CYBER SENTRIX FINAL

Presented By:

LastSeenIn2026

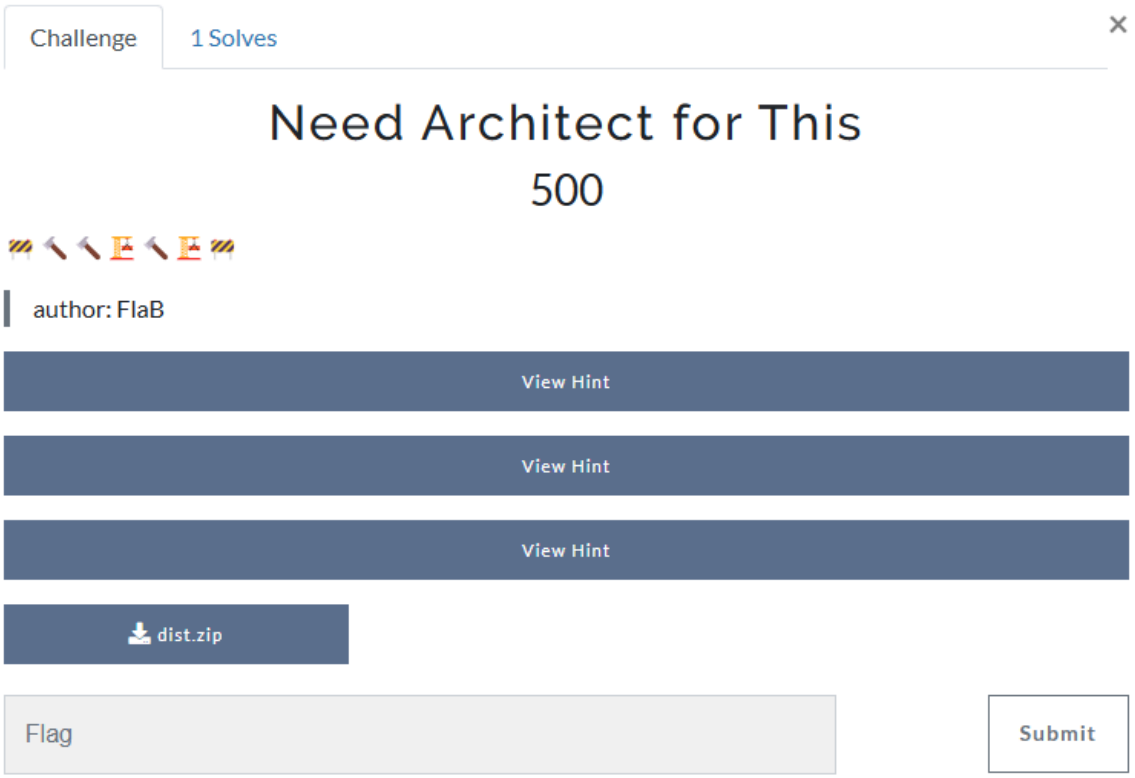
Sugeng Dwi Hermanto (SMKN 1 Cibinong)
Deffreus Theda (SMA Pradita Dirgantara)
Riduan (SMKN 2 Pangkalpinang)

[DAFTAR ISI]

[REVERSE ENGINEERING]

By Effie…

1. Need Architect for This



Deskripsi

🥲🥲🥲

Author: FlaB flap flap

Hints

1. To solve this, you must understand how to construct a strong foundation. Mastering how to connect and traverse through structures, just like linking nodes in a chain, will lead you to the solution.
2. You can swap two values without needing extra space. Think of an operation where combining the same value twice cancels it out. ^v^
3. Patching binary is the feature tha i like on ida, it can patch some operation that soo confusing, like xor, so you can see the output before xor happend.

Files

```
$ file chall output
chall:  ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV),
dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2,
BuildID[sha1]=3f1e5439c52e5e72f6c4de9f9269ac56216e289a, for GNU/Linux
3.2.0, stripped
output: data
$ du -h chall output
 16K  chall
4.0K  output
```

tl;dr

Given the output, I reverse a *recorded* singly linked list 'bubble' sort with the biggest byte as the seed for random xor encryption on the sorted flag. Actually prone to error and I swear to god– it shouldn't take 6 hours to solve…

