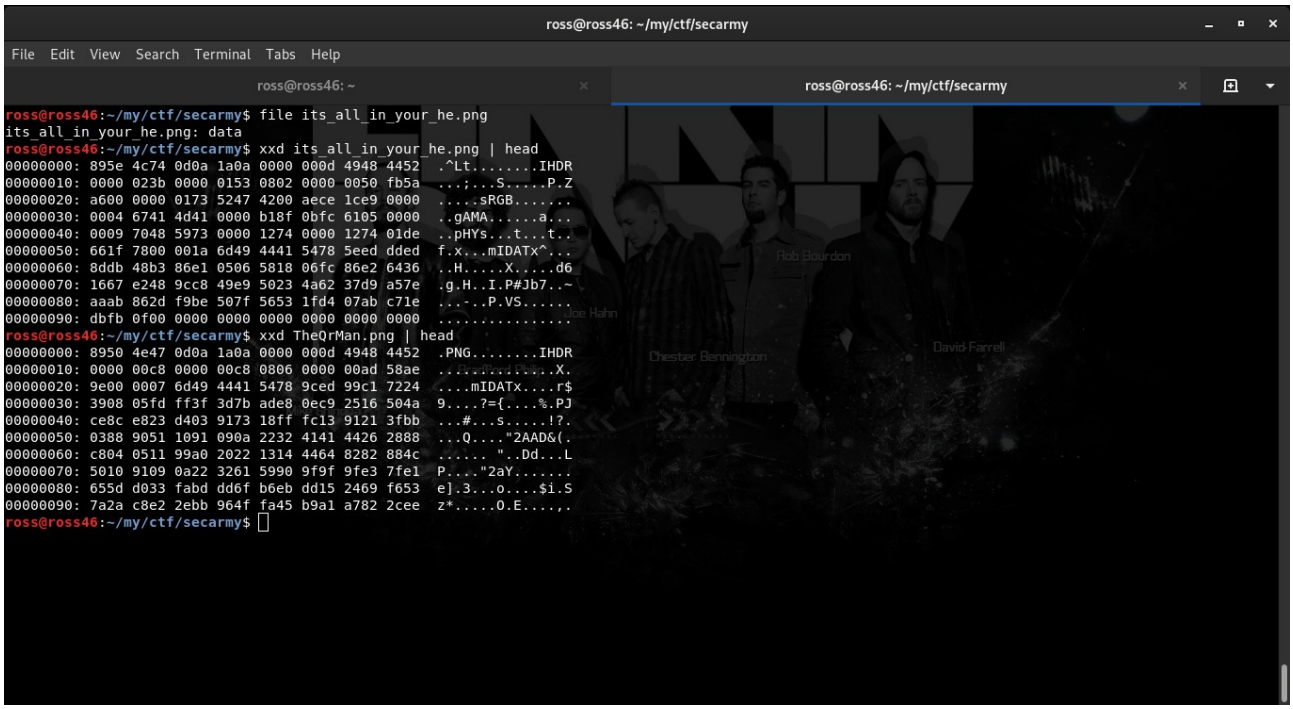
PART II of my writeups

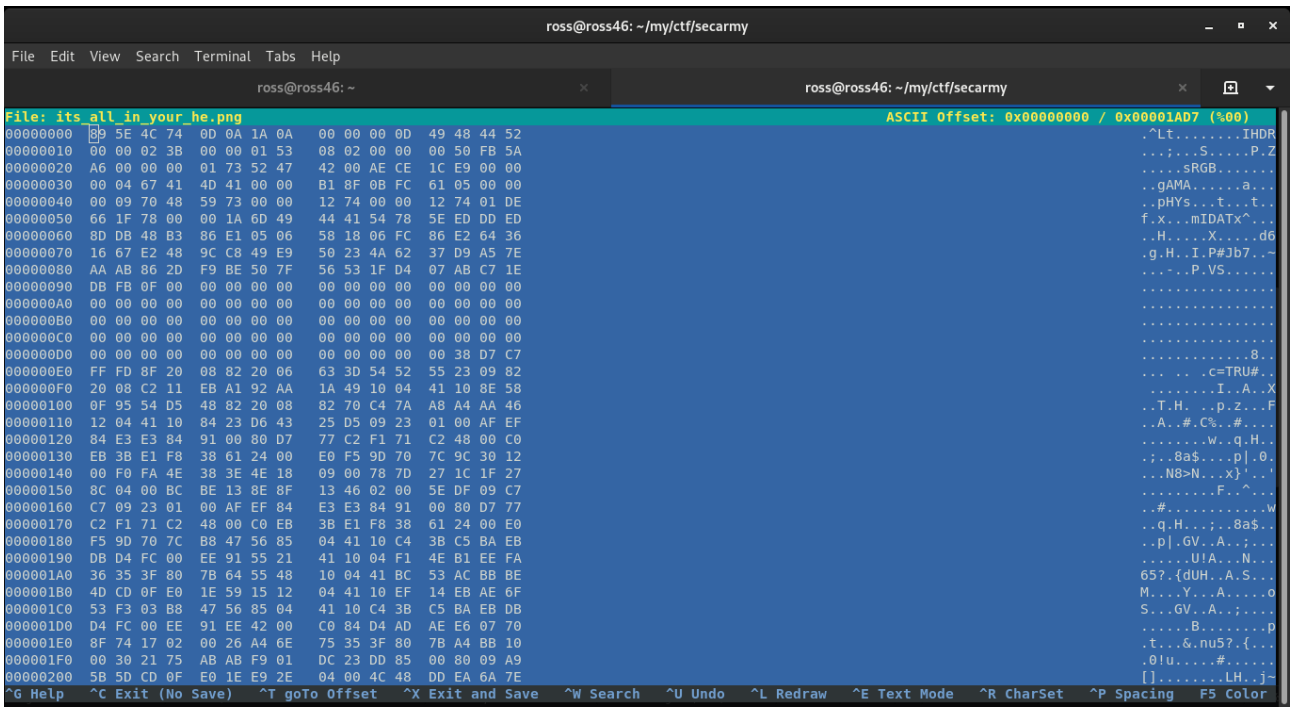
its_all_in_your_head.png:

analyzing the file type we get: its_all_in_your_head.png: data

we check the magic bytes of it as its extension is .png

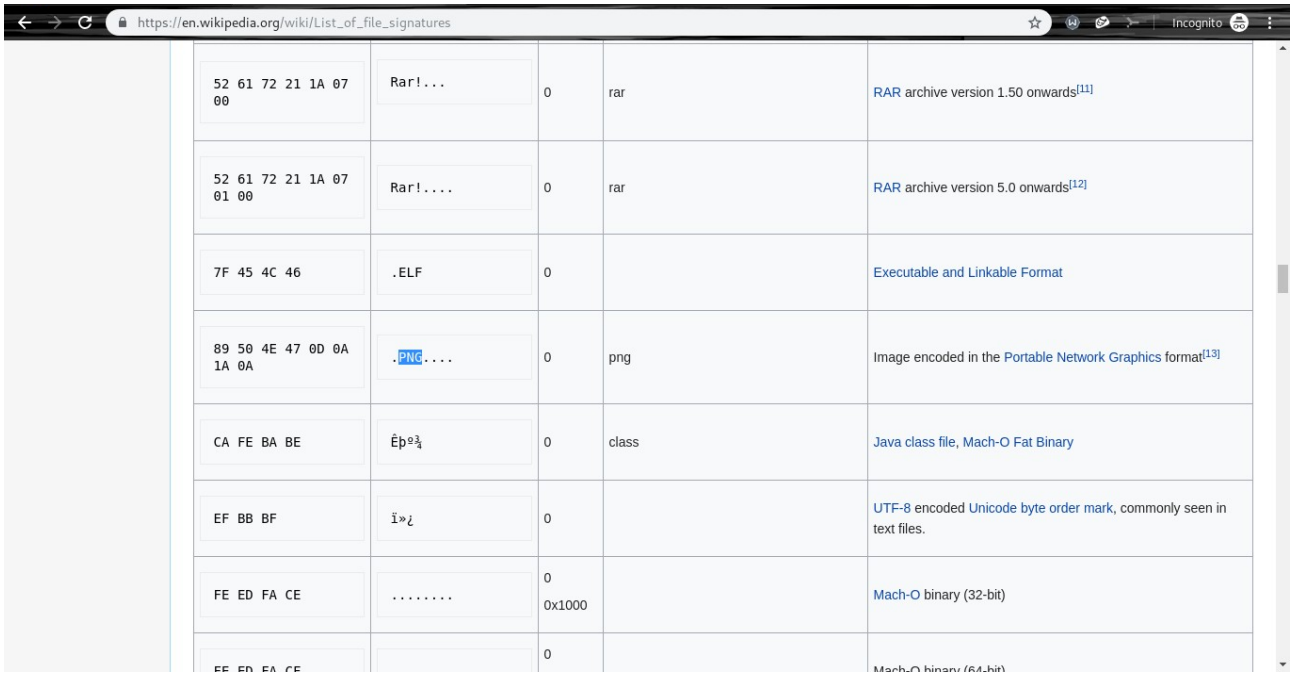


Comparing magic bytes of a proper PNG and its_all_in_your_head.png you can see there is a slight variation in the magic bytes.



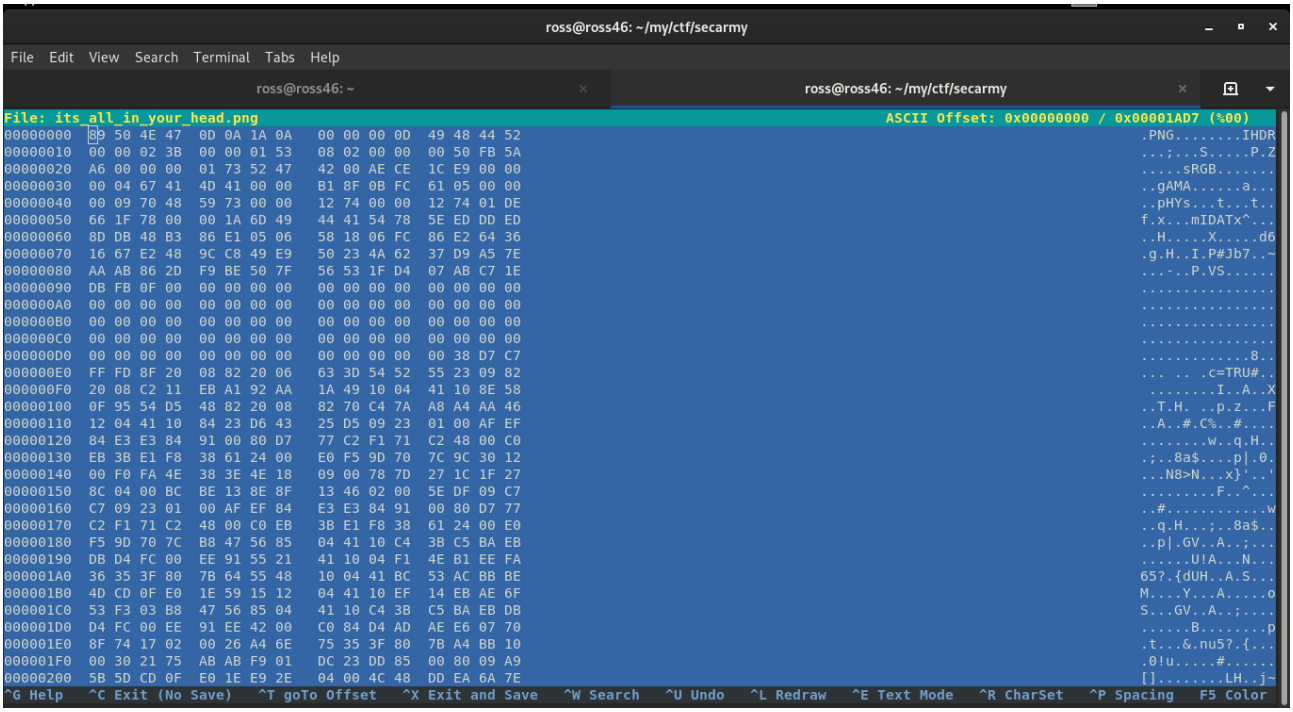
We open the its_all_in_your_head.png with hexeditor to edit its magic byte.
Hexeditor -b its_all_in_your_head.png and change the bytes
89 5E 4C 74 > 89 50 4E 47

