

## D Spooky Little Ghost

998 pts

Yu\_212, keymoon 님이 해결했습니다.

crypto

### Description

**Slide** away from this spooky little GHOST!

 Translate

[📄 문제 파일 다운로드](#)

이번 CTF 문제들을 다 훑어봤는데 그나마 이 문제가 시도해볼만 한 것 같아서 골라봤다  
파일을 다운받아보니 코드가 적힌 파일이 여러 개 있었다

Dockerfile

문제의 환경 세팅에 대해 알려주는 Dockerfile을 먼저 확인해보면,

FROM

ubuntu:22.04@sha256:965fbcae990b0467ed5657caceaec165018ef44a4d2d46c7cdea80a9d  
ff0d1ea

RUN apt update

RUN apt-get install -y socat python3

COPY ./deploy/flag /flag

COPY ./deploy/prob.py /prob.py

COPY ./deploy/utls.py /utls.py

COPY ./deploy/cipher.py /cipher.py

RUN chmod 755 /flag /prob.py

EXPOSE 1112

CMD socat TCP-LISTEN:1112,reuseaddr,fork EXEC:/prob.py

먼저 ubuntu 22.04 버전의 우분투 이미지를 사용한다는 것을 알 수 있다

그리고 run apt 명령어를 보니 패키지를 설치해야 하는 것 같은데, 찾아보니 socat은 네트워크 연결을 제어할 수 있는 유틸리티이고, 클라이언트와 서버 사이의 연결을 관리하도록 돕는다고 한다 파이썬도 설치해 준 걸 보니 파이썬을 활용해 뭔갈 실행해야 하는 것 같다 이 부분은 다른 파일을 보면 알 수 있을 것 같다

다음으로는 COPY 명령어들이 나열되어 있다 저기 적혀 있는 파일들은 문제 파일 중 Dockerfile을 제외한 deploy 폴더 안에 있는 파일들인데, 그 파일들을 Docker 컨테이너 안으로 복사하는 명령어이다

