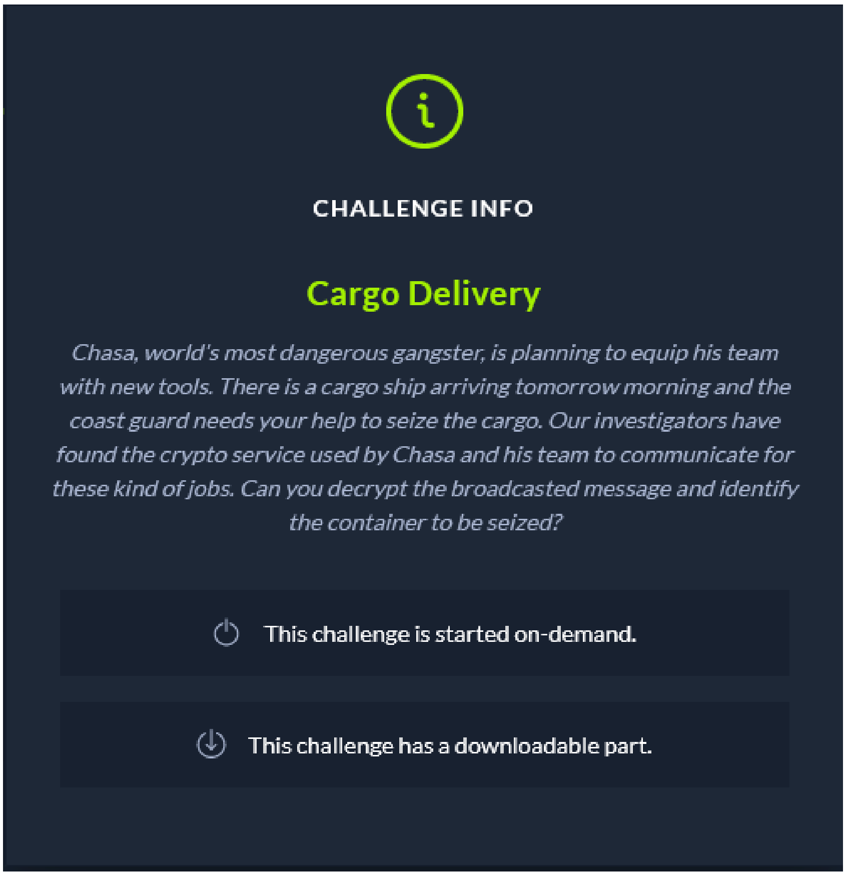
CARGO_DELIVERY



CHALLENGE INFO

Cargo Delivery

Chasa, world's most dangerous gangster, is planning to equip his team with new tools. There is a cargo ship arriving tomorrow morning and the coast guard needs your help to seize the cargo. Our investigators have found the crypto service used by Chasa and his team to communicate for these kind of jobs. Can you decrypt the broadcasted message and identify the container to be seized?

This challenge is started on-demand.

This challenge has a downloadable part.

MATERIAL:
server.py

FLAG:
HTB{CBC_0r4cl3}

SOLVER:
M1gnus

Foothold

The challenge presents us the code of a server:

```
from Crypto.Cipher import AES
import socketserver
import signal
import os
import random
import time
from secret import flag

KEY_LENGTH = 16
BLOCK_SIZE = AES.block_size

key = os.urandom(KEY_LENGTH)

def add_padding(msg):
    pad_len = BLOCK_SIZE - (len(msg) % BLOCK_SIZE)
    padding = bytes([pad_len]) * pad_len
    return msg + padding

def remove_padding(data):
    pad_len = data[-1]
    if pad_len < 1 or pad_len > BLOCK_SIZE:
    return None
    for i in range(1, pad_len):
    if data[-i-1] != pad_len:
        return None
    return data[:-pad_len]

def encrypt(msg):
    iv = os.urandom(BLOCK_SIZE)
    cipher = AES.new(key, AES.MODE_CBC, iv)
    return (iv + cipher.encrypt(add_padding(msg))).hex()


def decrypt(data):
    iv = data[:BLOCK_SIZE]

    cipher = AES.new(key, AES.MODE_CBC, iv)
    return remove_padding(cipher.decrypt(data[BLOCK_SIZE:]))

def is_padding_ok(data):
    if decrypt(data) is not None:
    return 'This is a valid ciphertext!\n'
    else:
    return 'Invalid ciphertext\n'



def challenge(req):
    req.sendall(bytes('This crypto service is used for Chasa\'s delivery system!\n'
    'Not your average gangster.\n'
    'Options:\n'
    '1. Get encrypted message.\n'
    '2. Send your encrypted message.\n', 'utf-8'))
    try:
    choice = req.recv(4096).decode().strip()

    index = int(choice)

    if index == 1:
        req.sendall(bytes(encrypt(flag) + '\n','utf-8'))
    elif index == 2:
        req.sendall(bytes('Enter your  ciphertext:\n', 'utf-8'))
        ct = req.recv(4096).decode().strip()
        req.sendall(bytes(is_padding_ok(bytes.fromhex(ct)), 'utf-8'))
    else:
        req.sendall(bytes('Invalid option!\n', 'utf-8'))
        exit(1)
    except:
    exit(1)
```

By analyzing the code is possible to see that the server offer a choice:

“1”: the user can receive a hexadecimal string in the following format:

iv||ciphertext

where “ciphertext” is the flag encrypted with AES using the CBC mode and “iv” is the initialization vector used in the encryption process.