Intro

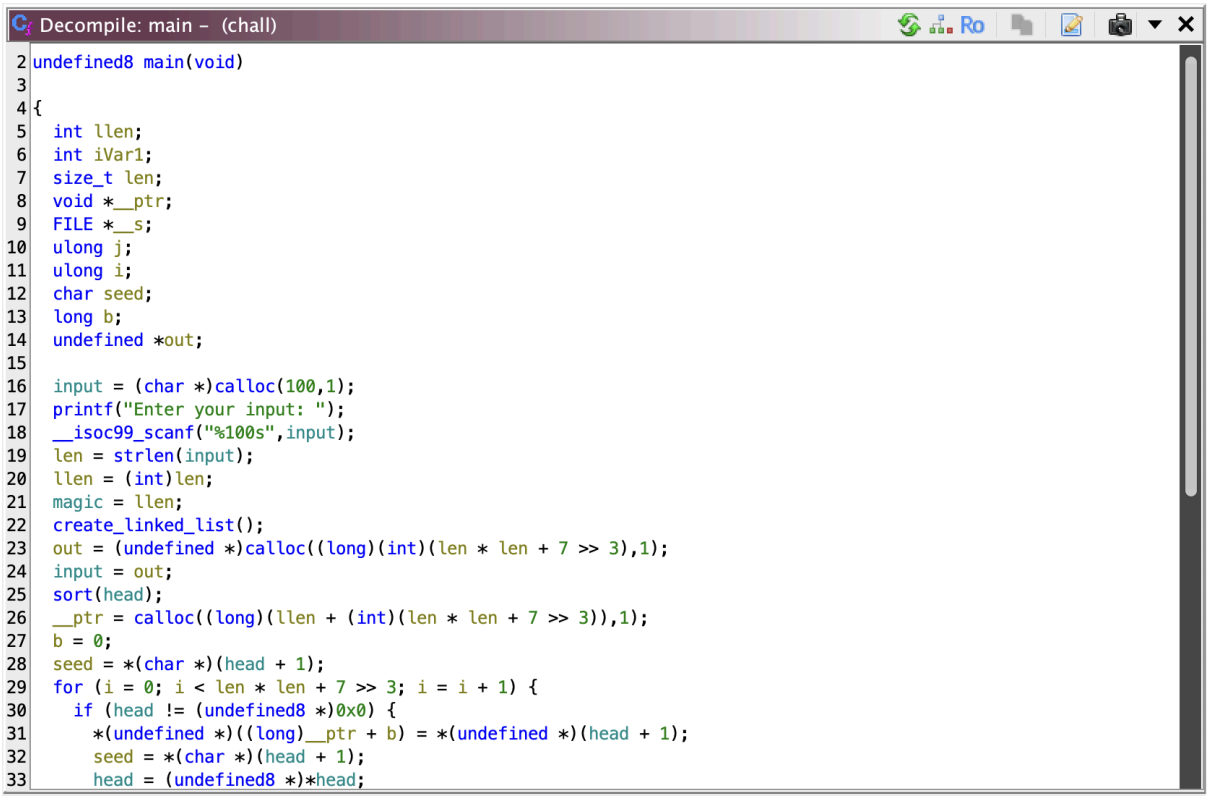
Read this blog about singly linked lists if you don't know about that already!! If this blog uses a class for a node, I just use a C struct for simplicity sake. The usage of the singly linked list in this challenge is not complicated and is not that different from just your normal data array. For bubble sort, you may read this if you wanna know more! Just treat the array element there as a node!

About styling and what they mean: *This is a code*, *also code but renamed*, and **important**; In Ghidra screenshots: global variable/data and local variable, so make sure that you know the difference!

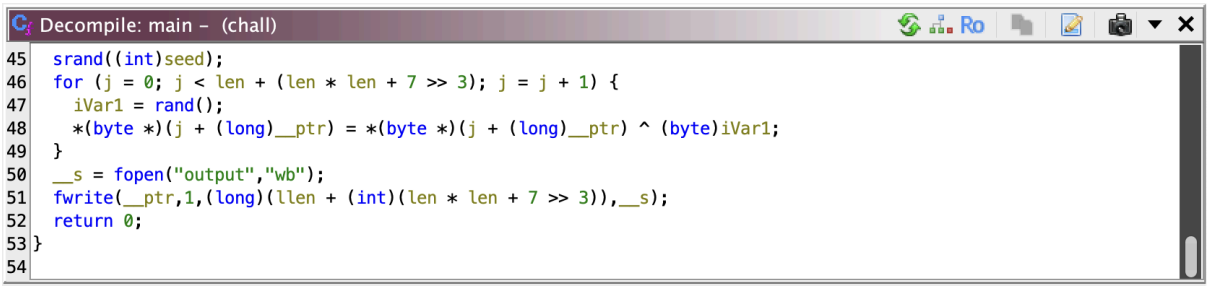
Solution

Thank god. Now let's jump right in! As the certified ARM enjoyer, IDA is unplayable so I'll do this with Ghidraawr <3. Just auto analyze, all checked,

and you'll be on *entry* in no time! The first argument to *__libc_start_main* is the main function, so jump there!

