvbuf(_bss_start, 0LL, 2, 0LL);
 *(_DWORD *)seed = 0;
 seed[4] = 0;
 printf("Hello, Mr. Eusk. \nPassword > ");
 __isoc99_scanf("%4s", seed);
 changed = 0;
 for ( i = 0; i <= 3; ++i )
 {
  if ( seed[i] )
   changed = 1;
 }
 if ( !changed )
 {
  puts("You gotta give an input!");
  result = 1;
 }
 else
 {
  fp = pickFunction(seed);
  ((void (__fastcall *)(char *))fp)(seed);
  result = 0;
 }
 return result;
}
```

pickFunction()

```
int (*__cdecl pickFunction(char *seed))(void)
{
  char res;

  res = (seed[2] | seed[3]) & (*seed | seed[1]);
  if ( res == 73 )
    return funTwo;
  if ( res > 0 && res <= 31 )
    return funOne;
  if ( res > 31 && res <= 63 )
    return funThree;
  if ( res <= 63 || res > 95 )
    return funFive;
  return funFour;
}
```

This function took in a char array containing 4 chars and then used those to pick a function. I then needed to check which function had what I needed. I found out that funTwo() had what I needed:

```
int __cdecl funTwo()
{
  FILE *fp;
  char flag[25];
  unsigned __int64 v3;

  v3 = __readfsqword(0x28u);
  fp = fopen("flag", "r");
  fgets(flag, 25, fp);
  puts(flag);
  return 1;
}
```

Then do Z3 script to know which 4 char I need

```
# import z3
from z3 import *

# instantiate solver
s = Solver()

# create the 4 values
a = BitVec(f'a', 8)
b = BitVec(f'b', 8)
c = BitVec(f'c', 8)
d = BitVec(f'd', 8)

# add constraints
s.add(a < 127)
s.add(b < 127)
s.add(c < 127)
s.add(d < 127)
s.add(a > 32)
s.add(b > 32)
s.add(c > 32)
s.add(d > 32)
s.add(((c|d)&(a|b))==73)

# check the solve
print(s.check())
print(s.assertions())
m = s.model()
```

```
# print the model
print(m)
```

Running

```
sat
[a < 127,
 b < 127,
 c < 127,
 d < 127,
 a > 32,
 b > 32,
 c > 32,
 d > 32,
 (c | d) & (a | b) == 73]
[b = 48, a = 105, c = 64, d = 73]
```

So the output is i0@I

```
nc ctf2021.hackpack.club 10998

Hello, Mr. Eusk.
Password > i0@I
flag{c1RcU1t5_R_fUn!2!}
```

## Bell

nc dctf-chall-bell.westeurope.azurecontainer.io 5311

main()

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  unsigned int v3;
  unsigned int v5;

  v3 = time(0LL);
  srand(v3);
  v5 = rand() % 5 + 8;
  printf("%d\n", v5);
  process(v5);
  return 0;
}
```

Processs()

```
__int64 __fastcall process(int a1)
{
  int v2;
  int i;
  __int64 v4;
  __int64 v5;
  unsigned __int64 v6;

  v6 = __readfsqword(0x28u);
  v2 = 1;
  for ( i = 1; i <= a1; ++i )
  {
    v5 = triangle((unsigned int)a1, (unsigned int)i);
    __isoc99_scanf(&unk_AA4, &v4);
    if ( v5 != v4 )
      v2 = 0;
  }
  if ( v2 == 1 )
    system("cat flag.txt");
  else
    puts("Better luck next time.");
  return 0LL;
}
```

Triangle()

```
__int64 __fastcall triangle(unsigned int a1, int a2)
{
  __int64 v3;

  if ( a2 > (int)a1 )
    return 0LL;
  if ( a1 == 1 && a2 == 1 )
    return 1LL;
  if ( a2 == 1 )
    return triangle(a1 - 1, a1 - 1);
  v3 = triangle(a1, (unsigned int)(a2 - 1));
  return v3 + triangle(a1 - 1, (unsigned int)(a2 - 1));
}
```

Triangle() was just a function that did some math so heres the script

```
# import pwn
from pwn import *

# open remote connection
r = remote("dctf-chall-bell.westeurope.azurecontainer.io", 5311)

# get the first line
num = r.recvline()

# turn the line into a number
num = int(num.decode().replace("\n", ""))

# triangle function
def triangle(a1, a2):
        if a2 > a1:
                return 0
        if a1 == 1 and a2 == 1:
                return 1
        if a2 == 1:
                return triangle(a1-1, a1-1)
        v3 = triangle(a1, a2 -1)
        return v3 + triangle(a1-1, a2-1)

# loop through X amount of times
for i in range(1, num + 1):

  # send the calculated value
        r.sendline(str(triangle(num, i)))

# print the flag
```

```
print(r.recvline())
```

```
[+] Opening connection to dctf-chall-bell.westeurope.azurecontainer.io on port 5311: Done
b'dctf{f1rst_step_t0wards_b3ll_l4bs}\n'
```

## Break making

### Main()

```
__int64 __fastcall main(int a1, char **a2, char **a3)
{
  __int64 v3;
  __int64 v4;
  __int64 v5;
  __int64 v6;
  int v7;
  char v9[136];
  unsigned __int64 v10;

  v10 = __readfsqword(0x28u);
  setbuf(stdin, 0LL);
  setbuf(stdout, 0LL);
  setbuf(stderr, 0LL);
  signal(14, handler);
  v3 = 0LL;
  qword_6440 = 0LL;
  do
  {
    alarm(*(_DWORD *)*(&off_6020 + v3));
    puts(*((const char **)*(&off_6020 + qword_6440) + 1));
    do
    {
      if ( fgets(v9, 128, stdin) )
      {
        v9[strcspn(v9, "\n")] = 0;
        v4 = (__int64)*(&off_6020 + qword_6440);
        v5 = *(_QWORD *)(v4 + 24);
        if ( v5 )
        {
          v6 = 0LL;
          while ( strcmp(v9, *(const char **)(v4 + 16 * v6 + 32)) )
          {
            if ( v5 == ++v6 )
              goto LABEL_17;
          }
          v7 = (*(__int64 (**)(void))(v4 + 16 * v6 + 40))();
          if ( v7 != -1 )
            continue;
        }
      }
LABEL_17:
      sub_24A0();
    }
    while ( v7 );
    ++qword_6440;
    puts("");
    v3 = qword_6440;
  }
  while ( (unsigned __int64)qword_6440 <= 0xA );
  alarm(0);
  puts("it's the next morning");
```

```
 if ( dword_641C )
 {
  if ( dword_6418 )
  {
   if ( dword_6414 )
   {
    if ( dword_6410 )
    {
     if ( dword_640C )
      sub_25C0();
     else
      puts("mom finds the fire alarm in the laundry room and accuses you of making bread");
    }
    else
    {
     puts("mom finds the window opened and accuses you of making bread");
    }
   }
   else
   {
    puts("mom finds burnt bread on the counter and accuses you of making bread");
   }
  }
  else
  {
   puts("mom finds flour on the counter and accuses you of making bread");
  }
 }
 else
 {
  puts("mom finds flour in the sink and accuses you of making bread");
 }
 return 0LL;
}
```

From this I could see that we had to enter several strings in order to pass the "tests" and make bread. The first thing I did here was look to see all the possible strings I could input and I found the following:

```
strings bread

/lib64/ld-linux-x86-64.so.2
...
flag.txt
it's the next morning
mom doesn't suspect a thing, but asks about some white dots on the bathroom floor
couldn't open/read flag file, contact an admin if running on server
mom finds flour in the sink and accuses you of making bread
mom finds flour on the counter and accuses you of making bread
mom finds burnt bread on the counter and accuses you of making bread
mom finds the window opened and accuses you of making bread
mom finds the fire alarm in the laundry room and accuses you of making bread
the tray burns you and you drop the pan on the floor, waking up the entire house
the flaming loaf sizzles in the sink
the flaming loaf sets the kitchen on fire, setting off the fire alarm and waking up the entire house
pull the tray out with a towel
there's no time to waste
pull the tray out
the window is closed
the fire alarm is replaced
you sleep very well
```

