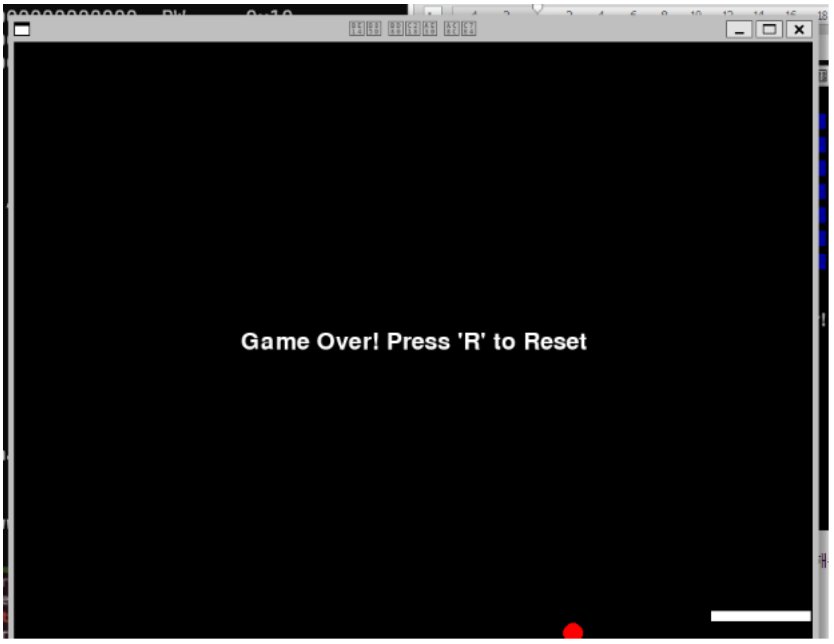
2024 3S CTF

태그 ctf 웹해킹 포너블 포렌식

▼ Can solve without thinking

이유는 모르겠는데 게임을 클리어해도 플래그가 뜨지 않았다.



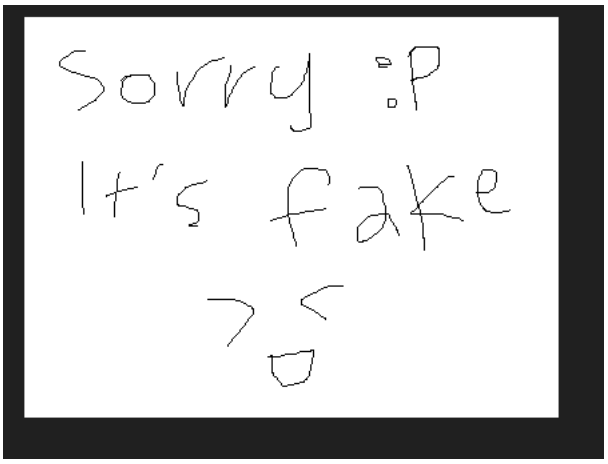
어쨌든 클리어 조건(게임 완수)은 채웠으니 그냥 남의 라업을 참고해서 정답을 가져왔다.

💡 정답: 3S{this_game_is_funny_isn_t_it?}

▼ DUM DUM :P

```
Invocation: foremost -a -v dump.bin -o dumpres
Output directory: /root/tmp/dumpres
Configuration file: /etc/foremost.conf
Processing: dump.bin
|------------------------------------------------------------------
File: dump.bin
Start: Sat Jul 27 14:23:43 2024
Length: 8 KB (9180 bytes)

Num      Name (bs=512)         Size      File Offset     Comment

0:      00000002.png           3 KB            1024       (400 x 300)
1:      00000008.png           2 KB            4198       (400 x 300)
*|
Finish: Sat Jul 27 14:23:43 2024

2 FILES EXTRACTED

png:= 2
------------------------------------------------------------------
```

파일 추출 툴 foremost로 bin 파일 안 png 두 개를 추출했다.

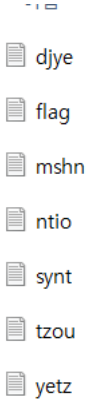


fake 두 개. foremost가 추출하지 못 한 잔여파일이 있을까 싶어 bin 파일을 다시 살펴봤다.

```
 0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  0123456789ABCDEF
37 7A BC AF 27 1C 00 03 DA 72 1D 2A 7C 01 00 00 7z¼¯'...Úr.*|...
00 00 00 00 23 00 00 00 00 00 00 00 D0 BA BB D2 ....#.......Đº»Ò
00 19 94 CB 64 52 18 E3 EB BA 33 A5 EA 50 3D E9 ..”ËdR.ãëº3¥êP=é
29 98 E4 7A 3F 73 6F D0 35 47 AD 86 29 23 A3 20 )˜äz?soÐ5G-†)#£
97 3C 08 59 E3 EF 9B DF 32 34 05 21 92 04 31 71 —<.Yãï›ß24.!’.1q
39 A2 6E EF B3 7C 59 9E 0C 6B 46 0D BA FC 47 E0 9¢nï³|Yž.kF.ºüGà
```

7z 파일 시그니처.

