



Introducere în Inginerie inversă pentru aplicații executabile, scripturi sau documente

Resurse utile pentru incepatori din UNbreakable România

unbreakable.ro

Declinarea responsabilității	3
Introducere	4
La ce sunt utile conceptele de inginerie inversa (reverse engineering)?	4
Procesul de decompilare	5
Analiza dinamică a unui program.	5
Analiza statică a unui program	6
Introducere în dezasamblarea codului sursa.	6
Operații matematice.	10
Bucle infinite (for si while).	14
Introducere în deobfuscarea codului sursa.	18
Introducere in steganografie	20
Resurse utile	23
Librarii si unelte utile în rezolvarea exercițiilor	24
Exerciții și rezolvări	25
Better-cat (usor)	25
Gogu (usor - mediu)	27
Mrrobot (mediu)	29
The_pass_pls (usor-mediu)	31
Crazy-Number (ușor)	36
Defuse-the-bomb (medium)	37
This-is-wendys (mediu)	43
Mastermind (mediu)	45
AgoodOne (easy)	49
Combined (easy-medium)	62
Leprechaun (ușor)	70
src (mediu)	73
Pyfuscation (mediu)	75
Contribuitori	78

Declinarea responsabilității

Aceste materiale și resurse sunt destinate exclusiv informării și discuțiilor, având ca obiectiv conștientizarea riscurilor și amenințarilor informatice dar și pregătirea unor noi generații de specialiști în securitate informatică.

Organizatorii și partenerii UNbreakable România nu oferă nicio garanție de niciun fel cu privire la aceste informații. În niciun caz, organizatorii și partenerii UNbreakable România, sau contractanții, sau subcontractanții săi nu vor fi răspunzători pentru niciun fel de daune, inclusiv, dar fără a se limita la, daune directe, indirecte, speciale sau ulterioare, care rezultă din orice mod ce are legătură cu aceste informații, indiferent dacă se bazează sau nu pe garanție, contract, delict sau altfel, indiferent dacă este sau nu din neglijență și dacă vătămarea a fost sau nu rezultată din rezultatele sau dependența de informații.

Organizatorii UNbreakable România nu aprobă niciun produs sau serviciu comercial, inclusiv subiectele analizei. Orice referire la produse comerciale, procese sau servicii specifice prin marca de servicii, marca comercială, producător sau altfel, nu constituie sau implică aprobarea, recomandarea sau favorizarea acestora de către UNbreakable România.

Organizatorii UNbreakable România recomandă folosirea cunoștințelor și tehnologiilor prezentate în aceste resurse doar în scop educațional sau profesional pe calculatoare, site-uri, servere, servicii sau alte sisteme informatice doar după obținerea acordului explicit în prealabil din partea proprietarilor.

Utilizarea unor tehnici sau unelte prezentate în aceste materiale împotriva unor sisteme informatice, fără acordul proprietarilor, poate fi considerată infracțiune în diverse țări.

În România, accesul ilegal la un sistem informatic este considerată infracțiune contra siguranței și integrității sistemelor și datelor informatice și poate fi pedepsită conform legii.

Introducere

Ingineria și dezvoltarea tehnologiilor în general este o industrie foarte versatilă, care devine mereu creativă. Folosind creativitatea și inovația, inginerii creează produse nemaivăzute, de care beneficiază comunitățile lor. Acestea joacă un rol cheie în extinderea economiei locale și în stimularea afacerilor. Cu toate acestea, rămâne o întrebare: cum inovează inginerii într-un mediu atât de rapid?

Răspunsul este invers, literalmente. Ingineria inversă (Reverse Engineering) joacă un rol imens în provocarea minților inovatoare și productive care produc necesități în fiecare industrie.

Ingineria inversă, în general, se referă la duplicarea produsului unui alt producător în urma unei examinări amănunțite a construcției sau a compoziției sale.

Aceasta implică dezasamblarea produsului pentru a înțelege cum funcționează. Aceasta face posibilă înțelegerea modului de lucru și a structurii sistemelor studiate. În contextul dezvoltării software, ingineria inversă presupune luarea unui sistem software și analizarea acestuia pentru a reproduce informațiile originale de proiectare și implementare.

La ce sunt utile conceptele de inginerie inversa (reverse engineering)?

Tehnicile de inginerie inversă sunt în general folosite pentru a:

- **Înțelege comportamentul unor aplicații malițioase (malware)** ce au încercat să anonimizeze serviciile cu care comunica, sau capabilitățile pe care le are o astfel de unealtă
- **Recupera codul sursă** în situația în care acesta a fost obfuscat / mascat pentru a proteja comportamentul real al acestuia
- **Identifica vulnerabilități** într-o aplicație ce a fost compilată (de eg. C/C++, C#, Java etc) prin recuperarea parțială sau integrală a codului sursă inițial folosind unelte de compilare, aplicații de debugging șamd

Adesea, hackerii dezvoltă metode de a induce în eroare un analist ce încearcă aplicarea unor tehnici de inginerie inversă astfel ca industria s-a dezvoltat și pe evitarea acestor tehnici.

De asemenea, hackerii folosesc tehnici de reverse engineering pentru a dezvolta “crack-uri” pentru diverse aplicații comerciale.

Uneori, ingineria inversa este ilegală datorită drepturilor de autor. Majoritatea software-urilor sunt proprietatea intelectuală a companiei care le-a creat.

Procesul de decompilare

Decompilarea este procesul de analiză a unui cod executabil sau de cod binar și de a scoate cod sursă într-un limbaj de programare precum C. Procesul implică traducerea unui fișier de la un nivel scăzut de abstractizare la un nivel mai ridicat de abstractizare, decompilator.

Software-ul poate fi inversat și decompilat. O mulțime de alte lucruri (cum ar fi hardware-ul) pot fi proiectate invers, dar nu decompilate, deoarece software-ul / firmware-ul lor este scris în limbaj de nivel scăzut (cod masina), fără o reprezentare de nivel superior sau, mai radical, nu au firmware în primul rând.

Analiza dinamică a unui program.

Analiza dinamica este analiza a software-ului de calculator care se realizează prin executarea de programe pe un procesor real sau virtual. Pe de altă parte, implică executarea programului și necesită instrumentarea blocurilor de bază, cum ar fi buclele, funcțiile etc. Informațiile colectate după analiză sunt de obicei utilizate pentru a optimiza aplicația efectuând derularea buclei cu un factor de derulare adecvat.

Cateva instrumente folosite pentru analiza dinamică (sistem de operare Windows):

- Immunity debugger.
- Ollydbg.
- WinDBG
- X64dbg
- dnSpy (.NET)
- Cheat Engine

Cateva instrumente folosite pentru analiza dinamica (sistem de operare Linux):

- GNU Debugger sau gdb.
- edb-debugger e un fel de immunity debugger doar ca pe sistemul Linux.

Analiza statică a unui program

Analiza statică este analiza a software-ului de calculator care se realizează fără a executa de fapt programul, practic este opusul analizei dinamice. Se bazează în principal pe găsirea de modele, numărarea referințelor de memorie iar în majoritatea cazurilor, analiza se efectuează pe o versiune a codului sursă, iar în celelalte cazuri, pe o formă a codului obiect.

Cateva instrumente folosite pentru analiza statică (sistem de operare Windows):

- **Ida Pro**
- **Ghidra**
- **dnspy** - acest tool este folosit pentru decompilarea aplicațiilor create în tehnologia .NET.
- **jd-gui** - este o aplicație pentru decompilat executabile create în limbajul de programare Java.

Cateva instrumente folosite pentru analiza statică (sistem de operare Linux):

- **Ida Pro**
- **Ghidra**
- **jd-gui**
- **radar2** sau **r2**
- **dex2jar**
- **Apktool (Android)**

Introducere în dezasamblarea codului sursa.

Vom începe prin a vizualiza în Ghidra cum se initializează o variabilă sau mai exact cum arată variabilele după dezasamblare. Vom scrie un scurt program în C apoi îl vom dezasamblare (dezasamblarea se face în limbajul de asamblare).

Programul conține doar variabile și este compilat cu gcc (gcc exemplu_1.c -o exemplu_1):

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

int main(){

    char sir_de_caractere[20]="Salut tuturor";
    int variabila_int=200;
```

Resurse utile pentru incepatori din UNbreakable România

```
bool variabila_boolean=true;
char variabila_char="D";
double variabila_double=3.1415;
float variabila_flotanta=2.5;
char vector[]={'a','b','c','d'};
}
```

Variabilele în Ghidra sau în Ida vor arăta astfel:

```
local_10= byte ptr -45h
local_18= dword ptr -38h
```

Acele numere (-45h, -38h) sunt offsetul fata de varful stivei.

Așa ca eu le-am redenumit ca sa înțelegem mai bine cam cum arată o variabila într-un decompilator.

```
variabila_boolean= byte ptr -3Ah
variabila_char= byte ptr -39h
variabila_int= dword ptr -38h
variabila_flotanta= dword ptr -34h
variabila_double= qword ptr -30h
vector_1= byte ptr -24h
vector_2= byte ptr -23h
vector_3= byte ptr -22h
vector_4= byte ptr -21h
vector= qword ptr -20h
var_18= qword ptr -18h
var_12= dword ptr -10h
sir_de_caractere= qword ptr -8
```

```
push    rbp
mov     rbp, rsp
sub     rsp, 40h
mov     rax, fs:28h
mov     [rbp+sir_de_caractere], rax
xor     eax, eax
mov     rax, 'ut tulaS'
mov     rdx, 'rorut'
mov     [rbp+vector], rax
mov     [rbp+var_18], rdx
mov     [rbp+var_12], 0
mov     [rbp+variabila_int], 0C8h
mov     [rbp+variabila_boolean], 1
lea     rax, off_788
mov     [rbp+variabila_char], al
movsd   xmm0, cs:qword_790
movsd   [rbp+variabila_double], xmm0
movss   xmm0, cs:dword_798
movss   [rbp+variabila_flotanta], xmm0
mov     [rbp+vector_1], 61h ; 'a'
mov     [rbp+vector_2], 62h ; 'b'
mov     [rbp+vector_3], 63h ; 'c'
mov     [rbp+vector_4], 64h ; 'd'
nop
mov     rax, [rbp+sir_de_caractere]
xor     rax, fs:28h
jz      short locret_6F8
```

Pe scurt, vedem variabila **sir_de_caractere** care va fi stocată în registrul RAX. RAX va avea valoare **75742074756C6153h** acel "h" de la sfarsit ne indica ca valoare este în hexadecimale, dacă convertim acea valoare în ascii vom avea ("ut tulaS" = "Salut tu") adresele de memorie se citesc de la dreapta la stanga. În continuare registrul RDX va avea următoarele caractere stocate ("rorut"=turor). Mai jos am convertit valorile din hex în ascii.

```
mov     rax, 'ut tulaS'
mov     rdx, 'rorut'
```

Următoarea variabila este cea de tip integer care este reprezentată în acest exemplu sub următoarea formă.

```
mov     [rbp+variabila_int], 0C8h
```

La fel ca în exemplul anterior valoarea **0C8h** este reprezentată în decimal ca fiind 200.

```
mov     [rbp+variabila_int], 200
```

Următoarea variabila este cea de tip boolean care este reprezentată în acest exemplu sub următoarea formă (valoarea 1 vine de la true, dacă era 0 atunci era false).

```
mov     [rbp+variabila_boolean], 1
```


Resurse utile pentru incepatori din UNbreakable România

Următoarea variabila este cea de tip caracter care este reprezentată în acest exemplu sub următoarea formă (off_788 ne va directiona catre offsetul dword_44 adică offsetul este 44h care în ascii este fix valoarea D).

```
lea    rax, off_788
mov    [rbp+variabila_char], al
```

Urmărim ruta off_788, care ne va duce către dword_44 adică valoarea D.

```
00000788 off_788          dq offset dword_44
```

Următoarele variabila sunt cele de tip double si float.

```
movsd  xmm0, cs:qword_790
movsd  [rbp+variabila_double], xmm0
movss  xmm0, cs:dword_798
movss  [rbp+variabila_flotanta], xmm0
```

Următoarele variabila sunt de tip vector, putem observa ca a și făcut conversia de la hex to ascii și le-a ordonat frumos.

```
mov     [rbp+vector_1], 61h ; 'a'
mov     [rbp+vector_2], 62h ; 'b'
mov     [rbp+vector_3], 63h ; 'c'
mov     [rbp+vector_4], 64h ; 'd'
```

În imaginea de jos puteți observa ca nu este o foarte mare diferenta cand il decompilati în GNU debugger (gdb) - instrucțiunile seamănă.

Resurse utile pentru incepatori din UNbreakable România

```
darius@bit-sentinel:~/Desktop/unbreakable/inginerie_inversa$ gdb -silent ./exemplul1
warning: ~/peda/peda.py source ~/peda/peda.py: No such file or directory
Reading symbols from ./exemplul1...(no debugging symbols found)...done.
gdb-peda$ disassemble main
Dump of assembler code for function main:
0x000000000000066a <+0>:    push    rbp
0x000000000000066b <+1>:    mov     rbp, rsp
0x000000000000066e <+4>:    sub     rsp, 0x40
0x0000000000000672 <+8>:    mov     rax, QWORD PTR fs:0x28
0x000000000000067b <+17>:   mov     QWORD PTR [rbp-0x8], rax
0x000000000000067f <+21>:   xor     eax, eax
0x0000000000000681 <+23>:   movabs  rax, 0x75742074756c6153
0x000000000000068b <+33>:   movabs  rdx, 0x726f727574
0x0000000000000695 <+43>:   mov     QWORD PTR [rbp-0x20], rax
0x0000000000000699 <+47>:   mov     QWORD PTR [rbp-0x18], rdx
0x000000000000069d <+51>:   mov     DWORD PTR [rbp-0x10], 0x0
0x00000000000006a4 <+58>:   mov     DWORD PTR [rbp-0x38], 0xc8
0x00000000000006ab <+65>:   mov     BYTE PTR [rbp-0x3a], 0x1
0x00000000000006af <+69>:   lea     rax, [rip+0xd2]          # 0x788
0x00000000000006b6 <+76>:   mov     BYTE PTR [rbp-0x39], al
0x00000000000006b9 <+79>:   movsdi xmm0, QWORD PTR [rip+0xcf] # 0x790
0x00000000000006c1 <+87>:   movsdi QWORD PTR [rbp-0x30], xmm0
0x00000000000006c6 <+92>:   movssi xmm0, DWORD PTR [rip+0xca] # 0x798
0x00000000000006ce <+100>:  movssi  DWORD PTR [rbp-0x34], xmm0
0x00000000000006d3 <+105>:  mov     BYTE PTR [rbp-0x24], 0x61
0x00000000000006d7 <+109>:  mov     BYTE PTR [rbp-0x23], 0x62
0x00000000000006db <+113>:  mov     BYTE PTR [rbp-0x22], 0x63
0x00000000000006df <+117>:  mov     BYTE PTR [rbp-0x21], 0x64
0x00000000000006e3 <+121>:  nop
0x00000000000006e4 <+122>:  mov     rax, QWORD PTR [rbp-0x8]
0x00000000000006e8 <+126>:  xor     rax, QWORD PTR fs:0x28
0x00000000000006f1 <+135>:  je      0x6f8 <main+142>
0x00000000000006f3 <+137>:  call   0x540 <__stack_chk_fail@plt>
0x00000000000006f8 <+142>:  leave
0x00000000000006f9 <+143>:  ret
End of assembler dump.
```

Operații matematice.

În această secțiune, vom trece în revistă următoarele funcții matematice:

- Adunare
- Scadere
- Inmultire
- Impartire
- Operatia AND
- Operatia OR
- Operatia XOR
- Operatia NOT
- Bitwise la dreapta
- Bitwise la stanga

Codul sursa de la exemplul 2 este:

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

void functii_matematice(){
```

Resurse utile pentru incepatori din UNbreakable România

```
int A = 12;
int B = 15;

int adunare = A+B;    //Adunare
int scadere = A-B;    //Scadere
int inmultire = A*B;  //Inmultire
int impartire = A/B;  //Impartire
int operatia_and = A&B; //Operatia SI
int operatia_or = A|B; //Operatia SAU
int operatia_xor = A^B; //Operatia de XOR
int operatia_not = ~A; //Operatia NOT
int rshit = A >> B;   //Mutare biți de la dreapta la stanga
int lshift = A << B;   //Mutare biți de la stanga la dreapta

return 0;

}
int main(){

functii_matematice();

return 0;
}
```

De data asta în Ida nu vom mai analiza funcția main și vom analiza funcția “functii matematice”. Pentru început vom denumi din nou variabilele.

```
A= dword ptr -30h
B= dword ptr -2Ch
adunare= dword ptr -28h
scadere= dword ptr -24h
inmultire= dword ptr -20h
impartire= dword ptr -1Ch
operatia_AND= dword ptr -18h
operatia_OR= dword ptr -14h
operaita_XOR= dword ptr -10h
operatia_NOT= dword ptr -0Ch
rshift= dword ptr -8
lshift= dword ptr -4
```

```
push    rbp
mov     rbp, rsp
mov     [rbp+A], 0Ch
mov     [rbp+B], 0Fh
mov     edx, [rbp+A]
mov     eax, [rbp+B]
add     eax, edx
mov     [rbp+adunare], eax
mov     eax, [rbp+A]
sub     eax, [rbp+B]
mov     [rbp+scadere], eax
mov     eax, [rbp+A]
imul    eax, [rbp+B]
mov     [rbp+inmultire], eax
mov     eax, [rbp+A]
cdq
idiv    [rbp+B]
mov     [rbp+impartire], eax
mov     eax, [rbp+A]
and     eax, [rbp+B]
mov     [rbp+operatia_AND], eax
mov     eax, [rbp+A]
or      eax, [rbp+B]
mov     [rbp+operatia_OR], eax
mov     eax, [rbp+A]
xor     eax, [rbp+B]
mov     [rbp+operaita_XOR], eax
mov     eax, [rbp+A]
not     eax
mov     [rbp+operatia_NOT], eax
mov     eax, [rbp+B]
mov     edx, [rbp+A]
mov     ecx, eax
sar     edx, cl
mov     eax, edx
mov     [rbp+rshift], eax
mov     eax, [rbp+B]
mov     edx, [rbp+A]
mov     ecx, eax
shl     edx, cl
mov     eax, edx
mov     [rbp+lshift], eax
nop
pop     rbp
retn
functii_matematice endp
```

Prima data trebuie sa identificam unde in code i se atribuie variabilei A și variabilei B valorile 12 respectiv 15.