How we got the magic bytes:
check the link and image
https://en.wikipedia.org/wiki/List_of_file_signatures

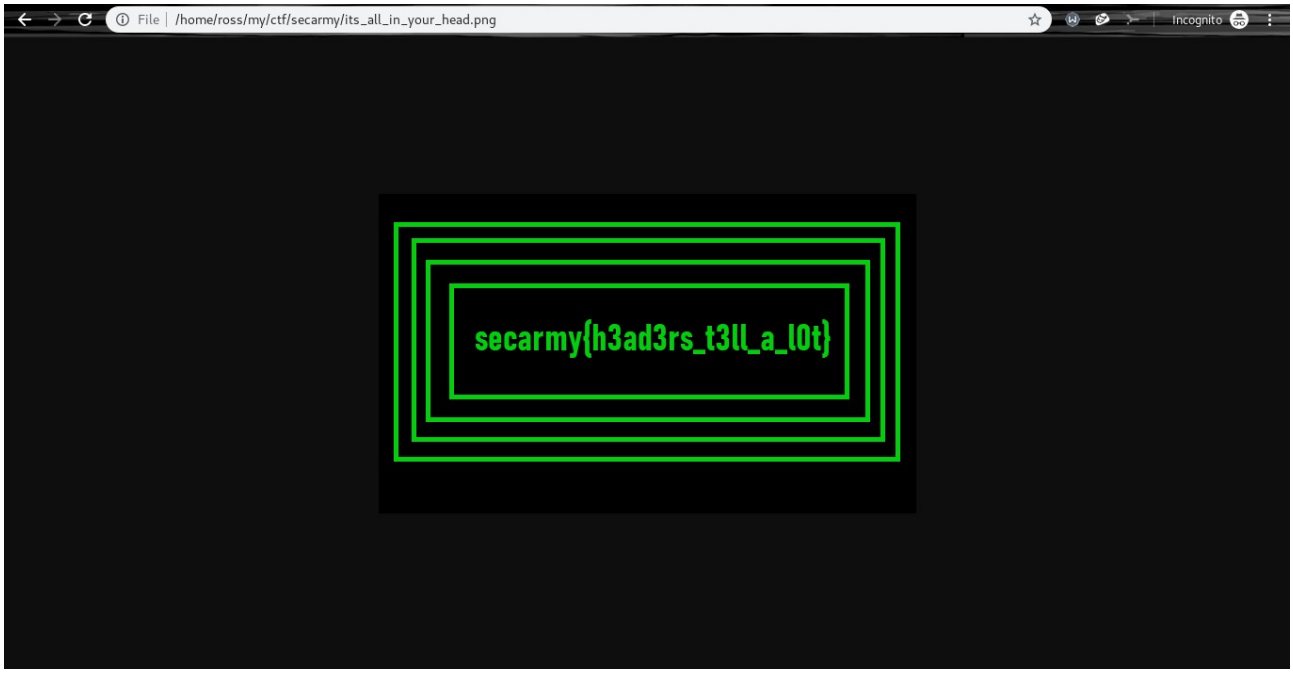


52 61 72 21 1A 07 00	Rar!...	0	rar	RAR archive version 1.50 onwards[11]
52 61 72 21 1A 07 01 00	Rar!....	0	rar	RAR archive version 5.0 onwards[12]
7F 45 4C 46	.ELF	0		Executable and Linkable Format
89 50 4E 47 0D 0A 1A 0A	.PNG....	0	png	Image encoded in the Portable Network Graphics format[13]
CA FE BA BE	Êþº¾	0	class	Java class file, Mach-O Fat Binary
EF BB BF	ï»¿	0		UTF-8 encoded Unicode byte order mark, commonly seen in text files.
FE ED FA CE	.....	0 0x1000		Mach-O binary (32-bit)
FE ED FA CF		0		Mach-O binary (64-bit)

so the proper format it



now when we type :
file its_all_in_your_head.png
We get: its_all_in_your_head.png: PNG image data, 571 x 339, 8-bit/color RGB, non-interlaced
Open it and you get the flag



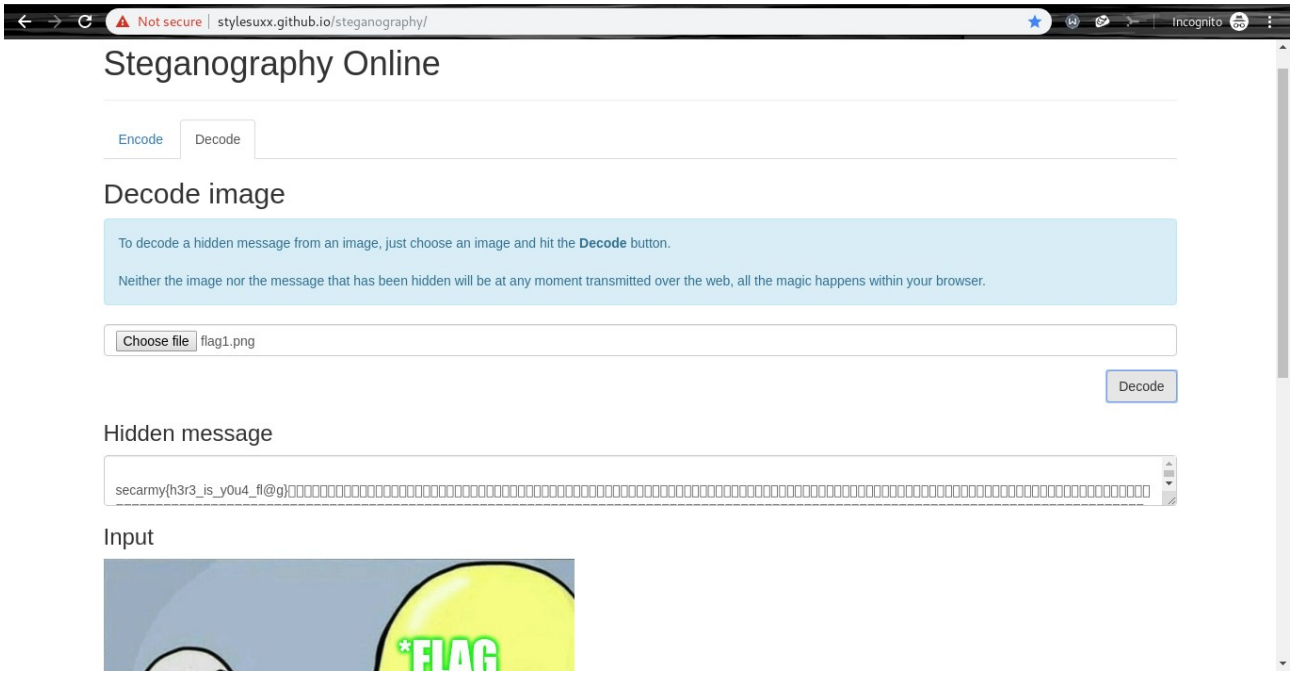
Flag: secarmy{h3ad3rs_t3ell_a_l0t}

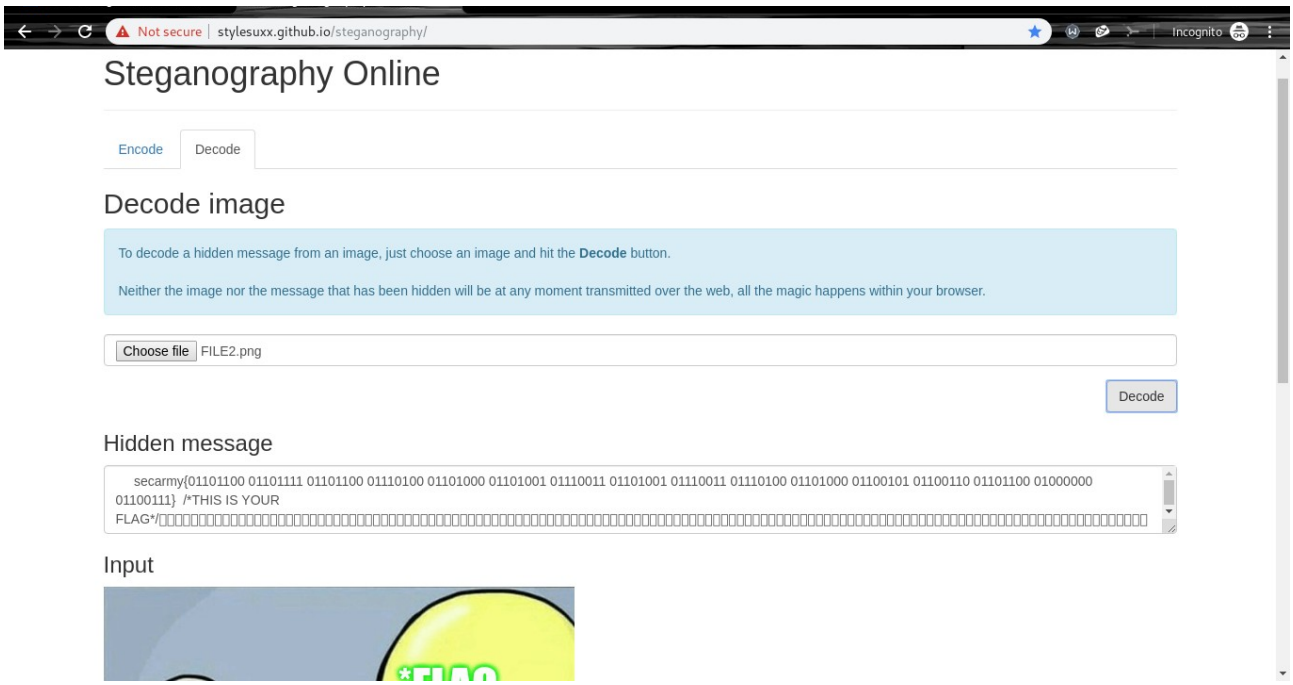
The_Confusion:

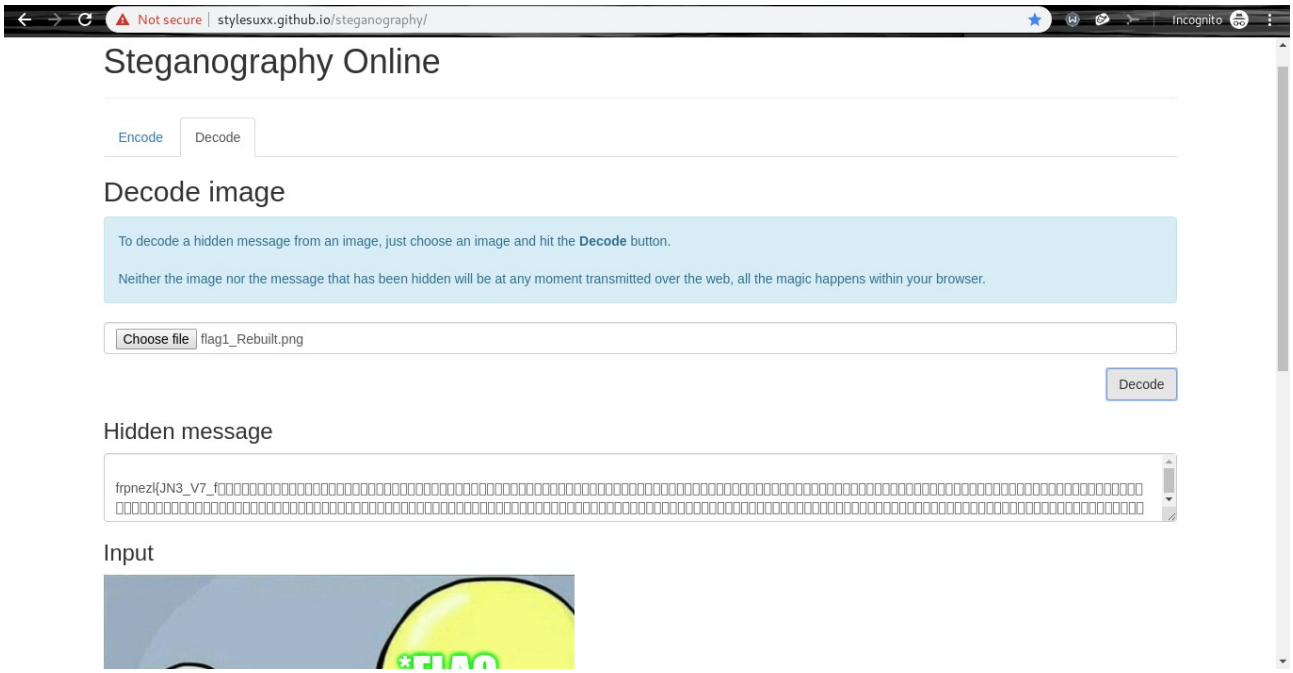
Unzip the flags.zip we get 4 images:
FLAG3.png
flag1_Rebuilt.png
flag1.png
FILE2.png