압축을 풀자 다음과 같이 텍스트 파일 목록이 나왔다.




```
3S{EA5Y_DUMP_F1L3_G00D_:P}
```

💡 **FLAG : 3S{EA5Y_DUMP_F1L#_G00D_:P}**

▼ lucky cook

```
root@DESKTOP-Q3EA4E7:~/tmp# ./lucky_cook
Let's Make Your Own FOOD!
You should make food start with "F"!
But, this is Unlucky word.I'll give you a chance.
Try with 'D'.
```

```
DOn't
DOn't
```

오답을 입력할 경우 단순히 입력값을 되풀이해 출력한다.

```
     00000000000000f0  0000000000000018
[12] .init             PROGBITS
     000000000000001b  0000000000000000
```

Readelf로 살펴본 섹션 헤더. init이 메인 함수이지 않을까?

```
v6 = __readfsqword(0x28u);
puts("Let's Make Your Own FOOD!");
rand = get_rand("Let's Make Your Own FOOD!", argv);
if ( rand == 76 )
{
  printf("You are a Lucky cook!");
  lucky();
}
```

아이다로 변환한 코드 → rand == 76일 경우 lucky() 함수 호출.

```
}
else
{
  printf("You should make food start with \"%c\"!\n", (unsigned int)rand);
  if ( rand != 68 )
  {
    puts("But, this is Unlucky word.I'll give you a chance.");
```

또한 문자 rand의 아스키코드 값을 기준으로 분기문을 설정하는 것으로 추측된다.

```
if ( rand == 76 )
```

아스키 코드가 76인 문자를 찾아보자.

```
>>> chr(76)
'L'
```

```
Let's Make Your Own FOOD!
You should make food start with "K"!
But, this is Unlucky word.I'll give you a chance.
Try with 'D'.
L
L
```

'L'을 입력했지만 "You are a LUcky cook!"이 출력되지 않음.

get_rand("Let's make your own food!", argv) 값이 76이 되면 lucky() 호출.

get_rand()

```
v6 = __readfsqword(0x28u);
//위치의 메모리 내용 읽어옴.
v0 = time(0LL);
//long long 0; (사실상 무한대)
srand(v0);
strcpy(v5, "ABCDEFGHIJKLMNOPQRSTUVWXYZ"); //스트링 복사
strcpy(v4, "ABCDEFGHIJKMNOPQRSTUVWXYZ"); //L만 빠져있음
v1 = rand();
//rand:near(?)
v3 = v1
   - 1569475904 * (((__int64)((unsigned __int128)(0x5DBE70A4CF21BC8DLL * (__int128)v1)
if ( v3 == 11 )
  return (unsigned int)v5[11];
//'K'
else
  return (unsigned int)v4[v3 % 26];
```

이 정도가 해석의 한계다.

```
}
if ( *(_DWORD *)food == -559038737 )
{
  puts("Wait. Let's doing another one.\n");
  lucky();
}
```

이쪽을 살펴보자. food 변수가 특정 값일 경우 lucky 실행.

저 값은... 아스키 값은 아닌 것 같다. 우선 food 변수가 포인터이니 변수 주소를 보자.

```
.bss:0000000000404011                align 4
.bss:0000000000404014                public food
.bss:0000000000404014 ; char food[]
.bss:0000000000404014 food           dd ?                    ; DATA XREF: lucky+76↑o
```

```
39:                             ; CODE XREF: main+EC↑j
        nop
        mov     eax, cs:food
        cmp     eax, 0DEADBEEFh
        jnz     short loc_401562
        lea     rax, aWaitLetSDoingA ; "Wait. Let's doing another one.\n"
        mov     rdi, rax        ; s
        call    _puts
        mov     eax, 0
        call    lucky
        jmp     short loc_401571
-----------------------------------------------------------------------
```

eax에 food 대입 후 0DEADBEEFh와 비교.

0DEADBEEFh

이게 무슨 수지?(헥스는 아닌 듯)

내 생각에는 그냥 jnz를 jz로 바꾸는 편이 빠를 것 같다. 오답이어도 정답문으로 점프하도록.

이거 아이다로는 코드 수정이 불가능한가? 검색으로 찾은 방법을 적용하려고는 하는데, 어셈블리 코드나 c 코드를 직접적으로 수정하는 방법이 마뜩치 않아 당황스럽다. 올리디버거로는 아예 파일이 열리지 않음.