Resurse utile pentru incepatori din UNbreakable România

```
mov    [rbp+A], 0Ch
mov    [rbp+B], 0Fh
```

După cum putem observa avem 0x0C și 0x0F în decimal, acestea sunt fix valorile pe care i le-am atribuit lui A și B.

```
mov    [rbp+A], 12
mov    [rbp+B], 15
```

Adunarea se face cu ajutorul instrucțiunii **add**. Logica este foarte simplă, i se alocă valoarea lui A adică 12 în registrul **edx** (**edx=12**) și lui **eax** valoarea 15 (**eax=15**) apoi cu ajutorul instrucțiunii **add eax,edx** (12+15) se face adunarea. Pe același principiu funcționează și următoarele instrucțiuni.

```
mov    edx, [rbp+A]
mov    eax, [rbp+B]
add    eax, edx
```

Scaderea se face cu ajutorul instrucțiunii **sub**

```
mov    eax, [rbp+A]
sub    eax, [rbp+B]
mov    [rbp+scadere], eax
```

Înmulțirea se face cu ajutorul instrucțiunii **imul** sau **mul**.

```
mov    eax, [rbp+A]
imul   eax, [rbp+B]
mov    [rbp+inmultire], eax
```

Împărțirea se face cu ajutorul instrucțiunii **idiv** sau **div**.

```
mov    eax, [rbp+A]
cdq
idiv   [rbp+B]
mov    [rbp+impartire], eax
```

Operația AND se face cu ajutorul instrucțiunii **and**.

```
mov    eax, [rbp+A]
and    eax, [rbp+B]
mov    [rbp+operatia_AND], eax
```

Operația OR se face cu ajutorul instrucțiunii **or**.

```
mov    eax, [rbp+A]
or     eax, [rbp+B]
mov    [rbp+operatia_OR], eax
```

Resurse utile pentru incepatori din UNbreakable România

Operația XOR se face cu ajutorul instrucțiunii **xor**.

```
mov    eax, [rbp+A]
xor     eax, [rbp+B]
mov     [rbp+operaita_XOR], eax
```

Operația NOT se face cu ajutorul instrucțiunii **not**.

```
mov     eax, [rbp+A]
not     eax
mov     [rbp+operatia_NOT], eax
```

Operația de rshift se face cu instrucțiunea **sar**.

```
mov     eax, [rbp+B]
mov     edx, [rbp+A]
mov     ecx, eax
sar     edx, cl
mov     eax, edx
mov     [rbp+rshift], eax
```

Operația de lshift se face cu instrucțiunea **shl**.

```
mov     eax, [rbp+B]
mov     edx, [rbp+A]
mov     ecx, eax
shl     edx, cl
mov     eax, edx
mov     [rbp+lshift], eax
```

Bucle infinite (for si while).

Vom începe cu o bucla infinită de tip **for**. Vom analiza exemplul 3.

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

void functia_for(){

    for (int i=0; i < 10;i++){
        printf("%i \n",i);
    }
}

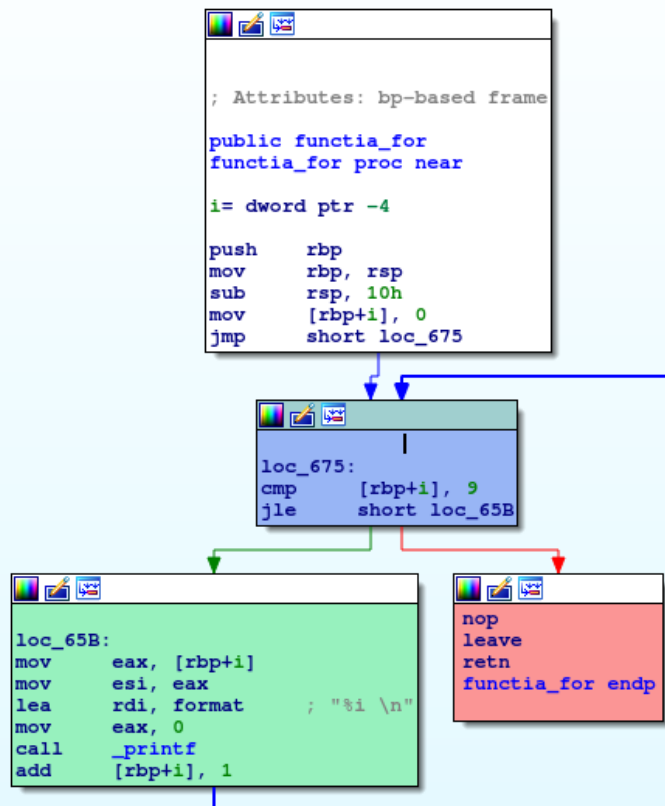
int main(){

    functia_for();
```

Resurse utile pentru incepatori din UNbreakable România

```
return 0;  
}
```

Pe scurt acest program va afișa cifrele de la 0 la 9. Reprezentarea in cod de asamblare a funcției **for** va arata ceva de genul.

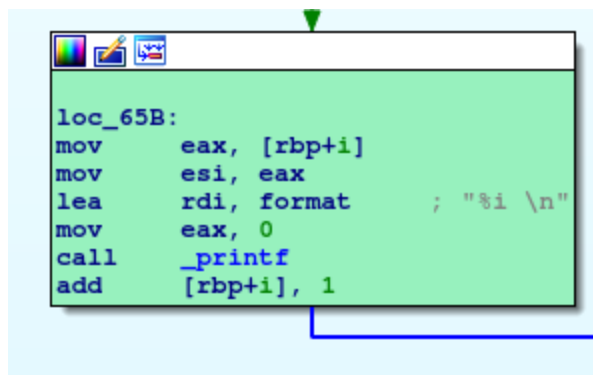


Pentru început lui "i" i se atribuie valoarea 0 apoi face un jump către blocul cu culoare albastra.

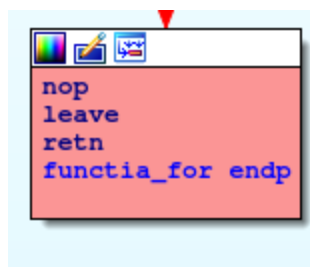
```
mov    [rbp+i], 0  
jmp    short loc_675
```

Resurse utile pentru incepatori din UNbreakable România

În blocul albastru se compara valoarea lui "i" cu valoarea 9, dacă "i" este 9 atunci se va ieși din bucla, adică va merge la blocul de culoare roșu, dar fiindcă "i" are valoarea 1 va merge în blocul de culoare verde.



În acest bloc se va afișa valoarea lui "i" apoi se va adăuga +1([add \[rsb+i\],1](#)) valori lui "i" și se va întoarce iară în blocul albastru să compare valorile. Va face acest lucru până "i" va fi egal cu 9, apoi va ieși din bucla și va merge la blocul roșu unde se termina execuția programului.



Exemplul 4, bucla while.

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

void functia_while(){
    int A = 0;

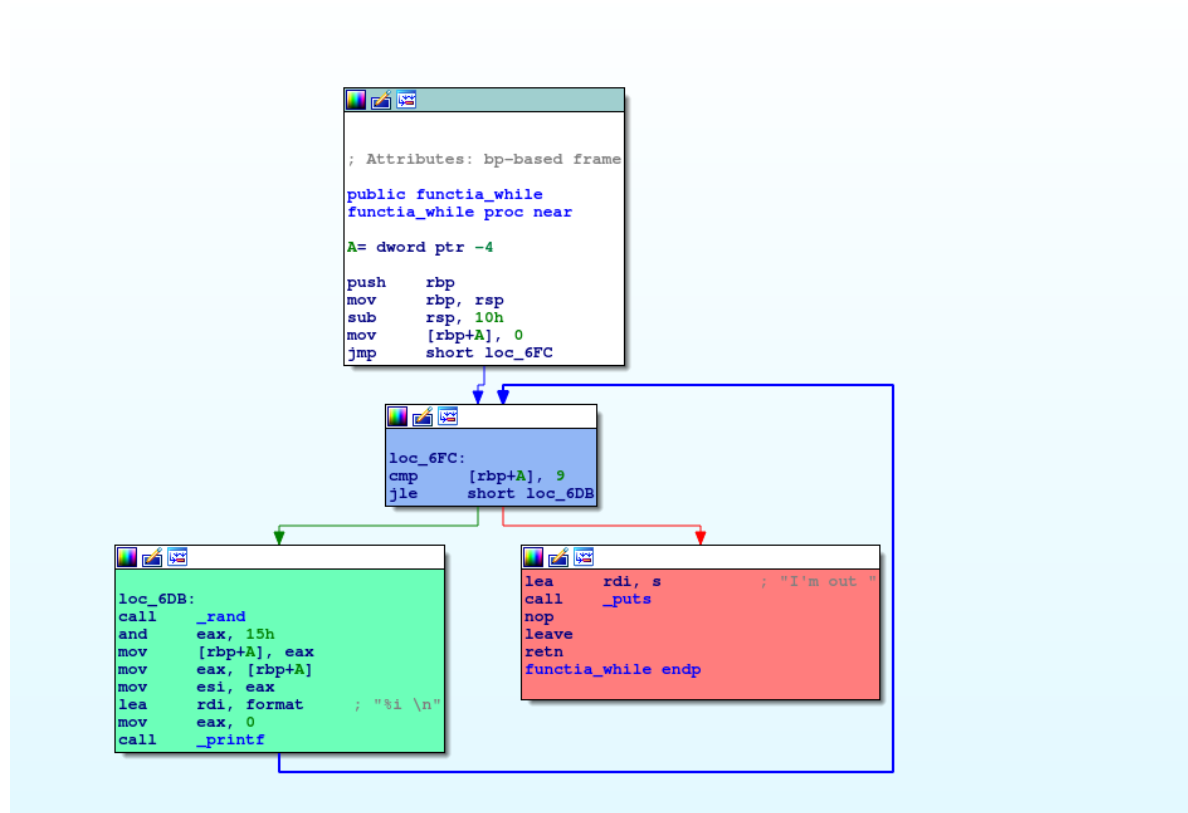
    while(A<10){
        A = 0 + (rand() & (int)(20-0+1));
        printf("%i \n",A);
    }
    printf("I'm out \n");
}
```


Resurse utile pentru incepatori din UNbreakable România

```
}  
int main(){  
  
    functia_while();  
    return 0;  
}
```

```
darius@bit-sentinel:~/Desktop/unbreakable/inginerie_inversa$ ./exemplu4  
5  
4  
1  
17  
I'm out
```

În această buclă, tot ceea ce facem este să generăm un număr aleatoriu între 0 și 20. Dacă numărul este mai mare de 10, ieșim din buclă și imprimăm „**I'm out**” altfel, continuăm să facem o buclă. În ansamblu, variabila A este generată și setată la 0 inițial, apoi inițializăm bucla comparând A cu numărul 10 (în decompilator va apărea 9 pentru ca de la 0 la 9 sunt 10 cifre). Dacă A nu este mai mare sau egal cu 10, generăm un nou număr aleatoriu care este apoi setat la A și continuăm înapoi la comparație. Dacă A este mai mare sau egal cu 10, ieșim din buclă, imprimăm „**I'm out**” și apoi ne întoarcem. Seamănă cu exemplul 3.



Introducere în deobfuscarea codului sursa.

Deobfuscarea este tehnica prin care un ethical hacker decodeaza sau decripteaza informațiile pe care un atacator intenționează sa le folosească. De obicei, un atacator folosește tehnica de obfuscare în scopul de a face cat mai greu citibil codul sursa a unei aplicații pe care o executa în scop malițios sau pentru a trece de anumite protectii cum sunt cele de firewall sau de antivirus.

Exemplu de code obfusc.

```
var _0x5377=["\x48\x65\x6C\x6C\x20\x57\x6F\x72\x6C\x64\x21"];var
a=_0x5377[0];function MsgBox(_0x82a8x3){alert(_0x82a8x3);};MsgBox(a);
```

Resurse utile pentru incepatori din UNbreakable România

Acesta este unul dintre cele mai ușoare exemple de deobfuscare a unui cod de javascript. În primul rând ne vom folosi de [JavaScript Beautifier](#) este un tool online care ne va face codul mult mai frumos.

```
var a = 'Hello World!';

function MsgBox(_0x82a8x3) {
    alert(_0x82a8x3);
};
MsgBox(a);
```

Pe scurt acest tool a decodat caracterele din hex to ascii.

```
\x48\x65\x6C\x6C\x6F\x20\x57\x6F\x72\x6C\x64\x21 -- 48656c6c6f20576f726c6421
```

Un alt exemplu de obfuscare este prin metoda XOR.

```
#!/bin/python3

A = "Salut tuturor"
B = "asdadasdasada"
lista=[chr(ord(a)^ord(b)) for a,b in zip(A,B)]
print(lista)
```

Outputul va fi ceva de genul.

```
darius@bit-sentinel:~/Desktop/unbreakable/inginerie_inversa$ python xor.py
['2', '\x12', '\x08', '\x14', '\x10', 'A', '\x07', '\x11', '\x15', '\x06', '\x13', '\x0b', '\x13']
```

Ca sa putem deobfusca această valoare trebuie doar sa mai folosim încă o dată xor cu una din valorile A sau B.

```
#!/bin/python3

A = "2\x12\x08\x14\x10A\x07\x11\x15\x06\x13\x0b\x13"
B="asdadasdasada"
lista=[chr(ord(a)^ord(b)) for a,b in zip(A,B)]
print(lista)
```

```
darius@bit-sentinel:~/Desktop/unbreakable/inginerie_inversa$ python decrypt.py  
['S', 'a', 'l', 'u', 't', ' ', 't', 'u', 't', 'u', 'r', 'o', 'r']  
darius@bit-sentinel:~/Desktop/unbreakable/inginerie_inversa$
```

Introducere in steganografie

Steganografia este practica de a ascunde un mesaj într-un alt mesaj sau un obiect fizic. În contextele informatice / electronice, un fișier, mesaj, imagine sau videoclip al computerului este ascuns într-un alt fișier, mesaj, imagine sau videoclip.

Cele mai folosite instrumente folosite pentru steganografie sunt:

- Hide'N'Send
- SteganPEG
- OpenStego
- Our Secret
- SSuite Piscal
- Exiftool
- Binwalk
- Steghide

Vom ascunde într-o imagine un fișier de tip zip, care contine un mesaj secret inauntru.

```
darius@bit-sentinel:~/Desktop/unbreakable/inginerie_inversa$ cat 'security_dude.png' 'secret.zip' > imagine_secreta.png  
darius@bit-sentinel:~/Desktop/unbreakable/inginerie_inversa$
```

Imaginea a fost creata.

Resurse utile pentru incepatori din UNbreakable România



Acuma vom analiza ambele imagini folosind binwalk. Prima imagine (security_dude.png) nu contine arhiva zip.

```
darius@bit-sentinel:~/Desktop/unbreakable/inginerie_inversa$ binwalk security_dude.png
```

DECIMAL	HEXADECIMAL	DESCRIPTION
0	0x0	PNG image, 1024 x 684, 8-bit/color RGBA, non-interlaced
254	0xFE	Unix path: /www.w3.org/1999/02/22-rdf-syntax-ns#>
384	0x180	Zlib compressed data, default compression

```
darius@bit-sentinel:~/Desktop/unbreakable/inginerie_inversa$ binwalk imagine_secreta.png
```

DECIMAL	HEXADECIMAL	DESCRIPTION
0	0x0	PNG image, 1024 x 684, 8-bit/color RGBA, non-interlaced
254	0xFE	Unix path: /www.w3.org/1999/02/22-rdf-syntax-ns#>
384	0x180	Zlib compressed data, default compression
517028	0x7E3A4	Zip archive data, at least v2.0 to extract, uncompressed size: 13, name: secret.txt
517219	0x7E463	End of Zip archive

Resurse utile pentru incepatori din UNbreakable România

În cea de-a doua imagine putem observa ceva ciuda, ultimele 2 coloane conțin date despre alt fișier. Vom extrage acel zip folosind binwalk.

```
darius@bit-sentinel:~/Desktop/unbreakable/inginerie_inversa$ binwalk -e imagine_secreta.png
```

DECIMAL	HEXADECIMAL	DESCRIPTION
0	0x0	PNG image, 1024 x 684, 8-bit/color RGBA, non-interlaced
254	0xFE	Unix path: /www.w3.org/1999/02/22-rdf-syntax-ns#>
384	0x180	Zlib compressed data, default compression
517028	0x7E3A4	Zip archive data, at least v2.0 to extract, uncompressed size: 13, name: secret.txt
517219	0x7E463	End of Zip archive

```
darius@bit-sentinel:~/Desktop/unbreakable/inginerie_inversa$
```

Ne va descarca un folder cu următorul nume.

```
darius@bit-sentinel:~/Desktop/unbreakable/inginerie_inversa$ ls -la | grep _image
drwxr-xr-x 2 darius darius 4096 mar 11 10:55 _image_secreta.png.extracted
darius@bit-sentinel:~/Desktop/unbreakable/inginerie_inversa$
```

Acesta contine arhiva zip și conținutul arhivei.

```
darius@bit-sentinel:~/Desktop/unbreakable/inginerie_inversa/_image_secreta.png.extracted$ ls
180 180.zlib 7E3A4.zip secret.txt
darius@bit-sentinel:~/Desktop/unbreakable/inginerie_inversa/_image_secreta.png.extracted$ cat secret.txt
Mesaj Secret
darius@bit-sentinel:~/Desktop/unbreakable/inginerie_inversa/_image_secreta.png.extracted$
```

Resurse utile

- [LiveOverflow](#)
- [Assembly Programming Tutorial](#)
- [Davy Wybiral - Intro to x86 Assembly Language](#)
- [Modern x64 Assembly Language](#)
- [PC Assembly Language](#)
- [Reverse Engineering IDA tutorial](#)

Librarii si unelte utile în rezolvarea exercițiilor

- [dnSpy](#) (decompilare .NET)
- [Ghidra](#)
- IDA
- [x64dbg](#)
- [Jd-gui](#), Android-Studio, apk-tools (.apk's)
- [radare2](#) + [cutter](#)
- gdb
- ImmunityDebugger
- [Binary Ninja](#)
- z3
- angr

Exerciții și rezolvări

Better-cat (usor)

Concurs: UNbreakable #1 (2020)

Descriere:

You might need to look for a certain password.