RUN chmod 755 명령어를 사용해 flag와 prob.py 파일에 읽기와 쓰기, 실행 권한을 설정 해주었다

EXPOSE 1112는 찾아보니 컨테이너가 1112번 포트에서 통신할 수 있도록 설정해주는 명령어라고 한다

마지막으로 CMD 명령어는 socat을 통해 1112번에서 TCP 연결을 하고, 연결이 되면 prob.py 파일이 실행되도록 한다

Flag

먼저 flag 파일을 열어보니 다음과 같이 나왔다

그냥 플래그 형식을 알려줄 뿐, 별다른 정보는 없는 것 같아 이쯤에서 넘어가야겠다

DH{sample flag sample flag sample flag sample flag sample flag }

Cipher

다음으로 cipher 파일을 열어보았다

```
from utils import *
```

```
class GHOST:
```

```
    sbox = (  
        (0xC, 0x4, 0x6, 0x2, 0xA, 0x5, 0xB, 0x9, 0xE, 0x8, 0xD, 0x7, 0x0, 0x3, 0xF, 0x1),  
        (0x6, 0x8, 0x2, 0x3, 0x9, 0xA, 0x5, 0xC, 0x1, 0xE, 0x4, 0x7, 0xB, 0xD, 0x0, 0xF),  
        (0xB, 0x3, 0x5, 0x8, 0x2, 0xF, 0xA, 0xD, 0xE, 0x1, 0x7, 0x4, 0xC, 0x9, 0x6, 0x0),  
        (0xC, 0x8, 0x2, 0x1, 0xD, 0x4, 0xF, 0x6, 0x7, 0x0, 0xA, 0x5, 0x3, 0xE, 0x9, 0xB),  
        (0x7, 0xF, 0x5, 0xA, 0x8, 0x1, 0x6, 0xD, 0x0, 0x9, 0x3, 0xE, 0xB, 0x4, 0x2, 0xC),  
        (0x5, 0xD, 0xF, 0x6, 0x9, 0x2, 0xC, 0xA, 0xB, 0x7, 0x8, 0x1, 0x4, 0x3, 0xE, 0x0),  
        (0x8, 0xE, 0x2, 0x5, 0x6, 0x9, 0x1, 0xC, 0xF, 0x4, 0xB, 0x0, 0xD, 0xA, 0x3, 0x7),  
        (0x1, 0x7, 0xE, 0xD, 0x0, 0x5, 0x8, 0x3, 0x4, 0xF, 0xA, 0x6, 0x9, 0xC, 0xB, 0x2)  
    )
```

```
    def __init__(self, key: bytes) -> None:
```

```
        self._block_size = 8
```

```
        self._round_keys = self._expand_key(key)
```

```
        self._state = []
```

```
    def _expand_key(self, key: bytes) -> list[list[int]]:
```

```
        assert len(key) == 8
```

```
        key_bits = bytes_to_bits(key)
```

```
        round_keys: list[list[int]] = []
```

```
        for i in range(0, 64, 32):
```

```
            round_keys.append(key_bits[i:i+32])
```

```
return round_keys
```

```
def _substitution(self, bit32: list[int]) -> list[int]:
```

```
    res: list[int] = []
```

```
    for i,j in enumerate(range(0,32,4)):
```

```
        res += int_to_bits(self.sbox[7-i][bits_to_int(bit32[j:j+4])], 4)
```

```
    return res
```

```
def _round_function(self, round_n: int, is_enc: bool) -> None:
```

```
    round_key_idx = round_n%2
```

```
    state_hi = self._state[:32]
```

```
    state_lo = self._state[32:]
```

```
    state_lo = add_mod_2_32(state_lo, self._round_keys[round_key_idx])
```

```
    state_lo = self._substitution(state_lo)
```

```
    state_lo = rol11(state_lo)
```

```
    state_lo = xor_lst(state_lo, state_hi)
```

```
    if (is_enc and round_n == 31) or (not is_enc and round_n == 0):
```

```
        self._state[:32] = state_lo
```

```
    else:
```

```
        self._state[:32] = self._state[32:]
```

```
        self._state[32:] = state_lo
```

```
def _encrypt(self, plaintext: bytes) -> bytes:
```

```
    self._state = bytes_to_bits(plaintext)
```

```
    for i in range(32):
```

```
        self._round_function(i, is_enc=True)
```

```
    return bits_to_bytes(self._state)
```

```
def _decrypt(self, ciphertext: bytes) -> bytes:
```

```
    self._state = bytes_to_bits(ciphertext)
```

```
    for i in range(31, -1, -1):
```

```
        self._round_function(i, is_enc=False)
```

```
    return bits_to_bytes(self._state)
```

```
def encrypt(self, plaintext: bytes) -> bytes:
```

```
    assert len(plaintext)%self._block_size == 0
```

```
    ciphertext = b''
```

```
    for i in range(0, len(plaintext), self._block_size):
```

```
        ciphertext += self._encrypt(plaintext[i:i + self._block_size])
```

```
    return ciphertext
```

```
def decrypt(self, ciphertext: bytes) -> bytes:
```

```
    assert len(ciphertext)%self._block_size == 0
```

```
    plaintext = b''
```

```
    for i in range(0, len(ciphertext), self._block_size):
```

```
        plaintext += self._decrypt(ciphertext[i:i + self._block_size])
```

```
    return plaintext
```

```

def test():

    import os

    trial = 0x1000

    for _ in range(trial):

        key = os.urandom(8)

        g = GHOST(key)

        plain = os.urandom(8)

        cipher = g.encrypt(plain)

        assert plain == g.decrypt(cipher)

if __name__ == '__main__':

    test()

```

이 파일은 암호화를 구현하는 코드인 것 같다 문제부터 스푸키 어쭈구더니 암호 이름도 Ghost인가보다.. 먼저 코드를 바탕으로 ghost 클래스를 간단히 써보자면 64비트 크기의 key와 64비트 크기의 평문을 처리한다 각 블록은 32라운드의 암호화 / 복호화 과정을 거쳐서 처리된다