▼ sick xss

▼ 분야: web

```yaml
services:
  app:
    build:
      context: .
      dockerfile: Dockerfile
    ports:
      - "9999:9999"
    restart: always
    container_name: sick_xss
    volumes:
      - ./app:/app
```

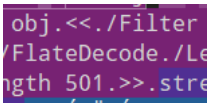
```
FROM node:latest

RUN mkdir /app


COPY app/ ./

WORKDIR /app



RUN apt-get update
RUN apt-get install -y gconf-service libasound2 li
libdbus-1-3 libexpat1 libfontconfig1 libgcc1 libgc
libgtk-3-0 libnspr4 libpango-1.0-0 libpangocairo-1
libxcb1 libxcomposite1 libxcursor1 libxdamage1 lib
libxrender1 libxss1 libxtst6 ca-certificates fonts
-release xdg-utils wget libgbm1


RUN npm install

EXPOSE 9999

CMD node app.js 9999
```

도커 컨테이너를 건축하자.

▼ CUTE_TIGER

https://blog.forensicresearch.kr/4

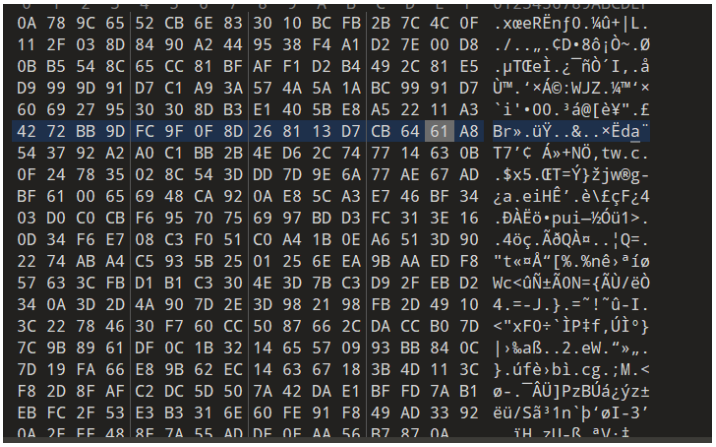
[참고자료] pdf 자료구조 (stream 뒤의 암호문 복호화)



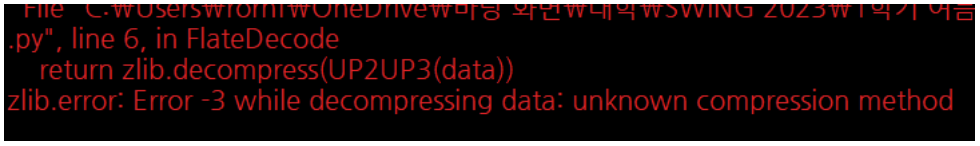
1. stream - end stream 사이의 암호문 복호화 시도.

```python
import zlib
import sys
import binascii

def FlateDecode(data):
    return zlib.decompress(UP2UP3(data))

def UP2UP3(string):
    if sys.version_info[0] > 2:
        return bytes([ord(x) for x in string])

if __name__ == "__main__":
    data = ""
    result = FateDecode(data)
    print(result)
```

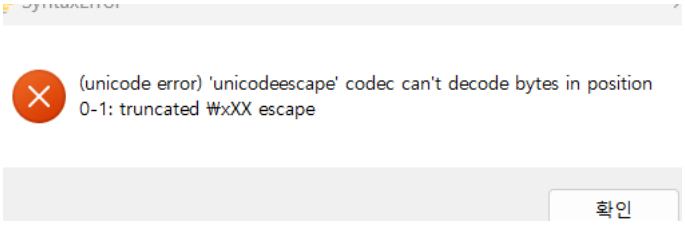
어떤 이유에서인지 헥사 코드가 붙여 넣기가 되지 않아 일일이 옮겨적지 않으면 안 되는 상황이 되었다. hxd를 경유해 복붙하려니 '올바르지 않은 문자열' 탓에 입력이 불가능하다는 에러 메시지가 출력되었다.



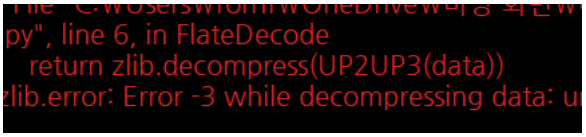
'stream' 내 헥사 코드 + 문자열.