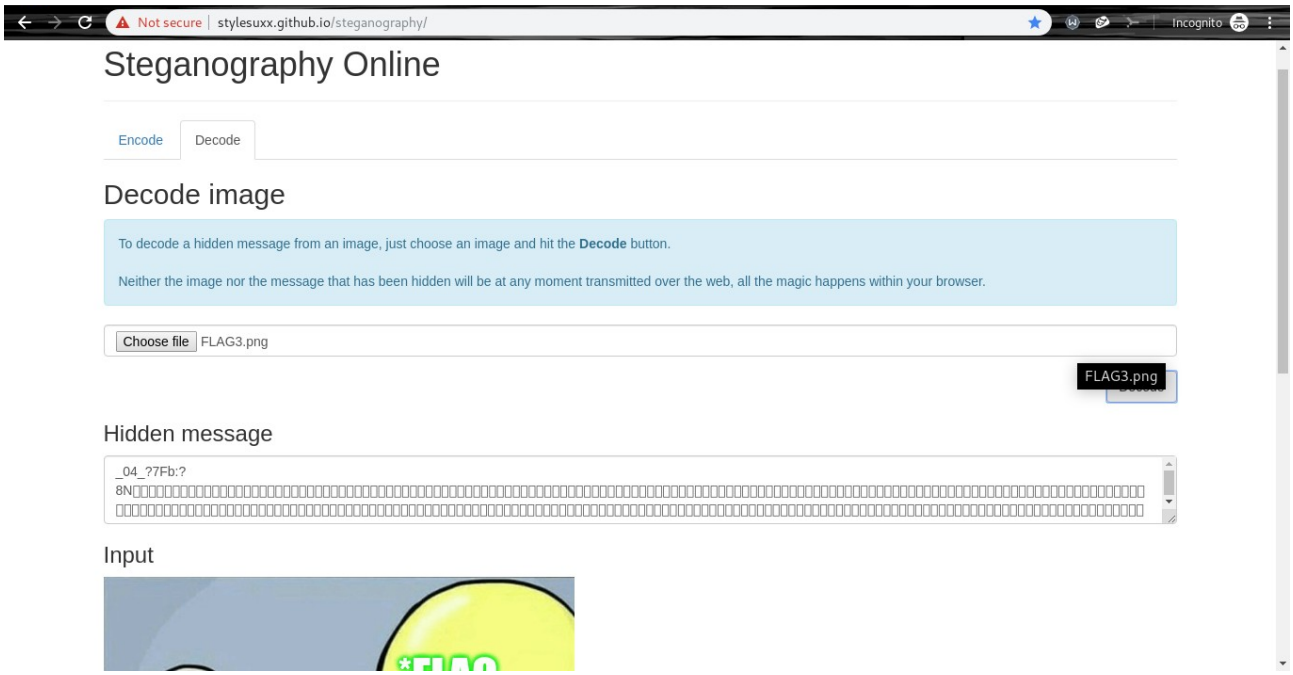
once again we use the help of our faithful steganography tool:
http://stylesuxx.github.io/steganography/

We open all the 4 images:









We get 4 strings:
secarmy{h3r3_is_y0u4_fl@g}
secarmy{01101100 01101111 01101100 01110100 01101000 01101001 01110011 01101001 01110011 01110100 01101000 01100101 01100110 01101100 01000000 01100111}
frpnezl{JN3_V7_f
_04_?7Fb:?8N

The first one is obviously ruled out cause it is the flag format, mentioned in welcome

second one also is wrong

now the suspicious strings :
frpnezl{JN3_V7_f
_04_?7Fb:?8N

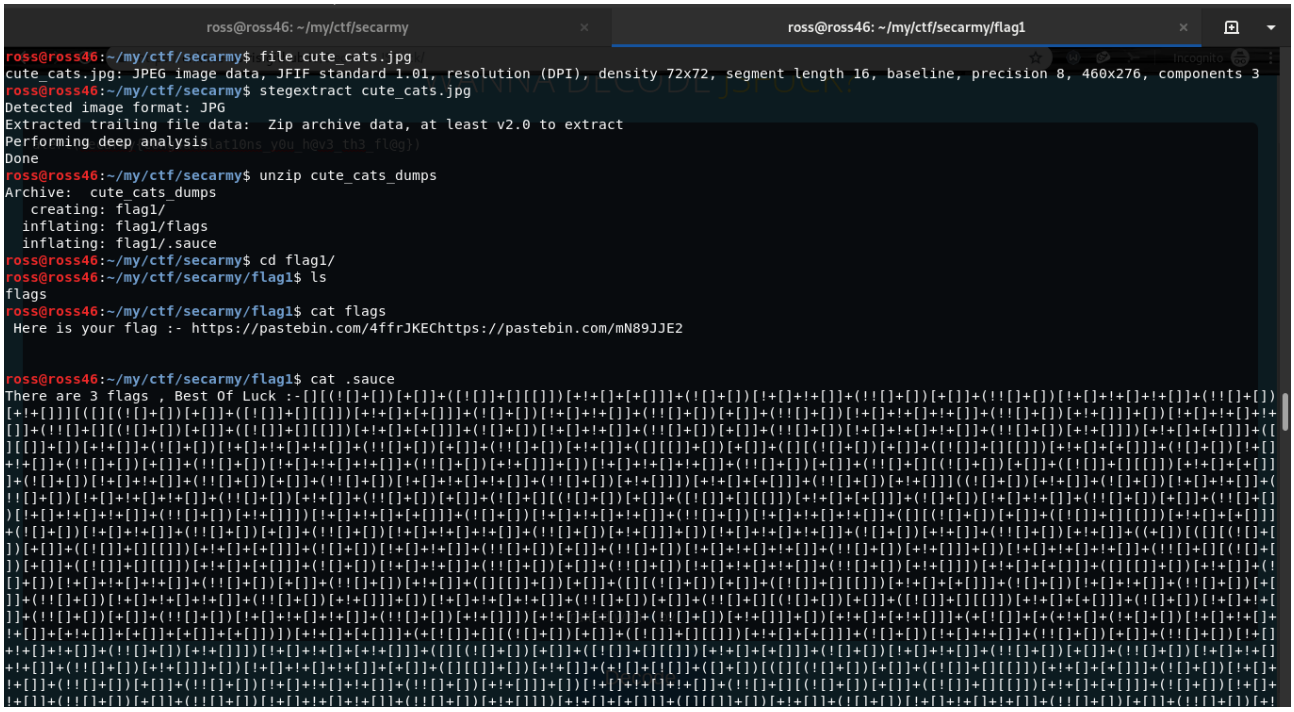
Lets try ROT

Site: dcode.fr
Why I prefer this is?, you have brute-force option which saves a lot of time
it performs ROT from 0-47 including special characters at once and has a lot of options

when we try to ROT together we get weird strings
so lets try individually :
first part brute force we get: “secarmy{WA3_I7_s” from ROT-13
second part we get: “0_c0nfu3ing}” from ROT-47
combining it together
Flag: secarmy{WA3_I7_s0_c0nfu3ing}

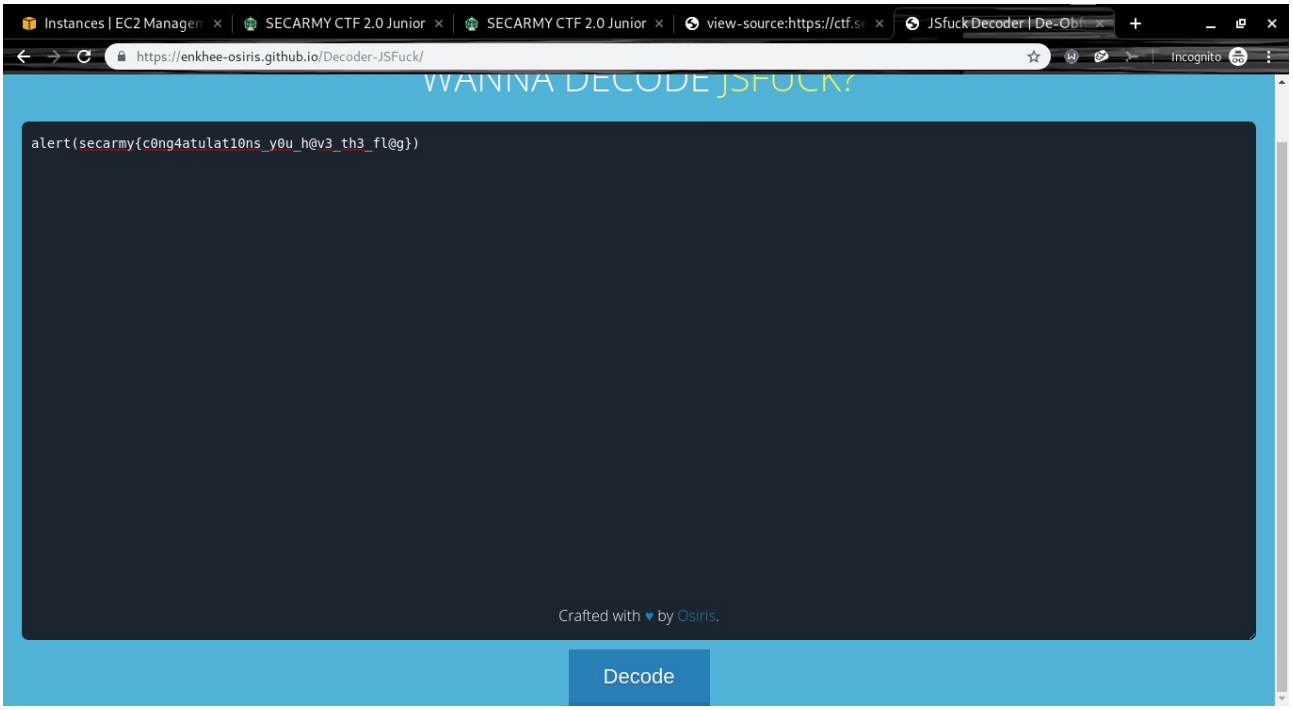
Save them:
We get a zip file called cute_cats.zip Unzipping it we get a .jpg image named cute_cats.jpg
Using stegextract we find a zip file called cute_cats_dumps.
It consists of directory flag1
moving into that directory we find two files flags and a hidden file .sauce

opening flags we get:
Here is your flag :- https://pastebin.com/4ffrJKEChttps://pastebin.com/mN89JJE2 which is a hoax
so we open .sauce file



we get a jsfuck code, now we gotta decode
link: https://enkhee-osiris.github.io/Decoder-JSFuck/

pasting the jsfuck code and decoding we get:
alert(secarmy{c0ng4atulat10ns_y0u_h@v3_th3_fl@g})