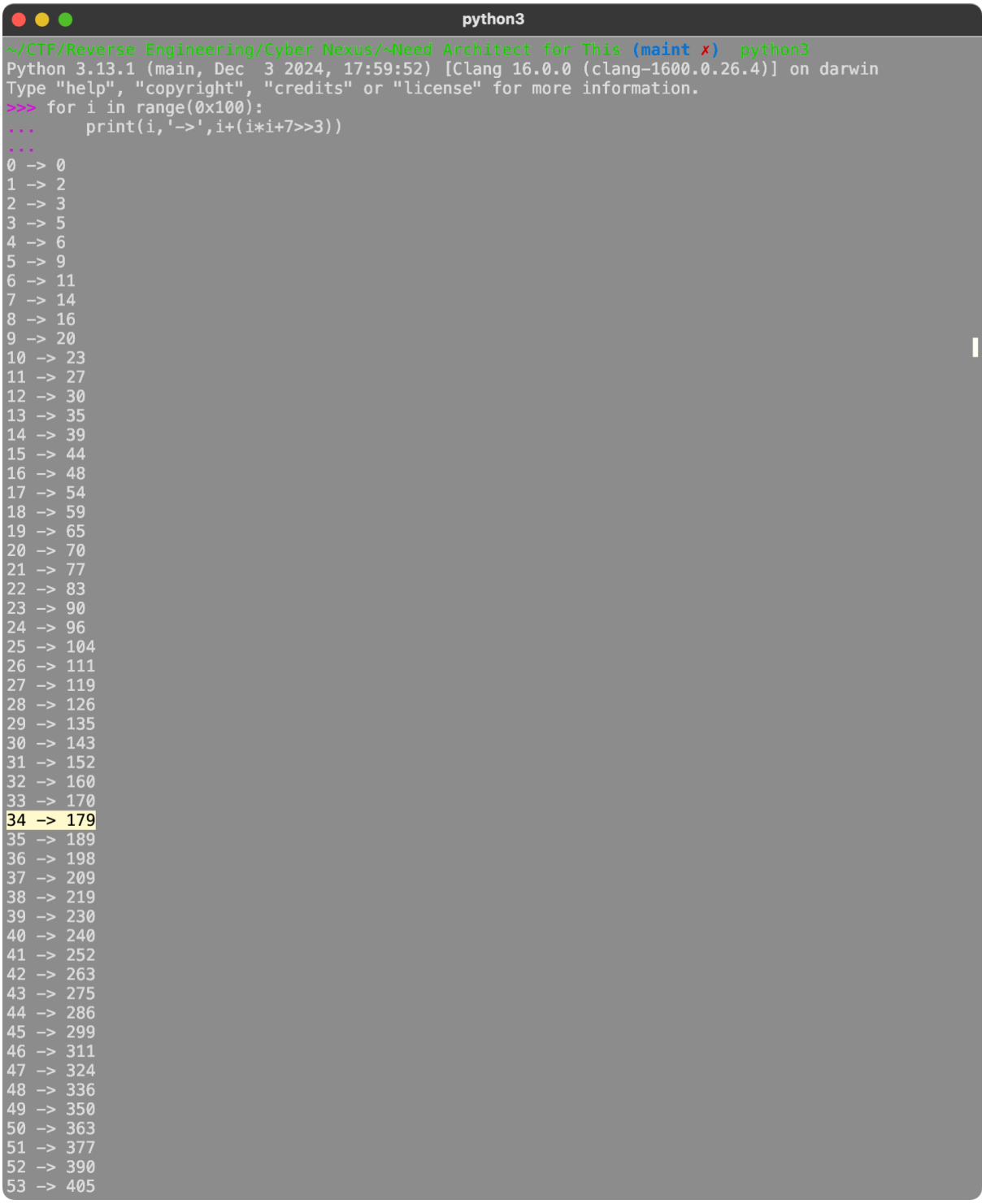


The main function is where the main program flow takes place. If you notice, the argument to *calloc* is this weird expression of *len * len + 7 >> 3*, and tbh I don't really understand exactly why either :p, but I'm sure it has something to do with the bubble sort! So basically, the program takes an input, sets input length to the global data *magic*, and calls *create_linked_list* and *sort*. **Note that *input* is pointing to a newly allocated *out*.**



Before going into details about these functions, we'll go about figuring out some meta information about the flag. The output is random xored, and because of the (thankfully) deterministic behavior of *rand()*, we can reverse this given we know the seed for *srand()*.

If you open it in a hex editor, the file `output` is 179 bytes in size. As we can see in the for loop condition, *j* is iterated from *0* to *len + (len * len + 7 >> 3)*, and we can find the input length by trying out which *len* cause that expression to be 179:



So, now we know that **34 is the correct input length**! Things should be easier with that. Consequently, the **expression** *len * len + 7 >> 3* **result in 145**!

Decompile: main - (chall)

```
26   __ptr = calloc((long)(llen + (int)(len * len + 7 >> 3)),1);
27   b = 0;
28   seed = *(char *)(head + 1);
29   for (i = 0; i < len * len + 7 >> 3; i = i + 1) {
30     if (head != (undefined8 *)0x0) {
31       *(undefined *)((long)__ptr + b) = *(undefined *)(head + 1);
32       seed = *(char *)(head + 1);
33       head = (undefined8 *)*head;
34       b = b + 1;
35     }
36     *(undefined *)((long)__ptr + b) = *out;
37     out = out + 1;
38     b = b + 1;
39   }
40   for (; head != (undefined8 *)0x0; head = (undefined8 *)*head) {
41     *(undefined *)((long)__ptr + b) = *(undefined *)(head + 1);
42     seed = *(char *)(head + 1);
43     b = b + 1;
44   }
```

Just before writing the output, *__ptr* is populated with the sorted input and bytes of *out* (*input*) alternatively. This can be visualized like:

I0 M0 I1 M1 ... I32 M32 I33 M33 M34 M35 ... M144

This should be noted if we were to reverse this, especially with the random xor part. **I0-I33 is the sorted flag**, whereas **M0-M144 is the 'recording' of the bubble sort performed**. What's important is the seed of *srand*, and because it's the biggest (last) character of the sorted flag, and that should be '}' which has a value of 125. So now we know the result of this program, next, let's have a look at *create_linked_list*!