헥사값 입력 뒤 오류.




```
if __name__ == "__main__":

    data = "78 9C CD 55 CF 6B DD
    d = data.split(" ")
    s = ''.join(d)
    result = FlateDecode(s)
    print(result)
```



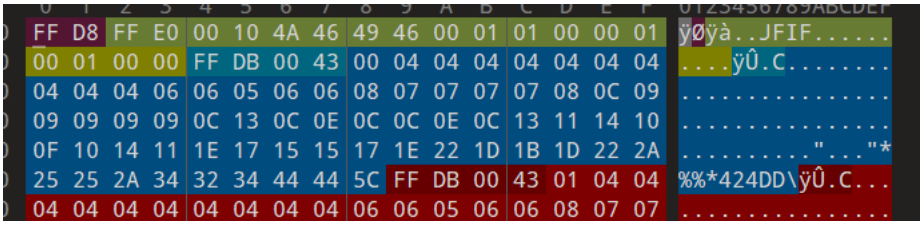
flatedecode(복호화 코드)에서 지속적 오류 발생.

pdf 파일 탐지 툴 peepdf를 다운로드 받아보자.

https://eternal-todo.com/tools/peepdf-pdf-analysis-tool

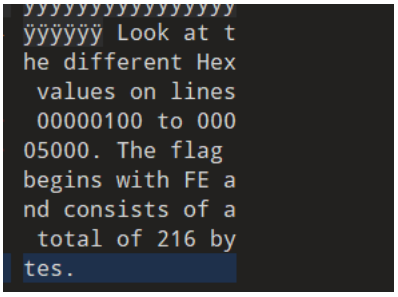
peepdf 다운로드 과정에서 문제 발생(usr/bin에 python파일 부재 문제 같은데 정확히는 모르겠다.)

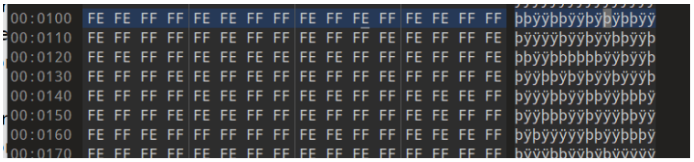
pypdf 를 다운로드 받았으나 생각했던 툴이 아니라 uninstall했다.



pdf 속 링크에 있는 이미지를 010 editor로 분석해 봤으나 특이점은 없다.

▼ Broken Hearted





이전에 LSB 추출에서 막혀 풀이를 완료하지 못 했다. 다른 사람들의 풀이를 참조해 보니 216 바이트를 8바이트씩 27줄로 정렬한 뒤 FE를 0, FF를 1로 치환해 결과를 최종적으로 아스키 코드로 변환했다고 한다.

이진수를 아스키 코드로 변환하는 건 그렇다 쳐도 FE FF를 2진수로 변환하는 아이디어는 어디서 나온거지?

아무튼 그렇게 8바이트 * 27줄로 정렬하면

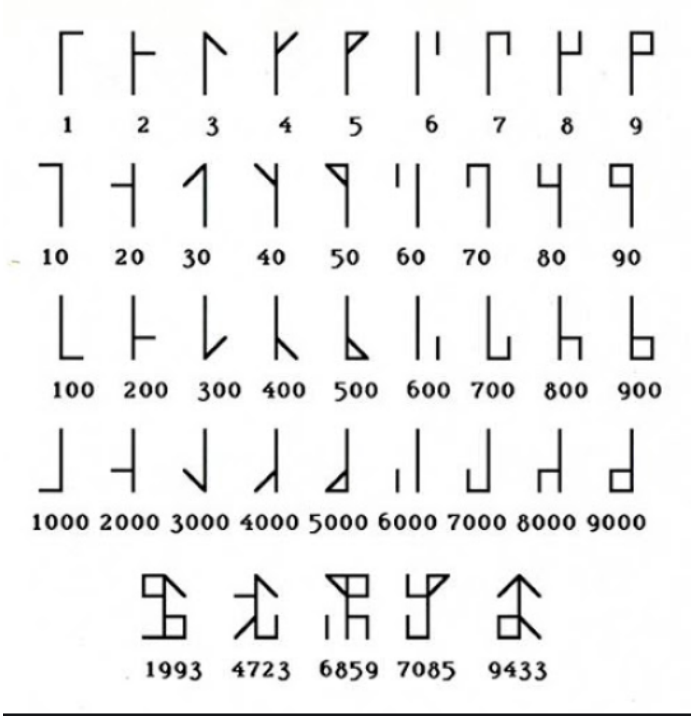
다른 분의 라업을 참조하니 LSB '최하위 비트'를 바꿔서 메시지를 숨기는 방식이란다. 최하위 단위, 즉 '0, 1'만 변조... 아.

https://bgm2020.tistory.com/40

이제 어떤 개념인지 이해가 좀 가는 것 같다.

▼ What is this

이미지 검색 결과 룬 문자와 다음 문자가 나왔다. 내 생각엔 이 편이 더 비슷한 것 같다.