time to go to sleep
close the window
replace the fire alarm
brush teeth and go to bed
you've taken too long and fall asleep
the dough has risen, but mom is still awake
the dough has been forgotten, making an awful smell the next morning
the dough has risen
the bread needs to rise
wait 2 hours
wait 3 hours
the oven makes too much noise, waking up the entire house
the oven glows a soft red-orange
the dough is done, and needs to be baked
the dough wants to be baked
preheat the oven
preheat the toaster oven
mom comes home and finds the bowl
mom comes home and brings you food, then sees the bowl
the ingredients are added and stirred into a lumpy dough
mom comes home before you find a place to put the bowl
the box is nice and warm
leave the bowl on the counter
put the bowl on the bookshelf
hide the bowl inside a box
the kitchen catches fire, setting off the fire alarm and waking up the entire house
the bread has risen, touching the top of the oven and catching fire
45 minutes is an awfully long time
you've moved around too much and mom wakes up, seeing you bake bread
return upstairs
watch the bread bake
the sink is cleaned
the counters are cleaned
everything appears to be okay
the kitchen is a mess
wash the sink
clean the counters
get ready to sleep
the half-baked bread is disposed of
flush the bread down the toilet
the oven shuts off
cold air rushes in
there's smoke in the air
unplug the oven
unplug the fire alarm
open the window
you put the fire alarm in another room
one of the fire alarms in the house triggers, waking up the entire house
brother is still awake, and sees you making bread
you bring a bottle of oil and a tray
it is time to finish the dough
you've shuffled around too long, mom wakes up and sees you making bread
work in the kitchen
work in the basement
flour has been added
yeast has been added
salt has been added
water has been added
add ingredients to the bowl
add flour
add yeast
add salt
add water

```
we don't have that ingredient at home!
the timer makes too much noise, waking up the entire house
the bread is in the oven, and bakes for 45 minutes
you've forgotten how long the bread bakes
the timer ticks down
use the oven timer
set a timer on your phone
...
```

From here I then stated debugging and each time before inputting a string I was able to see which choices were possible for each item, I then got the following list:

```
add flour
add yeast
add salt
add water
hide the bowl inside a box
wait 3 hours
work in the basement
preheat the toaster oven
set a timer on your phone
watch the bread bake
pull the tray out with a towel
unplug the fire alarm
open the window
unplug the oven
clean the counters
flush the bread down the toilet
wash the sink
get ready to sleep
close the window
replace the fire alarm
brush teeth and go to bed
```

then pasted this into the netcat and the program ran successfully:

```
nc mc.ax 31796

add ingredients to the bowl
add flour
add yeast
add salt
add water
hide the bowl inside a box
wait 3 hours
work in the basement
preheat the toaster oven
set a timer on your phone
watch the bread bake
pull the tray out with a towel
unplug the fire alarm
open the window
unplug the oven
clean the counters
flush the bread down the toilet
wash the sink
get ready to sleep
close the window
replace the fire alarm
brush teeth and go to bedflour has been added
```

```
yeast has been added
salt has been added
water has been added

the ingredients are added and stirred into a lumpy dough
the box is nice and warm

the bread needs to rise
the dough has risen

it is time to finish the dough
you bring a bottle of oil and a tray

the dough is done, and needs to be baked
the oven glows a soft red-orange

the bread is in the oven, and bakes for 45 minutes
the timer ticks down

45 minutes is an awfully long time
the bread has risen, touching the top of the oven and catching fire

there's no time to waste
the flaming loaf sizzles in the sink

there's smoke in the air
you put the fire alarm in another room
cold air rushes in
the oven shuts off

the kitchen is a mess
the counters are cleaned
the half-baked bread is disposed of
the sink is cleaned
everything appears to be okay

time to go to sleep
the window is closed
the fire alarm is replaced

you sleep very well

it's the next morning
mom doesn't suspect a thing, but asks about some white dots on the bathroom floor
flag{m4yb3_try_f0ccac1a_n3xt_t1m3???0r_dont_b4k3_br3ad_at_m1dnight}
```

## Jumprope

### Main()

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  puts("Ice cream!");
  puts("Soda Pop!");
  puts("Cherry on top!");
  puts("Is your flag exact?");
  puts("Well, let's find out!");
  sleep(1u);
  puts("\nEighty-eight characters!");
  puts("A secret well kept!");
  puts("If you get it right,");
  puts("I'll shout CORRECT!\n");
  if ( !(unsigned int)checkFlag("I'll shout CORRECT!\n", argv) )
    printf("Nope!");
  return 0;
}
```

checkFlag()

```
__int64 checkFlag()
{
  char vars0[8];
  void *retaddr;

  printf(">>> ");
  __isoc99_scanf("%88s%c", &retaddr, &dead);
  for ( count = 8; count <= 95; ++count )
  {
    val = next(val);
    vars0[count] ^= x[count - 8] ^ val;
  }
  return 0LL;
}
```

From here I can see that the code reads our input and then we xor that data with some other xor'ed data.

Next()

```
unsigned __int64 __fastcall next(unsigned __int64 a1)
{
  int j;
  unsigned __int64 v4;
  __int64 v5;
  int i;

  for ( i = 0; i <= 7; ++i )
  {
    v5 = 0LL;
    v4 = a1;
    for ( j = 0; j <= 7; ++j )
    {
      if ( (unsigned int)test((unsigned int)(j + 1)) )
        v5 ^= v4 & 1;
      v4 >>= 1;
    }
    a1 = (v5 << 7) + (a1 >> 1);
  }
  return a1;
}
```

Test()

```
__int64 __fastcall test(int a1)
{
  int i; // [rsp+10h] [rbp-4h]

  if ( a1 == 1 )
    return 0LL;
  for ( i = 2; i < a1 - 1; ++i )
  {
    if ( !(a1 % i) )
      return 0LL;
  }
  return 1LL;
}
```

The x array

```
[0x00000000000000FD, 0x000000000000003C, 0x00000000000000C4, 0x000000000000000E,
0x0000000000000076, 0x00000000000000FF, 0x000000000000004B, 0x0000000000000045,
0x000000000000001F, 0x0000000000000040, 0x00000000000000F4, 0x00000000000000E6,
0x0000000000000080, 0x00000000000000B8, 0x00000000000000B5, 0x00000000000000E8,
0x0000000000000076, 0x000000000000008E, 0x000000000000003B, 0x00000000000000F8,
0x00000000000000E4, 0x00000000000000BD, 0x00000000000000C9, 0x00000000000000C7,
0x000000000000003F, 0x00000000000000E6, 0x00000000000000CF, 0x0000000000000015,
0x0000000000000094, 0x000000000000009A, 0x000000000000008A, 0x0000000000000028,
0x000000000000004E, 0x000000000000005E, 0x000000000000001E, 0x000000000000003F,
0x0000000000000025, 0x00000000000000D4, 0x000000000000002C, 0x00000000000000A9,
0x0000000000000036, 0x0000000000000028, 0x0000000000000042, 0x0000000000000040,
0x0000000000000093, 0x000000000000008D, 0x000000000000000F, 0x00000000000000FF,
0x00000000000000AE, 0x000000000000002B, 0x000000000000002B, 0x00000000000000DF,
0x000000000000007E, 0x000000000000001A, 0x000000000000004E, 0x0000000000000005,
0x0000000000000063, 0x00000000000000D0, 0x0000000000000088, 0x00000000000000E1,
0x00000000000000A1, 0x000000000000001F, 0x000000000000005A, 0x000000000000003D,
0x0000000000000036, 0x000000000000004F, 0x00000000000000AE, 0x0000000000000089,
0x000000000000007B, 0x00000000000000D7, 0x0000000000000027, 0x00000000000000D0,
0x0000000000000029, 0x00000000000000C0, 0x000000000000009E, 0x00000000000000F0,
0x0000000000000020, 0x00000000000000DF, 0x0000000000000069, 0x0000000000000077,
0x0000000000000094, 0x00000000000000E9, 0x0000000000000058, 0x000000000000000F,
0x00000000000000B8, 0x00000000000000EC, 0x00000000000000F9, 0x0000000000000024]
```

The script