먼저 S-Box에 대해 설명해보자면, 암호화 과정의 각 라운드에서 가장 먼저 수행되는 연산이며, 8개의 4비트 S-box로 구성된다 각 4비트 값을 다른 값으로 매핑하는 치환 테이블이기 때문에, 각 라운드마다 치환 연산을 수행해 비트를 섞어 암호화한다 (참

고 : <https://mintnlatte.tistory.com/108> )

다음으로 \_expand\_key와 같은 경우, 8바이트 키를 32비트씩 나눠서 두 개의 라운드 키를 만든다 ghost 암호화의 총 32라운드가 모두 진행될 때 이 두 개의 라운드 키를 반복적으로 사용한다

암호화 과정을 살펴보자면, 가장 먼저 plaintext를 비트로 변환해 내부상태 (\_state)로 바꾼다 다음으로 \_round\_function을 진행해야 하는데, 이때 각 라운드는 상위 32비트와 하위 32비트로 상태를 나눠 진행한다 먼저 하위 32비트는 라운드 키와 더한 다음 S-box로

치환하고, 11로 shift, 마지막으로 상위 32 비트로 XOR 연산을 진행한다 이런 식으로 암호화를 완료하면 비트를 바이트로 변환해 암호문을 반환하게 된다 복호화 과정은 암호화의 반대 순서로 진행할 수 있다

Prob

이제 중요해보이는 파일인 Prob 파일을 열어 확인해보자

```
#!/usr/bin/env python3
```

```
import os
```

```
from cipher import GHOST
```

```
from utils import *
```

```
def menu() -> int:
```

```
    print('1. encrypt')
```

```
    print('2. decrypt')
```

```
    print('3. flag')
```

```
    print('4. exit')
```

```
    return int(input('> '))
```

```
if __name__ == '__main__':
```

```
    with open('flag', 'rb') as f:
```

```
        flag = f.read()
```

```
    key = os.urandom(8)
```

```
    g = GHOST(key)
```

```
    while True:
```

```

i = menu()

if i == 1:

    msg = input('plaintext(hex)> ')

    enc = g.encrypt(bytes.fromhex(msg))

    print('ciphertext(hex)>', enc.hex())

elif i == 2:

    enc = input('ciphertext(hex)> ')

    msg = g.decrypt(bytes.fromhex(enc))

    print('plaintext(hex)>', msg.hex())

elif i == 3:

    new_key = xor_bytes(key, bytes.fromhex('deadbeefcafebabe'))

    new_g = GHOST(new_key)

    enc_flag = new_g.encrypt(flag)

    print('encrypted_flag(hex)> ', enc_flag.hex())

else:

    break

```

이 파이썬 코드는 네트워크 상에서 암호화/복호화를 수행하고, 플래그 (근데 암호화된...)를 제공하는 내요의 코드이다

먼저 키 설정 부분을 보자면, 8바이트의 랜덤 키를 생성하고, 그 키를 사용해 ghost 암호화 알고리즘 객체 g를 만들게 된다 이 키는 암호화 복호화를 진행하는데 사용된다

다음으로 메뉴 부분을 구현함 함수를 보자면, 사용자에게 메뉴를 보여주고, 암호화, 복호화, 플래그 확인, 종료 기능이 있다는 걸 확인할 수 있다

메인 부분에는 암호화/복호화/플래그암호화 부분을 구성하고 있는데, 먼저 암호화 파트에서는 사용자가 입력한 평문을 16진수로 받고, 이를 ghost 암호화 알고리즘을 사용해 암호화하고, 출력한다 이때 평문은 bytes.fromhex(msg)로 변환된 다음 암호화되고, 그 결과가 다시 16진수로 바뀌어 출력된다