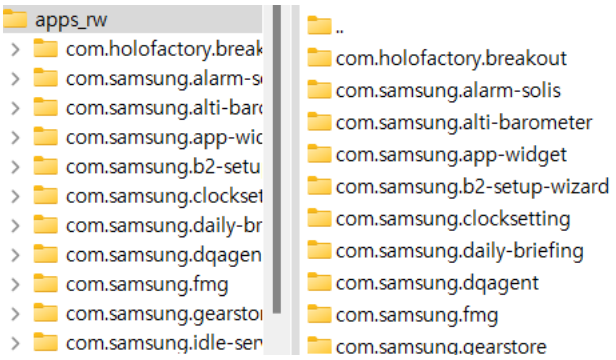
상하반전.ver

보이는 대로 끼워맞추자면:

9, 300, 1993, ? , 4723, 7085, ?, 6869, 4723, 9433

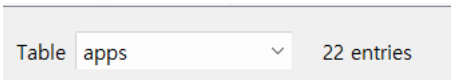
▼ Wat ch??

▼ 포렌식



우선 파일을 옵토시로 열어 삭제 파일이 있나 확인

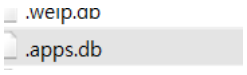
▼ 앱 개수



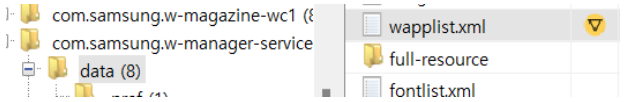
22개

였는데.... 다른 분 라업을 보니 21개가 정답인 모양이다.

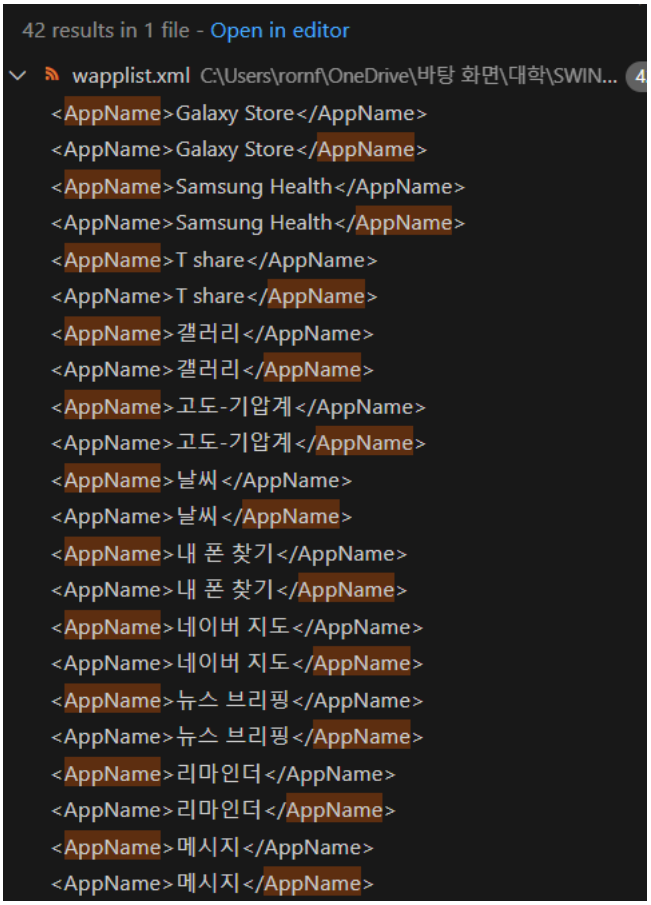
왜지?



나는 db 파일의 행 개수를 참고했다.



라업 작성하신 분은 .xml 파일을 참고하신 것 같다. 키:값 형식으로 데이터를 저장해 두는 파일이라고 일전에 잠깐 들은 적이 있다.



Xml 파일 추출해 비쥬얼 코드로 조회. 옵토시 내에서는 파일 내용이 제대로 보이지 않았는데, 추출해서 조회해야만 내용물이 제대로 보이나 보다.

아무튼 중복을 제거하고 카운팅하면 <AppName>은 21개.

▼ 3번째 리마인더 작성 시간

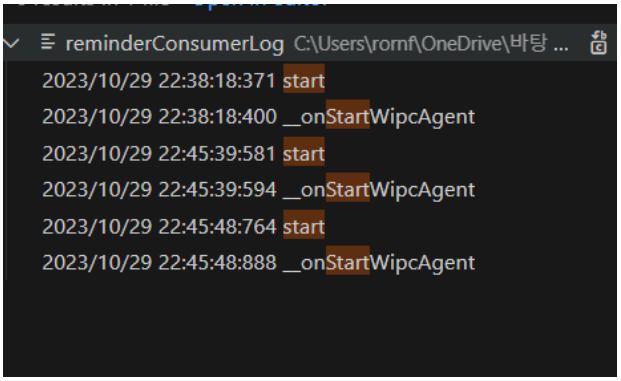
저번에 여기서 막혔었다. 리마인더 db에 항목이 두 개밖에 없었기 때문이다.

instanceId	id	title	dueTime	modTime	repeat
1	-1	칼럼작성	0	1698587138306	0
2	-1	스윙	1698588000	1698587148096	0

instanceId	id	title	dueTime	modTime	repeat
1	-1	칼럼작성	0	*2023/10/29 22:45:38*	0
2	-1	스윙	1698588000	*2023/10/29 22:45:48*	0

그래서 이미 완료된 리마인더가 있겠거니 하고 마지막으로 작성된 리마인더 시간을 환산해 적었다.