```
x = [0x00000000000000FD, 0x000000000000003C, 0x00000000000000C4, 0x000000000000000E,
0x0000000000000076, 0x00000000000000FF, 0x000000000000004B, 0x0000000000000045,
0x000000000000001F, 0x0000000000000040, 0x00000000000000F4, 0x00000000000000E6,
0x0000000000000080, 0x00000000000000B8, 0x00000000000000B5, 0x00000000000000E8,
0x0000000000000076, 0x000000000000008E, 0x000000000000003B, 0x00000000000000F8,
0x00000000000000E4, 0x00000000000000BD, 0x00000000000000C9, 0x00000000000000C7,
0x000000000000003F, 0x00000000000000E6, 0x00000000000000CF, 0x0000000000000015,
0x0000000000000094, 0x000000000000009A, 0x000000000000008A, 0x0000000000000028,
0x000000000000004E, 0x000000000000005E, 0x000000000000001E, 0x000000000000003F,
0x0000000000000025, 0x00000000000000D4, 0x000000000000002C, 0x00000000000000A9,
0x0000000000000036, 0x0000000000000028, 0x0000000000000042, 0x0000000000000040,
0x0000000000000093, 0x000000000000008D, 0x000000000000000F, 0x00000000000000FF,
0x00000000000000AE, 0x000000000000002B, 0x000000000000002B, 0x00000000000000DF,
0x000000000000007E, 0x000000000000001A, 0x000000000000004E, 0x0000000000000005,
0x0000000000000063, 0x00000000000000D0, 0x0000000000000088, 0x00000000000000E1,
0x00000000000000A1, 0x000000000000001F, 0x000000000000005A, 0x000000000000003D,
0x0000000000000036, 0x000000000000004F, 0x00000000000000AE, 0x0000000000000089,
0x000000000000007B, 0x00000000000000D7, 0x0000000000000027, 0x00000000000000D0,
0x0000000000000029, 0x00000000000000C0, 0x000000000000009E, 0x00000000000000F0,
0x0000000000000020, 0x00000000000000DF, 0x0000000000000069, 0x0000000000000077,
0x0000000000000094, 0x00000000000000E9, 0x0000000000000058, 0x000000000000000F,
0x00000000000000B8, 0x00000000000000EC, 0x00000000000000F9, 0x0000000000000024]

val = 2

def test(b):
        if b == 1:
                return 0
        for i in range(2, b-1):
                if not (b%i):
                        return 0
        return 1

def next(a):
        for i in range(0, 8):
                v5 = 0
                v4 = a
                for j in range(0, 8):
                        if test(j+1):
                                v5 ^= v4 & 1
                        v4 >>= 1
                a = (v5 << 7) + (a >> 1)
        return a

def printA(s):
        n = ""
        for i in s:
                n+=i
        print(n)

s = []

for i in range(8, 96):
        val = next(val)
        s.append(chr(x[i-8] ^ val))
printA(s)
```

Then got

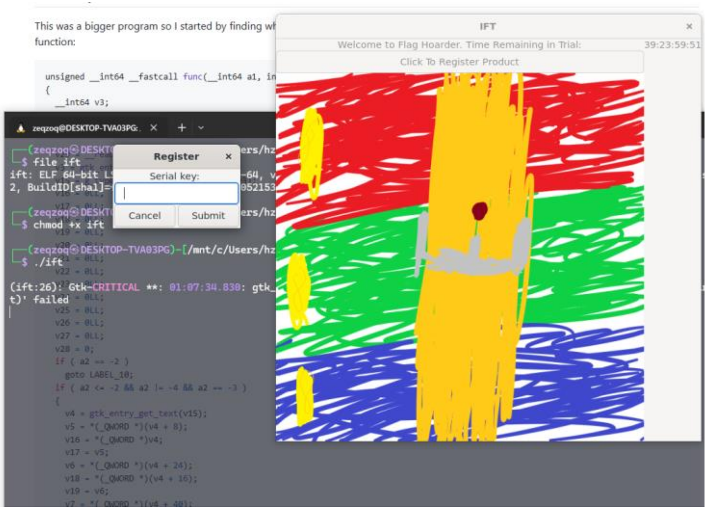
xq4f{n0tÔx!st_ni´¨CLbut_Zz%_nigh/M"ef0ref7enty_"5r_haczw2s_camîK!_kn0c¥Ã¹_at_økd00r}

the some bruteforcing and guessing

ictf{n0t_last_night_but_the_night_bef0re_twenty_f0ur_hackers_came_a_kn0cking_at_my_d00r}

## Infinite free trial

When we run the elf file we got this


This was a bigger program so I started by finding wh
function:

```
unsigned __int64 __fastcall func(__int64 a1, in
{
  __int64 v3;
```

```
┌──(zeqzoq㉿DESKT
└─$ file ift
ift: ELF 64-bit L
2, BuildID[sha1]=
┌──(zeqzoq㉿DESKT
└─$ chmod +x ift
┌──(zeqzoq㉿DESKTOP-TVA03PG)-[/mnt/c/Users/hz
└─$ ./ift

(ift:26): Gtk-CRITICAL **: 01:07:34.830: gtk_
t)' failed
```

```
    v25 = 0LL;
    v26 = 0LL;
    v27 = 0LL;
    v28 = 0;
    if ( a2 == -2 )
      goto LABEL_10;
    if ( a2 <= -2 && a2 != -4 && a2 == -3 )
    {
      v4 = gtk_entry_get_text(v15);
      v5 = *(_QWORD *)(v4 + 8);
      v16 = *(_QWORD *)v4;
      v17 = v5;
      v6 = *(_QWORD *)(v4 + 24);
      v18 = *(_QWORD *)(v4 + 16);
      v19 = v6;
```

zeqzoq@DESKTOP-TVA03PG:
Register
Serial key:
Cancel
Submit
IFT
Welcome to Flag Hoarder. Time Remaining in Trial:
39:23:59:51
Click To Register Product

But in main() there no string such as “serial key” or “register”. And the functions in the main() also doesn’t have the “main” feature. So need to check one by one.

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  __int64 v3; // rdi
  __int64 type; // rdx
  __int64 v5; // rax
  int v8; // [rsp+14h] [rbp-Ch]

  v3 = gtk_application_new((__int64)"win.rars.ift", 0LL, (__int64)envp);
  g_signal_connect_data(v3, "activate", activate, 0LL, 0LL, 0LL);
  type = g_application_get_type();
  v5 = g_type_check_instance_cast(v3, type);
  v8 = g_application_run(v5, (unsigned int)argc, argv);
  g_object_unref(v3);
  return v8;
}
```

So found func() which have serial key string

```
unsigned __int64 __fastcall func(__int64 a1, int a2, __int64 a3)
{
  __int64 type; // rax
  __int64 text; // rax
  __int64 v5; // rbx
  __int64 v6; // rbx
  __int64 v7; // rbx
  __int64 v8; // rbx
  __int64 v9; // rbx
  __int64 v10; // rbx
  __int64 v11; // rax
  __int64 v12; // rax
  __int64 v15; // [rsp+28h] [rbp-88h]
  __int64 v16; // [rsp+30h] [rbp-80h] BYREF
  __int64 v17; // [rsp+38h] [rbp-78h]
  __int64 v18; // [rsp+40h] [rbp-70h]
  __int64 v19; // [rsp+48h] [rbp-68h]
  __int64 v20; // [rsp+50h] [rbp-60h]
  __int64 v21; // [rsp+58h] [rbp-58h]
  __int64 v22; // [rsp+60h] [rbp-50h]
  __int64 v23; // [rsp+68h] [rbp-48h]
  __int64 v24; // [rsp+70h] [rbp-40h]
  __int64 v25; // [rsp+78h] [rbp-38h]
  __int64 v26; // [rsp+80h] [rbp-30h]
  __int64 v27; // [rsp+88h] [rbp-28h]
  int v28; // [rsp+90h] [rbp-20h]
  unsigned __int64 v29; // [rsp+98h] [rbp-18h]

  v29 = __readfsqword(0x28u);
  type = gtk_entry_get_type();
  v15 = g_type_check_instance_cast(a3, type);
  v16 = 0LL;
  v17 = 0LL;
  v18 = 0LL;
  v19 = 0LL;
  v20 = 0LL;
  v21 = 0LL;
  v22 = 0LL;
  v23 = 0LL;
  v24 = 0LL;
  v25 = 0LL;
  v26 = 0LL;
  v27 = 0LL;
  v28 = 0;
  if ( a2 == -2 )
    goto LABEL_10;
  if ( a2 <= -2 && a2 != -4 && a2 == -3 )
  {
    text = gtk_entry_get_text(v15);
    v5 = *(_QWORD *)(text + 8);
    v16 = *(_QWORD *)text;
    v17 = v5;
    v6 = *(_QWORD *)(text + 24);
    v18 = *(_QWORD *)(text + 16);
    v19 = v6;
    v7 = *(_QWORD *)(text + 40);
    v20 = *(_QWORD *)(text + 32);
    v21 = v7;
    v8 = *(_QWORD *)(text + 56);
    v22 = *(_QWORD *)(text + 48);
    v23 = v8;
    v9 = *(_QWORD *)(text + 72);
    v24 = *(_QWORD *)(text + 64);
    v25 = v9;
    v10 = *(_QWORD *)(text + 88);
    v26 = *(_QWORD *)(text + 80);
    v27 = v10;
    v28 = *(_DWORD *)(text + 96);
    if ( (unsigned int)do_crc_check(&v16) )
    {
      if ( (unsigned int)do_xor_check(&v16) )
        registered = 1;
      else
        puts("Invalid Serial Key 2");
    }
    else
    {
      puts("Invalid Serial Key 1");
    }
LABEL_10:
    v11 = gtk_widget_get_type();
    v12 = g_type_check_instance_cast(a1, v11);
    gtk_widget_destroy(v12);
  }
  return v29 - __readfsqword(0x28u);
}
```