Flag: secarmy{c0ng4atulat10ns_y0u_h@v3_th3_fl@g}

Thats the end of forensics

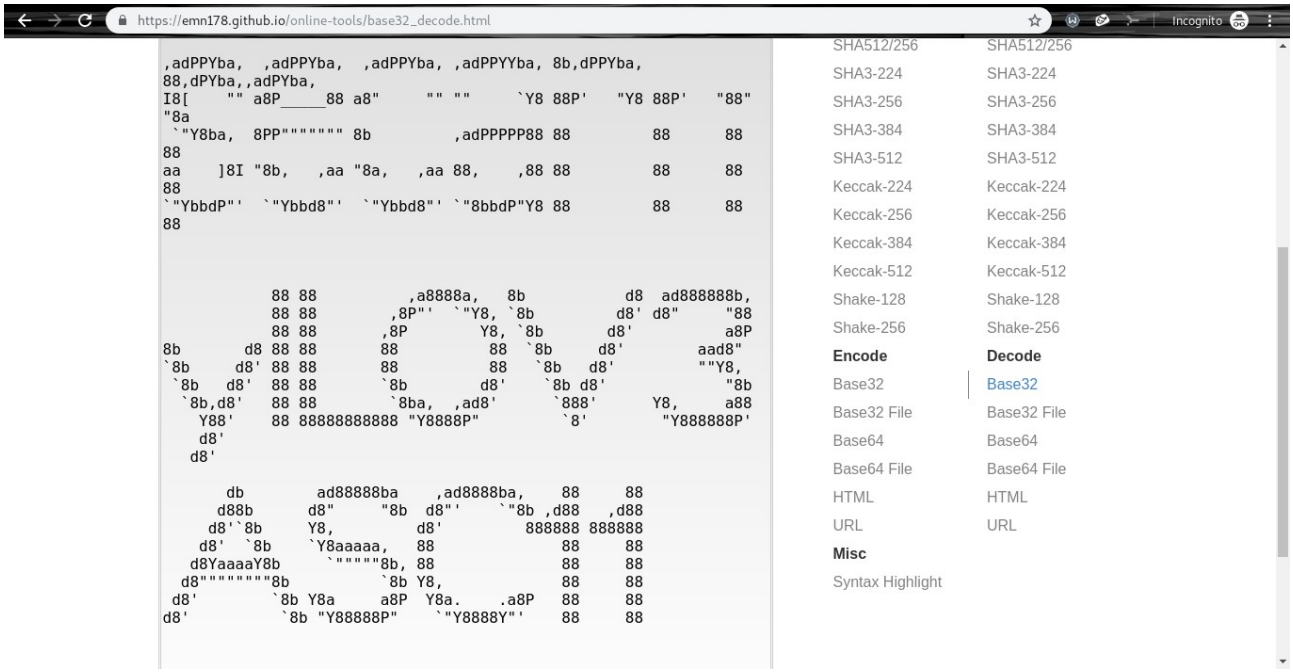
MISC:

Travel:
Question:
Well , I need to discover the path the packet takes . ,
Answer
secarmy{traceroute}

Get_Me:
We have a hint: BTW I LOVE ASCII
Unzip helpmetosolvezip.zip, we get flag2.txt
opening it we get a base32 string, how is it base32? cuz of "====" at the end

Link:https://emn178.github.io/online-tools/base32_decode.html

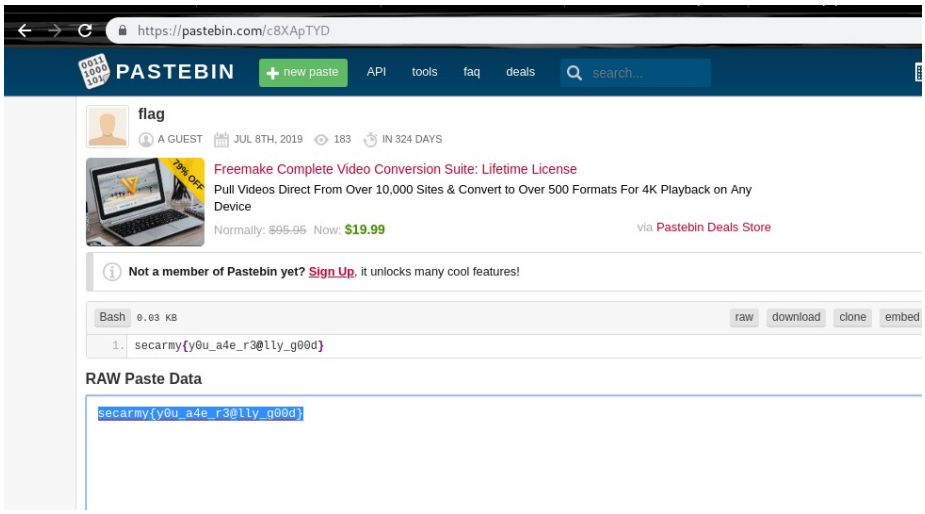
Use the link to decode, we get:
secarmyIL0V3ASC11
note that it is not letter "O" but Zero "0"



so we get Flag: secarmy{I_L0V3_ASC11}

THE PASTE:

we get “c8XApTYD”, it says i love bash and pasted and internet will help you.
So a hint it is a paste bin link:
https://pastebin.com/c8XApTYD



Flag:secarmy{y0u_a4e_r3@lly_g00d}

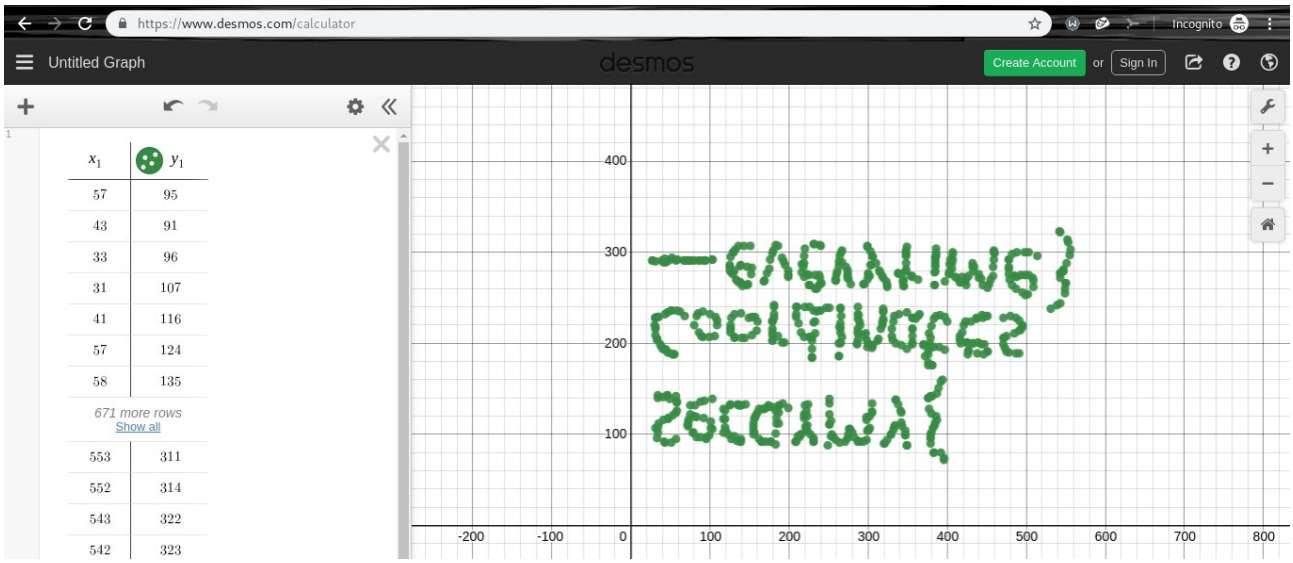
Constructor Ninja:
opening the cpp code we see no expicitly defined constructor but for:
    Student s1 = Student(1, "John", 89);

answer:
secarmy{parameterizedconstructor}

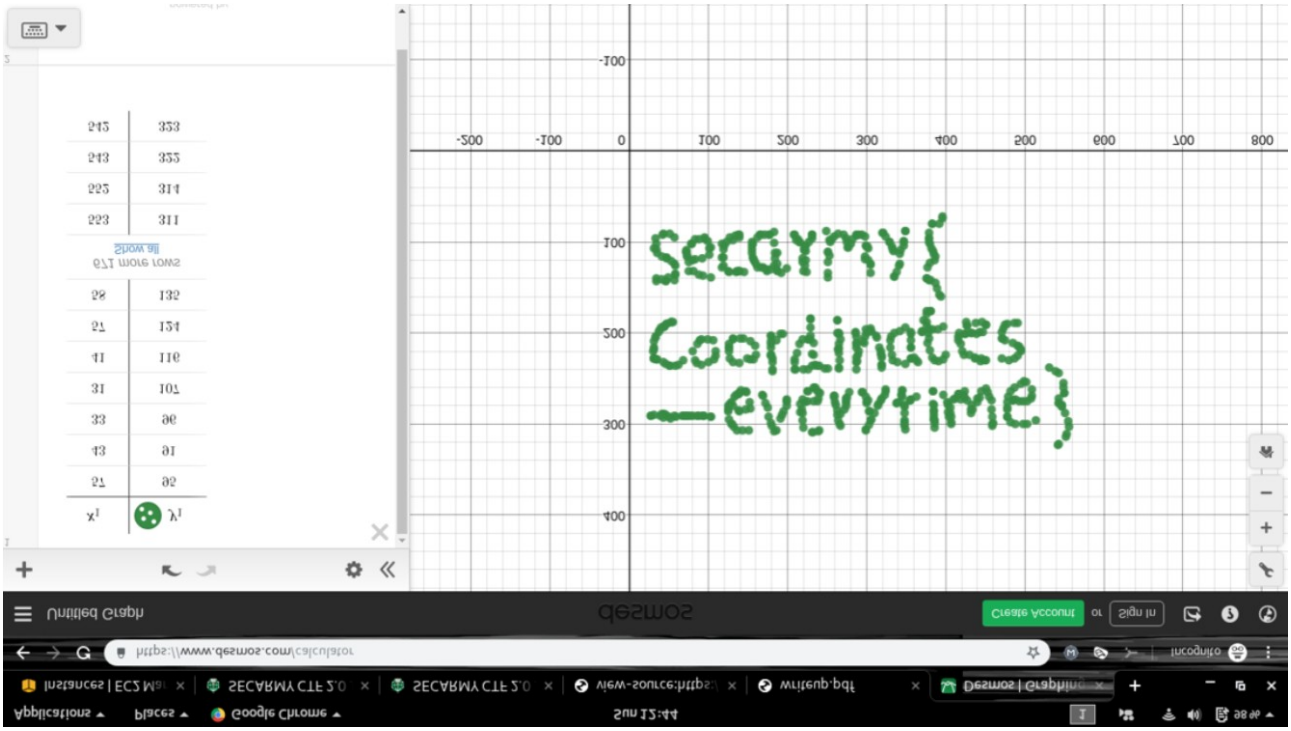
Poor prisoner:
so we get a file radio_capture.txt which consists of co ordinates:

we need to plot those, we can use:
https://www.desmos.com/calculator

copy the entire co ordinates and paste it.
We get the key in reverse order



Mirroring the image and inverting it we get the flag:



Flag: secarmy{coordinates-everytime}.
NOTE: it is a “-” and not a “_”

WEB:

Prizes:
link: https://ctf.sec.army/prizes

view the source code:

We have a base64 string: c2VjYXJteXtzMHVyYzNfaTVfbjNjZXM1YXJ5fQo=

decode it: secarmy{s0urc3_i5_n3ces5ary}

Flag: secarmy{s0urc3_i5_n3ces5ary}

Silly Mongolian 2.0:
The most eazy and loved challenge:

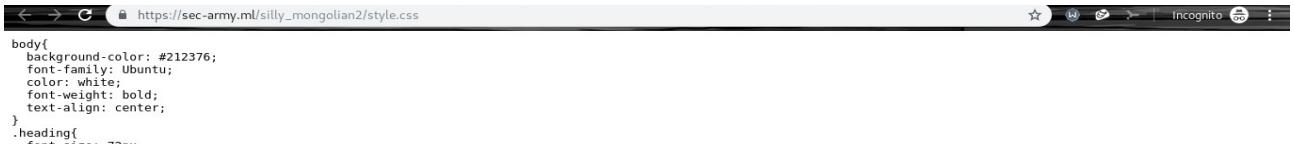
view source code:
first part of flag

Second part is in index.js


```
function returnb() {
  window.open("index.html");
}
const utf8bytes = code -> {
  code >>>= 0;
  let array;
  if(code < 0x0080) {
    array = [
      code
    ];
  } else if(code < 0x0800) {
    array = [
      0xc0 | ((code & 0x780) >> 5),
      0x80 |  (code & 0x07f)
    ];
  } else if(code < 0x10000) {
    array = [
      0xe0 | ((code & 0xf000) >> 12),
      0x80 | ((code & 0x0fc0) >> 6),
      0x80 |  (code & 0x003f)
    ];
  } else if(code < 0x200000) {
    array = [
      0xf0 | ((code & 0x1c0000) >> 18),
      0x80 | ((code & 0x03f000) >> 12),
      0x80 | ((code & 0x000fc0) >> 6),
      0x80 |  (code & 0x00003f)
    ];
  } else {
    throw "undefined code: U+" + code.toString(16);
  }
  return new Uint8Array(array);
  console.log(flag);
  /*Here's the second part of flag :
  _1s_th1s_
  */
};
```

Third part is in process.js:


```
function login() {
  window.open("error404.html");
  /*Here's the third part of flag :
  m0ng0li@n_$uch
  */
}
```



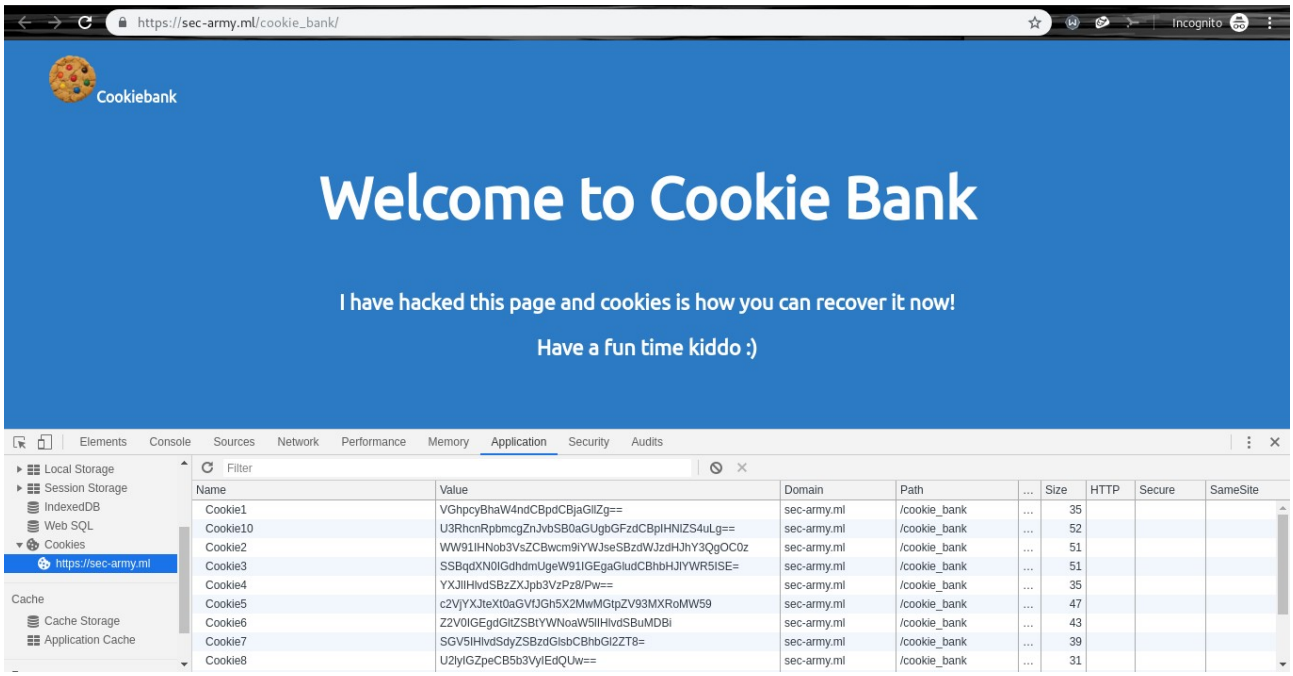

```
body{
  background-color: #212376;
  font-family: Ubuntu;
  color: white;
  font-weight: bold;
  text-align: center;
}
.heading{
  font-size: 72px;
}

.pass{
  margin-top: 20px;
}
/*Here's the fourth part of flag :
_@_f00l}
*/
input{
  height: 40px;
  width: 320px;
  border-radius: 18px;
  border: none;
  text-align: center;
  background-color: #181818;
  color: white;
  font-family: Ubuntu;
}
.btn{
  margin-top: 20px;
  height: 30px;
  width: 120px;
  border: none;
  border-radius: 18px;
  font-family: Ubuntu;
}
nav{
  padding-top: 30px;
}
```

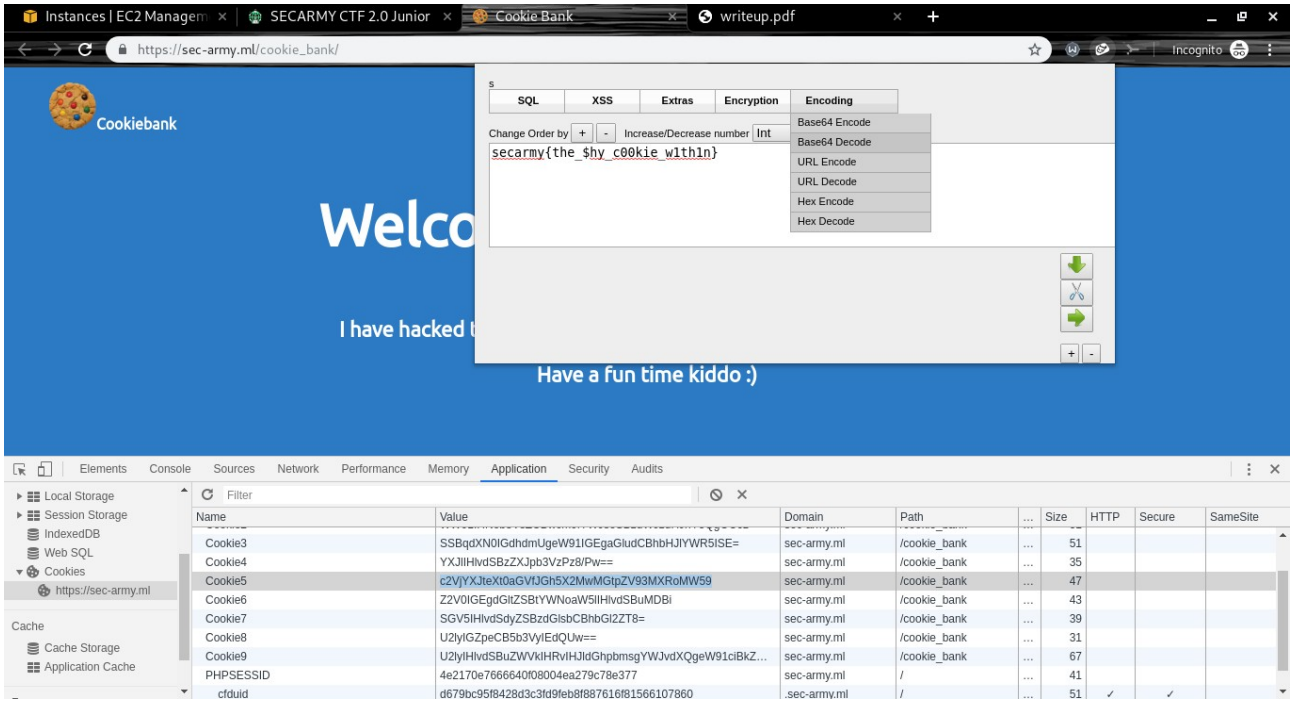
Fourth part is in css:
the four parts are:
secarmy{why
_1s_th1s_
m0ng0li@n_$uch
_@_f00l}
Flag: secarmy{why_1s_th1s_m0ng0li@n_$uch_@_f00l}

Cookie bank:

View the cookies in console: application:

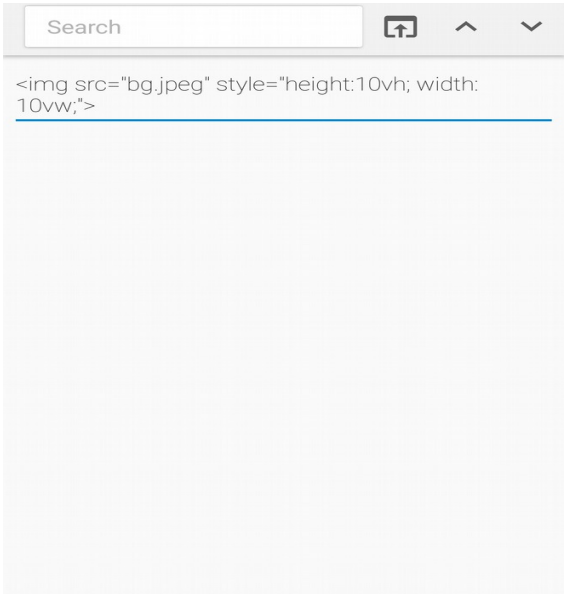


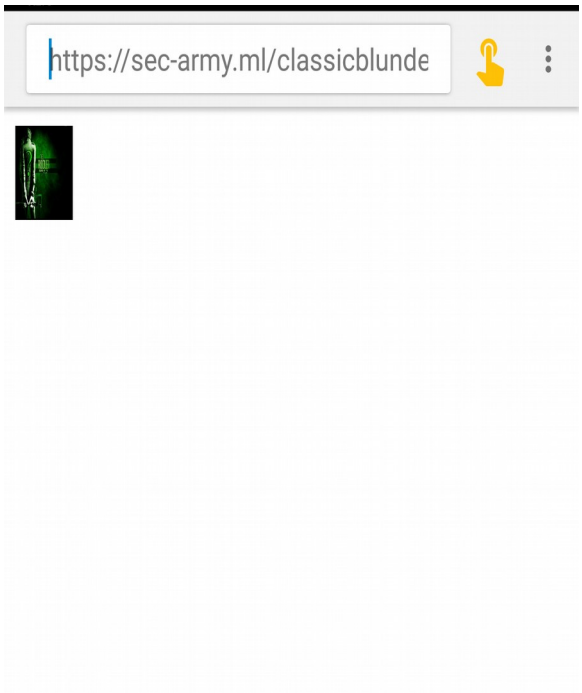
Cookie 5:c2VjYXJteXt0aGVfJGh5X2MwMGtpZV93MXRoMW59
base64 decoded: secarmy{the_$hy_c00kie_w1th1n}