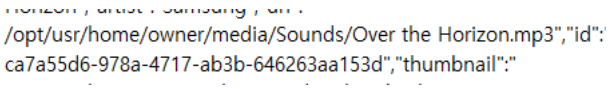
다른 분 라업을 참고하니 역시나 파일 폴더에서 파일을 추출하여 풀이하셨다.



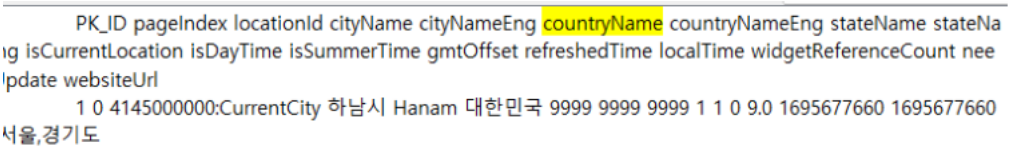
나 역시 파일을 추출해 데이터를 조회하니 플래그를 확인할 수 있었다.

22:45:48

▼ 음악 파일명



▼ 김소리 거주지



대한민국 하남시

일기예보 데이터니 거주 지역 데이터를 기반으로 할 것이라 예상함 + 문제에서 안내한 변수명과(countryName cityName) 정확히 일치함.

▼ 1695650088에 뉴스를 발행한 author

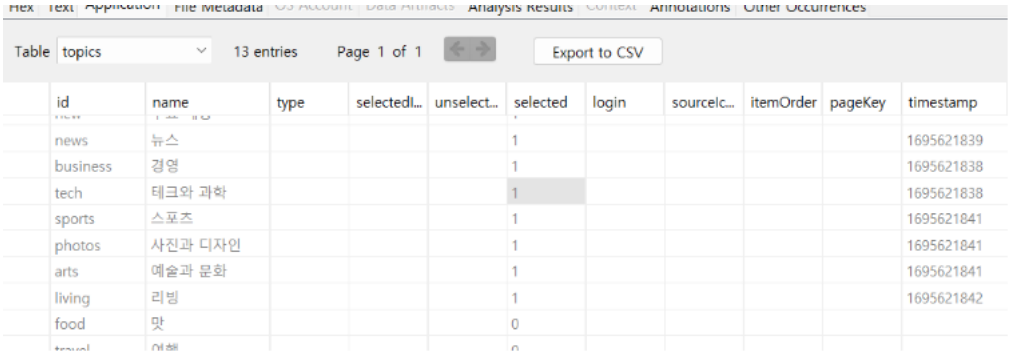
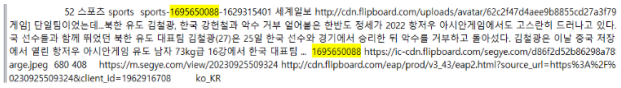


Table topics 13 entries Page 1 of 1 Export to CSV

id	name	type	selectedI...	unselect...	selected	login	sourceIc...	itemOrder	pageKey	timestamp
news	뉴스				1					1695621839
business	경영				1					1695621838
tech	테크와 과학				1					1695621838
sports	스포츠				1					1695621841
photos	사진과 디자인				1					1695621841
arts	예술과 문화				1					1695621841
living	리빙				1					1695621842
food	맛				0					

magazine.db에 1695650088라는 타임스탬프가 없길래 그냥 숫자를 검색해서 조회했다.



52 스포츠 sports sports-1695650088-1629315401 세계일보 http://cdn.filpboard.com/uploads/avatar/62c2f47d4aee9b8855cd27a3f79
게임] 단일팀이었는데..북한 유도 김철광, 한국 강헌철과 악수 거부 얼어붙은 한반도 정세가 2022 항저우 아시안게임에서도 고스란히 드러나고 있다.
국 선수들과 함께 뛰었던 북한 유도 대표팀 김철광(27)은 25일 한국 선수와 경기에서 승리한 뒤 악수를 거부하고 돌아섰다. 김철광은 이날 중국 저장
에서 열린 항저우 아시안게임 유도 남자 73kg급 16강에서 한국 대표팀 .. 1695650088 https://ic-cdn.flipboard.com/segye.com/d86f2d52b86298a78
arge.jpeg 680 408 https://m.segye.com/view/20230925509324 http://cdn.flipboard.com/eap/prod/v3_43/eap2.html?source_url=https%3A%2F%
0230925509324&client_id=1962916708 ko_KR

author라는 변수는 없다. 기사를 직접 검색해 조회했으나 역시나 기자명이 명시되어 있지 않다.

