Proof of Concept- GPCSSI Buddy 52

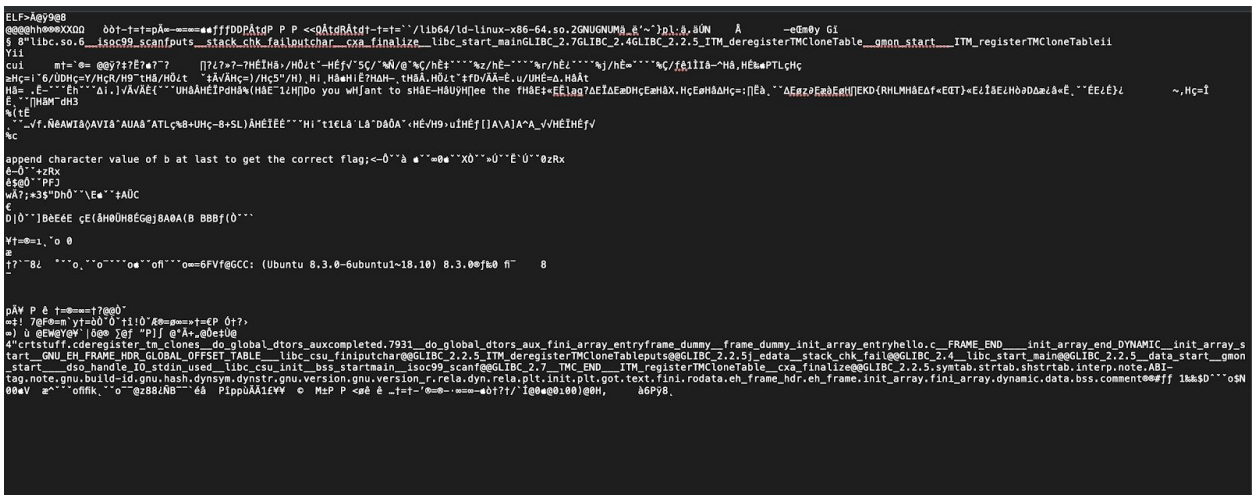
HACKQUEST ROUND 2

Team Members :

Loveneesh Dhir-TEC-253
Parth Sikka TEC-153
KRISHNENDU SAMANTA SCH-046
Divyanshi Sharma Tec -054
Laksh Rawat NTE -066

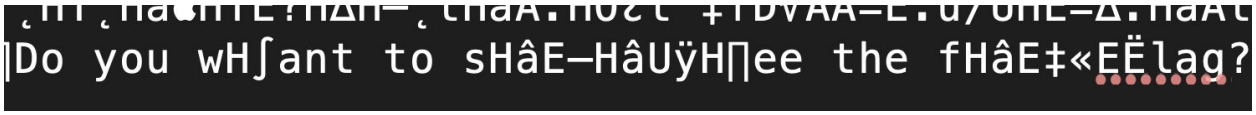
Challenge 1 :

In this challenge, we decided to read the raw file of the challenge file (fooled) and we got the following :



Looking the image above, we could find it clearly written : “append character value of b at last to get the correct flag”

However, on closely glancing at the text, we could also find it’s written :



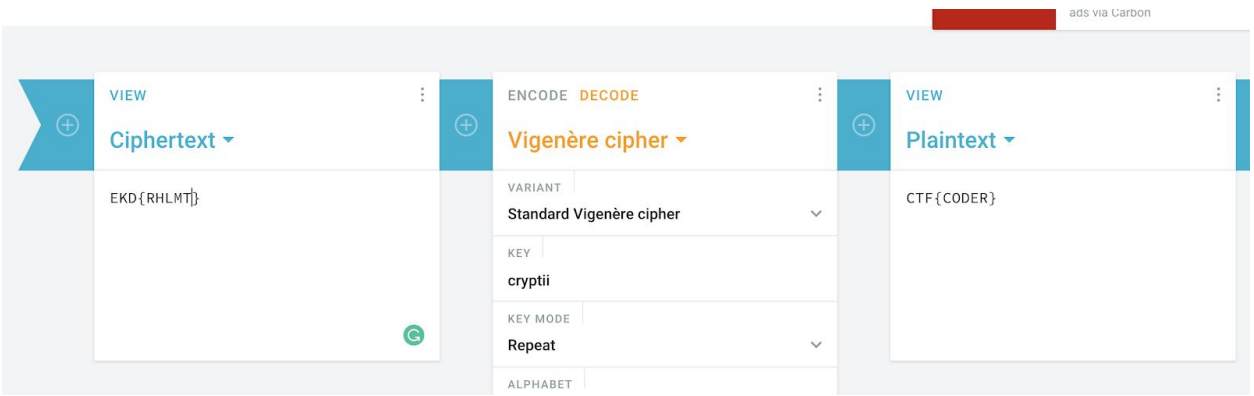
Which translates to : “*Do you want to see the flag?*”