복호화 부분은 암호문을 16진수로 입력받고 (당연함 암호문이 최종으로 16진수로 출력 됨) 이를 복호화해 평문을 출력하는데, 이때 복호화 과정은 암호화의 반대 순서로 진행된다

마지막으로 플래그 암호화 부분은 `xor_bytes(key, bytes.fromhex('deadbeefcafebabe'))`로 새로운 키로 변환하고 새로운 키로 `new_g`를 생성한다 그 새 키로 `flag`를 암호화하고 이를 출력한다

utils

이 파일에는 암호화 알고리즘에 필요한 다양한 유틸리티 함수들이 들어가 있다

```
def xor_lst(a: list[int], b: list[int]) -> list[int]:
```

```
    return [x^y for x,y in zip(a,b)]
```

```
def xor_bytes(a: bytes, b: bytes) -> bytes:
```

```
    return bytes([x^y for x,y in zip(a,b)])
```

```
def int_to_bits(d: int, bits_len:int = 8) -> list[int]:
```

```
    return [int(x) for x in bin(d)[2:].zfill(bits_len)]
```

```
def bits_to_int(bits: list[int]) -> int:
```

```
    return int(''.join([str(x) for x in bits]),2)
```

```
def bytes_to_bits(m: bytes) -> list[int]:
```

```
    bits = []
```

```
    for b in m:
```

```
        bits += [int(x) for x in bin(b)[2:].zfill(8)]
```

```
    return bits
```

```
def bits_to_bytes(bits: list[int]) -> bytes:
```

```
    return bits_to_int(bits).to_bytes(len(bits)//8, 'big')
```

```
def add_mod_2_32(bit32: list[int], key32: list[int]) -> list[int]:
```

```
    return int_to_bits((bits_to_int(bit32) + bits_to_int(key32)) % (2**32), 32)
```

```
def rol11(bit32: list[int]) -> list[int]:
```

```
    return bit32[11:]+bit32[:11]
```

xor\_lst(a: list[int], b: list[int]) -> list[int]: : 두 리스트 a와 b의 각 요소를 XOR해서 결과 리스트를 반환한다 이때, XOR 연산은 대칭적이기 때문에 동일한 값 두 개로 XOR하면 원래 값을 복원할 수 있다

xor\_bytes(a: bytes, b: bytes) -> bytes: : 두 바이트 배열 a와 b의 각 바이트를 XOR해서 결과 바이트 배열을 반환한다 이는 GHOST 알고리즘 키를 XOR해서 새로운 키를 만들 때 사용한다

int\_to\_bits(d: int, bits\_len: int = 8) -> list[int]: 정수 d를 주어진 길이만큼 이진수로 변환하고, 그 이진수를 리스트로 반환한다 이때 각 자릿수는 0 또는 1로 이루어진 리스트로 표현된다

bits\_to\_int(bits: list[int]) -> int: : 비트 리스트 bits를 정수로 변환한다 이때 리스트로 표현된 이진수를 정수로 다시 변환해서 반환된다

bytes\_to\_bits(m: bytes) -> list[int]: : 바이트 배열 m을 비트 리스트로 변환한다 각 바이트를 8비트 이진수로 변환한 다음 모든 비트를 리스트로 반환한다

bits\_to\_bytes(bits: list[int]) -> bytes: : 비트 리스트 bits를 바이트 배열로 변환한다 리스트로 표현된 이진수를 정수로 변환한 다음 그 정수를 바이트로 변환해서 반환한다

add\_mod\_2\_32(bit32: list[int], key32: list[int]) -> list[int]: : 두 32비트 비트 리스트를 32비트 정수로 변환한 후 더하고, 32비트에서 모듈로 연산을 적용한 결과를 반환한다

rol11(bit32: list[int]) -> list[int]: : 32비트 비트 리스트를 11비트 왼쪽으로 회전한 결과를 반환한다 GHOST 알고리즘에서 라운드별 비트 셔플링을 하는데 사용된다