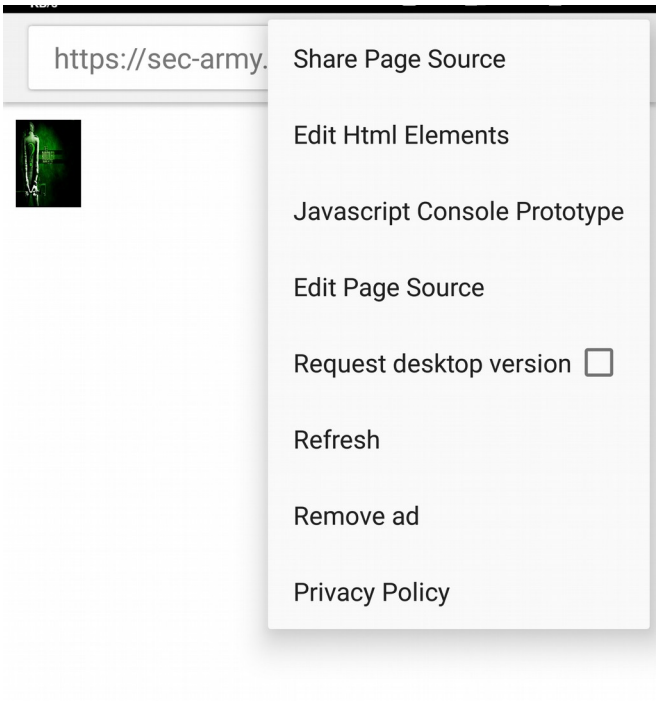


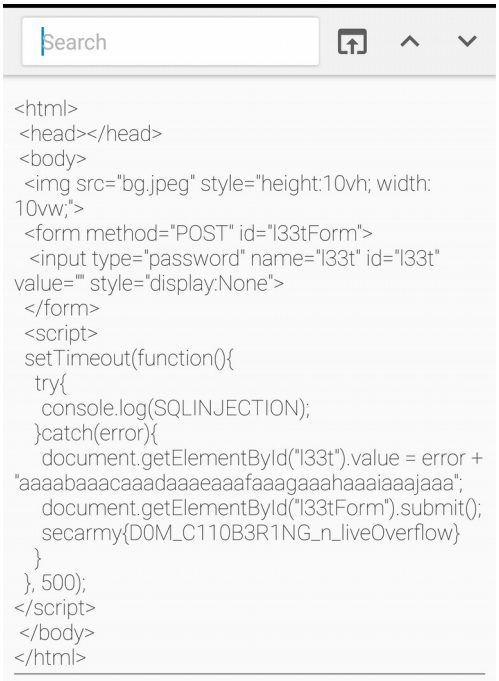
Classic blunder:
Honestly i have no idea what happened and why this happend, i was travelling and dint have access to my laptop, so tried it in my phone using an app, it was total fluke the way i got this.
I dont have any explanation at the time of writing this.
link:https://apkpure.com/inspect-and-edit-html-live/web.dassem.livewebsiteeditorfree

I present the screenshots from my phone:









What i did :
1. Opened the link: http://sec-army.ml/classicblunder
2. opened the inspect element and resized the image from 100vh 100wh to 10vh 10wh
3. opened the source code, and i have the Flag: secarmy{D0M_C110B3R1NG_n_liveOverflow}

CRYPTOGRAPHY

Exchange:

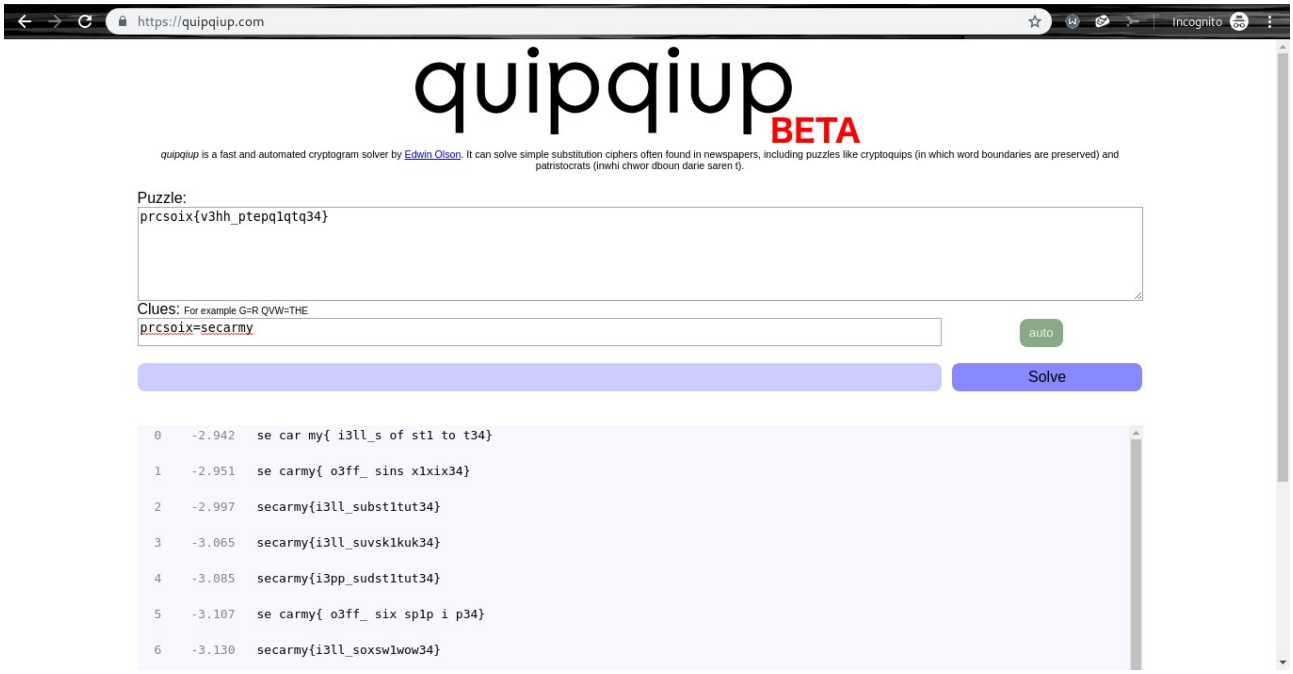
we get a text.zip file, unziping the file we get a imp_f1le.

Well I only know this :-prcsoix{v3hh_ptepq1qtq34}

prcsoix is secarmy

so cipher is quipquip, i had a wild guess for this

link:https://quipqiup.com/



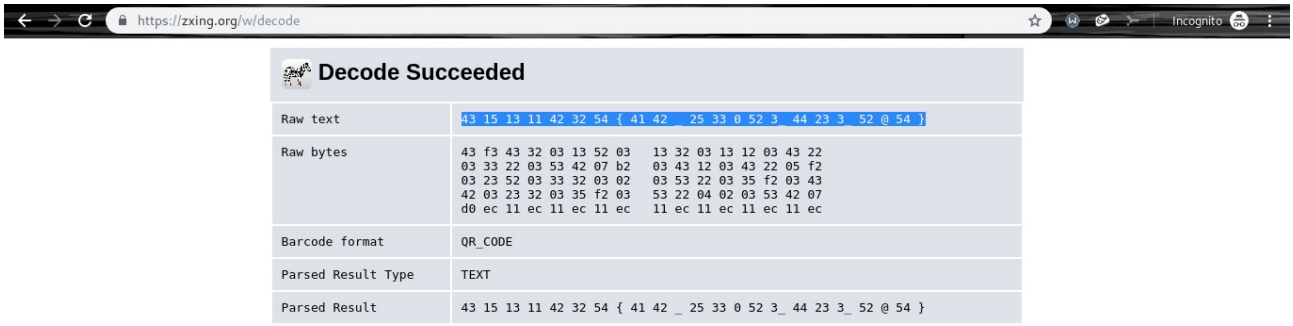
decoding it: secarmy{i3ll_subst1tut34}

Flag:secarmy{w3ll_subst1tut34}

SQUARE:

we get a zip file ThrQrMan.zip having ThrQrMan.png which is a qr code

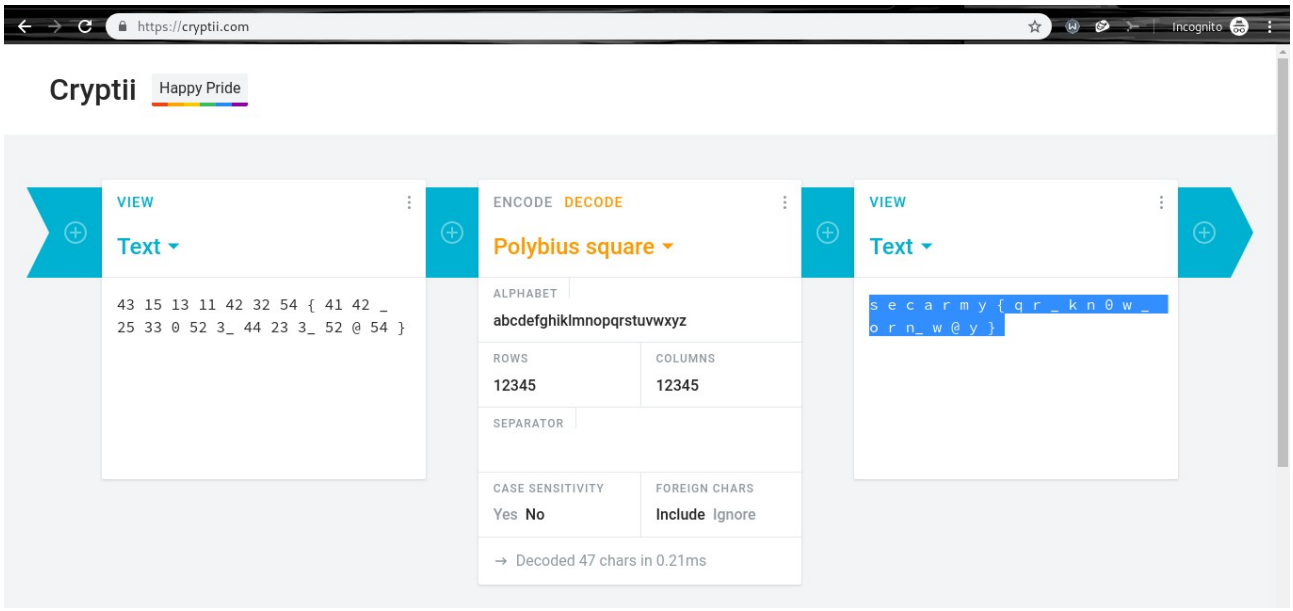
uploading and scanning it we get:



```
43 15 13 11 42 32 54 { 41 42 _ 25 33 0 52 3_ 44 23 3_ 52 @ 54 }
```

Using polybius cipher we can decode it:

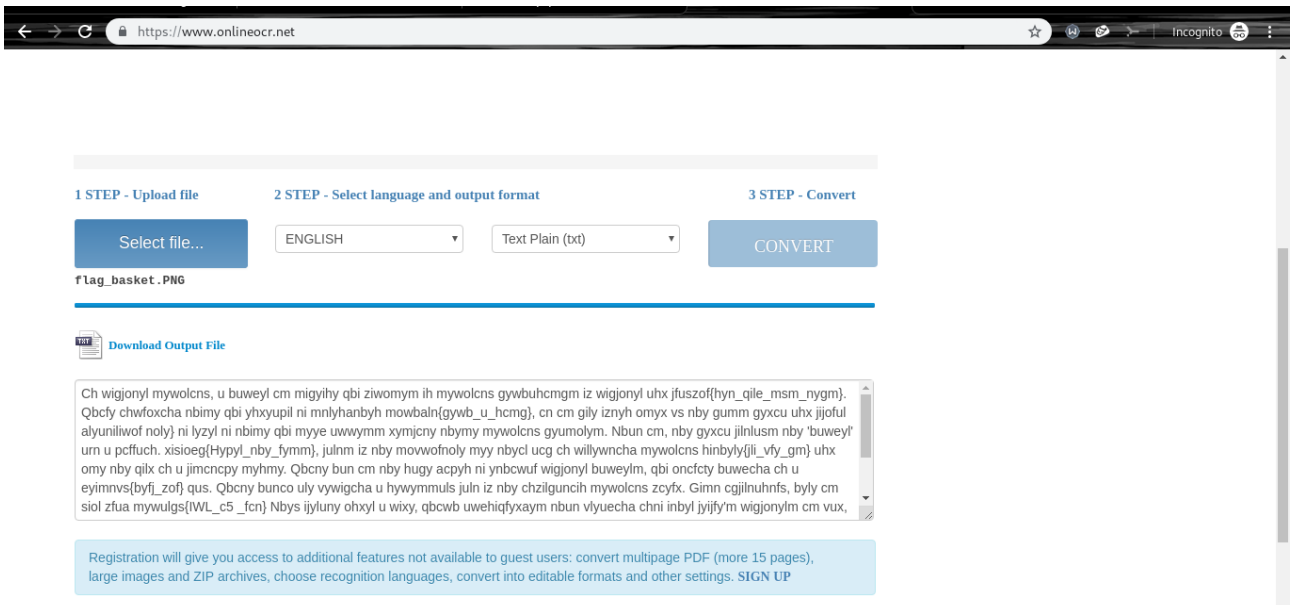
link: cryptii.com:



Flag: [secarmy{qr_kn0w_orn_w@y}]

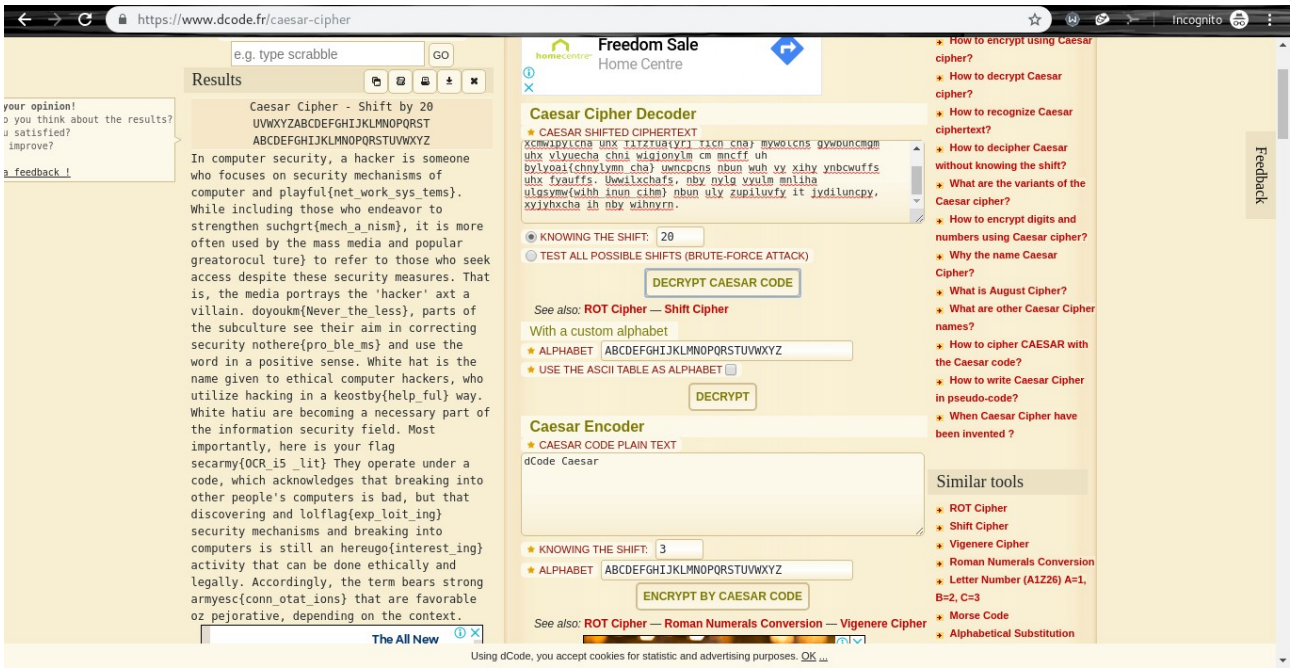
Flag Basket:

we have an image flag_basket.png with text in. We use OCR to extract the characters



It is cipher text, which is bruteforced in dcode.fr for substitution cipher,

it is a shift of 20 alphabets and we get our flag



Flag: secarmy{OCR_i5 _lit}

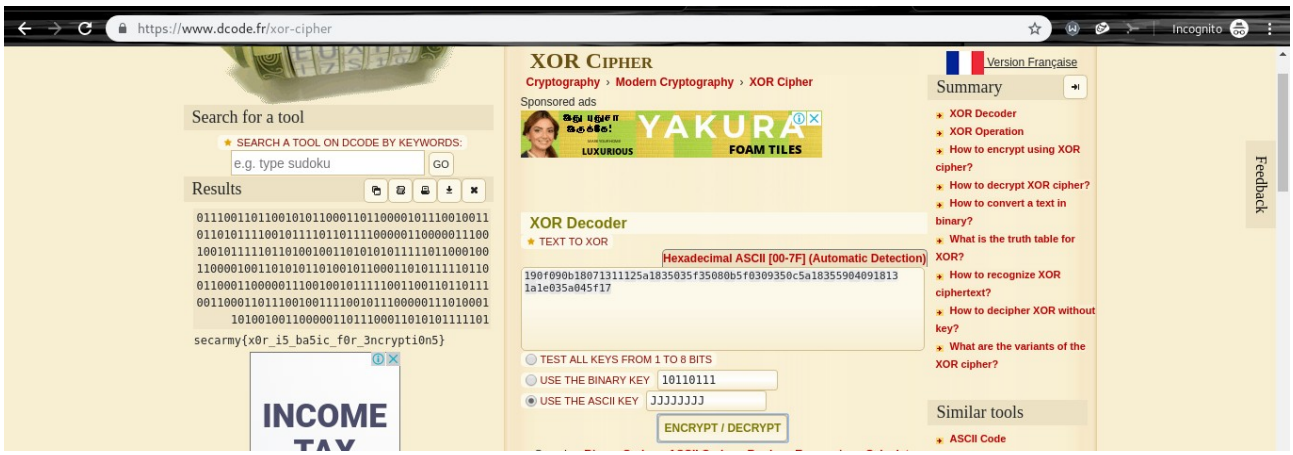
Exclusive OR Non-Exclusive:

We get a wierd string:
190f090b18071311125a1835035f35080b5f0309350c5a183559040918131a1e035a045f17

Name has it XOR,

trying few different keys I did not get anything , but the dcode.fr site gave me an hint.

“USE THE BINARY KEY” has 0 and 1 so I just thought of using iiiiiiii (i*8) in place of 1, did not work out but i tried jjjjjjjj (j*8) i got the key, Flag: secarmy{x0r_i5_ba5ic_f0r_3ncrypti0n5}



The 3ASY:

This challenge was right on your face but u missed it,

use https://cryptii.com and https://v2.cryptii.com and https://dcode.fr for this

we get a string which is combination of number letters and morse code:

25111111251111251111111111251111111111112511111111112511ndaekjaadadaabaaaaaajmadebab.-
-.- .- .- .- -.- .- -.- .- .- -.- .- -.- .- .- -.- .- -.- .- .- .- .- .- .- .- .- -.- -.-

Decoding the number: polybius square

25111111251111251111111111251111111111112511111111112511 =>
kaaakaakaaaaakaaaaaakaaaaka

Decoding the alphabets: trifid cipher

ndaekjaadadaabaaaaaajmadebab => kkkakkaaaaakaaaaaaaakaakkaak

Decoding morse code:

.- -.- .- .- .- -.- .- -.- .- .- -.- .- -.- .- .- -.- .- -.- .- .- .- .- .- .- .- .- -.- -. = >
akaaakakaakakaakakaaaaaaaaakk

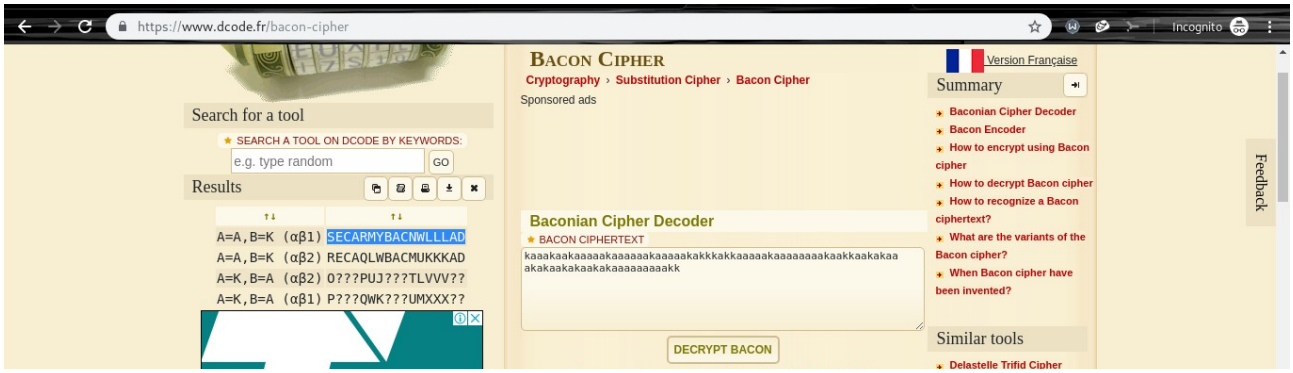
combining all together

kaaakaakaaaaakaaaaaakaaaaakakkkakkaaaaakaaaaaaaakaakkaakakaaakakaakakaakakaaaaaaaaakk

It is bacon cipher, decoding it: SECARMYBACNWLLLAD

Flag: secarmy{bacnwlllad} does not make any sense, but that is the flag (-_-),

I know it hurst you guys spent a lot of time, but this was it.



Rivest Shamir Adelman:

We have 3 files, the encryption file, encrypted image and the private key,

the code used for it:

https://pastebin.com/vwYqhbxP

use this code and run: python2 <filename.py>

you have the decoded image:

Flag:secarmy[RSA_ba51c5tobe_13arn7}

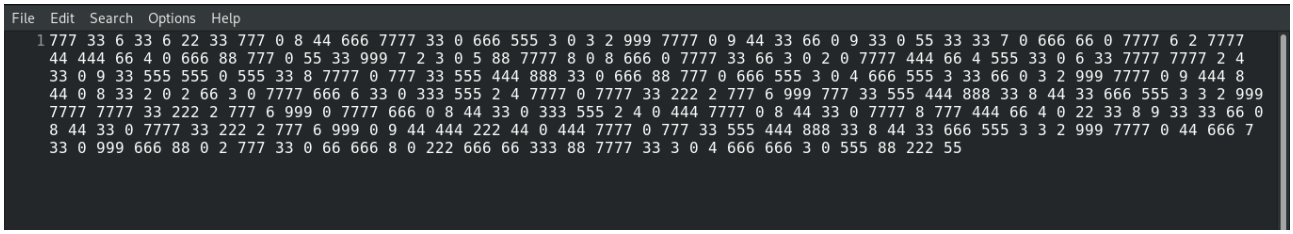
Old Days:

only 90’s kids will remember this XD

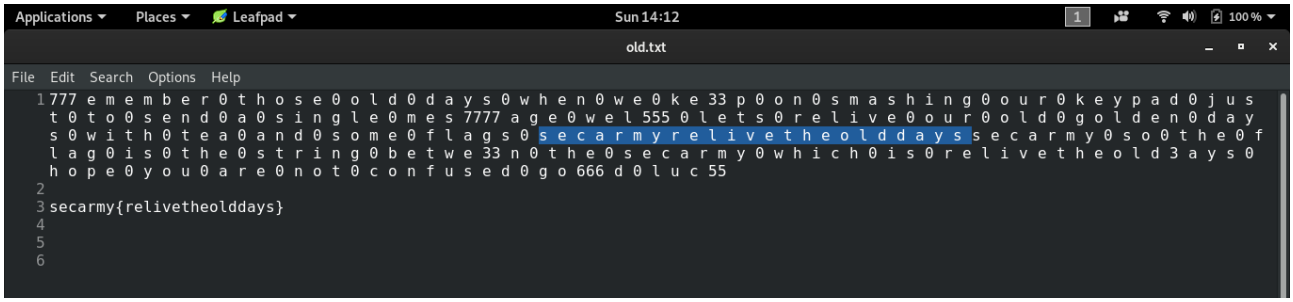
The numbers correspond to the number of times it has to be pressed to get the letters in the phone which was before smartphone era,

The pain of typing long text.... phew.