ON SORTING THE TEXT IN THE FILE, WE COULD SEE THE FOLLOWING :

```
98
EKD{RHLMT}
append character value of b at last to get the correct flag
```

THIS TEXT EKD{RHLMT} GAVE ME A HINT THAT THIS MUST BE A CAESAR CIPHER OR A ROT13 CIPHER, BECAUSE THE FIRST 3 LETTERS MUST TRANSLATE TO CTF.

ON SPENDING SOME TIME DECIPHERING THE TEXT, I CAME TO NOTICE THAT THIS WAS INDEED A VIGNERE CIPHER.



THIS CONVERTED TO PLAINTEXT FORM OF CTF{CODER}

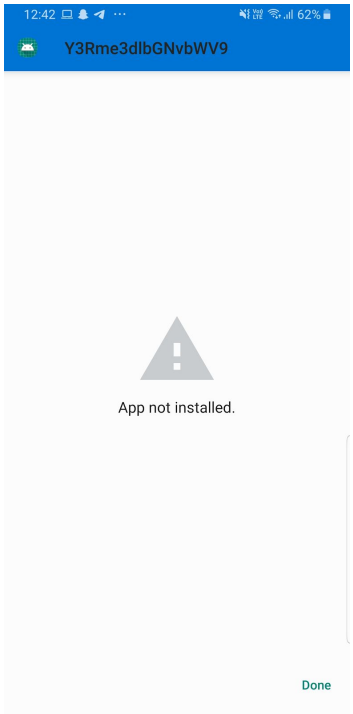
Appending the Character value of b in the end gave us the following flag :

FLAG : CTF{CODER098}

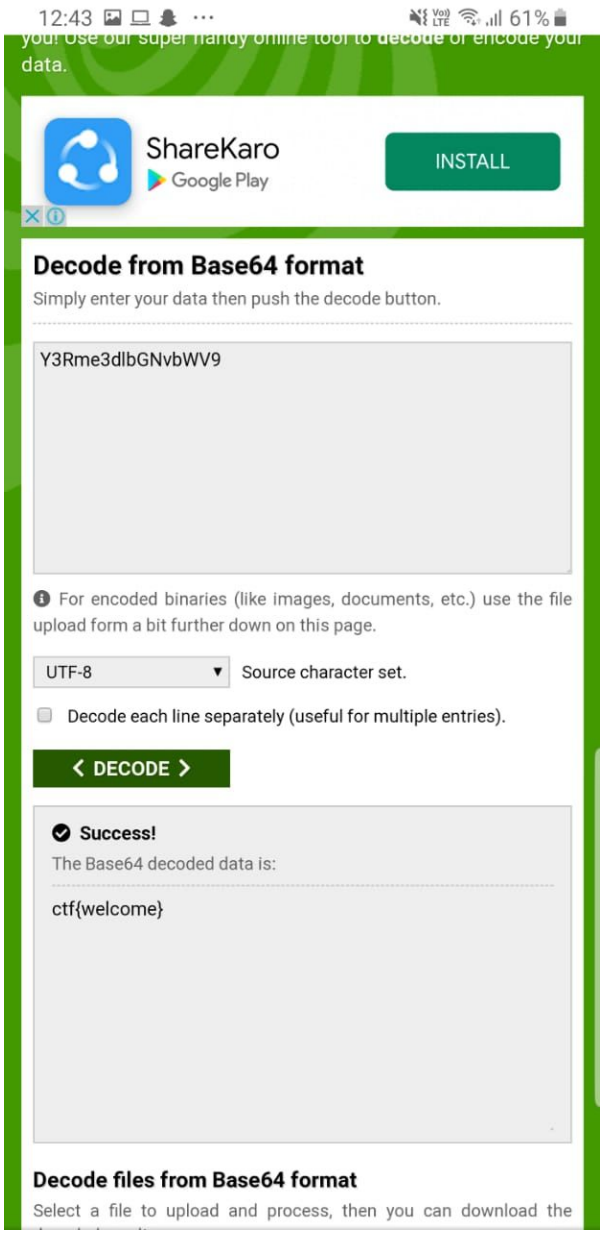
Challenge 4(ANDROID ANDROID ANDROID) :

This challenge required us to check for an apk.