## Do_crc_check()

```
_BOOL8 __fastcall do_crc_check(__int64 a1)
{
  int i; // [rsp+1Ch] [rbp-4h]

  for ( i = 0; i <= 6; ++i )
    crcout[i] = crccheck[(unsigned __int8)crc8(6 * i + a1, 6LL)];
  return memcmp(crcout, "w1nR4rs", 7uLL) == 0;
}
```

```
__int64 __fastcall crc8(unsigned __int8 *a1, int a2)
{
  unsigned int v3; // [rsp+18h] [rbp-14h]
  int i; // [rsp+1Ch] [rbp-10h]

  v3 = 0;
  while ( a2 )
  {
    v3 ^= *a1 << 8;
    for ( i = 8; i; --i )
    {
      if ( (v3 & 0x8000) != 0 )
        v3 ^= 0x8380u;
      v3 *= 2;
    }
    --a2;
    ++a1;
  }
  return v3 >> 8;
}
```

Do_xor_check()

```
_BOOL8 __fastcall do_xor_check(__int64 a1)
{
  int i; // [rsp+1Ch] [rbp-4h]

  for ( i = 0; i <= 5; ++i )
    xor_block(6 * i + a1, 6 * (i + 1) + a1, (char *)&xorout + 6 * i, 6LL);
  return memcmp(&xorout, &xorcheck, 0x24uLL) == 0;
}
```

```
__int64 __fastcall xor_block(__int64 a1, __int64 a2, __int64 a3, int a4)
{
  __int64 result; // rax
  unsigned int i; // [rsp+28h] [rbp-4h]

  for ( i = 0; ; ++i )
  {
    result = i;
    if ( (int)i >= a4 )
      break;
    *(_BYTE *)((int)i + a3) = *(_BYTE *)((int)i + a1) ^ *(_BYTE *)((int)i + a2);
  }
  return result;
}
```

For do_crc_check we take the input string, pass the string 6 bytes at a time to crc8, and use the output from that as an index for the crccheck array and add it to a string. That string then needs to equal w1nR4rs. So crc_check array:

```
[0xD6, 0xD4, 0x4D, 0x46, 0x53, 0xCD, 0x3E, 0xC7, 0x41, 0x6D, 0x50, 0x8A, 0x22, 0xBF, 0x2C, 0x8E, 0x09, 0x9C,
0x01, 0x55, 0x10, 0x35, 0xF4, 0xC5, 0x6B, 0x68, 0xD8, 0x4F, 0xD5, 0x15, 0x13, 0xA8, 0x08, 0xD3, 0x42, 0x32, 0x54,
0x06, 0x94, 0xA1, 0xE0, 0xFB, 0xAD, 0xFF, 0x5F, 0x9E, 0x31, 0x82, 0x02, 0xCA, 0x1E, 0xF2, 0x4A, 0xD7, 0xE2, 0x47,
0x48, 0x66, 0x80, 0x14, 0x67, 0xDA, 0x27, 0x2D, 0x62, 0xE8, 0x40, 0x11, 0x23, 0x21, 0x84, 0x81, 0x74, 0x17, 0xBE,
0xCE, 0x9B, 0x92, 0xB5, 0x0E, 0xC6, 0xF0, 0x99, 0xF7, 0xA6, 0xDF, 0x3A, 0x76, 0xDD, 0x7C, 0xD1, 0xF6, 0xA9,
0xE9, 0xB7, 0x07, 0x97, 0x7A, 0xC2, 0x7E, 0x90, 0xB3, 0x4C, 0x30, 0x5D, 0xFD, 0x45, 0x85, 0xA3, 0x75, 0xE3, 0xF3,
0x49, 0xBD, 0x0D, 0x38, 0xB4, 0x8B, 0xB9, 0xFA, 0xAA, 0x59, 0xB2, 0x2B, 0x6A, 0xCF, 0x0B, 0xE6, 0x05, 0x63,
0x3C, 0xBC, 0xE5, 0x87, 0x79, 0x88, 0xA5, 0x03, 0x34, 0x43, 0xEF, 0x1D, 0x7D, 0x89, 0xF1, 0x58, 0x33, 0xB1, 0x78,
0x83, 0x95, 0x7F, 0xDB, 0x7B, 0xB6, 0xF5, 0x1B, 0x2F, 0xBA, 0x37, 0x8D, 0x18, 0x12, 0xD0, 0x73, 0xE7, 0x3F, 0x70,
0xA7, 0x0C, 0x0A, 0x64, 0x9F, 0x71, 0x6C, 0xAE, 0x28, 0xEB, 0x96, 0xB8, 0xA2, 0x19, 0x8F, 0x86, 0xD9, 0x0F, 0xDC,
0xC9, 0xF9, 0x39, 0x5E, 0xAB, 0x51, 0xCB, 0xC1, 0x25, 0x20, 0x65, 0x44, 0xEE, 0x5C, 0x3B, 0xA4, 0x1F, 0xCC,
0xAF, 0x29, 0xC8, 0x2A, 0x60, 0xAC, 0x61, 0x5A, 0xF8, 0x5B, 0x4B, 0x93, 0xEC, 0x8C, 0x9D, 0xA0, 0xC3, 0xDE,
0x98, 0xBB, 0x36, 0xE4, 0xEA, 0x72, 0x00, 0x3D, 0xB0, 0x24, 0x4E, 0x77, 0x6F, 0x52, 0xFE, 0xC0, 0x1A, 0x91, 0x69,
0x56, 0x2E, 0x9A, 0x16, 0xFC, 0x04, 0xE1, 0x26, 0x1C, 0x57, 0xED, 0xD2, 0x6E, 0xC4]
```

The xor check takes the input string 12 bytes at a time, it then splits those 12 in half and xor's the two bytes from each half, saving them into xorout. It then checks this against xorcheck.
Xorcheck array:

```
[0x09, 0x16, 0x17, 0x0F, 0x17, 0x56, 0x16, 0x44, 0x3A, 0x18, 0x53, 0x6F, 0x14, 0x03, 0x2A, 0x06, 0x6F, 0x31, 0x1C,
0x47, 0x2A, 0x06, 0x2D, 0x5F, 0x51, 0x1B, 0x00, 0x46, 0x4A, 0x00, 0x04, 0x55, 0x66, 0x50, 0x01, 0x4C]
```