Flag format: ctf{sha256}

Goal: In this challenge you have to obtain the password string or flag from the binary file.

The challenge was created by Bit Sentinel.

Rezolvare:

Ne este dat și un fișier, **cat.elf**. Putem determina tipul fișierului folosind comanda **file**:

```
yakuhito@furry-catstation:~/ctf/unbr1/better-cat$ file cat.elf
cat.elf: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), dynamically linked,
interpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux 3.2.0,
BuildID[sha1]=cfcb7ab0ad0834a3ee237b8c4e6ee136978d4b26, not stripped
```

Fișierul este un executabil care cere o parola atunci cand este rulat. Putem folosi comanda **strings** pentru a vedea șirurile de caractere citibile din acesta:

```
yakuhito@furry-catstation:~/ctf/unbr1/better-cat$ strings -7 cat.elf | head -n 24
/lib64/ld-linux-x86-64.so.2
libc.so.6
__isoc99_scanf
__stack_chk_fail
__cxa_finalize
__libc_start_main
```

```
GLIBC_2.7
GLIBC_2.4
GLIBC_2.2.5
_ITM_deregisterTMCloneTable
__gmon_start__
_ITM_registerTMCloneTable
parola12
Well donH
e, your H
special H
flag is:H
ctf{a81H
8778ec7aH
9fc19887H
[redactat]
b42e998eH
b09450eaH
b7f1236eH
yakuhito@furry-catstation:~/ctf/unbr1/better-cat$
```

Comanda **strings -7 cat.elf** arată toate stringurile din fișierul **cat.elf** care au lungimea mai mare sau egala decat 7. Output-ul este apoi trecut prin comanda **head -n 24**, care afișează doar primele 24 de linii ale primei comenzi.

Se pot observa mai multe parti ale flag-ului, precum si un string care pare a fi parola pe care o cere programul: **parola12**. Putem obține flag-ul și dacă rulăm **cat.elf** și introducem **parola12** ca parola:

```
yakuhito@furry-catstation:~/ctf/unbr1/better-cat$ ./cat.elf
https://www.youtube.com/watch?v=oHg5SJYRHA0
The password is: parola12
Well done, your special flag is:
ctf{a818778ec7a9fc1988724ae3[redactat]50eab7f1236e53bfdcd923878}
yakuhito@furry-catstation:~/ctf/unbr1/better-cat$
```

Rezolvare în engleză: <https://blog.kuhi.to/unbreakable-romania-1-writeup#better-cat>

Gogu (usor - mediu)

Concurs: UNbreakable #1 (2020)

Descriere:

For sure obfuscated values are secure.

Flag format: ctf{sha256}

Goal: In this challenge you have to bypass various anti-debugging and anti-reverse techniques in order to recover the flag.

The challenge was created by Bit Sentinel.

Rezolvare:

Dacă rulăm aplicația data, putem vedea ca printează un string care nu este citibil:

```
yakuhito@furry-catstation:~/ctf/unbr1/gogu$ ./gogu.exe
Welcome to gogu!
Good luck!
a961f71e0f287ac52a25aa93be854377
yakuhito@furry-catstation:~/ctf/unbr1/gogu$
```

După ce deschidem binarul într-un decompilator precum IDA Pro, ne dăm seama ca acesta este o aplicație de **go** compilată. Am presupus ca hash-ul afisat reprezinta flag-ul 'encodat' si ca flag-ul se afla în memorie la un moment dat. Pentru ca analiza statică a binarelor generate de **go** fără simboluri de debug (cazul de fata) ia foarte mult timp, am decis sa încerc sa analizez programul dinamic.

Cum programul afișează ceva pe ecran, sigur va folosi syscall-ul **write**. Presupunand ca flag-ul e în memorie cand hash-ul e afișat, am putea scrie continutul stack-ului si al heap-ului într-un fisier in care sa cautam ulterior flag-ul. Putem face asta cu ajutorul debugger-ului **gdb**:

Resurse utile pentru incepatori din UNbreakable România

```
yakuhito@furry-catstation:~/ctf/unbr1/gogu$ gdb ./gogu.exe
Reading symbols from ./gogu.exe...(no debugging symbols found)...done.
gdb-peda$ catch syscall write
Catchpoint 1 (syscall 'write' [1])
gdb-peda$ r
[...]
gdb-peda$ c
[...]
gdb-peda$ c
[...]
gdb-peda$ c
[...]
gdb-peda$ c
[...]
gdb-peda$ vmmap
Start      End      Perm    Name
0x00400000 0x00484000 r-xp    /home/yakuhito/ctf/unbr1/gogu/gogu.exe
0x00484000 0x00517000 r--p    /home/yakuhito/ctf/unbr1/gogu/gogu.exe
0x00517000 0x0052b000 rw-p    /home/yakuhito/ctf/unbr1/gogu/gogu.exe
0x0052b000 0x0054a000 rw-p    [heap]
0x000000c000000000 0x000000c000001000 rw-p    mapped
0x000000c41fff8000 0x000000c420100000 rw-p    mapped
0x00007fff7f5a000 0x00007fff7ffa000 rw-p    mapped
0x00007fff7ffa000 0x00007fff7ffd000 r--p    [vvar]
0x00007fff7ffd000 0x00007fff7fff000 r-xp    [vdso]
0x00007fffffde000 0x00007fffffff000 rw-p    [stack]
0xffffffff600000 0xffffffff601000 r-xp    [vsyscall]
gdb-peda$ dump memory mem.dump 0x000000c41fff8000 0x000000c420100000
gdb-peda$ quit
yakuhito@furry-catstation:~/ctf/unbr1/gogu$ strings mem.dump | grep ctf{
ctf{1fe6954870babd55ba6e5d[redacted]533397b985c890749cbfc7e306}
ctf{1fe6954870babd55ba6e5d[redacted]b70533397b985c890749cbfc7e306}
```

Rezolvare în engleză: <https://blog.kuhi.to/unbreakable-romania-1-writeup#gogu>

Mrrobot (mediu)

Concurs: UNbreakable #2 (2020)

Descriere:

```
Let's secure the network using some special routers. We need to decrypt this message first :  
013032224029145C2047711D11562831021F077A1406782B28
```

```
flag = ctf{decrypt_message(sha256)}
```

Rezolvare:

Pagina exercițiului ne oferă și un binar, însă output-ul programului **strings** nu prea ne este de ajutor:

```
yakuhito@furry-catstation:~/ctf/unr2/mrrobot$ strings ./encrypt  
/lib64/ld-linux-x86-64.so.2  
libc.so.6  
[...]  
encrypt  
! Error: %s  
Encrypt message: %s  
Message was encrypted: %s  
dsfd;kfoA,.iyewrkldJKDHSUBsgvca69834ncxv  
[...]  
yakuhito@furry-catstation:~/ctf/unr2/mrrobot$
```

Deși stringul **dsfd;kfoA,.iyewrkldJKDHSUBsgvca69834ncxv** ne poate atrage atenția, nu știm ce am putea face cu el. Dacă rulăm programul, vom vedea că acesta criptează un string pe care-l controlăm, însă nu știm algoritmul folosit. Pentru a descoperi mai multe informații, deschidem fișierul în IDA Pro și decompilăm funcția care criptează input-ul:

Resurse utile pentru incepatori din UNbreakable România

```
1 _BYTE *__fastcall sub_C81(const char *a1)
2 {
3     char v1; // dl
4     unsigned int v2; // eax
5     char v3; // al
6     char v4; // al
7     unsigned int v5; // ST20_4
8     char v7; // [rsp+1Bh] [rbp-15h]
9     unsigned int v8; // [rsp+1Ch] [rbp-14h]
10    unsigned int i; // [rsp+20h] [rbp-10h]
11    unsigned int v10; // [rsp+24h] [rbp-Ch]
12    _BYTE *v11; // [rsp+28h] [rbp-8h]
13
14    v10 = strlen(a1);
15    v11 = malloc(2 * v10 + 3);
16    if ( v10 > 0x19 )
17        v10 = 25;
18    v8 = rand() % 16;
19    if ( v8 <= 9 )
20        v1 = 48;
21    else
22        v1 = 49;
23    *v11 = v1;
24    v11[1] = v8 % 0xA + 48;
25    for ( i = 2; i <= 2 * v10; i = v5 + 1 )
26    {
27        v2 = v8++;
28        v7 = a1[(i >> 1) - 1] ^ off_202010[v2];
29        if ( (char)(v7 >> 4) > 9 )
30            v3 = (v7 >> 4) + 55;
31        else
32            v3 = (v7 >> 4) + 48;
33        v11[i] = v3;
34        if ( (v7 & 0xF) > 9 )
35            v4 = (v7 & 0xF) + 55;
36        else
37            v4 = (v7 & 0xF) + 48;
38        v5 = i + 1;
39        v11[v5] = v4;
40    }
41    v11[i] = 0;
42    return v11;
43 }
```

Cum funcția returnează **v11**, putem deduce ca aceasta variabila contine input-ul criptat. După puțină muncă, ne putem da seama ce face, de fapt, programul: ia input-ul și face XOR dintre acesta și o cheie din memorie, apoi reprezintă output-ul în hex. Cheia cu care se face XOR-ul este, de fapt, stringul găsit anterior cu ajutorul programului **strings**. Putem găsi flag-ul folosind python:

```
yakuhito@furry-catstation:~/ctf/unr2/mrrobot$ python
Python 3.6.9 (default, Oct 8 2020, 12:12:24)
[GCC 8.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from pwn import xor
>>> xor(bytes.fromhex('013032224029145C2047711D11562831021F077A1406782B28'),
```

Resurse utile pentru incepatori din UNbreakable România

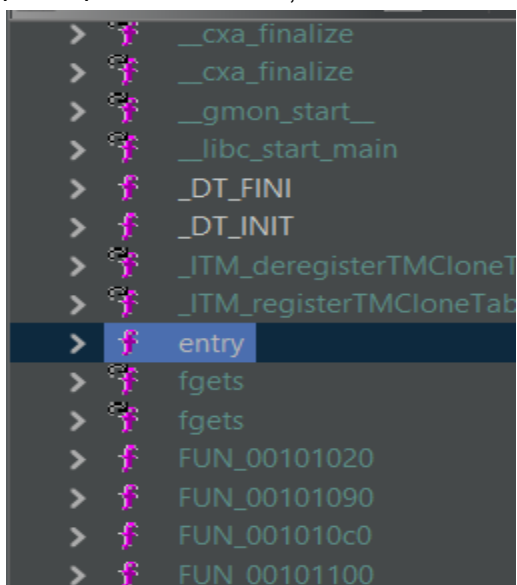
```
b'dsfd;kfoA,.iyewrkldJKDHSUBsgvca69834ncxv')  
b'eCTF{Br3ak_th3_Cisc0_B0x}CCUT#H"e\x18tEsr.^[  
>>>
```

O alta metoda de rezolvare a exercitiului este recunoașterea algoritmului folosit pentru criptarea inputului, fie din formatul flag-ului criptat, fie din codul sursa reasamblat: stringul criptat e un **CISCO Router Password Hash** si poate fi decodat cu ajutorul [acestui site](https://blog.kuhi.to/unbreakable-romania-2-writeup#mrrobot).

Rezolvare în engleză: <https://blog.kuhi.to/unbreakable-romania-2-writeup#mrrobot>

The_pass_pls (usor-mediu)

Voi folosi Ghidra pentru a decompila acest executabil. După ce o sa îl incarcam, vom observa prima problema. Funcția **main** nu “exista”.



De fapt, funcția exista, dar numele a fost șters la compilare. Cand nu găsim funcția **main**, funcția pe care trebuie sa o respectam este **entry**.

```
void entry(undefined8 param_1,undefined8 param_2,undefined8 param_3)  
{  
    undefined8 in_stack_00000000;  
    undefined auStack8 [8];  
  
    __libc_start_main(FUN_00101199,in_stack_00000000,&stack0x00000008,FUN_00101200,FUN_00101260,  
        param_3,auStack8);  
  
    do {  
        /* WARNING: Do nothing block with infinite loop */  
    } while( true );  
}
```

Resurse utile pentru incepatori din UNbreakable România

Funcția **entry** apelează funcția [__libc_start_main](#), funcție importantă (face parte din [LIBC](#), împreună cu alte funcții, precum `fgets`, `gets`, `puts`, etc.).

Synopsis

```
int __libc_start_main(int (*main) (int, char * *, char * *), int argc, char * * ubp_av, void (*init) (void), void (*fini) (void), void (*rtld_fini) (void), void (*stack_end));
```

Description

The `__libc_start_main()` function shall perform any necessary initialization of the execution environment, call the `main` function with appropriate arguments, and handle the return from `main()`. If the `main()` function returns, the return value shall be passed to the `exit()` function.

Funcția **__libc_start_main**, pe langa altele, apelează funcția **main**, primind ca prim parametru un pointer către funcția **main**. Deci, **FUN_00101199** este **main**. Redenumim funcția, si o accesam.

```
undefined8 main(void)

{
    char cVar1;
    char local_38 [48];

    puts("Salutare. Pentru a continua, introduceti parola: ");
    fgets(local_38,0x30,stdin);
    cVar1 = FUN_00101145(local_38);
    if (cVar1 == '\0') {
        puts("Parola incorecta. La revedere");
    }
    else {
        puts("Parola corecta. Bine v-am regasit.");
    }
    return 0;
}
```

Funcția afișează pe ecran “*Salutare.....*”, citește de la tastatura maximum 0x30 caractere în variabila `local_38`, și apoi apelează funcția **FUN_00101145** cu parametrul `local_38`. Apoi, în funcție de valoarea returnată de funcție, hotărăște dacă parola este corectă sau nu. Inspectăm funcția **FUN_00101145**:

Resurse utile pentru incepatori din UNbreakable România

```
undefined FUN_00101145(long param_1)
{
    int local_10;

    local_10 = 0;
    while( true ) {
        if (0x1e < local_10) {
            return 1;
        }
        if ((* (byte *) (param_1 + local_10) ^ 0xf3) != (&DAT_00104060)[local_10]) break;
        local_10 = local_10 + 1;
    }
    return 0;
}
```

Funcția declara variabila **local_10** si o inițializează cu 0. La intrare in **while**, se verifica dacă **local_10** este mai mare decat **0x1e**. Daca este mai mare, returnează **1**. Cand ne uitam in **main**, orice alta valoare in afara de 0 înseamnă “Parola corecta”. Apoi, un **if** care verifica dacă parametrul + **local_10**, xorat cu 0xf3 este diferit de ce se afla la **&DAT_00104060[local_10]**. După **if**, se incrementeaza **local_10**. Pentru a face funcția mai citibila, trebuie sa facem niște ajustări. In primul rand, parametrul funcției este **char ***, nu **long**. Ghidra a observat ca are size-ul 8, si s-a gandit ca este long int. Putem sa redenumim și **local_10** la **index**, deoarece este clar ca este folosit ca index pentru parametru.

```
undefined FUN_00101145(long param_1)
{
    int local_10;

    local_10 = 0;
    while( true ) {
        if (0x1e < local_10)
            return 1;
    }
    if ((* (byte *) (param_
        local_10 = local_10 +
    }
    return 0;
})
```

Edit Function Signature	
Rename Variable	L
Retype Variable	Ctrl-L
Auto Create Structure	Shift-Open Bracket
Commit Params/Return	P
Commit Local Names	
Highlight	>
Secondary Highlight	>
Copy	Ctrl-C
Comments	>
Find...	Ctrl-F
References	>
Properties	

```
undefined FUN_00101145(char *param_1)
{
    int index;

    index = 0;
    while( true ) {
        if (0x1e < index) {
            return 1;
        }
        if ((byte)(param_1[index] ^ 0xf3U) != (&DAT_00104060)[index]) break;
        index = index + 1;
    }
    return 0;
}
```

Putem observa că acest **while** poate fi rescris ca un **for** (for este mai favorabil decât while, deoarece este mai ușor de urmărit) :

```
for(index = 0; index < 0x1e; index++)
{
    if((byte)param_1[index] ^ 0xf3U) != (&DAT_00104060)[index])
        return 0;
}
return 1;
```

Variabilele **DAT_** sunt de fapt variabile globale. Deci acest **if** xoreaza fiecare caracter din input cu 0xf3, și verifica dacă este diferit de ce se afla în variabila globală **DAT_00104060**. Dacă este diferit, se executa **break**, și funcția returnează **0**, ceea ce înseamnă ca parola nu este corecta. Folosindu-ne de proprietățile operației XOR, putem afirma ca:

$$A \oplus B = C \Leftrightarrow B \oplus C = A$$

În cazul nostru, **A** = caracterele din input, **B** = 0xf3, iar **C** = caracterele din variabila globala. Deci, ca sa aflam **A**-ul(caracterele din input, astfel incat daca le xoram, sa fie egale cu variabila globală), trebuie să xoram **B**-ul cu **C**-ul.