Decompile: create_linked_list - (chall)

```
1
2  void create_linked_list(void)
3
4  {
5    long *node;
6    undefined8 *next;
7    ulong i;
8
9    node = (long *)malloc(0x10);
10   head = node;
11   *(undefined *)(node + 1) = 0;
12   *node = 0;
13   for (i = 0; i < (ulong)(long)magic; i = i + 1) {
14     next = (undefined8 *)malloc(0x10);
15     *(undefined *)(next + 1) = *(undefined *)(i + input);
16     *next = 0;
17     for (; *head != 0; head = (long *)*head) {
18     }
19     *head = (long)next;
20   }
21   head = (long *)*node;
22   return;
23 }
24
```

As you may already guessed, it creates a **singly linked list** of the input bytes from start to end. *head* is set to the head of the linked list. Each node is 16 (0x10) bytes long, consisting of a pointer to the next node and then the data it holds, each 8 bytes long.

The main function then allocates a lot of bytes to *out* which is then pointed by *input*. Take note of that! Here, we'll go into *sort*, a recursive outer loop implementation of the bubble sort algorithm.

Decompile: sort - (chall)

```
1
2 void sort(undefined8 head)
3
4 {
5   magic = magic + -1;
6   if (0 < magic) {
7     sort(head);
8     xor();
9   }
10  ::head = head;
11  return;
12 }
13
```

As I said before, *magic* is initially set to input length, and what this does is stack a bunch of *sort* and *xor* calls for *magic* (34) times. What's important is how it calls *xor*, and given this code, *xor* would be first called when *magic* is 0. Note that, after *xor* is called, *head* is resetted, meaning subsequent *xor* also starts on the head of the linked list.

Decompile: xor - (chall)

```
1
2 void xor(void)
3
4 {
5   if (*head != 0) {
6     if (*(char *)(*head + 8) < *(char *)(head + 1)) {
7       *(byte *)(*head + 8) = *(byte *)(head + 1) ^ *(byte *)(*head + 8);
8       *(byte *)(head + 1) = *(byte *)(*head + 8) ^ *(byte *)(head + 1);
9       *(byte *)(*head + 8) = *(byte *)(head + 1) ^ *(byte *)(*head + 8);
10      input();
11    }
12    head = (long *)*head;
13    magic = magic + 1;
14    if (7 < magic) {
15      input = input + 1;
16      magic = 0;
17    }
18    xor();
19  }
20  return;
21 }
22
```

xor is the part where the comparison and swapping happens. What I love about this is how it swaps the value of the nodes, by using xor! ISN'T

```
>>> def x(a, b):
...     b ^= a
...     a ^= b
...     b ^= a
...     return (a, b)
...
>>> x(10,10)
(10, 10)
>>> x(10,13)
(13, 10)
```

THAT COOL?? Like, I've never seen anyone do that before, so it's really mind blowing when I realized what it actually does.. Anyway, I know that decompiled code kinda seems hard to understand so I made a much more beautiful version of it:

```
1   void xor() {
2     if (head->next != NULL) {
3       if (head->next->data < head->data) {
4         head->next->data = head->data ^ head->next->data;
5         head->data = head->next->data ^ head->data;
6         head->next->data = head->data ^ head->next->data;
7         record();
8       }
9       head = head->next;
10      magic = magic + 1;
11      if (7 < magic) {
12        ++input;
13        magic = 0;
14      }
15      xor();
16    }
17  }
```

It compares the current node with the next node, swapping and recording if it's smaller. *magic* is used here for recording if swaps (denoted by 1) happen or not, into a memory in the region pointed by *input*. *xor* is called recursively, meaning that this will continue until the next pointer is null.

Decompile: record - (chall)

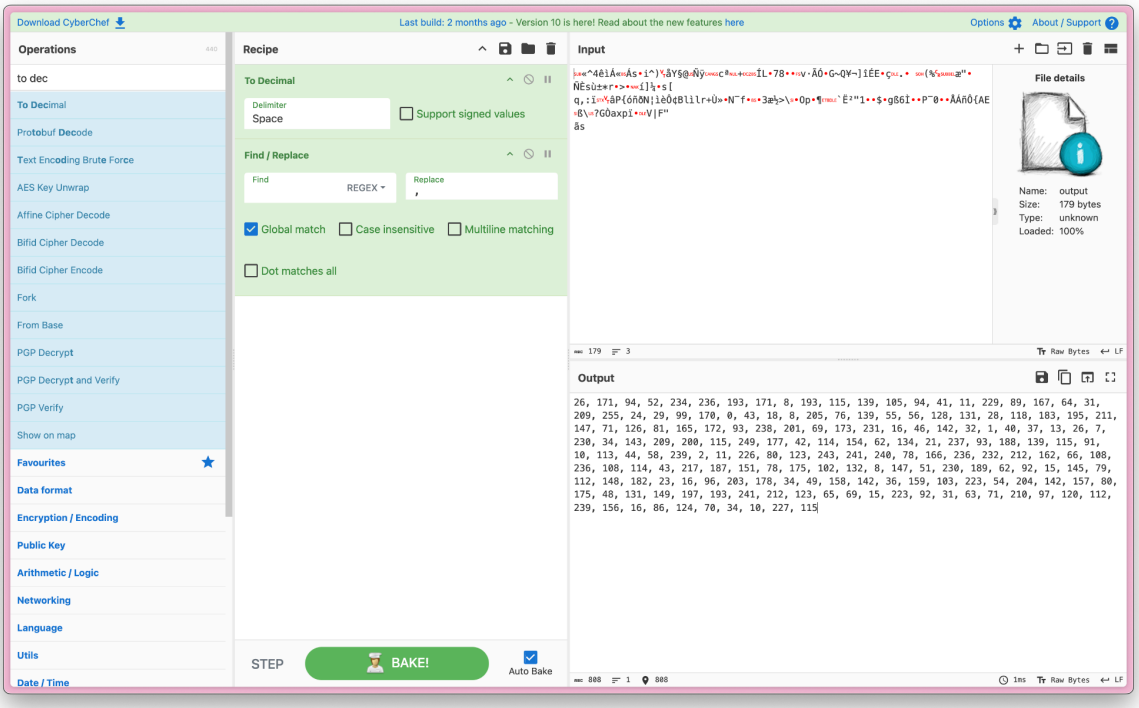
```
1
2 void record(void)
3
4 {
5   *input = *input | (byte)(1 << ((byte)magic & 0x1f));
6   return;
7 }
8
```