On installing the APK file, I could see this :



The text in the top was base64 encoded and when decoded, it gave us the Flag :



FLAG: CTF{welcome}

Challenge 2 :

This challenge provided us with two files one being “encrypted.jpg” and crackthecode.py

On giving the code a thorough read, I could make out that **the encrypted.jpg file had some contents stored in its text file and the way it was to be decrypted was to use the *crackthecode.py* file.**

The code itself was made to run using python2 and had to be altered(just to change the name of the function **raw_input** to **input**) to make it compatible for python3.

I set the image file in the code to encrypted.jpg, and then ran the code multiple times.

```
KeyboardInterrupt
(base) Parths-MacBook-Pro:NANI ANNIVERSARY parthsikka$ python3 crackthecode.py
enter the file name to be encrypt:      encrypted.png
enter the no of iterations :    200
random integers are 10 , 78
encryption done in 8.83658480644226 sec.
(base) Parths-MacBook-Pro:NANI ANNIVERSARY parthsikka$ python3 crackthecode.py
enter the file name to be encrypt:      encrypted.png
enter the no of iterations :    1000
random integers are 161 , 203
encryption done in 43.0792818069458 sec.
(base) Parths-MacBook-Pro:NANI ANNIVERSARY parthsikka$ python3 crackthecode.py
enter the file name to be encrypt:      parth.txt
enter the no of iterations :    25
random integers are 122 , 125
Traceback (most recent call last):
  File "/Users/parthsikka/opt/anaconda3/lib/python3.7/site-packages/PIL/Image.py", line 2082, in save
    format = EXTENSION[ext]
KeyError: '.txt'

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "crackthecode.py", line 119, in <module>
    im2.save("enc_result/" + file_name)
  File "/Users/parthsikka/opt/anaconda3/lib/python3.7/site-packages/PIL/Image.py", line 2084, in save
    raise ValueError("unknown file extension: {}".format(ext))
ValueError: unknown file extension: .txt
(base) Parths-MacBook-Pro:NANI ANNIVERSARY parthsikka$ python3 crackthecode.py
enter the file name to be encrypt:      new.png
enter the no of iterations :    50
random integers are 95 , 38
encryption done in 2.236041307449341 sec.
(base) Parths-MacBook-Pro:NANI ANNIVERSARY parthsikka$ python3 crackthecode.py
enter the file name to be encrypt:      new2.png
enter the no of iterations :    2000
random integers are 177 , 72
encryption done in 119.33310198783875 sec.
```

But, this did not result in a proper output of a flag.

So after spending around half an hour on this question, I decided to move to ques 5.