Accesam variabila **DAT_00104060**, pentru a copia caracterele.

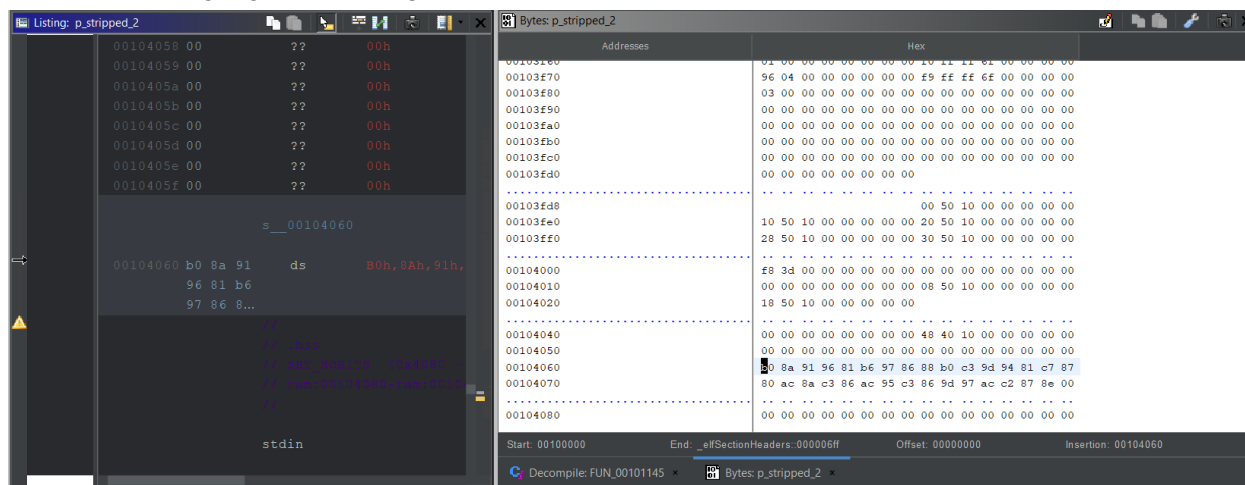
Resurse utile pentru incepatori din UNbreakable România

DAT_00104060		
00104060	b0	?? B0h
00104061	8a	?? 8Ah
00104062	91	?? 91h
00104063	96	?? 96h
00104064	81	?? 81h
00104065	b6	?? B6h
00104066	97	?? 97h
00104067	86	?? 86h
00104068	88	?? 88h
00104069	b0	?? B0h



Le putem copia de mana, sau ne putem folosi de **Display Bytes**.

Activam **Display Bytes**, si mergem cu cursorul la variabila noastra(00104060).



Selectam cei 31 de byte și (31 pentru pentru că la 31(0x1f), while se oprește), ii copiem, și îi introducem într-un script de python.

```
encrypted =
b"\xb0\x8a\x91\x96\x81\xb6\x97\x86\x88\xb0\xc3\x9d\x94\x81\xc7\x87\x80\xac\x8a\xc3\x86\xac\x95\xc3\x86\x9d\x97\xac\xc2\x87\x8e"

flag = "".join([chr(i ^ 0xf3) for i in encrypted])

print(flag)
```

Si gasim parola/flag-ul:

Resurse utile pentru incepatori din UNbreakable România

```
edmund@DESKTOP-FC4GM8U:/mnt/d/CTFS/cyberedu/exer$ ./the_pass_pls
Salutare. Pentru a continua, introduceti parola:
CyberEdu{<DATA EXPUNGED>}
Parola corecta. Bine v-am regasit.
```

Crazy-Number (ușor)

Concurs: UNbreakable 2021 #Individual

Descriere:

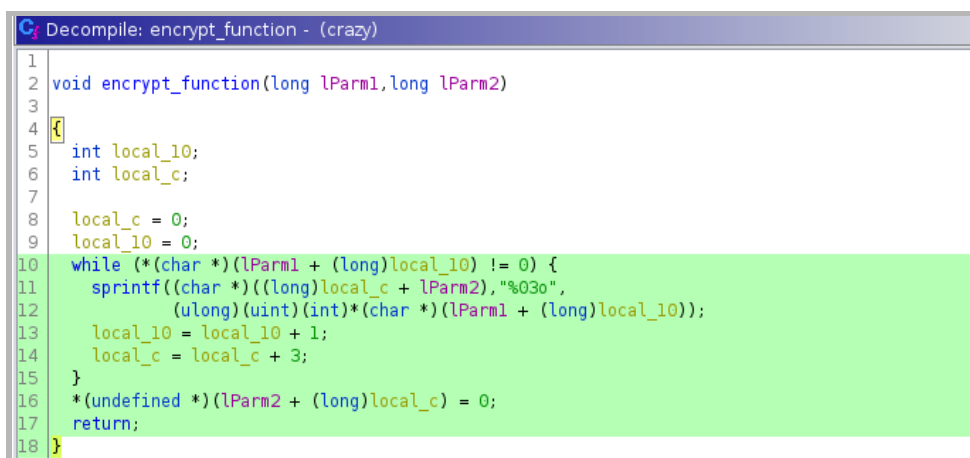
Hi edmund. I have some problem with this strange message
(10312410617307106706214406206006607014514406107106106414306514214607014314
50640640600710711440610640660640671410650631431460630610610631460701450600
62061060065071063146144071144066071061144145066067062064175). Can you help me
to figure out what it is?

Format flag: CTF{sha256}

Rezolvare:

Folosind Ghidra, apelați `encrypt_function` și acolo veți vedea un loop care ia fiecare caracter din șirul furnizat și folosește `sprintf((char *)((long)local_c + lParm2), "%03o")` pentru a pune toate caracterele în zona de memorie care conține șirul criptat.

În acest caz, `"%03o"` este cunoscut sub numele de `format string` - acesta modifică datele de intrare într-un mod previzibil. Vă arată cum `"%03o"` convertește caracterul în octal și alătura numărului rezultat zerouri până când șirul ajunge la o lungime de 3.



```
C: Decompiler: encrypt_function - (crazy)
1
2 void encrypt_function(long lParm1, long lParm2)
3
4 {
5     int local_10;
6     int local_c;
7
8     local_c = 0;
9     local_10 = 0;
10    while (*(char *) (lParm1 + (long) local_10) != 0) {
11        sprintf((char *) ((long) local_c + lParm2), "%03o",
12              (ulong) (uint) (int) *(char *) (lParm1 + (long) local_10));
13        local_10 = local_10 + 1;
14        local_c = local_c + 3;
15    }
16    *(undefined *) (lParm2 + (long) local_c) = 0;
17    return;
18 }
```

Exploit final:

Resurse utile pentru incepatori din UNbreakable România

```
def octal_to_str(octal_str):
    str_converted = ""
    for octal_char in octal_str.split(" "):
        str_converted += chr(int(octal_char, 8))
    return str_converted

print(octal_to_str("103 124 106 173 071 067 062 144 062 060 066 070 145 144 061 071 061
064 143 065 142 146 070 143 145 064 064 060 071 071 144 061 064 066 064 067 141 065
063 143 146 063 061 061 063 146 070 145 060 062 061 060 065 071 063 146 144 071 144
066 071 061 144 145 066 067 062 064 175 000"))
```

```
darius@bit-sentinel:~/Desktop/unbreakable/unr21-1/rev/crazy-number$ python solver.py
CTF{972d2068ed1914c5bf8ce44099d14647a53cf3113f8e0210593fd9d691de6724}
```

Defuse-the-bomb (medium)

Concurs UNbreakable 2021 #Individual

Descriere:

You are the last CT alive, you have a defuse kit, and the bomb is planted.
You need to hurry, but what??
Those Terrorists made the bomb defusal-proof...they locked it with a password.
Find the password before the bomb explodes.

Flag format: CTF{sha256}

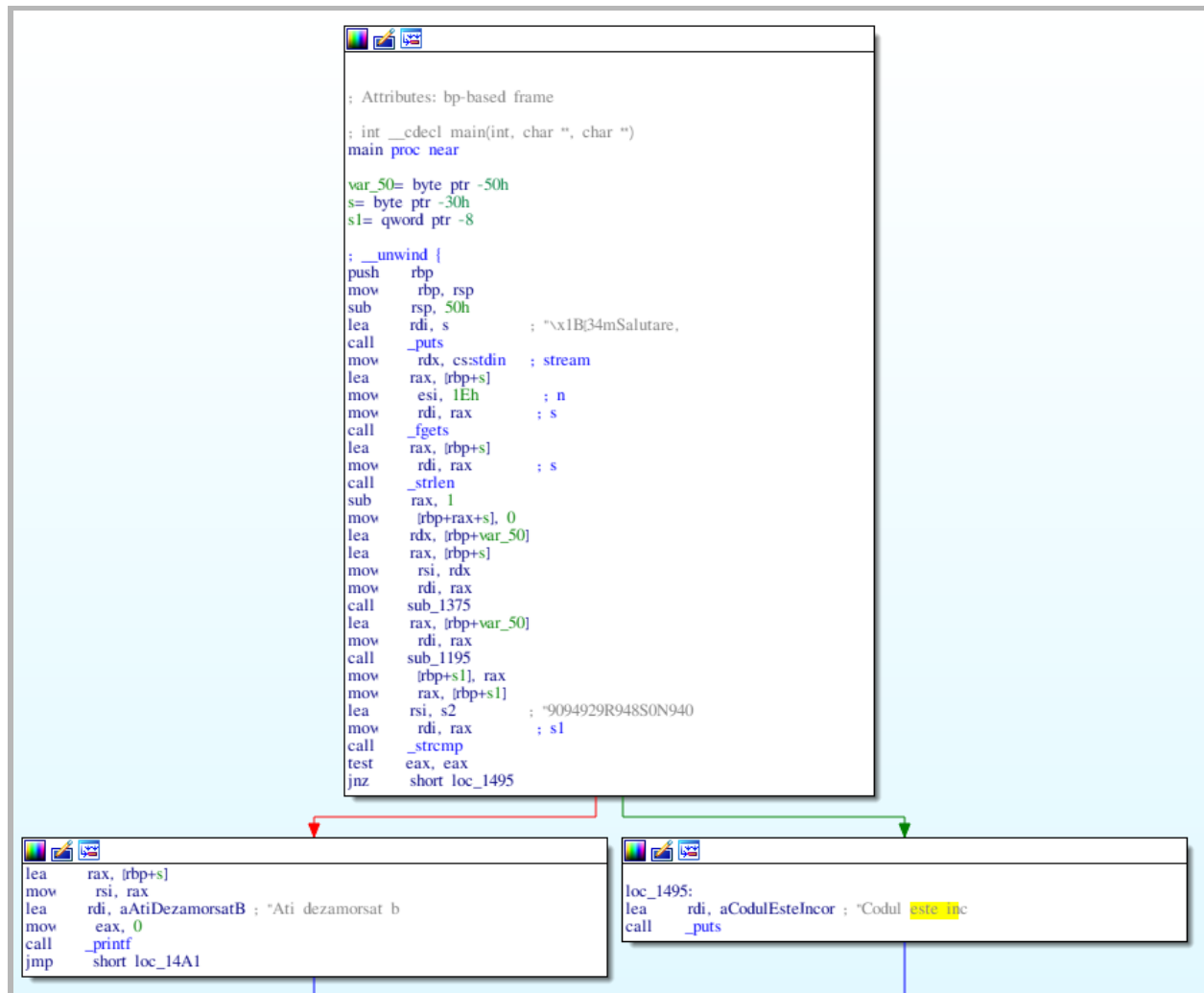
Rezolvare:

Fișierul primit pare să fie un executabil ce are nevoie de o parolă pentru execuție:

```
yakuhito@furry-catstation:~/ctf/unr21-ind$ file defuse_kit
defuse_kit: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), dynamically linked,
interpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux 3.2.0,
BuildID[sha1]=c8fa35d3efd7268cc1a7129249cab4eb20afd030, stripped
yakuhito@furry-catstation:~/ctf/unr21-ind$ chmod +x defuse_kit
yakuhito@furry-catstation:~/ctf/unr21-ind$ ./defuse_kit
Salutare, CT. Introdu codul pentru dezamorsarea bombei:
1337
Codul este incorect. Bomba a explodat. Iar ai ajuns in silver II.
yakuhito@furry-catstation:~/ctf/unr21-ind$
```

Resurse utile pentru incepatori din UNbreakable România

Să deschidem fișierul binar în IDA Pro. Din nefericire, nu toate funcțiile au un nume clar, ceea ce înseamnă că este posibil ca binarul să fi fost golit de simboluri. Cu toate acestea, IDA identifică automat funcția principală, care arată astfel:



Lopp-ul ia fiecare caracter din șirul de intrare și îl procesează folosind următorul cod:

```
mov     eax, [rbp+var_4]
movsxd  rdx, eax
mov     rax, [rbp+var_18]
add     rax, rdx
movzx   eax, byte ptr [rax]
movsx   eax, al
mov     edx, [rbp+var_8]
movsxd  rcx, edx
mov     rdx, [rbp+var_20]
add     rcx, rdx
```

```
mov    edx, eax
lea    rsi, format    ; "%02X"
mov    rdi, rcx        ; s
mov    eax, 0
call   _sprintf
add    [rbp+var_4], 1
add    [rbp+var_8], 2
```

Codul de mai sus apelează **sprintf** folosind șirul **"%02X"** ca parametru **"format"**. Aceasta va lua caracterul care este procesat, îl va converti în hexazecimal și îl va umple cu zerouri până când va ajunge la o lungime de 2. Practic, această funcție ia un șir de caractere și îl convertește în hexazecimal. Să ne uităm acum la cealaltă funcție, **sub_1195**.

Funcția are o mulțime de ramificații (care se traduc prin if-uri în limbaje de nivel superior, cum ar fi C). De fapt, **sub_1195** este o implementare simplă a algoritmului de codificare **ROT13**.

Știind cum este transformată intrarea înainte de a fi comparată cu șirul codificat, putem crea un script de rezolvare care să recupereze parola originală:

```
enc = "9094929R948S0N94039496920794"

def sub_1195(s):
    dec = ""
    for ch in s:
        ch = ord(ch)
        if ch > 65 and ch < 90: # 65 = 'A'; 90 = 'Z'
            ch = ch + 13
            if ch > 90:
                ch = ch - 90 + 65 - 1
            dec += chr(ch)
        elif ch >= 97 and ch <= 122: # 97 = 'a'; 122 = 'z'
            ch = ch + 13
            if ch > 122:
                ch = ch - 122 + 97 - 1
            dec += chr(ch)
        elif ch >= 48 and ch <= 57: # 48 = '0'; 57 = '9'
            ch = ch - 48 + 13
            ch = ch % 10
            ch = ch + 48
            dec += chr(ch)
        else:
            dec += chr(ch)
```

```
return dec

def rev_sub_1195(s):
    dec = ""
    for ch in s:
        ch = ord(ch)
        # the hardcoded value doesn't contain lowercase chars
        if ch >= ord('A') and ch <= ord('Z'):
            ch = ch + 13
            if ch > ord('Z'):
                ch = ch - ord('Z') + ord('A') - 1
        else:
            ch = ch - ord('0')
            ch = (ch - 13) % 10
            ch = ch + ord('0')
        dec += chr(ch)
    return dec

def sub_1375(s):
    return s.encode().hex() # shortcut :)

def rev_sub_1375(s):
    return bytes.fromhex(s).decode()

flag = rev_sub_1375(rev_sub_1195(enc))
print(flag)
```

Notă: Am întâlnit o piedică în timpul acestei provocări. În timp ce $\text{rot13}(\text{rot13}(\text{caracter})) = \text{caracter}$ este adevărat pentru caracterele minuscule și majuscule, nu este adevărat pentru cifre (lungimea alfabetului este $\text{len}('0123456789') = 10$). Acest lucru înseamnă că funcția de decriptare trebuie să sustragă 13 din toate cifrele în loc să-l adauge, deoarece adăugarea ar avea ca rezultat o valoare diferită de cea inițială.