record sets the bit of the byte pointed by *input* depending on the value of *magic* to 1. What this means is that bytes in the input memory region hold the 'replay' of the bubble sort performed on the actual flag. To play forward, It iterates from bytes to bytes, but for each bit it goes from the least (right-most) significant bit to the most (left-most) significant bit.

But, we're here to reverse it so just flip all that back! This Python expression would produce a stream of 1 and 0 that can be used for reversing the bubble sort: *s = ''.join(bin(m)[2:].rjust(8, '0') for m in magic[::-1])[**X**:]*, where ***X*** is the length difference that can be calculated $145 \times 8 - 34^2 = 4$

Solver

First, we reverse the random xor. I turn the output file into a C array with cyberchef and a C script:




```c
#include <stdio.h>
#include <stdlib.h>

int main() {
  srand(125);

  char output[] = {(char)26, (char)171, (char)94, (char)52, (char)234, (char)236, (char)193, (char)171, (char)8, (char)193, (char)115, (char)139, (char)105, (char)94, (char)41, (char)11, (char)229, (char)89, (char)167, (char)64, (char)31, (char)209, (char)255, (char)24, (char)29, (char)99, (char)170, (char)0, (char)43, (char)18, (char)8, (char)205, (char)76, (char)139, (char)55, (char)56, (char)128, (char)131, (char)28, (char)118, (char)183, (char)195, (char)211, (char)147, (char)71, (char)126, (char)81, (char)165, (char)172, (char)93, (char)238, (char)201, (char)69, (char)173, (char)231, (char)16, (char)46, (char)142, (char)32, (char)1, (char)40, (char)37, (char)13, (char)26, (char)7, (char)230, (char)34, (char)143, (char)209, (char)200, (char)115, (char)249, (char)177, (char)42, (char)114, (char)154, (char)62, (char)134, (char)21, (char)237, (char)93, (char)188, (char)139, (char)115, (char)91, (char)10, (char)113, (char)44, (char)58, (char)239, (char)2, (char)11, (char)226, (char)80, (char)123, (char)243, (char)241, (char)240, (char)78, (char)166, (char)236, (char)232, (char)212, (char)162, (char)66, (char)108, (char)236, (char)108, (char)114, (char)43, (char)217, (char)187, (char)151, (char)78, (char)175, (char)102, (char)132, (char)8, (char)147, (char)51, (char)230, (char)189, (char)62, (char)92, (char)15, (char)145, (char)79, (char)112, (char)148, (char)182, (char)23, (char)16, (char)96, (char)203, (char)178, (char)34, (char)49, (char)158, (char)142, (char)36, (char)159, (char)103, (char)223, (char)54, (char)204, (char)142, (char)157, (char)80, (char)175, (char)48, (char)131, (char)149, (char)197, (char)193, (char)241, (char)212, (char)123, (char)65, (char)69, (char)15, (char)223, (char)92, (char)31, (char)63, (char)71, (char)210, (char)97, (char)120, (char)112, (char)239, (char)156, (char)16, (char)86, (char)124, (char)70, (char)34, (char)10, (char)227, (char)115};

  for (int i = 0; i < 179; ++i) {
    printf("%d, ", output[i] ^ (char)rand());
```

```
  11    |   }
  12    | }
```

```
┌──(effie㉿theda)-[/media/…/CTF/Reverse Engineering/Cyber Nexus/~Need Architect for Th
└─$ gcc unxor.c; ./a.out
48, 237, 48, 255, 48, 255, 48, 255, 49, 72, 51, 255, 52, 255, 52, 255, 67, 64, 68, 246
 110, 111, 64, 111, 212, 111, 148, 114, 110, 115, 0, 116, 212, 116, 148, 117, 106, 123
2, 0, 0, 144, 42, 0, 0, 144, 42, 0, 0, 128, 42, 0, 0, 128, 42, 0, 0, 0, 42, 0, 0, 0, 4
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

Then, we filter the data and the bubble sort recording out:

```
   1    | output = [48, 237, 48, 255, 48, 255, 48, 255, 49, 72, 51, 255, 52, 255, 52, 255, 67, 64,
68, 246, 69, 220, 71, 111, 78, 64, 83, 214, 83, 220, 85, 110, 88, 64, 89, 214, 95, 220, 95, 110, 95
       | , 64, 95, 214, 99, 212, 104, 110, 111, 64, 111, 212, 111, 148, 114, 110, 115, 0, 116, 212,
116, 148, 117, 106, 123, 0, 125, 212, 148, 106, 0, 212, 148, 42, 0, 208, 148, 42, 0, 128, 148, 42
       | , 0, 128, 148, 42, 0, 0, 148, 42, 0, 0, 148, 42, 0, 0, 144, 42, 0, 0, 144, 42, 0, 0, 128,
42, 0, 0, 128, 42, 0, 0, 0, 42, 0, 0, 0, 40, 0, 0, 0, 40, 0, 0, 0, 40, 0, 0, 0, 40, 0, 0, 0, 32, 0
       | , 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
   2    | flag_length = 34
   3    |
   4    | data = [output[i] for i in range(0, flag_length * 2, 2)]
   5    | magic = [output[i] for i in range(1, flag_length * 2, 2)] + [output[i] for i in
range(flag_length * 2, len(output))]
   6    |
   7    | print(f'{data = }')
   8    | print(f'{len(data) = }')
   9    | print(f'{magic = }')
  10    | print(f'{len(magic) = }')
```

Now, since we got the data and the recording, we can re

```
~/CTF/Reverse Engineering/Cyber Nexus/~Need Architect for This (maint x)  python3 filter.py
data = [48, 48, 48, 48, 49, 51, 52, 52, 67, 68, 69, 71, 78, 83, 83, 85, 88, 89, 95, 95, 95, 95, 99, 104,
111, 111, 111, 114, 115, 116, 116, 117, 123, 125]
len(data) = 34
magic = [237, 255, 255, 255, 72, 255, 255, 255, 64, 246, 220, 111, 64, 214, 220, 110, 64, 214, 220, 110,
64, 214, 212, 110, 64, 212, 148, 110, 0, 212, 148, 106, 0, 212, 148, 106, 0, 212, 148, 42, 0, 208, 148, 4
2, 0, 128, 148, 42, 0, 128, 148, 42, 0, 0, 148, 42, 0, 0, 148, 42, 0, 0, 144, 42, 0, 0, 144, 42, 0, 0, 12
8, 42, 0, 0, 128, 42, 0, 0, 0, 42, 0, 0, 0, 40, 0, 0, 0, 40, 0, 0, 0, 40, 0, 0, 0, 40, 0, 0, 0, 32, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0]
len(magic) = 145
```

verse the bubble sort with this solver:

```
   1    | for X in range(256):
   2    |     data = [48, 48, 48, 48, 49, 51, 52, 52, 67, 68, 69, 71, 78, 83, 83, 85, 88, 89, 95, 95,
95, 95, 99, 10
       | 4, 111, 111, 111, 114, 115, 116, 116, 117, 123, 125]
   3    |     magic = [237, 255, 255, 255, 72, 255, 255, 255, 64, 246, 220, 111, 64, 214, 220, 110,
64, 214, 220, 11
       | 0, 64, 214, 212, 110, 64, 212, 148, 110, 0, 212, 148, 106, 0, 212, 148, 106, 0, 212, 148,
42, 0, 208, 148,
       |  42, 0, 128, 148, 42, 0, 128, 148, 42, 0, 0, 148, 42, 0, 0, 148, 42, 0, 0, 144, 42, 0, 0,
144, 42, 0, 0, 1
       | 28, 42, 0, 0, 128, 42, 0, 0, 0, 42, 0, 0, 0, 40, 0, 0, 0, 40, 0, 0, 0, 40, 0, 0, 0, 40, 0,
0, 0, 32, 0, 0,
       |  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0,
       | 0, 0, 0, 0, 0, 0, 0, 0]
```