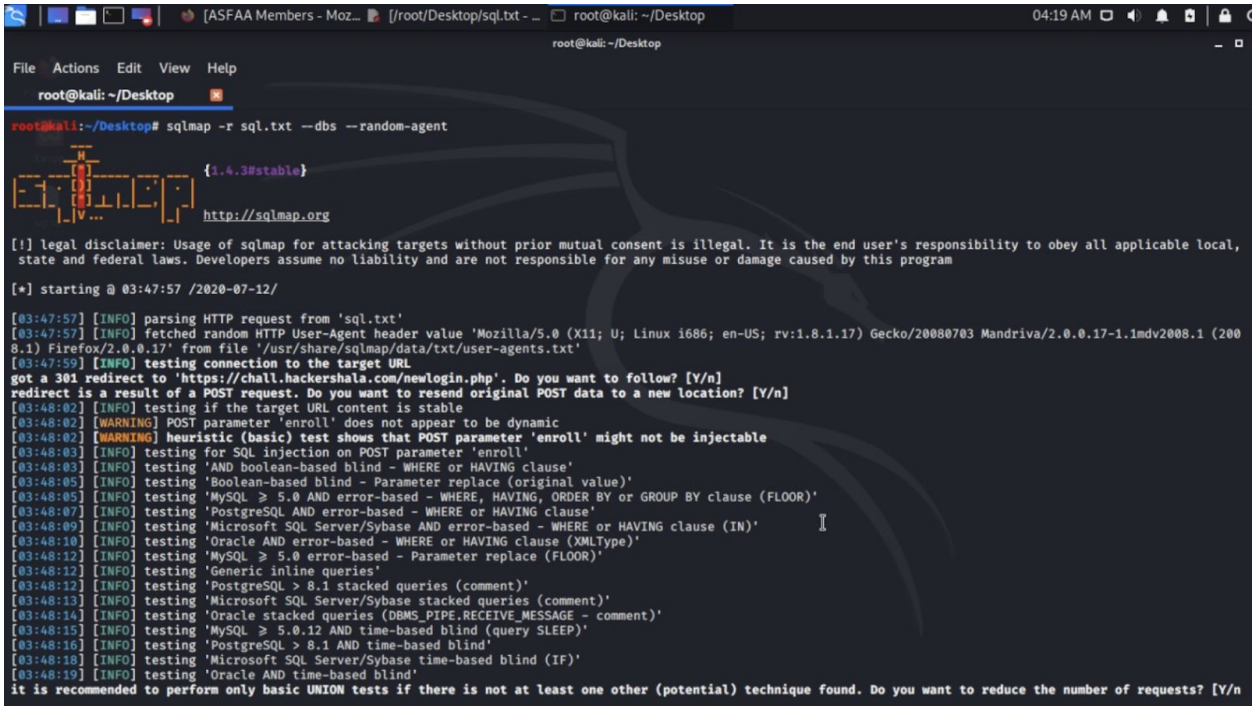
What I think could've been done : *BRUTE FORCING THE PROGRAM TO RUN IT WITH DIFFERENT ITERATIONS EACH RESULTING IN A DIFFERENT IMAGE, OUT OF WHICH ONE WOULD GIVE US THE RESULT.*

End finding : there is a rubics cube decryptor for an image as well which couldve been used but we could not due to time constraint.

Challenge 5 :

This challenge required us to check the database, and the first thing that popped to our minds was to check for any SQLinjection Vuln.

On finding none, we decided to check for any traffic using wireshark and then tried using ***SQLMAP*** to check if we could arrive anywhere:



However, due to the time constraint, we decided to give our last shot to question No. 3.

Challenge 3 :

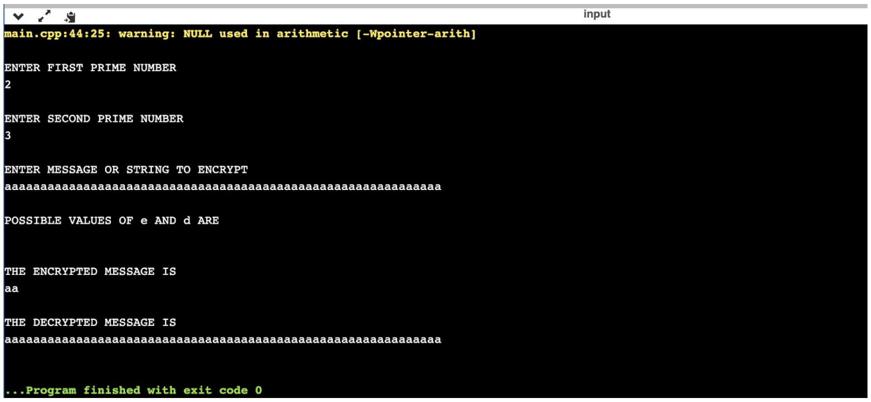
This challenge as quoted was Mathematical, however, on seeing the **important.cpp** file, I could make out that it was not the case.

The code was vulnerable to a **Buffer Overflow Attack** and I decided to do exactly that.

It included 4 functions and each of it accepted an array of **50 chars** before it jumped to the last function where we could get our generated flag.

I tried to overflow the buffer by running it on a remote server, however, it wasn't overflowing.

I noticed that the buffer started to overflow somewhere near **65 to 70 characters** when It actually should've overflowed at around **150 chars** according to what I had thought..



```
main.cpp:44:25: warning: NULL used in arithmetic [-Wpointer-arith]

ENTER FIRST PRIME NUMBER
2

ENTER SECOND PRIME NUMBER
3

ENTER MESSAGE OR STRING TO ENCRYPT
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa

POSSIBLE VALUES OF e AND d ARE


THE ENCRYPTED MESSAGE IS
aa

THE DECRYPTED MESSAGE IS
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa


...Program finished with exit code 0
```

In the end, I tried enumerating the files, but could not find anything.

Tried looking in the .git directory, but again found nothing other than the HEAD and Branch Addresses. Tried running them for the cpp code, however, all in vain.