So knowing all the required functions and the array, we can build a script with z3

```
# import z3
from z3 import *

# crccheck bytes
crccheck = [0xD6, 0xD4, 0x4D, 0x46, 0x53, 0xCD, 0x3E, 0xC7, 0x41, 0x6D, 0x50, 0x8A, 0x22, 0xBF, 0x2C, 0x8E,
0x09, 0x9C, 0x01, 0x55, 0x10, 0x35, 0xF4, 0xC5, 0x6B, 0x68, 0xD8, 0x4F, 0xD5, 0x15, 0x13, 0xA8, 0x08, 0xD3, 0x42,
0x32, 0x54, 0x06, 0x94, 0xA1, 0xE0, 0xFB, 0xAD, 0xFF, 0x5F, 0x9E, 0x31, 0x82, 0x02, 0xCA, 0x1E, 0xF2, 0x4A, 0xD7,
0xE2, 0x47, 0x48, 0x66, 0x80, 0x14, 0x67, 0xDA, 0x27, 0x2D, 0x62, 0xE8, 0x40, 0x11, 0x23, 0x21, 0x84, 0x81, 0x74,
0x17, 0xBE, 0xCE, 0x9B, 0x92, 0xB5, 0x0E, 0xC6, 0xF0, 0x99, 0xF7, 0xA6, 0xDF, 0x3A, 0x76, 0xDD, 0x7C, 0xD1,
0xF6, 0xA9, 0xE9, 0xB7, 0x07, 0x97, 0x7A, 0xC2, 0x7E, 0x90, 0xB3, 0x4C, 0x30, 0x5D, 0xFD, 0x45, 0x85, 0xA3, 0x75,
0xE3, 0xF3, 0x49, 0xBD, 0x0D, 0x38, 0xB4, 0x8B, 0xB9, 0xFA, 0xAA, 0x59, 0xB2, 0x2B, 0x6A, 0xCF, 0x0B, 0xE6,
0x05, 0x63, 0x3C, 0xBC, 0xE5, 0x87, 0x79, 0x88, 0xA5, 0x03, 0x34, 0x43, 0xEF, 0x1D, 0x7D, 0x89, 0xF1, 0x58, 0x33,
0xB1, 0x78, 0x83, 0x95, 0x7F, 0xDB, 0x7B, 0xB6, 0xF5, 0x1B, 0x2F, 0xBA, 0x37, 0x8D, 0x18, 0x12, 0xD0, 0x73, 0xE7,
0x3F, 0x70, 0xA7, 0x0C, 0x0A, 0x64, 0x9F, 0x71, 0x6C, 0xAE, 0x28, 0xEB, 0x96, 0xB8, 0xA2, 0x19, 0x8F, 0x86, 0xD9,
0x0F, 0xDC, 0xC9, 0xF9, 0x39, 0x5E, 0xAB, 0x51, 0xCB, 0xC1, 0x25, 0x20, 0x65, 0x44, 0xEE, 0x5C, 0x3B, 0xA4, 0x1F,
0xCC, 0xAF, 0x29, 0xC8, 0x2A, 0x60, 0xAC, 0x61, 0x5A, 0xF8, 0x5B, 0x4B, 0x93, 0xEC, 0x8C, 0x9D, 0xA0, 0xC3,
0xDE, 0x98, 0xBB, 0x36, 0xE4, 0xEA, 0x72, 0x00, 0x3D, 0xB0, 0x24, 0x4E, 0x77, 0x6F, 0x52, 0xFE, 0xC0, 0x1A, 0x91,
0x69, 0x56, 0x2E, 0x9A, 0x16, 0xFC, 0x04, 0xE1, 0x26, 0x1C, 0x57, 0xED, 0xD2, 0x6E, 0xC4]

# xorcheck bytes
xorcheck = [0x09, 0x16, 0x17, 0x0F, 0x17, 0x56, 0x16, 0x44, 0x3A, 0x18, 0x53, 0x6F, 0x14, 0x03, 0x2A, 0x06, 0x6F,
0x31, 0x1C, 0x47, 0x2A, 0x06, 0x2D, 0x5F, 0x51, 0x1B, 0x00, 0x46, 0x4A, 0x00, 0x04, 0x55, 0x66, 0x50, 0x01, 0x4C]

# string that we check against
crcstr = "w1nR4rs"

# turn string into array of indices that get the string we want
s = []
for i in crcstr:
        for j in crccheck:
                if i == chr(j):
                        s.append(crccheck.index(j))
crcstr = s

# crc8 function
def crc8(a1, a2):
        v3 = 0
        c = 0
        while a2 != 0:
                v3 ^= (a1[c] << 8)
                for i in range(8, 0, -1):
                        v3 = If((v3 & 0x8000) != 0, v3 ^ 0x8380, v3)
                        v3 *= 2
                a2 -= 1
                c += 1
        return v3 >> 8

# xorblock function
def xor_block(s, a1, a2, flag):
        xorout = [0] * 6
        for i in range(0, 6):
                xorout[i] = flag[a1 + i] ^ flag[a2 + i]
        s.add(xorout[0] == xorcheck[a1])
        s.add(xorout[1] == xorcheck[a1+1])
        s.add(xorout[2] == xorcheck[a1+2])
        s.add(xorout[3] == xorcheck[a1+3])
        s.add(xorout[4] == xorcheck[a1+4])
        s.add(xorout[5] == xorcheck[a1+5])

# initialize the flag
flag = [BitVec(f'flag[{i}]', 32) for i in range(0,42)]

# create solver
s = Solver()

# add contraint for characters
for i in range(0, len(flag)):
        s.add(flag[i] >= 32)
        s.add(flag[i] <= 127)

# add do_crc_check function constraints
for i in range(0, 7):
        s.add(crcstr[i] == (0xFF & crc8([flag[(i * 6)], flag[(i * 6) + 1], flag[(i * 6) + 2], flag[(i * 6) + 3], flag[(i * 6) + 4],
flag[(i * 6) + 5]], 6)))

# add do_xor_check constraints
for i in range(0, 6):
        xor_block(s, i * 6, (i + 1) * 6, flag)

# Add rarctf{ as a constraint so we get the flag since there are multiple solutions
s.add(flag[0] == ord("r"))
s.add(flag[1] == ord("a"))
s.add(flag[2] == ord("r"))
s.add(flag[3] == ord("c"))
s.add(flag[4] == ord("t"))
s.add(flag[5] == ord("f"))
s.add(flag[6] == ord("{"))

# run the model
print(s.check())
m = s.model()

s = ""

# convert model to flag and print
for i in range(0, len(flag)):
        s += chr(int(str(m[flag[i]])))

print(s)
```

rarctf{welc0m3_t0_y0ur_new_tr14l_281099b9}

## Very TriVial Reversing

Main_main()

```
__int64 main__main()
{
  __int64 v0;
  __int64 result;
  __int64 v2[4];
  __int64 v3[2];

  v2[2] = (__int64)L_1252;
  v2[3] = 0x10000000ELL;
  v2[0] = os__input(L_1252, 0x10000000ELL);
  v2[1] = v0;
  memmove_plt(v3, v2, 16LL);
  result = (unsigned __int8)main__check(v3[0], v3[1]);
  if ( (_BYTE)result )
    result = println(L_1254, 0x100000003LL);
  return result;
}
```

This function read the user input into v2 as a pointer and then called main__check:

```
__int64 __fastcall main__check(__int64 a1, __int64 a2)
{
  int v2;
...
  __int64 v57;

  v56 = a1;
  v57 = a2;
  _new_array_with_default(v54, 0LL, 0LL, 1LL, 0LL);
  memmove_plt(v55, v54, 32LL);
  v51[0] = 19;
  v51[1] = 55;
  new_array_from_c_array(v52, 2LL, 2LL, 4LL, v51);
  memmove_plt(v53, v52, 32LL);
  for ( i = 0; i < (int)v57; ++i ) // loop through the length of the input
  {
    v49 = *(_BYTE *)(i + v56);
    v47 = &v29;
    memmove_plt(&v29, v53, 32LL);
    v6 = (_DWORD *)array_get(0, (unsigned int)v53, v2, v3, v4, v5, v29, v30, v31); // get a value
    v46 = *v6 ^ v49; // xor that value
    v47 = &v29;
    memmove_plt(&v29, v53, 32LL);
    v11 = (_DWORD *)array_get(1, (unsigned int)v53, v7, v8, v9, v10, v29, v30, v31); // get another value
    v48 = *v11 + v46; // add that value
    array_push(v55, &v48);
    v47 = &v29;
    memmove_plt(&v29, v53, 32LL);
    v44[0] = *(_DWORD *)array_get(1, (unsigned int)v53, v12, v13, v14, v15, v29, v30, v31);
    v47 = &v29;
    memmove_plt(&v29, v53, 32LL);
    v44[1] = *(_DWORD *)array_get(0, (unsigned int)v53, v16, v17, v18, v19, v29, v30, v31);
    new_array_from_c_array(v45, 2LL, 2LL, 4LL, v44); // swap the locations of the two values
    memmove_plt(v53, v45, 32LL);
  }
  memmove_plt(v41, v55, 32LL);
  v40 = v43;
  _new_array(v38, 0LL, v43, 4LL);
  memmove_plt(v39, v38, 32LL);
  for ( j = 0; j < v40; ++j )
  {
    v36 = *(_BYTE *)(j + v42);
    v35 = anon_fn_e2d96d4126f6333f_byte__int_215(v36);
    array_push(v39, &v35);
  }
  v33[0] = 152; // create encrypted flag
  v33[1] = 105;
  v33[2] = 152;
  v33[3] = 103;
  v33[4] = 158;
  v33[5] = 100;
  v33[6] = 159;
  v33[7] = 119;
  v33[8] = 173;
  v33[9] = 101;
  v33[10] = 118;
  v33[11] = 118;
  v33[12] = 178;
  v33[13] = 105;
  v33[14] = 158;
  v33[15] = 115;
  v33[16] = 169;
  v33[17] = 87;
  v33[18] = 180;
  v33[19] = 35;
  v33[20] = 158;
  v33[21] = 119;
  v33[22] = 179;
  v33[23] = 146;
  v33[24] = 169;
  v33[25] = 88;
  v33[26] = 174;
  v33[27] = 45;
  v33[28] = 89;
  v33[29] = 101;
  v33[30] = 168;
  v33[31] = 21;
  v33[32] = 89;
  v33[33] = 33;
  v33[34] = 173;
  v33[35] = 102;
  v33[36] = 165;
  new_array_from_c_array(v34, 37LL, 37LL, 4LL, v33);
  v47 = &v29;
  memmove_plt(&v29, v34, 32LL);
  v47 = &v25;
  memmove_plt(&v25, v39, 32LL);
  if ( !(unsigned __int8)Array_int_arr_eq(
                   (unsigned int)&v25,
                   (unsigned int)v39,
                   v20,
                   v21,
                   v22,
                   v23,
                   v25,
                   v26,
                   v27,
                   v28,
                   v29,
                   v30,
                   v31) ) // check the flag
    return 0LL;
  v32 = 1;
  return 1LL;
}
```