```python
    s = ''.join(bin(m)[2:].rjust(8, '0') for m in magic[::-1])[X:]
    print(f'{len(s) = }')
    p = 32
    for ss in s:
        if p == -1:
            p = 32
        if ss == "1":
            # print(p, p + 1)
            tmp = data[p]
            data[p] = data[p + 1]
            data[p + 1] = tmp
        p -= 1
    print(''.join(chr(c) for c in data))
```

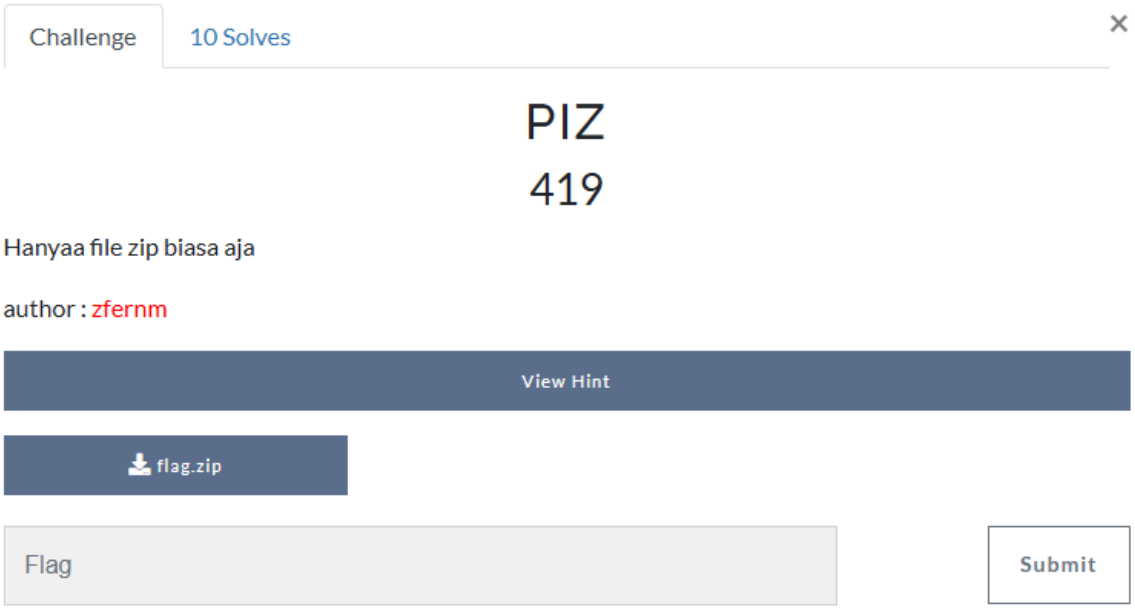
```
.../Reverse Engineering/Cyber Nexus/~Need Architect for This
~/CTF/Reverse Engineering/Cyber Nexus/~Need Architect for This (maint x)  python3 solve.py
len(s) = 1160
rNGoSED_CS44__3U1hX0c0_0o0You}tt{s
len(s) = 1159
NsSGrSED_CU44_c3X0oY0h1_0o0_o{}ttu
len(s) = 1158
SNtSEsUGD_4XC0ch4Y0o_1o3_0o0_r{}ut
len(s) = 1157
SSGtUEtXNCc0YD0ho4_1o_3o4_0r0_s{}u
len(s) = 1156
GUSNuXDtYS0h0_E0ooC_3r_4o4c0s1_t{}
len(s) = 1155
NEXUS{Y0u_S0o0_G0ooD_4s_4rCh1t3ct}
len(s) = 1154
0SGYXS}_0{_U0o0_N1orE_4tcCsDo3t4hu
len(s) = 1153
00SN_YU{_0}_X0o1_S3rsGcCthDtEo4u4o
len(s) = 1152
000US__Xo_0}_Y1r3cS4stNhDuoEtGo4{C
len(s) = 1151
0000XS__YD_1oc_3s4hU4ttSoE{oGuNrC}
len(s) = 1150
10000YU___rc3Eh_4t4oXCtuSoG}oN{SsD
len(s) = 1149
310000_Xc__Eh4Go_4tCoYDu{UoNsrS}St
len(s) = 1148
4300001_Yhc_to4Go_CuDo_E{}XrSNsStU
len(s) = 1147
44001003__oh_XoCuocD{Er_G}tYsSNtUS
len(s) = 1146
C41003004__oocSoDYrhE}Gs_NuS_tU{tX
len(s) = 1145
DC31004004c_oohYrE}soG{Nt_SUS_tX_u
len(s) = 1144
ED43100400Ch_roo{sG_toNXStcSU}_uY_
```

Flag: NEXUS{Y0u_S0o0_G0ooD_4s_4rCh1t3ct}

Editorial

[MISC]

1. PIZ

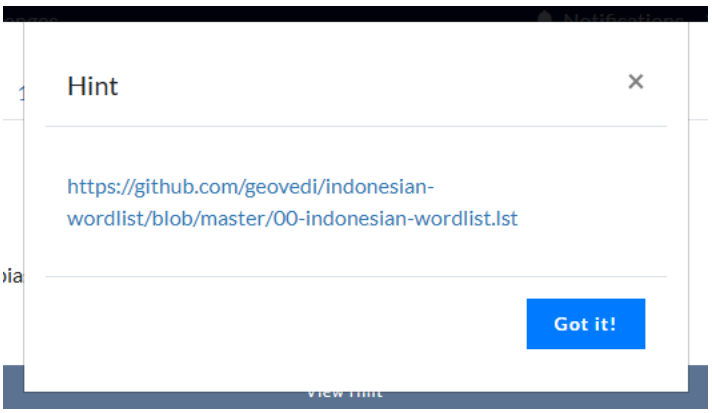


Overview:
Diberikan sebuah file **flag.zip** yang didalamnya terdapat sebuah file **flag.txt**. Lalu di tantangan kali ini saya harus memasukkan password dari flag.zip tersebut untuk mendapatkan flagnya, namun password nya tidak diberitahu, oleh karena itu salah satu caranya adalah melakukan **brute force password attack**.

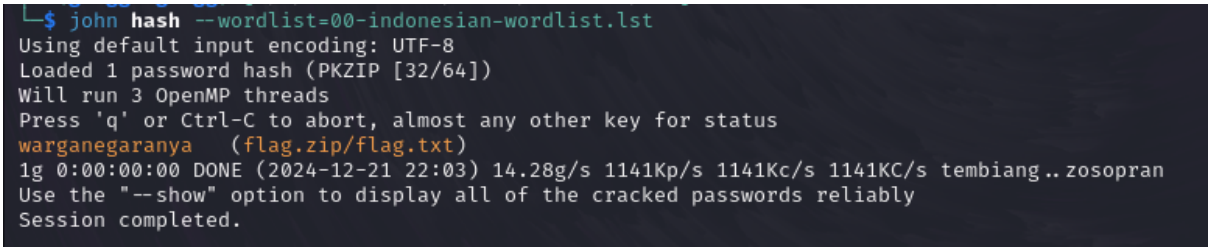
Solution:
Pertama-tama saya melakukan brute force attack password menggunakan tools **zip2john** dengan wordlist **rockyou.txt**

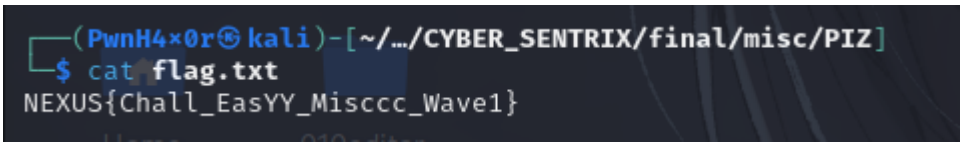
```
┌──(PwnH4×0r㉿kali)-[~/…/CYBER_SENTRIX/final/misc/PIZ]
└─$ zip2john flag.zip > hash
ver 2.0 flag.zip/flag.txt PKZIP Encr: TS_chk, cmplen=45, decmplen=31, crc=F7B6B474 ts=3C0E cs=3c0e type=8