Rularea scriptului de mai sus va imprima intrarea corectă.

```
yakuhito@furry-catstation:~/ctf/unr21-ind$ python solve.py
gaina_zapacita
yakuhito@furry-catstation:~/ctf/unr21-ind$ ./defuse_kit
Salutare, CT. Introdu codul pentru dezamorsarea bombei:
gaina_zapacita
```


Resurse utile pentru incepatori din UNbreakable România

```
Ati dezamorsat bomba cu succes.  
+300$  
Flag-ul este ctf{sha256(gai[REDACTAT]ita)}  
yakuhto@furry-catstation:~/ctf/unr21-ind$
```

Pentru cei curioși, mai jos puteți găsi și codul sursă al binarului folosit în acest exercițiu:

```
#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
  
char encrypted[] = "9094929R948S0N94039496920794";  
  
char *rot13(char *string) {  
    char *rot13d = malloc(strlen(string));  
    for(int i = 0; i < strlen(string); i++)  
    {  
        if(string[i] >= 65 && string[i] <= 90)  
        {  
            rot13d[i] = string[i] + 13;  
            if(rot13d[i] > 90)  
            {  
                rot13d[i] = rot13d[i] - 90 + 65 - 1;  
            }  
        }  
        else if(string[i] >= 97 && string[i] <= 122)  
        {  
            int j = (int)string[i];  
            j = j + 13;  
            if(j > 122) {  
                j = j - 122 + 97 - 1;  
            }  
            rot13d[i] = j;  
        }  
        else if(string[i] >= '0' && string[i] <= '9')  
        {  
            rot13d[i] = (string[i] - '0' + 13) % 10 + '0';  
        }  
        else {  
            rot13d[i] = string[i];  
        }  
    }  
}
```

```
    return rot13d;
}

void string2hexString(char* input, char* output)
{
    int loop;
    int i;

    i=0;
    loop=0;

    while(input[loop] != '\0')
    {
        sprintf((char*)(output+i), "%02X", input[loop]);
        loop+=1;
        i+=2;
    }
    //insert NULL at the end of the output string
    output[i++] = '\0';
}

int main()
{
    char input[30], tohex[30], *rotted;

    puts("\033[34mSalutare, CT. Introdu codul pentru dezamorsarea bombei: \033[00m");
    fgets(input, 30, stdin);

    input[strlen(input)-1] = '\x00';
    string2hexString(input, tohex);
    rotted = rot13(tohex);

    if(!strcmp(rotted, encrypted))
    {
        printf("Ati dezamorsat bomba cu succes.\n\033[92m+300$\033[00m\nFlag-ul este\nctf{sha256(%s)}\n", input);
    }
    else
    {
        printf("Codul este incorect. Bomba a explodat. Iar ai ajuns in silver II.\n");
    }
    return 0;
}
```

```
}
```

Flag: CTF{c63344dea9cdc97a00f20edca0867575292141b74021560c29c6a4429888d832}

This-is-wendys (mediu)

Concurs UNbreakable 2021 #Echipe

Descriere:

You went to Wendy's and found this weird keypad. What does it open?

Format flag: CTF{sha256}

Rezolvare:

Fișierul dat cu extensia .exe dată este o aplicație .NET - puteți utiliza dnSpy pentru a vedea codul sursă. Sistemul de operare este "Windows_NT". Pentru referință, iată codul sursă original:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace this_is_wendys
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();

            public char[] reee(string a, string b)
```

```
{
    char[] c = new char[b.Length];

    for(int i=0; i<b.Length ;i++)
    {
        c[i] = (char)(a[i%8] ^ b[i]);
    }
    return c;
}

private void button1_Click(object sender, EventArgs e)
{
    //flag = ctf{c384d5fdbdb1208ce9de3d2[REDACTAT]f6d5c6736fdbb793ab5}
    String s;
    char[] ham;
    string something =
"\x34\x1d\x08\x1f\x0c\x44\x4b\x6b\x33\x5c\x08\x00\x0d\x13\x11\x6e\x65\x59\x56\x07\x0a\x4e\x17\x3a\x64\x0d\x5c\x52\x57\x46\x46\x6f\x66\x58\x5e\x07\x0e\x45\x4b\x6d\x33\x5a\x0d\x56\x5b\x11\x46\x3e\x35\x0f\x58\x00\x5a\x14\x45\x68\x64\x5f\x08\x00\x0d\x15\x44\x66\x64\x08\x0c\x51\x12";

    s = Environment.GetEnvironmentVariable("OS");
    try
    {
        ham = reee(s,
System.Text.Encoding.UTF8.GetString(Convert.FromBase64String(textBox1.Text)));
        if (something == string.Concat(ham))
        {
            label2.Text = "Congrats. The flag is " +
System.Text.Encoding.UTF8.GetString(Convert.FromBase64String(textBox1.Text));
        }
        else
        {
            label2.Text = "You are close?";
        }
    }
    catch(FormatException)
    {
        label2.Text = "Try again.";
    }
}
```

```
}  
}
```

Flag: ctf{c384d5fdbdb1208ce9d[REDACTAT]4f5abf6d5c6736fdbb793ab5}

Mastermind (mediu)

Concurs UNbreakable 2021 #Echipe

Descriere:

Are you ready enough to be a master of the mind?

Flag format: CTF{sha256}

Rezolvare:

```
yakuhito@furry-catstation:~/ctf/unr21-tms$ file mastermind.bin  
mastermind.bin: ASCII text  
yakuhito@furry-catstation:~/ctf/unr21-tms$ head -n 5 mastermind.bin  
\x55\x48\x89\xe5\x41\x57\x41\x56  
\x41\x55\x41\x54\x53\x48\x81\xec  
\x18\x08\x00\x00\x64\x48\x8b\x04  
\x25\x28\x00\x00\x00\x48\x89\x45  
\xc8\x31\xc0\xc7\x85\xcc\xf7\xff  
yakuhito@furry-catstation:~/ctf/unr21-tms$
```

Putem converti datele folosind un script Python format din 3 linii:

```
data = open("mastermind.bin", "r").read().replace("\n", "").replace(" ", "")  
data = eval('b' + data + '')  
open("mastermind", "wb").write(data)
```

Cu toate acestea, formatul fișierului rămâne necunoscut. Deoarece aceasta este o provocare de inginerie inversă, am putea veni cu ideea de a rula datele de intrare ca și cod shell - și asta este ceea ce vom face! Iată codul sursă C al unui program care va rula shellcode-ul:

```
#include <sys/mman.h>  
#include <string.h>  
#include <stdio.h>  
  
char sc[] = "\x55\x48...\x5d\xc3";
```

Resurse utile pentru incepatori din UNbreakable România

```
int main(){
void * a = mmap(0, 4096, PROT_EXEC | PROT_READ | PROT_WRITE, MAP_ANONYMOUS
| MAP_SHARED, -1, 0);

printf("allocated executable memory at: %p\n", a);

((void (*)(void)) memcpy(a, sc, sizeof(sc)))();}
```

După compilarea și rularea fișierului binar cu gdb atașat, putem vedea un string ciudat în memoria programului:

```
yakuhito@furry-catstation:~/ctf/unr21-tms$ gcc run.c
yakuhito@furry-catstation:~/ctf/unr21-tms$ gdb ./a.out
GNU gdb (Ubuntu 8.1-0ubuntu3.2) 8.1.0.20180409-git
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./a.out...(no debugging symbols found)...done.
gdb-peda$ r
Starting program: /home/yakuhito/ctf/unr21-tms/a.out
allocated executable memory at: 0x7ffff7ff6000
^C
Program received signal SIGINT, Interrupt.

[-----registers-----]
RAX: 0x0
RBX: 0x5420212152454452 ('RDER!! T')
RCX: 0x0
RDX: 0x7fffffff420 --> 0x0
RSI: 0x4544524148205952 ('RY HARDE')
RDI: 0x7fffffff540 --> 0x555500000000 ("")
RBP: 0x7fffffffdb80 --> 0x7fffffffdba0 --> 0x555555554740 (<__libc_csu_init>: push r15)
```

```
RSP: 0x7fffffff340 --> 0x2
RIP: 0x7ffff7ff6203 --> 0xc283480289fa8948
R8 : 0x4452414820595254 ('TRY HARD')
R9 : 0x5952542021215245 ('ER!! TRY')
R10: 0x2152454452414820 (' HARDER!')
R11: 0x4148205952542021 ('! TRY HA')
R12: 0x5241482059525420 (' TRY HAR')
R13: 0x5254202121524544 ('DER!! TR')
R14: 0x2059525420212152 ('R!! TRY ')
R15: 0x2121524544524148 ('HARDER!!')
EFLAGS: 0x246 (carry PARITY adjust ZERO sign trap INTERRUPT direction overflow)
[-----code-----]
0x7ffff7ff61f8: mov    ecx,0x24
0x7ffff7ff61fd: mov    rdi,rdx
0x7ffff7ff6200: rep stos QWORD PTR es:[rdi],rax
=> 0x7ffff7ff6203: mov    rdx,rdi
0x7ffff7ff6206: mov    DWORD PTR [rdx],eax
0x7ffff7ff6208: add    rdx,0x4
0x7ffff7ff620c: movabs rax,0x4b4e554a4b4e554a
0x7ffff7ff6216: movabs rdx,0x65475231517c7c7c
[-----stack-----]
0000| 0x7fffffff340 --> 0x2
0008| 0x7fffffff348 --> 0xa00000000 ("")
0016| 0x7fffffff350 ("TRY HARDER!! TRY HARDER!! TRY HARDER!! TRY HARDER!! TRY
TRY HARDER!! TRY HARDER!! TRY HARDER!! TRY HARDER!! TRY HARDER!! TRY HARDER!!
TRY HARDER!! TRY HARDER!! TRY HARDER!! TRY HARDER!! TRY HARDER!! TRY H" ...)
0024| 0x7fffffff358 ("ER!! TRY HARDER!! TRY HARDER!! TRY HARDER!! TRY HARDER!!
TRY HARDER!! TRY HARDER!! TRY HARDER!! TRY HARDER!! TRY HARDER!! TRY
TRY HARDER!! TRY HARDER!! TRY HARDER!! TRY HARDER!! TRY HARDER!! TRY
HARDER!!")
0032| 0x7fffffff360 (" HARDER!! TRY HARDER!! TRY HARDER!! TRY HARDER!! TRY
TRY HARDER!! TRY HARDER!! TRY HARDER!! TRY HARDER!! TRY HARDER!! TRY HARDER!!
TRY HARDER!! TRY HARDER!! TRY HARDER!! TRY HARDER!! TRY HARDER!!")
0040| 0x7fffffff368 ("! TRY HARDER!! TRY HARDER!! TRY HARDER!! TRY HARDER!! TRY
TRY HARDER!! TRY HARDER!! TRY HARDER!! TRY HARDER!! TRY HARDER!! TRY HARDER!!
TRY HARDER!! TRY HARDER!! TRY HARDER!! TRY HARDER!!")
0048| 0x7fffffff370 ("RDER!! TRY HARDER!! TRY HARDER!! TRY HARDER!! TRY
TRY HARDER!! TRY HARDER!! TRY HARDER!! TRY HARDER!! TRY HARDER!! TRY HARDER!!
TRY HARDER!! TRY HARDER!! TRY HARDER!! TRY HARDER!!")
0056| 0x7fffffff378 ("RY HARDER!! TRY HARDER!! TRY HARDER!! TRY HARDER!! TRY
TRY HARDER!! TRY HARDER!! TRY HARDER!! TRY HARDER!! TRY HARDER!! TRY HARDER!!
TRY HARDER!! TRY HARDER!! TRY HARDER!!")
```

Resurse utile pentru incepatori din UNbreakable România

```
[-----]
Legend: code, data, rodata, value
Stopped reason: SIGINT
0x00007ffff7ff6203 in ?? ()
gdb-peda$ stack 100
0000| 0x7fffffd340 --> 0x2
0008| 0x7fffffd348 --> 0xa00000000 (")
0016| 0x7fffffd350 ("TRY HARDER!! TRY HARDER!! TRY HARDER!! TRY HARDER!! TRY
TRY HARDER!! TRY HARDER!! TRY HARDER!! TRY HARDER!! TRY HARDER!! TRY HARDER!! TRY HARDER!! TRY H" ...)
[...]
0528| 0x7fffffd550
("JUNKJUNK|||Q1RGezhmOTMxYzNhYTdjMThjNWEzZWFiYjg0OGRhYTkwZDc2MTk3ZWQ
zNDAXNDhhYTcwMmRIOTVkdODI4OGIxOTBhZWV9|||JUNKJUNKJUNK")
[...]
--More--(100/100)
gdb-peda$
```

Pentru a obține steagul, trebuie doar să decodăm în baza64 șirul dintre "JUNKJUNK " și "JUNKJUNK".

Flag: CTF{8f931c3aa7c18c5a3eab[REDACTAT]148aa702de95d8288b190aee}

AgoodOne (easy)

Concurs UNbreakable 2021 #Individual

Autor exercițiu: Octavian Purcaru

Contribuitori rezolvare: Niță Horia, Valentina Galea

Descriere:

One would simply want to be with the rest.

Rezolvare:

Pentru această probă vom analiza fișierul pe o mașină Linux folosind Ghidra și Python.

Binary file

Vom folosi comanda file din terminal pentru a vedea rapid cu ce tip de fișier avem de-a face.


```
file agoodone
file agoodone
agoodone: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV),
dynamically linked
```

Putem observa că este un fișier ELF (Executable and Linkable Format) și rulează pe 64 de biți. Aceasta înseamnă că, teoretic, îl putem executa pe mașina noastră Linux.

Analysis

Următoarele comenzi pot fi utilizate pentru a obține informații semnificative despre fișierul nostru înainte de a ne trece la analiza propriu-zisă.

```
strings agoodone
```

Comanda strings caută șiruri de caractere imprimabile într-un fișier. Un șir de caractere este orice secvență de 4 sau mai multe caractere imprimabile care se termină cu o nouă linie sau cu un caracter nul. Comanda strings este utilă pentru identificarea fișierelor cu obiecte aleatorii.

```
u+UH
[]A\A]A^A_
>#&v$qt$pr##tur}s$w#!'##&$!t} #qr $r}!qws$qr!u|r$q| v}uv#r |&u |s8
Usage: %s <key_value>
Voil
4, Correct Password!
:*3$
Try again or go home
Do you even try? :P
What was that?
Please shut down your computer
Really?
Of course you tried that
Still trying LOL
Is it that hard to enter the correct password
My dog can do reverse engineering better than you.
Just don't touch it again
Don't do that again pls pls pls
```

Resurse utile pentru incepatori din UNbreakable România

```
I am not sure what to tell you at this point...  
GCC: (Ubuntu 10.3.0-1ubuntu1) 10.3.0
```

Din output-ul generat putem vedea că există câteva șiruri de caractere care indică un mecanism ce ne cere o parolă și o mulțime de mesaje de eșec.

Acum este momentul să schimbăm permisiunile binarului și să îl executăm pentru a putea analiza viitoarele acțiuni.

```
chmod +x agoodone  
./agoodone  
Usage: ./agoodone <key_value>  
39395555555555  
Is it that hard to enter the correct password
```

Am încercat câteva parole la întâmplare, dar mă îndoiesc că acest lucru ar trebui să se întâmple.

ltrace este un program care execută pur și simplu comanda specificată până când iese. Acesta interceptează și înregistrează apelurile la bibliotecile dinamice care sunt apelate de procesul executat și semnalele care sunt primite de acest proces.

```
ltrace ./agoodone  
Usage: ./agoodone <key_value>  
+++ exited (status 255) +++
```

După execuția acestei comenzi, nu observăm nimic interesant, însă în această direcție mai putem folosi și **strace**.

strace este un instrument puternic ce se folosește în analiza programelor specifice sistemelor de operare Unix, cum ar fi Linux. Acesta captează și înregistrează toate apelurile de sistem efectuate de un proces și semnalele primite de respectivul proces.