Well lets solve:



each number represents the alphabet a converted format looks like this:



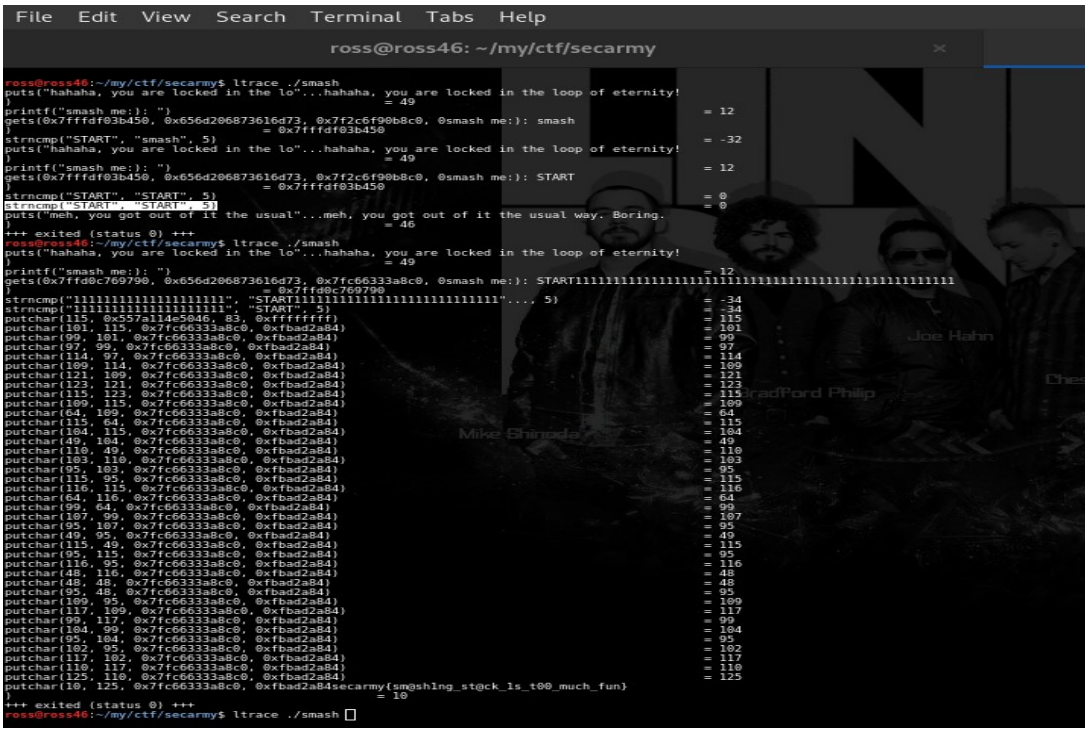
Flag: secarmy{relivetheolddays}

BINARY/REVERSING

Smash it:

classic buffer flow:

enter a very long string, and you are done XD



Flag: secarmy{sm@sh1ng_st@ck_1s_t00_much_fun}

if you run ltrace you can see there is strncmp()

strncmp("START", "START", 5)

It is checking for a limit of 5 chars,

you enter a lengthy string you bypass it and you get the flag.

Backyard COWs:

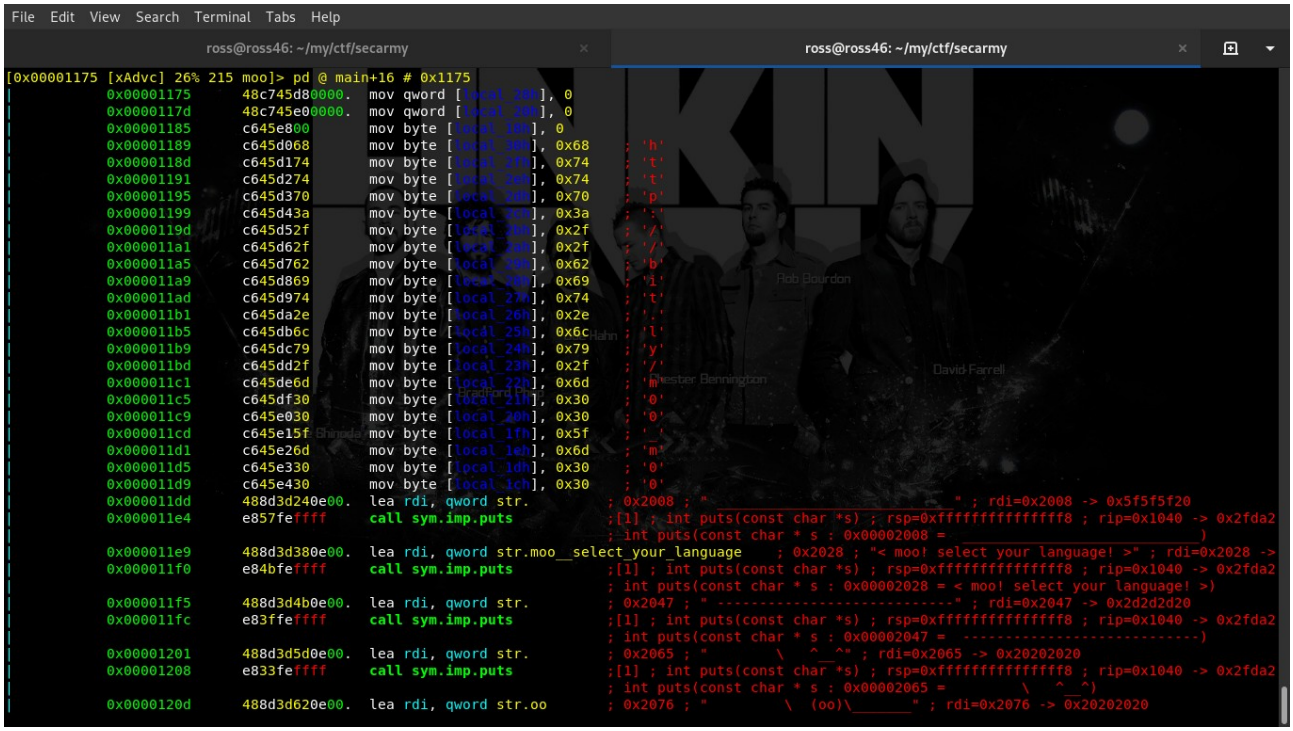
we have a elf file called moo

we use radare2 here, a crazy tool to do so

Flow:

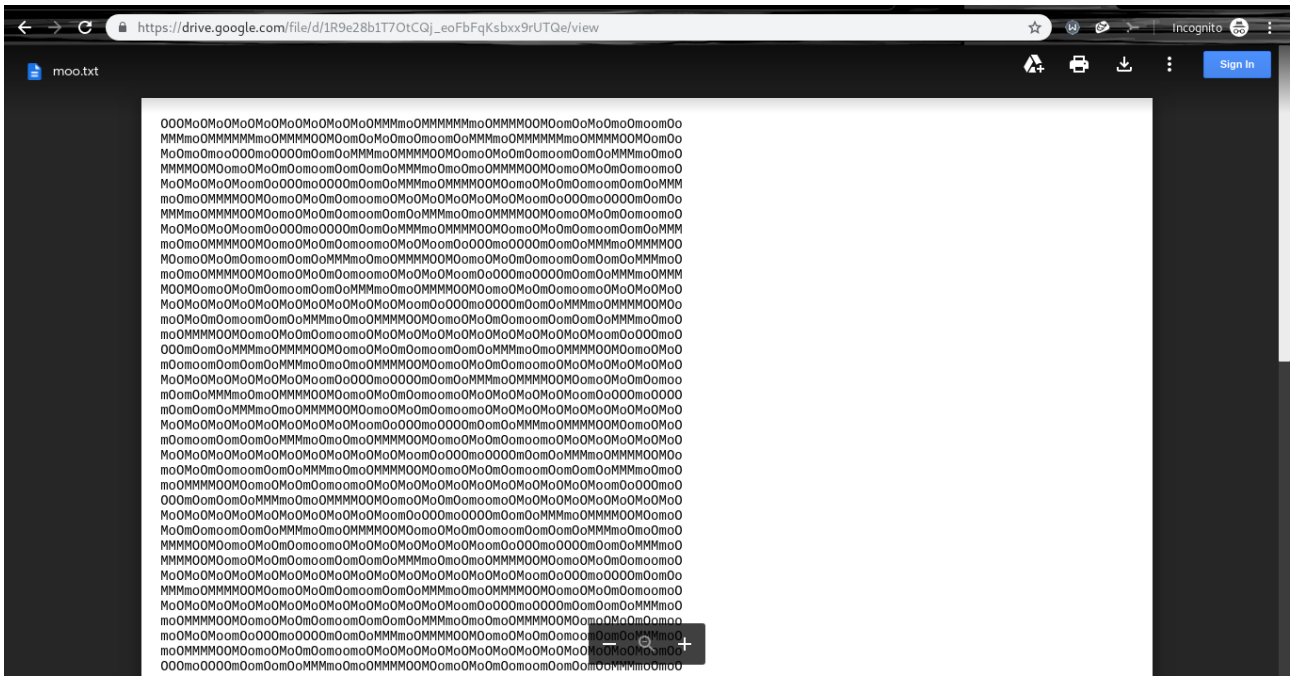
1. r2 moo => load the file
2. aa => to start analyzing the binary
3. s => main to start analyzing from main
4. p => to change the view mode

5. scroll down a bit and you will find a strange chars:



http://bit.ly/m00_m00

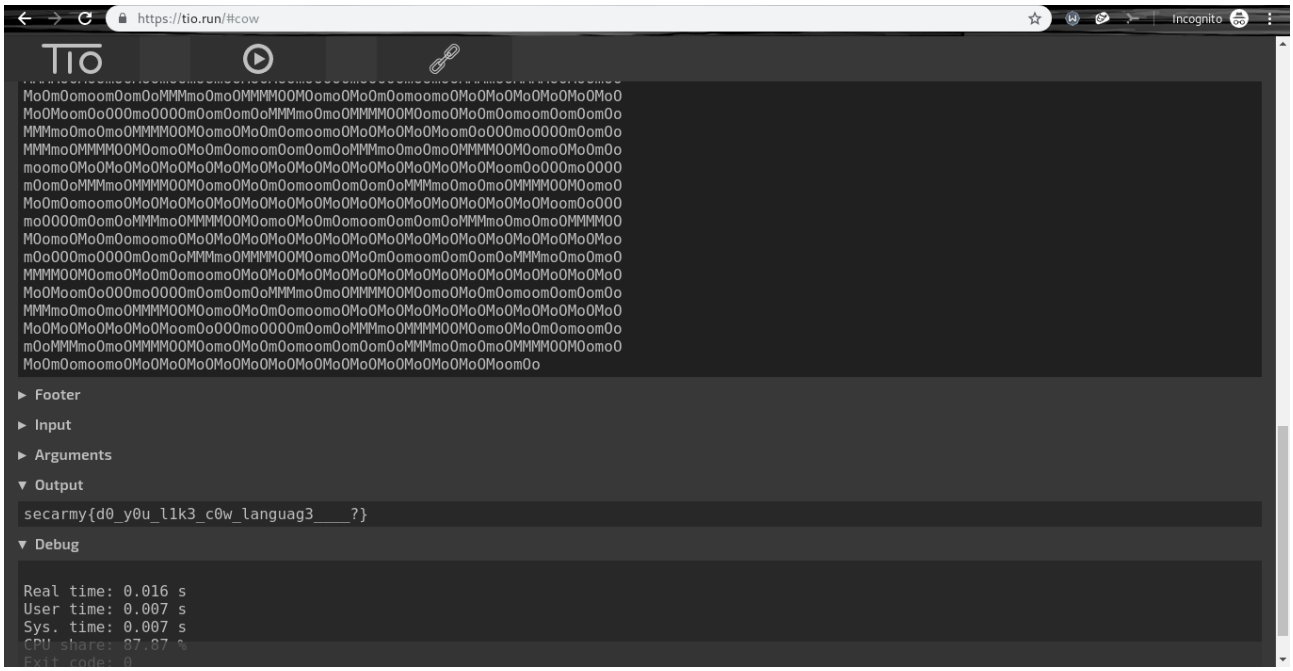
leads you to a google drive containing some text in cow lang



now we need to decode it,

Link: https://tio.run

choose cow and decode it



Flag: `secarmy{d0_y0u_l1k3_c0w_languag3____?}`