The program went through the entire input we had and xor'ed it by a, then added by b, then swapped a and b for the next one. The script:

```
# encrypted flag
```

```
x = [152, 105, 152, 103, 158, 100, 159, 119, 173, 101, 118, 118, 178, 105, 158, 115, 169, 87, 180, 35, 158, 119, 179,
146, 169, 88, 174, 45, 89, 101, 168, 21, 89, 33, 173, 102, 165]

# a and b
a = 0x13
b = 0x37

s = ""

# go through length of encrypted bytes
for i in range(len(x)):

  # subtract b and xor a
            s += chr(0xFF & ((x[i] - b) ^ a))

  # swap a and b
            c = a
            a = b
            b = c

# print flag
print(s)
```

rarctf{See,ThatWasn'tSoHard-1eb519ed}

## Medium

### Hash

flag{key1+key2}

Using strings on the file I noticed that this file had been packed with UPX so the first thing I did was unpack the file. Then run the file

```
$ ./keyjoinfile
Oops wrong path
Oops wrong path
```

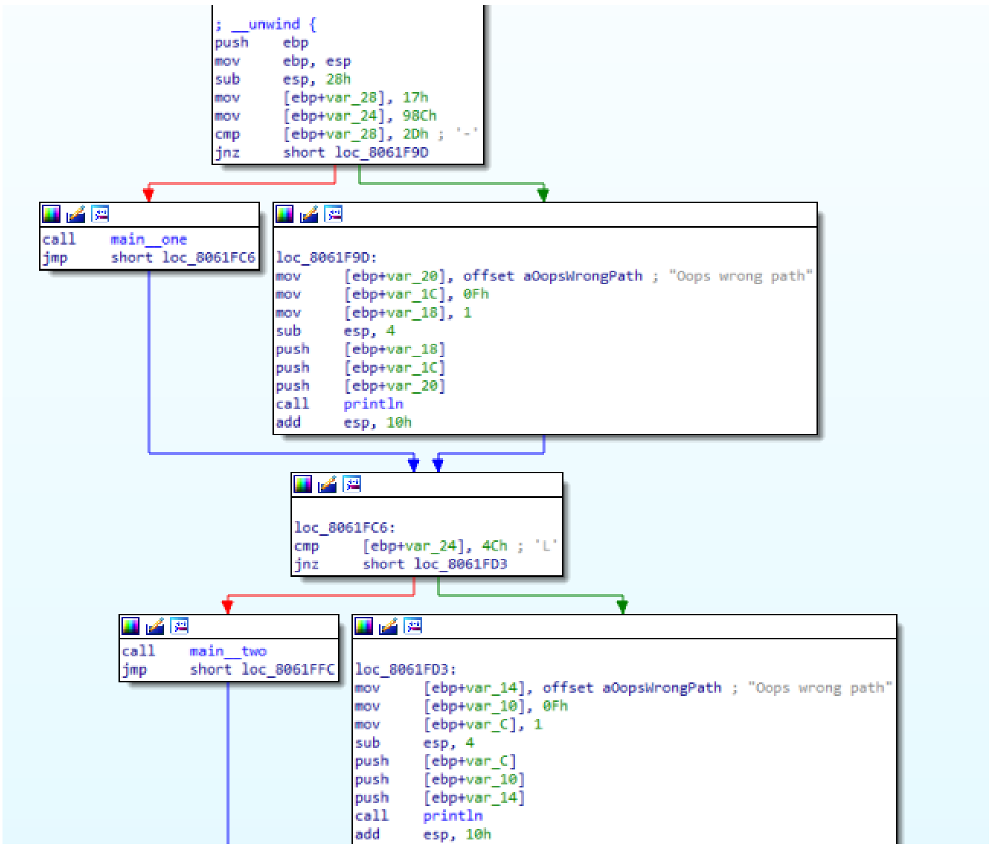
Main()

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
 vinit(argc, argv);
 main__main();
 return 0;
}
```

Main_main()

```
int main__main()
{
 println("Oops wrong path", 15);
 return println("Oops wrong path", 15);
}
```

Damn nothing. But for the main_main(), the graphical view said otherwise



```
; __unwind {
push    ebp
mov     ebp, esp
sub     esp, 28h
mov     [ebp+var_28], 17h
mov     [ebp+var_24], 98Ch
cmp     [ebp+var_28], 2Dh ; '-'
jnz     short loc_8061F9D
```

```
call    main__one
jmp     short loc_8061FC6
```

```
loc_8061F9D:
mov     [ebp+var_20], offset aOopsWrongPath ; "Oops wrong path"
mov     [ebp+var_1C], 0Fh
mov     [ebp+var_18], 1
sub     esp, 4
push    [ebp+var_18]
push    [ebp+var_1C]
push    [ebp+var_20]
call    println
add     esp, 10h
```

```
loc_8061FC6:
cmp     [ebp+var_24], 4Ch ; 'L'
jnz     short loc_8061FD3
```

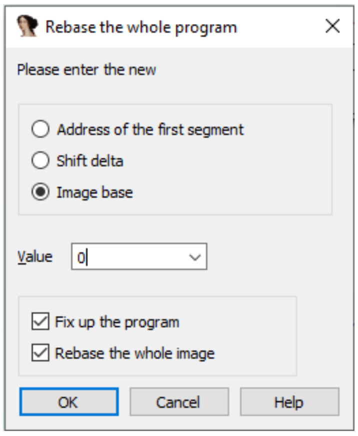
```
call    main__two
jmp     short loc_8061FFC
```

```
loc_8061FD3:
mov     [ebp+var_14], offset aOopsWrongPath ; "Oops wrong path"
mov     [ebp+var_10], 0Fh
mov     [ebp+var_C], 1
sub     esp, 4
push    [ebp+var_C]
push    [ebp+var_10]
push    [ebp+var_14]
call    println
add     esp, 10h
```

The code moves 17h into ebp+var_28 and then compares it to 2Dh. If these values are not equal, which they never are in this case, it never calls main_one. After the main_one call there is another compare, this time with ebp+var_24 to 4Ch. In order to make the code follow the right path I needed to patch the program so the cmp's had the same value.

To do this I first rebased the program to 0.

edit->segments->rebase program



Rebase the whole program

Please enter the new

- ( ) Address of the first segment
- ( ) Shift delta
- (●) Image base

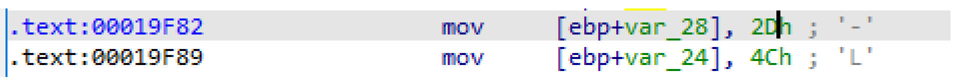
Value: 0

- [✓] Fix up the program
- [✓] Rebase the whole image

Now I have the hex offset to the opcodes I want to change:

```
.text:00019F82                  mov     [ebp+var_28], 17h
.text:00019F89                  mov     [ebp+var_24], 98Ch
```

Then patch it