“2”: the user send a hexadecimal string in the form:

iv||ciphertext

where “ciphertext” is a ciphertext chosen by the user and iv the initialization vector used in the original encryption process. The server will tell if the received ciphertext is derived from a well padded plaintext or not.

The server that adopt this kind of behavior is called “padding oracle” and this information leak make them vulnerable to a class of attack named “padding oracle attacks”.

The attack

In our specific case we have a server that act like a “CBC padding oracle”, so we have to perform a “CBC padding oracle attack”. Here is possible to find a clear explanation of the attack.

The implementation

```
from pwn import *
from binascii import unhexlify


def build_malicious_block(depth, C, B):
    padding_block = int.from_bytes(depth.to_bytes(1, "big")*depth, "big")
    B = int.from_bytes(B, "big")
    C = int.from_bytes(C, "big")
    return (padding_block ^ B ^ C).to_bytes(16, "big").hex()


r = remote('docker.hackthebox.eu', 30244)

r.recvuntil("Send your encrypted message.\n")
r.sendline(b"1")
iv_flag = r.recvline().decode()
iv = iv_flag[:32]
flag = iv_flag[32:]

print(f"iv: {iv}")
print(f"encrypted_flag: {flag}")
print(f".:flag:.")

B = b""

for i in range(16):
    for j in range(256):
        r.recvuntil("Send your encrypted message.\n")
        r.sendline(b"2")
        r.recvline()
        P = j.to_bytes(1, "big")+B
        block = build_malicious_block(i+1, unhexlify(iv), P)
        r.sendline(iv+block+flag)
        result = r.recvline()
        if b"Invalid" in result or i == j:
            continue
        else:
            B = chr(j).encode()+B
            print(f"{B}", end="\r")
            break
print()
```

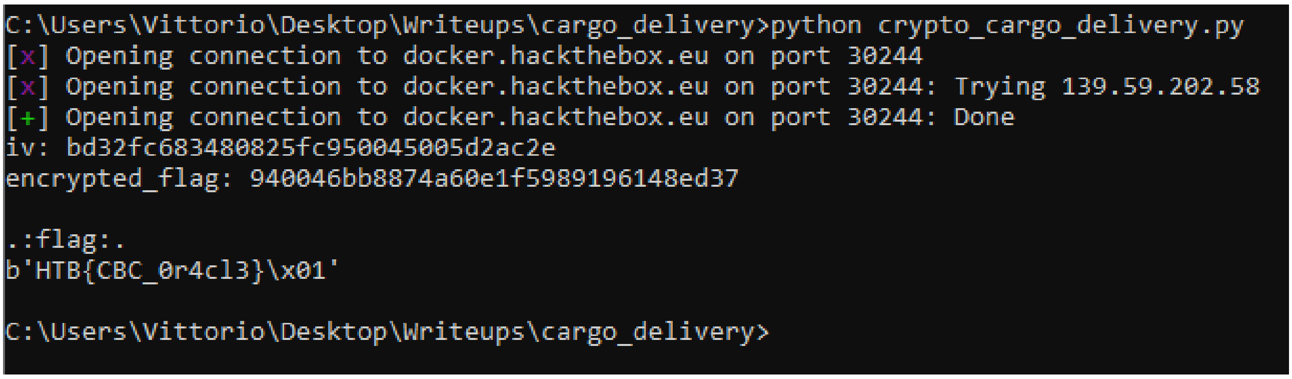
Recover the flag

To recover the flag we have only to run the script:

```
C:\Users\Vittorio\Desktop\Writeups\cargo_delivery>python crypto_cargo_delivery.py
[x] Opening connection to docker.hackthebox.eu on port 30244
[x] Opening connection to docker.hackthebox.eu on port 30244: Trying 139.59.202.58
[+] Opening connection to docker.hackthebox.eu on port 30244: Done
iv: bd32fc683480825fc950045005d2ac2e
encrypted_flag: 940046bb8874a60e1f5989196148ed37

.:flag:.
b'HTB{CBC_0r4cl3}\x01'
```

CHEESE!



```
C:\Users\Vittorio\Desktop\Writeups\cargo_delivery>python crypto_cargo_delivery.py
[x] Opening connection to docker.hackthebox.eu on port 30244
[x] Opening connection to docker.hackthebox.eu on port 30244: Trying 139.59.202.58
[+] Opening connection to docker.hackthebox.eu on port 30244: Done
iv: bd32fc683480825fc950045005d2ac2e
encrypted_flag: 940046bb8874a60e1f5989196148ed37

.:flag:.
b'HTB{CBC_0r4cl3}\x01'

C:\Users\Vittorio\Desktop\Writeups\cargo_delivery>
```