```
strace ./agoodone  
execve("./agoodone", ["./agoodone"], 0x7fff292d9af0 /* 63 vars */) = 0  
brk(NULL) = 0x5604a494d000  
arch_prctl(0x3001 /* ARCH_??? */, 0x7ffd6efb43f0) = -1 EINVAL (Invalid  
argument)  
access("/etc/ld.so.preload", R_OK) = -1 ENOENT (No such file or
```

```
directory)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=76146, ...},
AT_EMPTY_PATH) = 0
mmap(NULL, 76146, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f2a9911c000
close(3) = 0
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3,
"\177ELF\2\1\1\3\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\240\206\2\0\0\0\0"...,
832) = 832
pread64(3,
"\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784,
64) = 784
pread64(3, "\4\0\0\0
\0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0"..., 48, 848) = 48
pread64(3,
"\4\0\0\0\24\0\0\0\3\0\0\0GNU\0+H)\227\201T\214\233\304R\352\306\3379\220%"
..., 68, 896) = 68
newfstatat(3, "", {st_mode=S_IFREG|0755, st_size=1983576, ...},
AT_EMPTY_PATH) = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1,
0) =
0x7f2a9911a000
pread64(3,
"\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784,
64) = 784
mmap(NULL, 2012056, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) =
0x7f2a98f2e000
mmap(0x7f2a98f54000, 1486848, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x26000) = 0x7f2a98f54000
mmap(0x7f2a990bf000, 311296, PROT_READ,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x191000) = 0x7f2a990bf000
mmap(0x7f2a9910b000, 24576, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1dc000) = 0x7f2a9910b000
mmap(0x7f2a99111000, 33688, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f2a99111000
close(3) = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1,
0) =
0x7f2a98f2c000
arch_prctl(ARCH_SET_FS, 0x7f2a9911b580) = 0
mprotect(0x7f2a9910b000, 12288, PROT_READ) = 0
```

Resurse utile pentru incepatori din UNbreakable România

```
mprotect(0x5604a4009000, 4096, PROT_READ) = 0
mprotect(0x7f2a99161000, 8192, PROT_READ) = 0
munmap(0x7f2a9911c000, 76146) = 0
newfstatat(1, "", {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0), ...},
AT_EMPTY_PATH) = 0
brk(NULL) = 0x5604a494d000
brk(0x5604a496e000) = 0x5604a496e000
write(1, "Usage: ./agoodone <key_value>\n", 30Usage: ./agoodone <key_value>
) = 30
exit_group(-1) = ?
+++ exited with 255 +++
```

Din nou, rezultatul acestei comenzi nu are nicio semnificație reală pentru noi (niciun apel interesant).

Niciuna dintre comenzile de mai sus nu ne-a oferit informații utile.

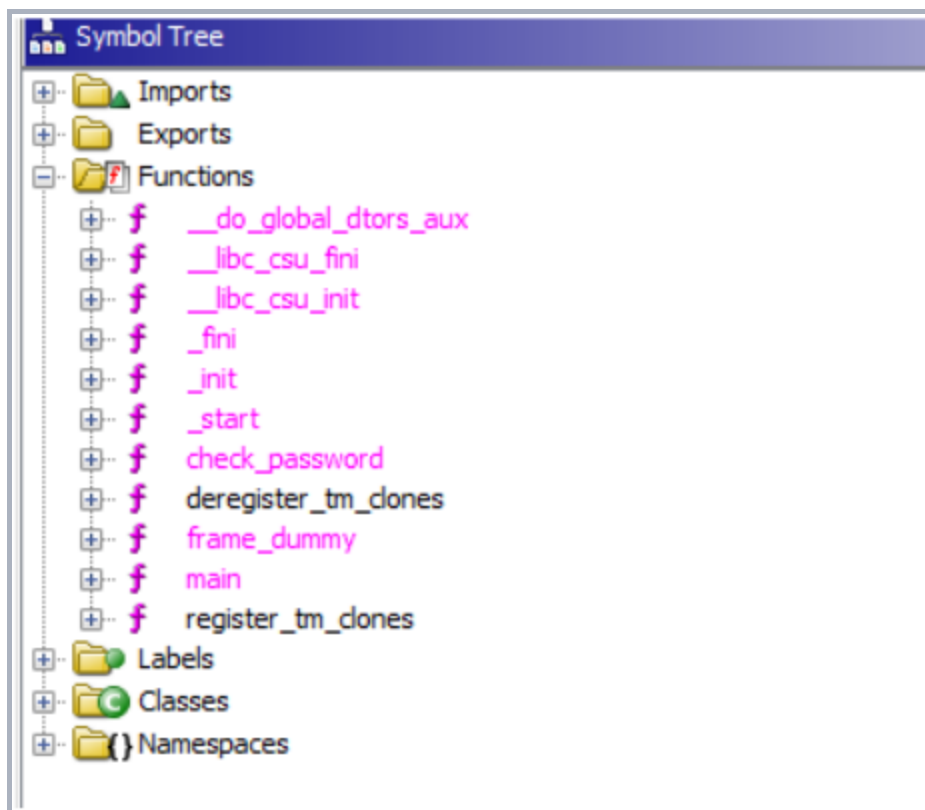
Așadar, va trebui să facem inginerie inversă legitimă.

Vom folosi Ghidra, dar puteți efectua analiza în orice decompiler la alegere.

Elements inside the binary

După ce am creat un nou proiect în Ghidra (vezi resurse) și am încărcat binarul nostru, vom începe procesul de analiza. Începem să ne uităm la structura acestuia și să vedem ce funcții sunt folosite. Funcția `main()` va fi prima care va fi analizată

Resurse utile pentru incepatori din UNbreakable România

`main()`



```
1
2 undefined8 main(int param_1,undefined8 *param_2)
3
4 {
5     char cVar1;
6     undefined4 uVar2;
7     int iVar3;
8     undefined8 uVar4;
9
10    uVar2 = func_0x001010e0(0);
11    func_0x001010d0(uVar2);
12    if (param_1 < 2) {
13        func_0x001010c0(&UNK_0010204e,*param_2);
14        uVar4 = 0xffffffff;
15    }
16    else {
17        iVar3 = func_0x001010b0(param_2[1]);
18        cVar1 = check_password(param_2[1],(long)iVar3);
19        if (cVar1 == '\0') {
20            iVar3 = func_0x001010f0();
21            func_0x001010c0(&UNK_0010207f,fail_msgs + (long)(iVar3 % 0xe) * 0x40);
22            uVar4 = 0xffffffff;
23        }
24        else {
25            func_0x001010a0(&UNK_00102065);
26            uVar4 = 0;
27        }
28    }
29    return uVar4;
30 }
```

Aici sunt câteva variabile cu nume fără sens (așa le afișează Ghidra) și câteva verificări simple.

Ceva care ne atrage atenția este funcția `check_password()`.

Să inspectăm această funcție mai departe.

```
Decompile: check_password - (agoodone)
1
2 _Bool check_password(char *passwd, size_t len)
3
4 {
5     char cVar1;
6     size_t sVar2;
7     int result;
8     int i;
9
10    result = 0;
11    for (i = 0; (ulong)(long)i < len; i = i + 1) {
12        cVar1 = passwd[i];
13        sVar2 = strlen(enc_flag);
14        result = result | (uint)sVar2 ^ (int)cVar1 ^ (uint)len;
15        printf("%d", (ulong)(uint)result);
16    }
17    printf("%d\n", (ulong)(uint)result);
18    return (_Bool)(result == 0);
19 }
20
32 }
33 }
34 if (local_10 == *(long *) (in_FS_OFFSET + 0x28)) {
35     return debugger_present;
36 }
37 /* WARNING: Subroutine does not return */
38 __stack_chk_fail();
39 }
40
```

Să analizăm acest lucru linie cu linie și să vedem ce informații putem obține despre modul în care funcționează:

Linia 2: Această funcție returnează o valoare booleană.

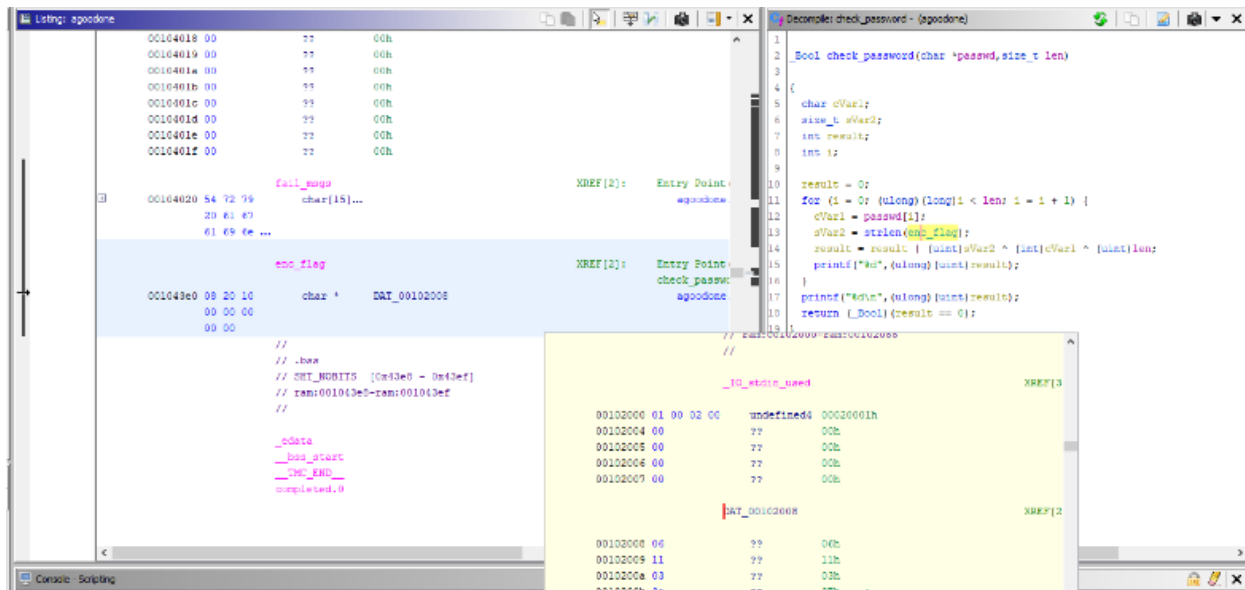
Resurse utile pentru incepatori din UNbreakable România

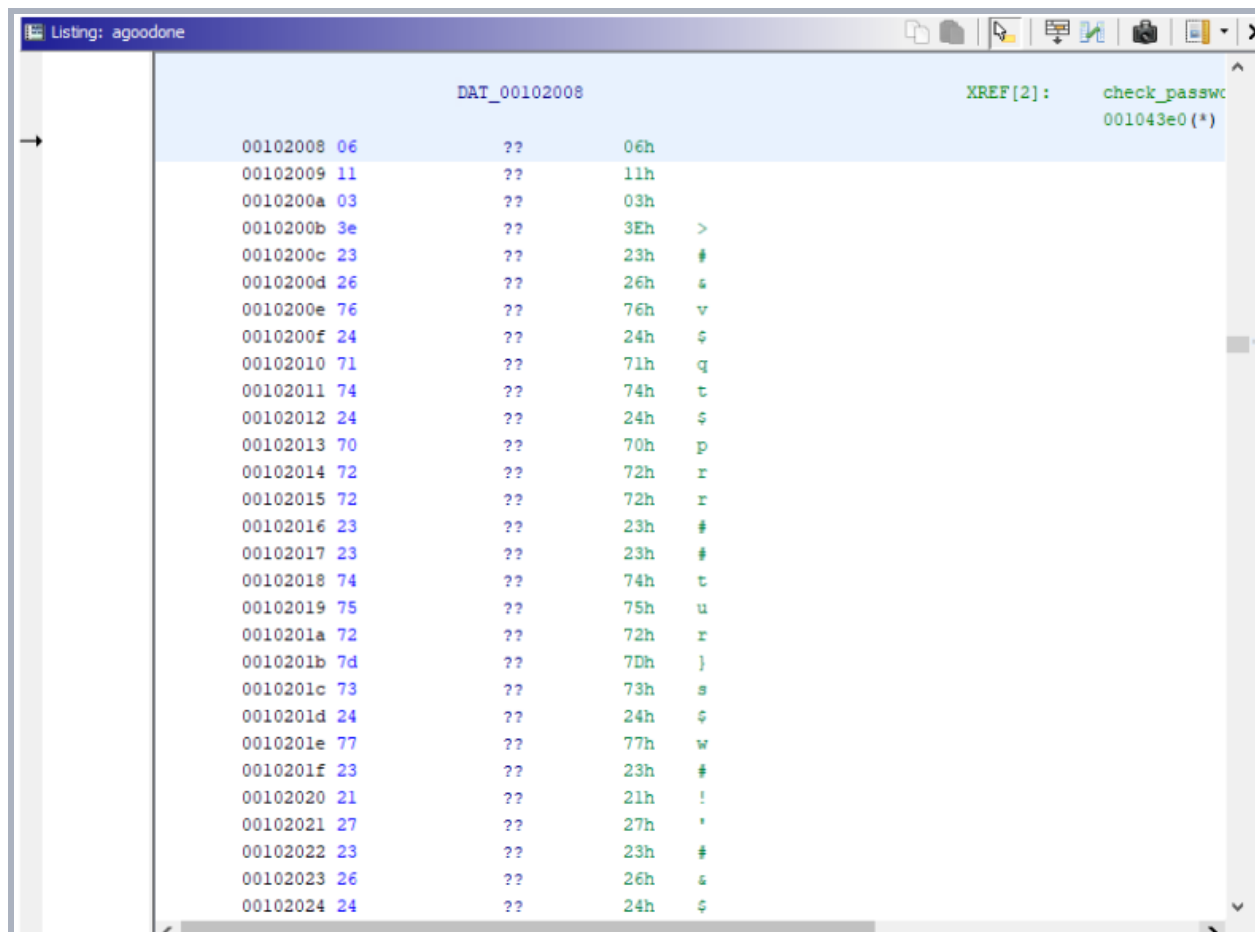
Rândul 11: **for()** face o buclă peste o lungime de ceva.

Linia 12: Aceasta pare a fi o verificare pentru fiecare caracter al unei parole.

Linia 13: Stocarea lungimii variabilei `enc_flag`. După cum sugerează și numele, aici ar putea fi locul în care ar trebui să se afle steagul.

Dând dublu clic pe această variabilă, vom vedea conținutul ei:





După cum putem vedea, stochează o listă de caractere. Aceeași secvență pe care am văzut-o atunci când am emis comanda `strings`

```
[A^A]A^A_
>#&v$qt$pr#tur}s$w#!'#$&$!t} #qr $r}!qws$qr!u|r$q| v}uv#r |&u |s8
Usage: %s <key_value
```

Putem observa că **enc_flag** conține următoarele valori hex

```
\x06\x11\x03\x3e\x23\x26\x76\x24\x71\x74\x24\x70\x72\x72\x23\x23\x74\x75\x72\x7d\x73\x24
\x
77\x23\x21\x27\x23\x26\x24\x21\x74\x7d\x20\x23\x71\x72\x20\x24\x72\x7d\x21\x71\x77\x73\x
2
4\x71\x72\x21\x75\x7c\x72\x24\x71\x7c\x20\x76\x7d\x75\x76\x23\x72\x20\x7c\x26\x75\x20\x7
```

```
c\  
x73\x38
```

Am extras și formatat valorile în acest mod, dar puteți face acest lucru oricum doriți.

Linia 14: Rezultatul stochează rezultatul unor operații XOR(^).

Să ne oprim pentru un moment aici și să încercăm să înțelegem această operațiune:

"a |= b" înseamnă "a = a | b", iar "a | b" va avea un 1 la fiecare bit care este 1 fie în a, fie în b.

Operatori de tip bitwise

Operatorul Bitwise lucrează pe biți și efectuează operații bit cu bit.

Tabelele de adevăr pentru &, | și ^ sunt următoarele:

p	q	p & q	p q	p ^ q
0	0	0	0	0
0	1	0	1	1
1	1	1	1	0
1	0	0	1	1

Deci, rezultatul începe cu 0 (după cum putem vedea la linia 10) și primește valori adăugate pe măsură ce bucla avansează, dar de îndată ce partea dreaptă are un 1 undeva, se va seta acel bit din rezultat.

Linia 18: Rezultatul este comparat cu 0 și este returnată o valoare booleană.

Resurse utile pentru incepatori din UNbreakable România

Așadar, acum știm că rezultatul trebuie să aibă toate valorile 0, astfel încât funcția să returneze true.

Un alt lucru pe care trebuie să-l știți este că $(a^b)^c$ este același lucru cu $a^{(b^c)}$.
Comutativitatea este o proprietate a operației XOR.

Acum, revenind la funcția noastră principală, vedem că `check_password()` este apelată cu 2 argumente:

- `argv[1]` -> ceea ce dăm ca argument atunci când pornim programul
- `sVar4` -> lungimea argumentului nostru

```
19 | sVar4 = strlen(argv[1]);  
20 | _Var1 = check_password(argv[1], (long) (int) sVar4);
```

Deci, pe baza a ceea ce am discutat până acum, singura modalitate ca rezultatul să rămână 0 este dacă fiecare caracter din

parola noastră este egal cu `"len ^ strlen(enc_flag)"`

De exemplu, dacă parola noastră are lungimea
= 3 și `strlen(enc_flag) = 40`, atunci fiecare caracter
trebuie să aibă valoarea `= 41^3 = 42`, care este *

Astfel, parola noastră (pentru acest scenariu ipotetic) va fi: ***

Resurse utile pentru incepatori din UNbreakable România

ASCII Hex Symbol	ASCII Hex Symbol	ASCII Hex Symbol	ASCII Hex Symbol
0 0 NUL	16 10 DLE	32 20 (space)	48 30 0
1 1 SOH	17 11 DC1	33 21 !	49 31 1
2 2 STX	18 12 DC2	34 22 "	50 32 2
3 3 ETX	19 13 DC3	35 23 #	51 33 3
4 4 EOT	20 14 DC4	36 24 \$	52 34 4
5 5 ENQ	21 15 NAK	37 25 %	53 35 5
6 6 ACK	22 16 SYN	38 26 &	54 36 6
7 7 BEL	23 17 ETB	39 27 '	55 37 7
8 8 BS	24 18 CAN	40 28 (56 38 8
9 9 TAB	25 19 EM	41 29)	57 39 9
10 A LF	26 1A SUB	42 2A *	58 3A :
11 B VT	27 1B ESC	43 2B +	59 3B ;
12 C FF	28 1C FS	44 2C ,	60 3C <
13 D CR	29 1D GS	45 2D -	61 3D =
14 E SO	30 1E RS	46 2E .	62 3E >
15 F SI	31 1F US	47 2F /	63 3F ?

ASCII Hex Symbol	ASCII Hex Symbol	ASCII Hex Symbol	ASCII Hex Symbol
64 40 @	80 50 P	96 60 `	112 70 p
65 41 A	81 51 Q	97 61 a	113 71 q
66 42 B	82 52 R	98 62 b	114 72 r
67 43 C	83 53 S	99 63 c	115 73 s
68 44 D	84 54 T	100 64 d	116 74 t
69 45 E	85 55 U	101 65 e	117 75 u
70 46 F	86 56 V	102 66 f	118 76 v
71 47 G	87 57 W	103 67 g	119 77 w
72 48 H	88 58 X	104 68 h	120 78 x
73 49 I	89 59 Y	105 69 i	121 79 y
74 4A J	90 5A Z	106 6A j	122 7A z
75 4B K	91 5B [107 6B k	123 7B {
76 4C L	92 5C \	108 6C l	124 7C
77 4D M	93 5D]	109 6D m	125 7D }
78 4E N	94 5E ^	110 6E n	126 7E ~
79 4F O	95 5F _	111 6F o	127 7F

Python script

Să vedem cum ne poate ajuta Python să înțelegem mai bine datele noastre.

Mai întâi, să verificăm lungimea lui **enc_flag**:

```
>>>
len("\x06\x11\x03\xe2\x23\x26\x76\x24\x71\x74\x24\x70\x72\x72\x23\x23\x74\x
75\x72\x7d\x73\x24\x77\x23\x21\x27\x23\x26\x24\x21\x74\x7d\x20\x23\x71\x72\x
x20\x24\x72\x7d\x21\x71\x77\x73\x24\x71\x72\x21\x75\x7c\x72\x24\x71\x7c\x20
\x76\x
7d\x75\x76\x23\x72\x20\x7c\x26\x75\x20\x7c\x73\x38")
69

strlen(enc_flag) = 69

>>> 69^3
```

70

Am ales să facem `XOR strlen(enc_flag)` cu 3, astfel încât fiecare caracter al parolei noastre trebuie să aibă

valoarea de 70 (F)

Așadar, parola, în acest caz, este: FFF

Test:

```
./agoodone FFF
0000
Voilà, Correct Password!
```

Deci, asta era parola, dar unde este steagul?

Flag

O modalitate de a aborda această problemă este de a încerca să faceți brute-force pe steagul `enc_flag` prin XOR cu 127

valorile ASCII:

```
enc_flag =
b'\x06\x11\x03\x3e\x23\x26\x76\x24\x71\x74\x24\x70\x72\x72\x23\x23\x74\x75\
\x72\x7d\x73\x24\x77\x23\x21\x27\x23\x26\x24\x21\x74\x7d\x20\x23\x71\x72\x20\
\x24\x72\x7d\x21\x71\x77\x73\x24\x71\x72\x21\x75\x7c\x72\x24\x71\x7c\x20\x7\
6\x7d\x75\x76\x23\x72\x20\x7c\x26\x75\x20\x7c\x73\x38'
for xor in range(127):
    print(''.join(chr(c ^ xor) for c in enc_flag))

DSA|ad4f36f200aa670?1f5aceadfc6?ba30bf0?c351f30c7>0f3>b4?74a0b>d7b>1z
ER@}`e5g27g311``761>0g4`bd`egb7>c`21cg1>b240g21b6?1g2?c5>65`1c?e6c?0{
BUGzgb2`50`466gg01697`3gecgb`e09dg56d`69e537`56e186`58d2912g6d8b1d87|
CTF{fc3a41a577ff10786a2fdbfcad18ef47ea78d426a47d097a49e3803f7e9c0e96}
@WExe`0b72b644ee234;5b1egae`bg2;fe74fb4;g715b74g3:4b7:f0;30e4f:`3f:5~
AVDyda1c63c755dd325:4c0df`dacf3:gd65gc5:f604c65f2;5c6;g1:21d5g;a2g;4
NYKvkn>I9<I8::kk<=:5;I?kioknli<5hk9:hl:5i9?;I9:i=4:I94h>5=>k:h4n=h4;p
OXJwjo?m8=m9;;jj=<;4:m>jhnjomh=4ij8;im;4h8>m8;h<5;m85i?4<?j;
i5o<i5;q

CTF{fc3a41a577ff1078[REDACTED]426a47d097a49e3803f7e9c0e96}
```

O altă modalitate ar fi să se facă un XOR doar cu lungimea **enc_flag**:

Resurse utile pentru incepatori din UNbreakable România

```
enc_flag =  
b'\x06\x11\x03\xe\x23\x26\x76\x24\x71\x74\x24\x70\x72\x72\x23\x23\x74\x75\  
\x72\x7d\x73\x24\x77\x23\x21\x27\x23\x26\x24\x21\x74\x7d\x20\x23\x71\x72\x20\  
\x24\x72\x7d\x21\x71\x77\x73\x24\x71\x72\x21\x75\x7c\x72\x24\x71\x7c\x20\x7\  
6\x7d\x75\x76\x23\x72\x20\x7c\x26\x75\x20\x7c\x73\x38'  
xor = 69  
print(''.join(chr(c ^ xor) for c in enc_flag))
```

```
CTF{fc3a41a577ff10786a2fd[REDACTED]a78d426a47d097a49e3803f7e9c0e96}
```

Combined (easy-medium)

Concurs UNbreakable 2021 #Individual

Autor exercițiu: Octavian Purcaru

Contribuitori rezolvare: Niță Horia, Valentina Galea

Descriere:

There are not that many combinations one can **do** here.

Rezolvare:

Pentru această probă vom analiza fișierul pe o mașină Linux folosind Ghidra și Python.

Inspecție binar

Vom folosi comanda file din terminal pentru a vedea rapid cu ce fel de fișier avem de-a face.

```
file combined  
combined_lvl0: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV),  
dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2
```

Putem vedea că este un fișier ELF (Executable and Linkable Format) și că rulează pe 64 de biți. Aceasta înseamnă că, teoretic, îl putem executa pe calculatorul nostru Linux.

Analiza binarului

Următoarele comenzi pot fi folosite pentru a obține informații semnificative despre fișierul nostru înainte de a trece la analiza propriu-zisă.

strings combined

Comanda strings caută șiruri de caractere imprimabile într-un fișier. Un șir de caractere este orice secvență de 4 sau mai multe caractere imprimabile care se termină cu o nouă linie sau cu un caracter nul.

Comanda strings este utilă pentru identificarea fișierelor cu obiecte aleatorii.

```
0x430xf10x250x0b0xac0xa20x2e0xb60xb20x540x3a0x7d0x4f0x6e0x1d0x2e0x7e0xd10x4
60x8a0x080xa30x600x970x330x8b0x1a0x7b0xb70x8c0x4a0x820x2f0x9b0xb10x440x660x
c90x510xd30x9c0x4b0x690xde0x0c0x650x050x6a0x4f0x370x170x000x670x230x340x110
xf00x6d0x650x810x400xc80xc90x300xa70xd30x4e0xc50xc00x0d0x2f0x970x320x5f0x1b
0xbe0x250x1a0x260x580x050x310xa80x290x090x9e0xf60xb60xbc0x680x380xf50xc40x3
60x810x290xdc0x650x440x330x8e0x310x890x6d0x220xda0x920x870x650x570xe20x100x
580x350x2e0x650xc80x610xc50x100x200x6f0x450x800x2a0xc50x330x420xcc0xd80xf30
xc00x590xfb0x7a0x300x3c0xed0xef0xdf0x020xb20x210x1a0x340x4c0xfb0x520x020x2f
0x4a0xd30x8a0x310xab0xf30x1b0x0a0x570xcc0x7e0xec0x370x5c0xa20xe90x6b0xbb0x4
70x490x550x660xef0x040x390xde0x150xc30xf00x970x350xfd0x470x280xcd0x330x380x
2a0x8e0x640x290xa30x910xf60x9e0xd60xee0x860x330xb40xbd0x5b0xa70x6b0xfd0xfd0
x020x330x440xfd0x1f0x5d0x4b0xe20x9c0x1f0x330x2e0x910xf50x830xe60x970xad0x0b
0x620x190x580xb40x650xc60x8c0xcc0x840x340x630xcd0xcc0xd30xdf0xec0x6a0xfa0x3
00x530x290x4e0x970x1d0x530x6f0x610x630xd50x6a0x1a0x1d0xdf0xea0x580xcf0x320x
2e0x860x7b0x990x2b0x940x780xf40x320xce0x150x360x960x930x540x330xa50x640x5d0
xe20x470x8d0x690xa00xf80xe90x390xbb0x010xdb0x1e0xd70x8a0x5c0xba0x620xaf0x70
0x410xcd0x7e0x440xf50x090x320xad0xb30x970xce0x680xfc0x3b0xe90x360x2b0xea0x9
30x900x3f0x0b0xd50xe00x630x610x6b0x9f0x790x480x430x680x320x310x020xc10xf40x
390xec0x3b0x0c0xde0x610x080xa60x3a0x880xb60x080xb90x490x650x0d0x920x7e0x210
x140x170xeb0xe30x620xea0xfb0x7f0x0e0x830x210xf60x1d0x650xcc0x4d0x510x970x01
0x060xb30x7d0x640x7b0x590x300xdb0x050x310xde0x590x340x150xe60x270xdf0x900x1
80x5e0x3b0x7d0x830x430xe80x780x2d0x2d0x0c0x530x870xd10xa20x340x2a0x140xfa0x
b30x470xd10x190x870xb40x7f0xb80xe30xc40xf10xb50x940x8b0xaa0x590x850xa30x040
x7a0x610xe50x860xe70x410x650x300xec0x4f0x610x860x4c0x2f0x5a0x0b0x420x760x20
GCC: (Ubuntu 10.3.0-1ubuntu1) 10.3.0
/home/oct/ctf_octavian/Combined_CTF
/usr/lib/gcc/x86_64-linux-gnu/10/include
/usr/include/x86_64-linux-gnu/bits
/usr/include/x86_64-linux-gnu/bits/types
/usr/include
combined.c
stddef.h
```

Resurse utile pentru incepatori din UNbreakable România

Din rezultat putem vedea că există câteva șiruri de caractere care indică o posibilă autentificare și o secvență lungă de caractere care ar putea părea interesantă.

După ce am făcut fișierul executabil cu `chmod +x` combinat, putem continua cu fișierul nostru și să emitem următoarele comenzi:

```
ltrace ./combined
Please enter validation: hello
Verifying your authentication
ERROR: Invalid authentication!
+++ exited (status 0) +++
```

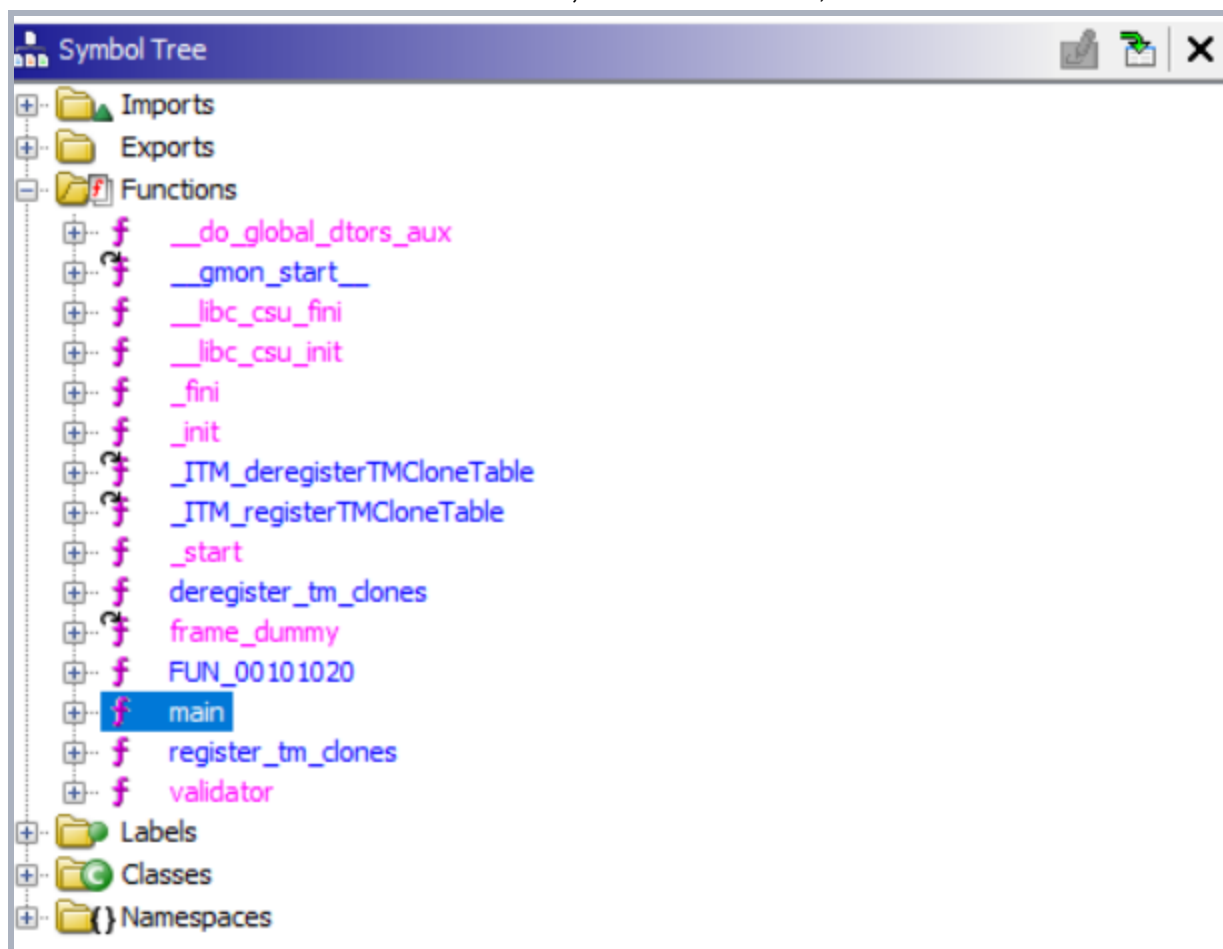
```
strace ./combined
execve("./combined", ["/combined"], 0x7ffccd202120 /* 53 vars */) = 0
brk(NULL) = 0x55b983617000
arch_prctl(0x3001 /* ARCH_??? */ , 0x7fffe28696c0) = -1 EINVAL (Invalid
argument)
access("/etc/ld.so.preload", R_OK) = -1 ENOENT (No such file or
directory)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=76146, ...},
AT_EMPTY_PATH) = 0
mmap(NULL, 76146, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7fecbdbbb000
close(3) = 0
6openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3,
"\177ELF\2\1\1\3\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\240\206\2\0\0\0\0" ...,
832) = 832
pread64(3,
"\6\0\0\0\4\0\0\0@\0\0\0\0\0\0@\0\0\0\0\0\0@\0\0\0\0\0\0" ..., 784,
64) = 784
pread64(3, "\4\0\0\0
\0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0" ..., 48, 848) = 48
pread64(3,
"\4\0\0\0\24\0\0\0\3\0\0\0GNU\0+H)\227\201T\214\233\304R\352\306\3379\220%"
..., 68, 896) = 68
newfstatat(3, "", {st_mode=S_IFREG|0755, st_size=1983576, ...},
AT_EMPTY_PATH) = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1,
0) =
0x7fecbdbbb9000
```



```
pread64(3,
"\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784,
64) = 784
mmap(NULL, 2012056, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) =
0x7fecbd9cd000
mmap(0x7fecbd9f3000, 1486848, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x26000) = 0x7fecbd9f3000
mmap(0x7fecbdb5e000, 311296, PROT_READ,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x191000) = 0x7fecbdb5e000
mmap(0x7fecbdbaa000, 24576, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1dc000) = 0x7fecbdbaa000
mmap(0x7fecbdbb0000, 33688, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7fecbdbb0000
close(3) = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1,
0) =
0x7fecbd9cb000
arch_prctl(ARCH_SET_FS, 0x7fecbdbba580) = 0
mprotect(0x7fecbdbaa000, 12288, PROT_READ) = 0
mprotect(0x55b98211a000, 4096, PROT_READ) = 0
mprotect(0x7fecbdc00000, 8192, PROT_READ) = 0
munmap(0x7fecbdbbb000, 76146) = 0
newfstatat(1, "", {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0x1), ...},
AT_EMPTY_PATH) = 0
brk(NULL) = 0x55b983617000
brk(0x55b983638000) = 0x55b983638000
newfstatat(0, "", {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0x1), ...},
AT_EMPTY_PATH) = 0
write(1, "Please enter validation: ", 25Please enter validation: ) = 25
read(0, hello
"hello\n", 1024) = 6
write(1, "Verifying your authentication\n", 30Verifying your authentication
) = 30
write(1, "ERROR: Invalid authentication!\n", 31ERROR: Invalid
authentication!
) = 31
exit_group(0) = ?
+++ exited with 0 +++
```

Resurse utile pentru incepatori din UNbreakable România

Este timpul să importăm binaural în Ghidra și să analizăm funcția main()



main()

```
C: Decompiler: main - (combined)
1
2 int main(int arg1, char **arg2, char **arg3)
3
4 {
5     int iVar1;
6     __ssize_t _Var2;
7     long in_FS_OFFSET;
8     int var498324;
9     char *var389534;
10    long local_10;
11
12    local_10 = *(long *) (in_FS_OFFSET + 0x28);
13    printf("Please enter validation: ");
14    var498324 = 0;
15    _Var2 = getline(&var389534, (size_t *) &var498324, stdin);
16    var498324 = (int) _Var2;
17    var389534[(long) var498324 + -1] = '\0';
18    puts("Verifying your authentication");
19    iVar1 = validator((long) var389534);
20    if (iVar1 == 1) {
21        puts("Correct authentication received");
22    }
23    else {
24        puts("ERROR: Invalid authentication!");
25    }
26    if (local_10 != *(long *) (in_FS_OFFSET + 0x28)) {
27        /* WARNING: Subroutine does not return */
28        __stack_chk_fail();
29    }
30    return 0;
31 }
```

```
validator()

C: Decompiler: validator - (combined)
1
2 int validator(long param_1)
3
4 {
5     int var1;
6     int var2341;
7
8     var2341 = 0;
9     while( true ) {
10         if (0x1d < var2341) {
11             return 1;
12         }
13         if ((int8_t (*) [4]) (long) * (char *) (param_1 + var2341) != verify[var2341 * 9]) break;
14         var2341 = var2341 + 1;
15     }
16     return -1;
17 }
18
```

validator()

Putem vedea că primul if nu se va executa deoarece `0x1d (29 în zecimal) > var2341` (care este declarat mai sus și are valoarea 0)

Așadar, a doua instrucțiune if rămâne să fie analizată în continuare. Vedem că, în esență, aceasta execută verifică fiecare al nouălea element din ceea ce pare a fi în prezent un `array -> verify[]`

După cum sugerează și numele, ne încurajează să mergem mai departe cu investigațiile noastre.