```
.text:00019F82                 mov     [ebp+var_28], 2Dh ; '-'
.text:00019F89                 mov     [ebp+var_24], 4Ch ; 'L'
```

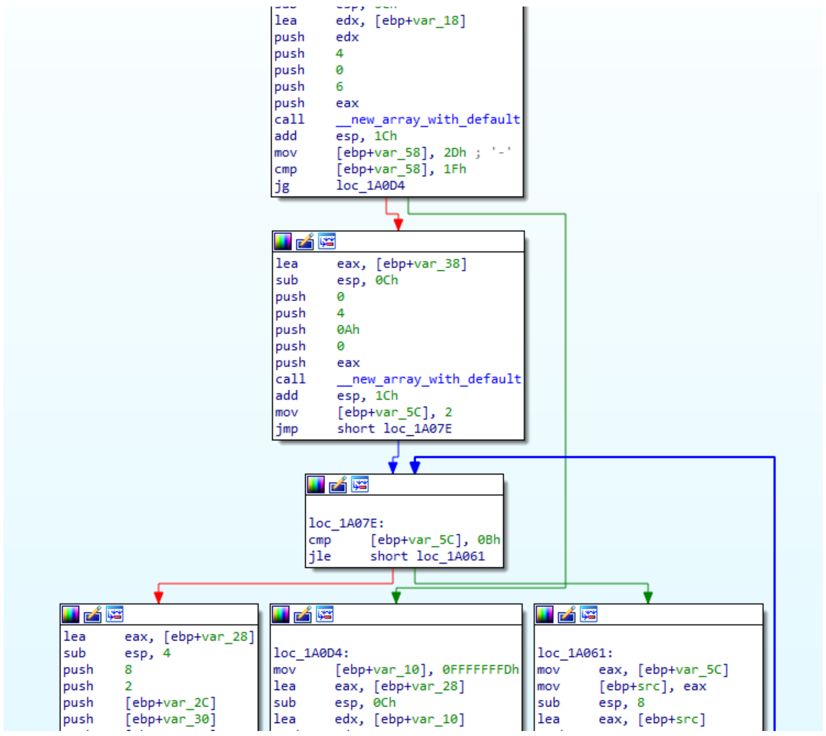
Edit > Patch program > Apply patches to input file

Then try running it again

```
└$ ./keyjoinfile
You don't have the first part of key yet
```

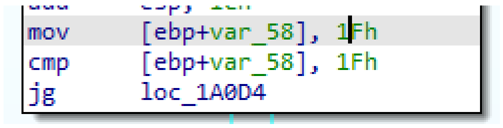
So need to check again

in main_one() also got cmp that will never reach



```
lea     edx, [ebp+var_18]
push    edx
push    4
push    0
push    6
push    eax
call    __new_array_with_default
add     esp, 1Ch
mov     [ebp+var_58], 2Dh ; '-'
cmp     [ebp+var_58], 1Fh
jg      loc_1A0D4

lea     eax, [ebp+var_38]
sub     esp, 0Ch
push    0
push    4
push    0Ah
push    0
push    eax
call    __new_array_with_default
add     esp, 1Ch
mov     [ebp+var_5C], 2
jmp     short loc_1A07E

loc_1A07E:
cmp     [ebp+var_5C], 0Bh
jle     short loc_1A061

lea     eax, [ebp+var_28]
sub     esp, 4
push    8
push    2
push    [ebp+var_2C]
push    [ebp+var_30]

loc_1A0D4:
mov     [ebp+var_10], 0FFFFFFFDh
lea     eax, [ebp+var_28]
sub     esp, 0Ch
lea     edx, [ebp+var_10]

loc_1A061:
mov     eax, [ebp+var_5C]
mov     [ebp+src], eax
sub     esp, 8
lea     eax, [ebp+src]
```

So need to patch it too

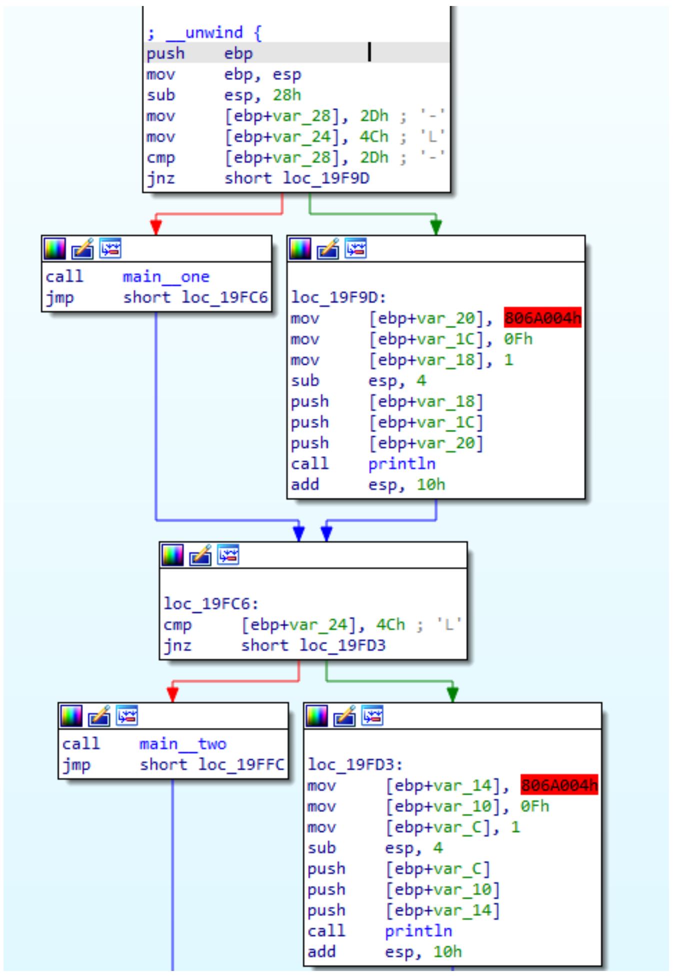


```
mov     [ebp+var_58], 1Fh
cmp     [ebp+var_58], 1Fh
jg      loc_1A0D4
```

Then try running it

```
└$ ./keyjoinfile
[4, 5, 6, 7, 8, 9]
```

The file seemed to output the first part of the key and then exit. So we are done in main_one()



```
; __unwind {
push    ebp
mov     ebp, esp
sub     esp, 28h
mov     [ebp+var_28], 2Dh ; '-'
mov     [ebp+var_24], 4Ch ; 'L'
cmp     [ebp+var_28], 2Dh ; '-'
jnz     short loc_19F9D
```

```
call    main__one
jmp     short loc_19FC6
```

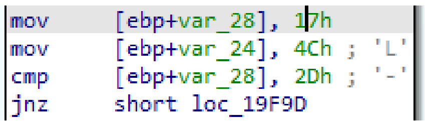
```
loc_19F9D:
mov     [ebp+var_20], 806A004h
mov     [ebp+var_1C], 0Fh
mov     [ebp+var_18], 1
sub     esp, 4
push    [ebp+var_18]
push    [ebp+var_1C]
push    [ebp+var_20]
call    println
add     esp, 10h
```

```
loc_19FC6:
cmp     [ebp+var_24], 4Ch ; 'L'
jnz     short loc_19FD3
```

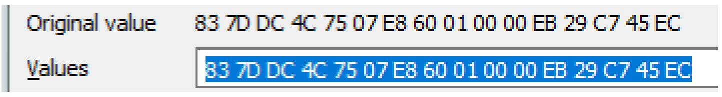
```
call    main__two
jmp     short loc_19FFC
```

```
loc_19FD3:
mov     [ebp+var_14], 806A004h
mov     [ebp+var_10], 0Fh
mov     [ebp+var_C], 1
sub     esp, 4
push    [ebp+var_C]
push    [ebp+var_10]
push    [ebp+var_14]
call    println
add     esp, 10h
```

Looking at the graphical view, we want the program the go to main_two(). SO instead of calling to main_one(), we want main_one to be the "oops wrong path" assuming there are two oops wrong path, one for main_one and one for main_two. So change the first cmp value to original.



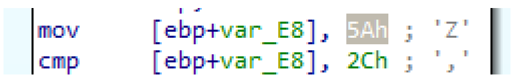
```
mov     [ebp+var_28], 17h
mov     [ebp+var_24], 4Ch ; 'L'
cmp     [ebp+var_28], 2Dh ; '-'
jnz     short loc_19F9D
```



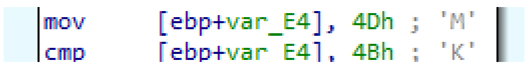
| | |
|---|---|
| Original value | 83 7D DC 4C 75 07 E8 60 01 00 00 EB 29 C7 45 EC |
| Values | 83 7D DC 4C 75 07 E8 60 01 00 00 EB 29 C7 45 EC |