┌──(PwnH4×0r㉿kali)-[~/…/CYBER_SENTRIX/final/misc/PIZ]
└─$ john hash --wordlist=/usr/share/wordlists/rockyou.txt
Using default input encoding: UTF-8
Loaded 1 password hash (PKZIP [32/64])
No password hashes left to crack (see FAQ)
```

Ternyata password tidak ditemukan di rockyou.txt, lalu saya mencoba berbagai wordlist namun tetap tidak dapat passwordnya.



Lalu author memberikan sebuah hint yaitu memakai wordlist **00-indonesianworlist.lst**.

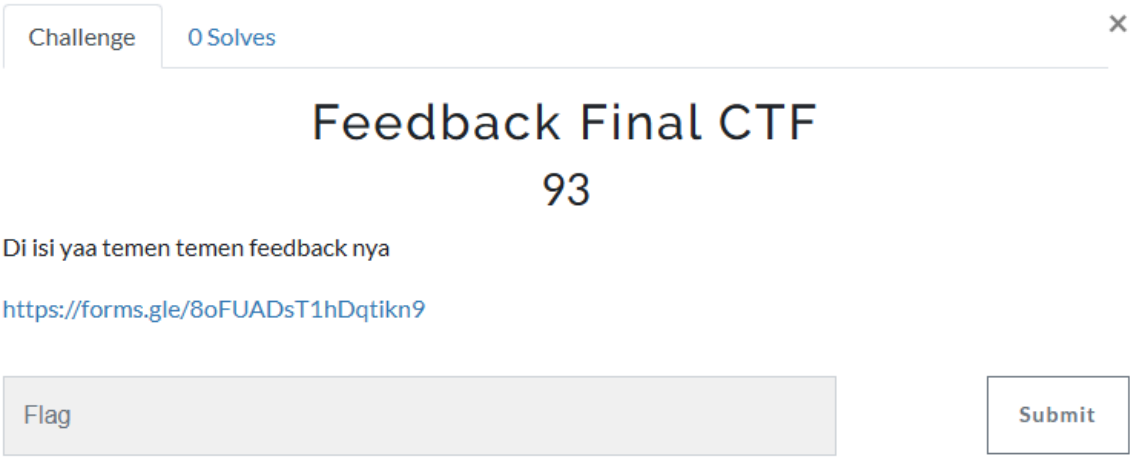




Flag: NEXUS{Chall_EasYY_Misccc_Wave1}

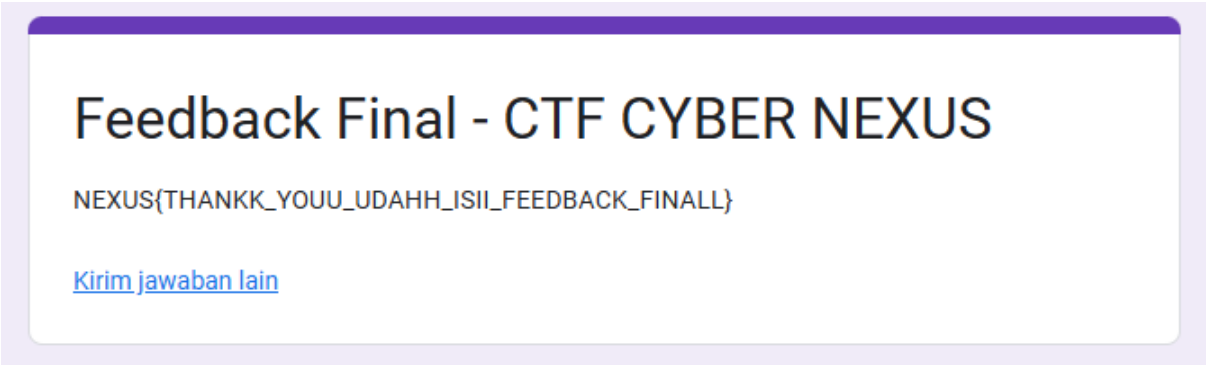
[Feedback CTF]

1. Feedback Final CTF



Overview:
Tinggal isi aja feedbacknya

Solution:



Flag: NEXUS{THANKK_YOUU_UDAHH_ISII_FEEDBACK_FINALL}