세계일보

뉴스 이슈 비즈 연예 스포츠 포토·영상

[아시안게임] 단일팀이었는데...북한 유도 김철광, 한국 강헌철과 악수 거부

기사입력 2023-09-25 13:54:48
기사수정 2023-09-25 13:54:47

김철광, 강헌철 꺾고서 매몰차게 뒤돌아서 퇴장
"국제대회마다 잘 지냈는데"...남북 정세 때문 관측도

얼어붙은 한반도 정세가 2022 항저우 아시안게임에서도 고스란히 드러나고 있다.

과거 남북 단일팀 멤버로 한국 선수들과 함께 뛰었던 북한 유도 대표팀 김철광(27)은 25일 한국 선수와 경기에서 승리한 뒤 악수를 거부하고 돌아섰다.

flag : 세계일보?

폴더 안 파일을 일일이 조회하지 않고 db 파일 내용물에 의존한 게 패착이었다. db 파일은 편리하지만 딱히 믿을 만한 존재는 못 되나 보다.

> FLAG : 3S{21_22:45:48_Over the Horizon_대한민국 하남시_세계일보}

▼ What's this song

▼ 크립토

빈도분석 + 아스키라는 힌트가 있었다.

File Edit Format Run Options Window Help

```python
import os
import random

def get_seed(size):
    return int(os.urandom(size).hex(), 16)

input = None
output = ""

salt = get_seed(16)
random.seed(salt)

crypto = "fedcba9876543210"
malware = list(crypto)
random.shuffle(malware)
malmware = ''.join(malware)

with open("input.txt", "r") as lyrics_file:
    input = lyrics_file.read()

for char in input:
    encoded_char = (bytes(char.encode()).hex())
    output += malware[crypto.index(encoded_char[0])]
    output += malware[crypto.index(encoded_char[1])]


with open("output.txt", "w") as result_file:
    result_file.write(str(output))
```

동봉된 파이썬 코드.

```
input = None
output = ""

salt = get_seed(16)

#salt에 16으로 만든 16진수 랜덤값 대입.
print(salt)

random.seed(salt)

#salt로 랜덤값(난수) 생성
#seed 사용시 특정 횟수에 출력되는 값이 고정됨.
print(random.seed(salt))

crypto = "fedcba9876543210"
malware = list(crypto)

print(malware)

random.shuffle(malware)
malmware = ''.join(malware)

print(malmware)

with open("input.txt", "r") as lyrics_file:
    input = lyrics_file.read()

for char in input:
    encoded_char = (bytes(char.encode()).hex())
    output += malware[crypto.index(encoded_char[0])]
    output += malware[crypto.index(encoded_char[1])]


with open("output.txt", "w") as result_file:
    result_file.write(str(output))
```

이 암호화 코드를 기반으로 복호화 코드를 짜는 것이 문제의 해결책이라 추측했다.

일단 중간중간 print()문을 넣어 확인한 암호화 과정은 이러하다.

```
g₩malware.py
282123513772825413218496152527409248614
None
['f', 'e', 'd', 'c', 'b', 'a', '9', '8', '7', '6', '5', '4', '3', '2', '1', '0']
3186cfd540e97b2a
Traceback (most recent call last):
  File "C:₩Users₩rornf₩OneDrive₩바탕 화면₩대학₩SWING 2023₩1학기 여름
lware.py", line 31, in <module>
    with open("input.txt", "r") as lyrics_file:
FileNotFoundError: [Errno 2] No such file or directory: 'input.txt'

= RESTART: C:₩Users₩rornf₩OneDrive₩바탕 화면₩대학₩SWING 2023₩1학
g₩malware.py
315406927295362501278122526721441611271
None
['f', 'e', 'd', 'c', 'b', 'a', '9', '8', '7', '6', '5', '4', '3', '2', '1', '0']
5a67b31e8d4c2f09
Traceback (most recent call last):
```