문제 풀이

먼저 Dockerfile에서 환경설정을 하는 법을 알려줬으니, 도커를 활용해 문제를 풀어야 한다

먼저 우분투를 열고 (포맷해서 우분투에 Docker 설치 안 되어있음) docker를 다음 명령어를 사용해 설치해준다

```
sudo apt update
```

```
sudo apt install -y docker.io
```

```
sudo systemctl start docker
```

```
sudo systemctl enable docker
```

그 다음으로 ctf\_docker\_project 디렉토리를 생성한다

```
mkdir ~/my_docker_project
```

```
cd ~/my_docker_project
```

deploy 디렉토리를 만들고 파일들을 저장한다

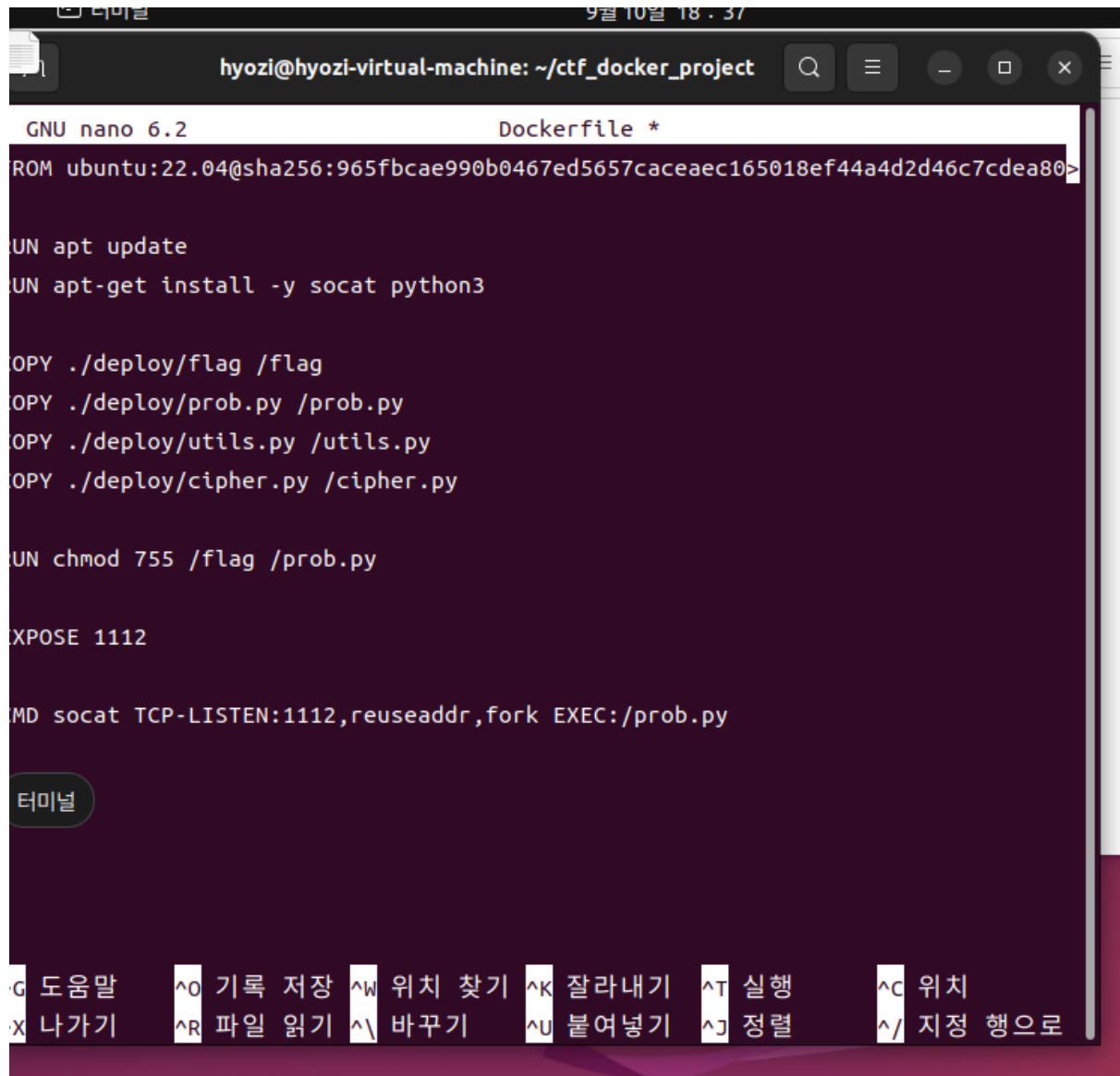
```
hyozi@hyozi-virtual-machine:~/ctf_docker_project$ mkdir deploy
hyozi@hyozi-virtual-machine:~/ctf_docker_project$ mv flag prob.py utils.py cipher.py deploy/
mv: 'flag' 상태 정보 확인 불가: 그런 파일이나 디렉터리가 없습니다
mv: 'prob.py' 상태 정보 확인 불가: 그런 파일이나 디렉터리가 없습니다
mv: 'utils.py' 상태 정보 확인 불가: 그런 파일이나 디렉터리가 없습니다
mv: 'cipher.py' 상태 정보 확인 불가: 그런 파일이나 디렉터리가 없습니다
hyozi@hyozi-virtual-machine:~/ctf_docker_project$ mv ~/Downloads/flag ~/Downloads/prob.py ~/Downloads/utils.py ~/Downloads/cipher.py ~/ctf_docker_project/deploy/
```