```
–$ ./keyjoinfile
[4, 5, 6, 7, 8, 9]
Oops wrong path
```

So now patch main_two too



```
mov     [ebp+var_E8], 5Ah ; 'Z'
cmp     [ebp+var_E8], 2Ch ; ','
```



```
mov     [ebp+var_E4], 4Dh ; 'M'
cmp     [ebp+var_E4], 4Bh ; 'K'
```

When running

```
$ ./keyjoinfile
```

```
[4, 5, 6, 7, 8, 9]
['J', 'K', 'L', 'q', '5', '9', 'U', '1', '3', '3', '7']
```

flag{456789+JKLq59U1337}

## X and Or

```
–$ file x-and-or
x-and-or: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-
x86-64.so.2, BuildID[sha1]=d75c2db8d7b1c77fd65762741f73b19aba4f2815, for GNU/Linux 3.2.0, not stripped
```

```
$ ./x-and-or
Enter the flag: test
That is not the flag.
```

### Main()

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  unsigned int v4; // [rsp+Ch] [rbp-114h]
  char s[264]; // [rsp+10h] [rbp-110h] BYREF
  unsigned __int64 v6; // [rsp+118h] [rbp-8h]

  v6 = __readfsqword(0x28u);
  printf("Enter the flag: ");
  fgets(s, 256, _bss_start);
  s[strcspn(s, "\r\n")] = 0;
  v4 = strnlen(s, 0x100uLL);
  if ( (unsigned int)code(s, v4) )
    puts("That is not the flag.");
  else
    puts("That is the flag!!!!");
  return 0;
}
```

## No Debug

```
$ file crackme
crackme: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-
x86-64.so.2, BuildID[sha1]=f4599ab6673e44ee040c43849f7af66cd81a3d45, for GNU/Linux 3.2.0, stripped
```

### Main()

```
__int64 __fastcall main(int a1, char **a2, char **a3)
{
  int v3; // eax
  char buf[264]; // [rsp+0h] [rbp-110h] BYREF
  ssize_t v6; // [rsp+108h] [rbp-8h]

  printf("Enter key: ");
  v6 = read(0, buf, 0xFFuLL);
  if ( v6 )
    buf[v6 - 1] = 0;
  if ( (unsigned int)sub_1738(buf, v6 - 1) )
  {
    printf("Congrats here is your flag: ");
    v3 = open("/flag", 0);
    sendfile(1, v3, 0LL, 0x100uLL);
  }
  else
  {
    puts("Invalid key");
  }
  return 0LL;
}
```

But when running the file in debugger it does not run the main()

```
$ ./crackme
Enter key: donno
Invalid key
```

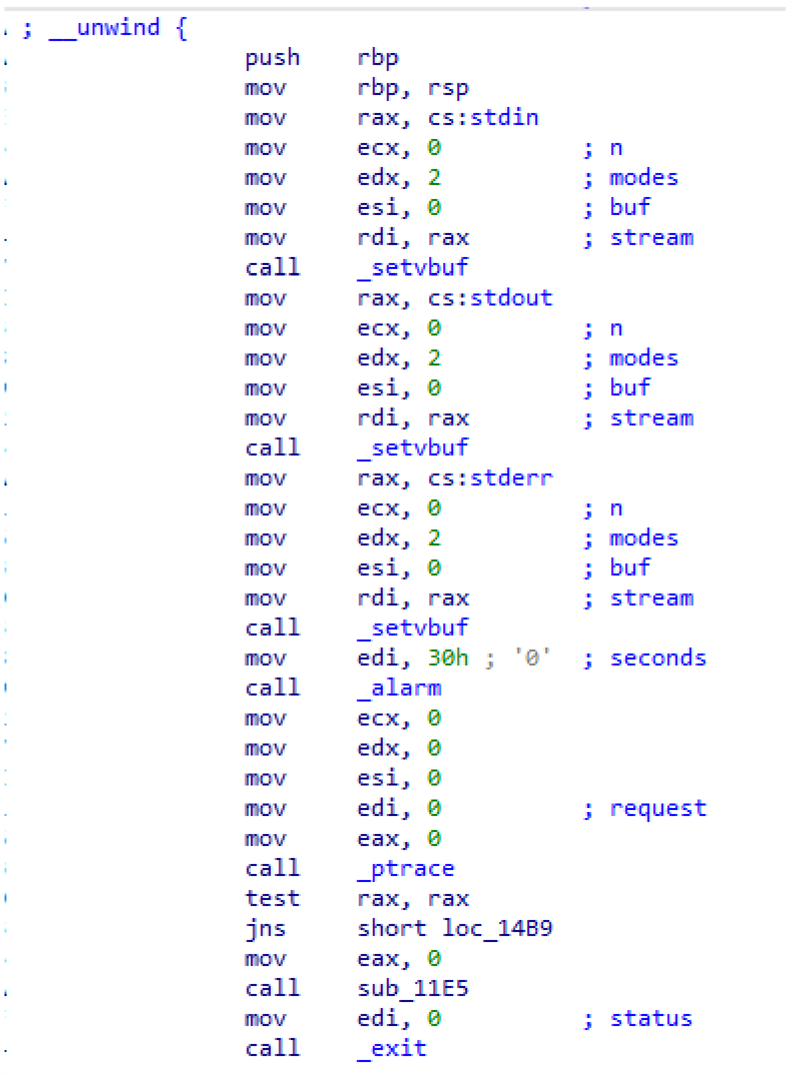
```
run
Starting program: /mnt/c/Users/hzqzz/Downloads/crackme
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Enter key: donno
Wrong password
```

I got "Wrong password" instead of "Invalid key". So need to find another function:

```
int sub_11E5()
{
  int v0; // eax
  char buf[32]; // [rsp+0h] [rbp-130h] BYREF
  __int64 s1[33]; // [rsp+20h] [rbp-110h] BYREF
  int v4; // [rsp+128h] [rbp-8h]
  int fd; // [rsp+12Ch] [rbp-4h]

  memset(s1, 0, 256);
  fd = open("/dev/urandom", 0);
  read(fd, buf, 0x20uLL);
  close(fd);
  v4 = 0;
  printf("Enter key: ");
  v4 = read(0, s1, 0xFFuLL);
  if ( v4 > 0 )
    *((_BYTE *)s1 + v4 - 1) = 0;
  if ( memcmp(s1, buf, 0x1FuLL) )
    return puts("Wrong password");
  puts("Congrats you are a super eleet hacker, here is your flag: ");
  v0 = open((const char *)s1, 0);
  return sendfile(1, v0, 0LL, 0x100uLL);
}
```

This function just iterates from 0-31 assigning 0LL to everything. Then found a function that called sub_11E5().

```
__int64 sub_141A()
{
  __int64 result; // rax

  setvbuf(stdin, 0LL, 2, 0LL);
  setvbuf(stdout, 0LL, 2, 0LL);
  setvbuf(stderr, 0LL, 2, 0LL);
  alarm(0x30u);
  result = ptrace(PTRACE_TRACEME, 0LL, 0LL, 0LL);
  if ( result < 0 )
  {
    sub_11E5();
    exit(0);
  }
  return result;
}
```



```
; __unwind {
                push    rbp
                mov     rbp, rsp
                mov     rax, cs:stdin
                mov     ecx, 0          ; n
                mov     edx, 2          ; modes
                mov     esi, 0          ; buf
                mov     rdi, rax        ; stream
                call    _setvbuf
                mov     rax, cs:stdout
                mov     ecx, 0          ; n
                mov     edx, 2          ; modes
                mov     esi, 0          ; buf
                mov     rdi, rax        ; stream
                call    _setvbuf
                mov     rax, cs:stderr
                mov     ecx, 0          ; n
                mov     edx, 2          ; modes
                mov     esi, 0          ; buf
                mov     rdi, rax        ; stream
                call    _setvbuf
                mov     edi, 30h ; '0'  ; seconds
                call    _alarm
                mov     ecx, 0
                mov     edx, 0
                mov     esi, 0
                mov     edi, 0          ; request
                mov     eax, 0
                call    _ptrace
                test    rax, rax
                jns     short loc_14B9
                mov     eax, 0
                call    sub_11E5
                mov     edi, 0          ; status
                call    _exit
```

Recur

Shop

Infinity Gauntlet

Jailbreak

Alienware

Exhell

FeistyCrypt

Dimensionality

Roolang