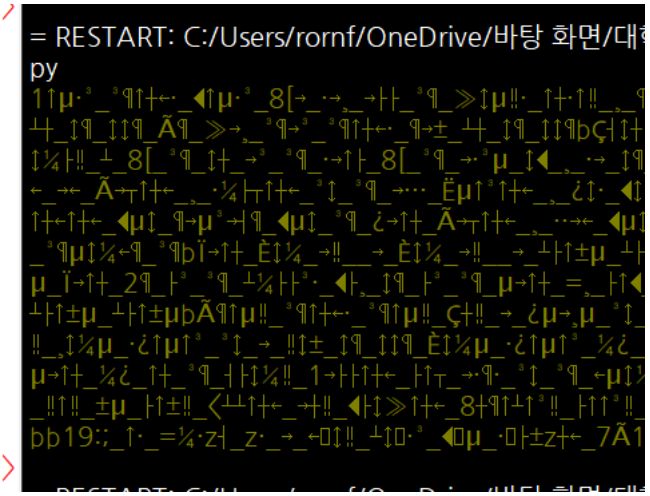
이후 gpt가 작성해준 디코딩 코드로 풀이 시도.

```python
import os
import random

def get_seed(size):
    return int(os.urandom(size).hex(), 16) #16진수 랜덤값 생성

def decrypt():
    crypto = "fedcba9876543210"
    salt = get_seed(16)

    # 암호화 시 사용된 salt와 같은 값을 사용하여 난수 생성기 초기화
    random.seed(salt)

    # malware 리스트를 복호화 시에 사용된 원래의 순서로 복구
    malware = list(crypto)
    random.shuffle(malware)  # 암호화 시와 같은 방식으로 섞음

    # 암호화된 텍스트 읽기
    with open("output.txt", "r") as result_file:
        output = result_file.read()

    # 복호화 작업
    decoded_chars = []
    for i in range(0, len(output), 2):
        encoded_hex = output[i:i+2]
        # encoded_hex를 malware의 인덱스를 이용해 16진수로 변환
        original_hex = ''.join([crypto[malware.index(char)] for char in encoded_hex])
        decoded_chars.append(chr(int(original_hex, 16)))

    decoded_text = ''.join(decoded_chars)
    print(decoded_text)

    # 복호화된 텍스트를 input.txt에 저장


# 복호화 함수 호출
decrypt()
```

▼ 암호문

Mebc\cGMKFb\DMebc\=VAI\bIm\I@@\cGJ\fHeCb\MKbMCJ\Am\GJIC\=VA\DMeJC\jI\IKC\cMeJC\HD\cGJ\fIm\
£GIc\cGMKFb\GIdJ\EJJK\HG\HHG\
£GJ\fIm\cGIc\cGMKFb\GIdJ\EJJK\HG\HHGÏ¢JBHKC\cGMKF\bJBHKC\3HKVc\mHj\cJ@@\AJ\¦GIc\mHj\cGMKN\c
£GJ\AIbcJe\HD\Am\bJI\HG\HHGÏ=\fIb\EeHNJK\DeHA\I\mHjKF\IFJ\
£INMKF\Am\bj@NMKF\cH\cGJ\AIbbJb\¦eMcMKF\Am\IHJAb\DHe\cGJ\DJf\£GIc\@HHN\Ic\AJ\cHHN\cH\AJ\
¢GHHN\cH\AJ\DJJ@MKF\AJÏ¢MKFMKF\DeHA\GJIecIBGJ\DeHA\cGJ\IIMK\
£INMKF\Am\AJbbIFJ\DeHA\cGJ\dJMKb\¢IJINMKF\Am\@JbbHK\DeHA\cGJ\EeIMK\
¢JJMKF\cGJ\EJIjcm\cGeHjFG\cGJÏ¬IMK\Hj\AICJ\AJ\I\Hj\AICJ\AJ\I\EJ@MJdJe\EJ@MJdJe\¬IMK\-
Hj\EeJIN\AJ\CHfK\9KC\EjM@C\AJ\jI\EJ@MJdJe\EJ@MJdJe\¬IMK\8G\@Jc\cGJ\Ej@@Jcb\D@m\HG\@Jc\cGJA\
Hj\AICJ\AJ\I\Hj\AICJ\AJ\I\EJ@MJdJe\EJ@MJdJeÏ£GMeC\cGMKFb\cGMeC\
¢JKC\I\IeImJe\cH\cGJ\HKJb\jI\IEHdJ\9@@\cGJ\GIcJ\cGIc\mHjVdJ\GJIeC\7Ib\cjeKJC\mHje\bIMeMc\cH\I\CHdJ\I
Hje\bIMeMc\jI\IEHdJ\HG\HHGÏ=\fIb\BGHNMKF\MK\cGJ\BeHfC\5jM@CMKF\Am\eIMK\jI\MK\cGJ\B@HjC\4I@@M
£M@@\Mc\EeHNJ\jI\IKC\Mc\eIMKJC\CHfK\=c\eIMKJC\CHfKÏÏ4096\Mb\1jb)B\)b\I\F,HC\EH,bc\D,e\b,@d)KF\2£4\

이걸 디코딩 코드 + 빈도분석 + 아스키로 복호화시키면 되겠거니 하고 일단 코드 실행.



암호화와 같이 디코딩문은 실행 시마다 출력 결과가 달라졌는데, 다시 생각해 보니 이게 바로 실패의 신호 아닌가 싶다. 아무튼 저 디코딩문을 빈도분석으로 해석하려다 실패했다.