그냥 옮기기를 하니까 경로가 달라 파일을 못 찾길래 downloads 폴더 안에 있는 파일을 디렉토리로 이동할 수 있도록 명령어를 새로 작성해줬더니 됐다

다음으로 작업 디렉토리 안에 Dockerfile을 만들고 문제 파일에 있던 코드를 복붙해줬다

touch Dockerfile

nano Dockerfile



```
GNU nano 6.2 Dockerfile *
FROM ubuntu:22.04@sha256:965fbcae990b0467ed5657caceaec165018ef44a4d2d46c7cdea80>

RUN apt update
RUN apt-get install -y socat python3

COPY ./deploy/flag /flag
COPY ./deploy/prob.py /prob.py
COPY ./deploy/utils.py /utils.py
COPY ./deploy/cipher.py /cipher.py

RUN chmod 755 /flag /prob.py

EXPOSE 1112

CMD socat TCP-LISTEN:1112,reuseaddr,fork EXEC:/prob.py
```

터미널

도움말 ^O 기록 저장 ^W 위치 찾기 ^K 잘라내기 ^T 실행 ^C 위치  
나가기 ^R 파일 읽기 ^\ 바꾸기 ^U 붙여넣기 ^J 정렬 ^/ 지정 행으로

다음으로 Docker 이미지를 빌드해주려고 했는데..... 안됐다

docker build -t customimage .

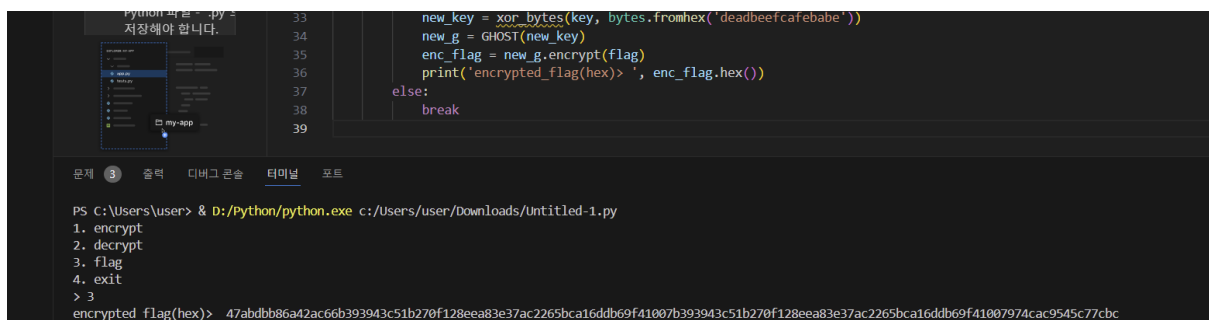
```
hyozi@hyozi-virtual-machine:~/ctf_docker_project$ docker build -t mycustomimage .
DEPRECATED: The legacy builder is deprecated and will be removed in a future release.

Install the buildx component to build images with BuildKit:
https://docs.docker.com/go/buildx/

Ubuntu Software Center failed while trying to connect to the Docker daemon socket at unix:///var/run/docker.sock: Post "http://%2Fvar%2Frun%2Fdocker.sock/v1.24/build?buildargs=%7B%7D&cachefrom=%5B%5D&cgroupparent=&cpuperiod=0&cpuquota=0&cpusetcpus=&cpusetmems=&cpushares=0&dockerfile=Dockerfile&labels=%7B%7D&memory=0&memswap=0&networkmode=default&rm=1&shmsize=0&t=mycustomimage&target=&ulimits=null&version=1": dial unix /var/run/docker.sock: connect: permission denied
hyozi@hyozi-virtual-machine:~/ctf_docker_project$ sudo usermod -aG docker $USER
[sudo] hyozi 암호:
죄송합니다만, 다시 시도하십시오.
[sudo] hyozi 암호:
hyozi@hyozi-virtual-machine:~/ctf_docker_project$ docker build -t mycustomimage .
```

사용자 권한도 줘보고 docker 실행도 해봤는데 여기서 막힌다 일단 우분투로 여는 건 포기하고 파이썬 파일들 보고 암호화 복호화만 도전해보자

vscode로 prob 파일 실행시켜서 암호화된 플래그를 얻어보자 3번을 누르면 암호화된 플래그가 출력된다



```
33 new_key = xor_bytes(key, bytes.fromhex('deadbeefcafebabe'))
34 new_g = GHOST(new_key)
35 enc_flag = new_g.encrypt(flag)
36 print('encrypted_flag(hex)> ', enc_flag.hex())
37 else:
38 break
39
```

```
PS C:\Users\user> & D:/Python/python.exe c:/Users/user/Downloads/Untitled-1.py
1. encrypt
2. decrypt
3. flag
4. exit
> 3
encrypted_flag(hex)> 47abdbb86a42ac66b393943c51b270f128eea83e37ac2265bca16ddb69f41007b393943c51b270f128eea83e37ac2265bca16ddb69f41007974cac9545c77cbc
```

암호화된 플래그는 다음과 같다

47abdbb86a42ac66b393943c51b270f128eea83e37ac2265bca16ddb69f41007b393943c51b270f128eea83e37ac2265bca16ddb69f41007974cac9545c77cbc

이제 저 암호화된 플래그를 GHOST 알고리즘으로 복호화하고, 그 결과를 XOR 연산을 반대로 해야 한다

그 과정을 파이썬으로 구현해 보았는데, 내가 정확히 암호화에 대한 개념을 이해하지 못한 것 같다 여러 번 수정했는데도 계속 오류가 떠서 여기까지 해야 할 것 같다....

```
4 enc_flag_hex = "47abdbb86a42ac66b393943c51b270f128eea83e37ac2265bca16ddb69f410"
5 enc_flag = bytes.fromhex(enc_flag_hex)
6
7 xor_value = bytes.fromhex('deadbeefcafebab')
8
9 def decrypt_flag(enc_flag: bytes, original_key: bytes) -> bytes:
10     new_key = xor_bytes(original_key, xor_value)
11
12     ghost = GHOST(new_key)
13
14     decrypted_flag = ghost.decrypt(enc_flag)
15
16     return decrypted_flag
17
18 flag = decrypt_flag(enc_flag, original_key)
19 print("복호화된 플래그:", flag.decode())
20
```

터미널 포트 Python