```
verify[]
Listing: combined
0010401f 00 ?? 00h
verify XREF[2]:
00104020 30 78 34 int8_t[4...
33 30 78
66 31 30 ...
00104020 30 78 34 33 int8_t[4] "0x43" [0]
00104020 [0] '0', 'x', '4', '3'
00104024 30 78 66 31 int8_t[4] "0xf1" [1]
00104028 30 78 32 35 int8_t[4] "0x25" [2]
0010402c 30 78 30 62 int8_t[4] "0x0b" [3]
00104030 30 78 61 63 int8_t[4] "0xac" [4]
00104034 30 78 61 32 int8_t[4] "0xa2" [5]
00104038 30 78 32 65 int8_t[4] "0x2e" [6]
0010403c 30 78 62 36 int8_t[4] "0xb6" [7]
00104040 30 78 62 32 int8_t[4] "0xb2" [8]
00104044 30 78 35 34 int8_t[4] "0x54" [9]
00104048 30 78 33 61 int8_t[4] "0x3a" [10]
0010404c 30 78 37 64 int8_t[4] "0x7d" [11]
00104050 30 78 34 66 int8_t[4] "0x4f" [12]
00104054 30 78 36 65 int8_t[4] "0x6e" [13]
00104058 30 78 31 64 int8_t[4] "0x1d" [14]
0010405c 30 78 32 65 int8_t[4] "0x2e" [15]
00104060 30 78 37 65 int8_t[4] "0x7e" [16]
00104064 30 78 64 31 int8_t[4] "0xd1" [17]
00104068 30 78 34 36 int8_t[4] "0x46" [18]
0010406c 30 78 38 61 int8_t[4] "0x8a" [19]
00104070 30 78 30 38 int8_t[4] "0x08" [20]
00104074 30 78 61 33 int8_t[4] "0xa3" [21]
00104078 30 78 36 30 int8_t[4] "0x60" [22]
```

Putem vedea că este un array de 450 de elemente cu 4 elemente de tip `int8_t`. Așadar, este o matrice 2D.

Valorile par a fi șiruri de caractere și conțin valori de `0x..`

În mod natural, vom fi înclinați să analizăm valorile hexagonale din interiorul șirurilor de caractere

Python script

Resurse utile pentru incepatori din UNbreakable România

Vom copia toate aceste valori și le vom formata astfel încât să obținem o listă hexagonală frumoasă și vom traduce în ASCII.

Puteți face acest lucru folosind editorul de text pe care îl alegeți.

Traducerea corectă a textului hexazecimal nu va oferi informații semnificative, dar pe baza verificării făcute în funcția `validator()` de mai devreme, vom alege să extragem fiecare al 9-lea element din interior.

```
data = [  
    0x43,  
    0xf1,  
    0x25,  
    # ... 450 values  
]  
for i in range(0, len(data), 9):  
    print(chr(data[i]), end="")
```

Flag

```
CTF{fe402183ea304[REDACTED]c22d9b26c1aebed4}
```

Leprechaun (ușor)

Concurs UNbreakable 2021 #Individual

Autor exercițiu: Răzvan Deaconescu

Contribuitori rezolvare: Niță Horia, Valentina Galea

Descriere:

```
Get the flag from the pot at the end of the rainbow.
```

Rezolvare:

Deschizând fișierul binar în IDA, observăm că funcția principală emite pur și simplu un șir de caractere în `stdout`.

Dacă examinăm mai mult putem vedea o funcție numită `sub_201010`.

Această funcție are un șir ciudat care spune `"dublin_guiness"`, așa că o putem executa cu gdb (deoarece decompilarea nu este posibilă fără a face modificări la pointerul pozitiv al stivei).

Resurse utile pentru incepatori din UNbreakable România

Binarul are PIE activat, așa că trebuie să calculăm valoarea poziției funcției `sub_201010` atunci când o executăm.

În IDA, aceasta este localizată la `0x201010`, așa că o adăugăm la valoarea din timpul execuției.

După ce rulăm info proc map în gdb, obținem `0x5555555400000` ca adresă de start pentru programul nostru.

```
0x5555555400000 + 0x201010 = 0x5555555601010
```

Așadar, setăm `$rip=0x5555555420101` pentru a rula instrucțiunile la această valoare. Funcția primește, de asemenea, un parametru. De asemenea, dorim să facem suficient spațiu pe stivă, astfel încât să putem obține fiecare parte a indicatorului.

Așadar, după ce am rulat pentru un timp, apăsăm pe:

```
sub rsp,0x60
```

Totuși, acest spațiu nu este suficient (încercare și eroare), și vom mai scădea încă un `0x40`.

```
gdb-peda$ p $rsp-0x40
$1 = (void *) 0x7ffffffdfb8
set $rsp=0x7ffffffdfb8
```

Minunat, acum, așa cum am spus, funcția primește un parametru, drept urmare setează RDI la `RBP-0x100` (atât cât am eliberat din stivă).

```
gdb-peda$ p $rbp - 0x100
$2 = (void *) 0x7ffffffdf58
set $rdi=0x7ffffffdf58
```

Acum stabilim un punct de întrerupere înainte de instrucțiunea `return` și continuăm să rulăm comanda până atunci.

```
0x555555601120: mov rax,QWORD PTR [rbp-0x8]
0x555555601124: xor rax,QWORD PTR fs:0x28
0x55555560112d: je 0x555555601134
0x55555560112f: call 0x555555601134
0x555555601134: leave
0x555555601135: ret
gdb-peda$ break *0x555555601124
```

Resurse utile pentru incepatori din UNbreakable România

```
Breakpoint 1 at 0x555555601124
gdb-peda$ c
```

Suntem opriți de canary, dar acest aspect nu ne impactează rezolvare exercițiului, așa că sărim peste.

```
Stopped reason: SIGSEGV
0x000055555560101c in ?? ()
gdb-peda$ set $rip=0x555555601025 # 1 instruction further
gdb-peda$ c
```

Flag-ul poate fi regăsit în registrul RDI:

```
RAX: 0x0
RBX: 0x0
RCX: 0x73 ('s')
RDX: 0x7d ('}')
RSI: 0x7d ('}')
20
RDI: 0x7fffffffdf58 ("UNR{doubloon_schiling_ducat}")
RBP: 0x7fffffff058 --> 0x0
RSP: 0x7fffffffdfb8 --> 0x0
RIP: 0x555555601124 --> 0x282504334864
R8 : 0x0
R9 : 0x0
R10: 0x0
R11: 0x0
R12: 0x0
R13: 0x0
R14: 0x0
R15: 0x0
EFLAGS: 0x202 (carry parity adjust zero sign trap INTERRUPT direction overflow)
```

```
UNR{doubloo[REDACTAT]ing_ducat}
```


src (mediu)

Concurs UNbreakable 2021 #Echipe

Autor exercițiu: Edmund

Contribuitori rezolvare: Andrei Albișoru, Teodor Duțu, Adina Smeu, Valentina Galea

Descriere:

Sometimes it's **not what it seems!**

Rezolvare:

Ni se oferă un executabil Windows numit src.exe, ideea era că în interiorul acestuia se află câteva artefacte python care pot fi extrase pentru a obține steagul care era, de asemenea, criptat.

Începem depanarea cu x64dbg (fișierul era pe 64bit). Am setat diferite puncte de întrerupere și la un moment dat am observat aceste linii:

```
00007FF6D5245D8F | FF15 93320200 | apel qword ptr  
ds:[<&GetCommandLine>]  
|  
00007FF6D5245D95 | 45:33C9 | xor r9d,r9d  
|  
00007FF6D5245D98 | 4C:8D4424 58 | lea r8,qword ptr ss:[rsp+58]  
|  
00007FF6D5245D9D | 48:8BD0 | mov rdx,rax  
| rax:L"C:\\Users\\andre\\Downloads\\src.exe\\"  
00007FF6D5245DA0 | 48:8D8C24 00010000 | lea rcx,qword ptr  
ss:[rsp+100]  
|  
00007FF6D5245DA8 | 48:8D4424 70 | lea rax,qword ptr ss:[rsp+70]  
|  
00007FF6D5245DAD | 48:894424 48 | mov qword ptr ss:[rsp+48],rax  
|  
00007FF6D5245DB2 | 48:8D8424 90000000 | lea rax,qword ptr ss:[rsp+90]  
|  
00007FF6D5245DBA | 48:894424 40 | mov qword ptr ss:[rsp+40],rax  
|  
00007FF6D5245DBF | 48:897C24 38 | mov qword ptr ss:[rsp+38],rdi  
|  
00007FF6D5245DC4 | 48:897C24 30 | mov qword ptr ss:[rsp+30],rdi  
|  
00007FF6D5245DC9 | 897C24 28 | mov dword ptr ss:[rsp+28],edi  
|
```

Resurse utile pentru incepatori din UNbreakable România

```
00007FF6D5245DCD | 895C24 20 | mov dword ptr ss:[rsp+20],ebx
|
00007FF6D5245DD1 | FF15 01330200 | apel qword ptr
ds:[<&CreateProcessW>]
```

Se pare că binarul rula un proces în urmă.
Acest proces ne cerea să introducem o parolă.

Acesta a fost creat dintr-o roată python. Observăm acest lucru prin apelarea comenzii strings pe src.exe. Extragem aceste date folosind un instrument pe care l-am găsit pe github:

<https://github.com/extremecoders-re/pyinstxtractor>

Folosim comanda:

```
python3.7 pyinstructor/pyinstructor.py src.exe
```

Următorului pas, și anume decompilarea .pyc.
Versiunea a afectat câțiva octeți magici de la începutul codului de bytes.
Și obținem fișierul .pyc.

Din acesta folosim din nou uncompyle6 pentru a extrage sursa py:

```
uncompyle6 -o . src.p
```

Astfel obținem `src.py`

```
# uncompyle6 version 3.8.0
# Python bytecode 3.7.0 (3394)
# Decompiled from: Python 3.8.10 (default, Sep 28 2021, 16:10:42)
# [GCC 9.3.0]
# Embedded file name: src.py
from Crypto.Cipher import Salsa20
k = bytes([202, 254, 186, 190, 80, 114, 97, 105, 115, 101, 84, 104, 101,
83, 117, 110])
no = bytes([76, 105, 118, 105, 97, 110, 222, 173])
cipher = Salsa20.new(key=k, nonce=no)
flag = [17, 223, 36, 154, 167, 98, 74, 38, 104, 27, 127, 85, 67, 37, 202,
173, 164, 152, 105, 244, 239, 123, 100, 178, 106, 212, 197, 10, 226, 141,
45, 74]
input = input('Enter the password: ').strip()
print(input)
encrypted = cipher.encrypt(input.encode('utf-8'))
if encrypted == bytes(flag):
print('O_O...Nice man xD...now that you learned some tricks, go get
```

Resurse utile pentru incepatori din UNbreakable România

```
your points :D')
else:
    print('Nope...good luck reversing this')
```

Steagul a fost criptat cu Salsa20. Modificăm scriptul astfel încât să corespundă nevoilor noastre:

```
from Crypto.Cipher import Salsa20
k = bytes([202, 254, 186, 190, 80, 114, 97, 105, 115, 101, 84, 104, 101,
83, 117, 110])
no = bytes([76, 105, 118, 105, 97, 110, 222, 173])
cipher = Salsa20.new(key=k, nonce=no)
flag = bytes([17, 223, 36, 154, 167, 98, 74, 38, 104, 27, 127, 85, 67, 37,
202, 173, 164, 152, 105, 244, 239, 123, 100, 178, 106, 212, 197, 10, 226,
141, 45, 74])
decrypted = cipher.decrypt(flag)
res = []
for i in decrypted:
    res.append(chr(i))
print("".join(res))
print('O_O...Nice man xD...now that you learned some tricks, go get your
points :D')
```

Îl rulăm și obținem:

```
UNBR{Nicee_u[REDACTAT]_py_trickz}
O_O...Nice man xD...now that you learned some tricks, go get your points :D
```

Pyfuscation (mediu)

Concurs UNbreakable 2021 #Echipe

Autor exercițiu: Lucian Nițescu

Contribuitori rezolvare: Andrei Albișoru, Teodor Duțu, Adina Smeu, Valentina Galea

Descriere:

```
Let's say that if you got the wrong flag, you did it wrong. I know is like wt*. Enjoy :)
```

Rezolvare:

Instalăm uncompile6 și apoi rulăm următorul script:

```
# uncompile6 version 3.8.0
# Python bytecode 3.6 (3379)
# Decompiled from: Python 3.8.10 (default, Sep 28
2021, 16:10:42) # [GCC 9.3.0]
# Embedded file name: ./chall.py
# Compiled at: 2021-09-23 00:48:10
# Size of source mod 2**32: 1508 bytes
import hashlib
version = 'Python 3.6.9'

def sauhd982w1d3jg23fwue(00000000000000000,
0000000000000000=2): 0000000000000000 =
0000000000000000.encode('utf-16-be')
0000000000000000 = []
for 0000000000000000 in range(0, len(0000000000000000),
0000000000000000):
0000000000000000 =
0000000000000000[0000000000000000:0000000000000000 +
0000000000000000]
0000000000000000.append(int.from_bytes(000000000000
00000, 'big'))

return str(0000000000000000)[1:-1]

def crazy_lol():
if 'aaaaaaaaaaaaaaaaaaaa' is 'aaaaaaaaaaaaaaaaaaaa':
if 'a' * 21 is 'aaaaaaaaaaaaaaaaaaaaaa':
return 'yuli'
else:
return 'w3y'
else:
return 'opl'

wufcwruewfhdwb = crazy_lol()
uehrgeriufqodhqf = 'xWjoy'
```

Resurse utile pentru incepatori din UNbreakable România

```
ourhecnuqwhdi = 'L3Hu'
uwoehsdia9j02m20 = sauhd982w1d3jg23fwue('ă')
fh983hf29hd28fh93 = 'ABvS'
jd2w0d9j20dwj22dj3grh = 'fmVeZ'
password = wufcwruewfhdb + uehrgeriufqodhgf +
ourhecnuqwhdi + uwoehsdia9j02m20 + fh983hf29hd28fh93
+ jd2w0d9j20dwj22dj3grh password_input =
input('Enter password to get the correct flag: ') if
password == password_input:
    print('CTF{' +
hashlib.sha256(password.encode('utf-8')).hexdigest() +
'})')
else:
    print('CTB{' +
hashlib.sha256(password_input.encode('utf-8')).hexdigest() +
'})')
```

Parola inițială utilizată a fost `yulixWjoyL3Hu259ABvSfmVeZ`.

După ce am rulat-o cu această parolă, obținem:

```
test.py:15: SyntaxWarning: "is" with a literal. Did you mean "=="?
if 'aaaaaaaaaaaaaaaaaaaaa' is 'aaaaaaaaaaaaaaaaaaaaa':
test.py:16: SyntaxWarning: "is" with a literal. Did you mean "=="?
if 'a' * 21 is 'aaaaaaaaaaaaaaaaaaaaa':
Enter password to get the correct flag: yulixWjoyL3Hu259ABvSfmVeZ
yulixWjoyL3Hu259ABvSfmVeZ
CTF{a89eaecced70954fb2ca4ed80bf6869a9da602fe568d414f30f62a4c42bb2ee7}
```

Rezultatele obținute nu sunt folositoare, drept urmare este necesar să modificăm scriptul:

```
# uncompile6 version 3.8.0
# Python bytecode 3.6 (3379)
# Decompiled from: Python 3.8.10 (default, Sep 28
2021, 16:10:42) # [GCC 9.3.0]
# Embedded file name: ./chall.py
# Compiled at: 2021-09-23 00:48:10
# Size of source mod 2**32: 1508 bytes
import hashlib
version = 'Python 3.6.9'
def sauhd982w1d3jg23fwue(000000000000000000,
0000000000000000=2): 0000000000000000 =
```

Resurse utile pentru incepatori din UNbreakable România

```
0000000000000000.encode('utf-16-be')
0000000000000000 = []
for 0000000000000000 in range(0,
len(0000000000000000), 0000000000000000):
    0000000000000000 =
0000000000000000[0000000000000000:0000000000000000 +
0000000000000000]
0000000000000000.append(int.from_bytes(000000000000
00000, 'big'))

return str(0000000000000000)[1:-1]

def crazy_lol():
    if 'aaaaaaaaaaaaaaaaaaaa' is 'aaaaaaaaaaaaaaaaaaaa':
        if 'a' * 21 is 'aaaaaaaaaaaaaaaaaaaa':
            return 'w3y'
        else:
            return 'w3y'
    else:
        return 'opl'

wufcwruewfhdwb = crazy_lol()
uehrgeriufqodhqf = 'xWjoy'
ourhecnuqwhdi = 'L3Hu'
uwoehsdia9j02m20 = sauhd982w1d3jg23fwue('ă')
fh983hf29hd28fh93 = 'ABvS'
jd2w0d9j20dwj22djc3grh = 'fmVeZ'
password = wufcwruewfhdwb + uehrgeriufqodhqf +
ourhecnuqwhdi + uwoehsdia9j02m20 + fh983hf29hd28fh93
+ jd2w0d9j20dwj22djc3grh password_input =
input('Enter password to get the correct flag: ') if
password == password_input:
    print('CTF{' +
hashlib.sha256(password.encode('utf-8')).hexdigest() +
'}')
else:
    print('CTB{' +
hashlib.sha256(password_input.encode('utf-8')).hexdigest() +
'}')
```

Resurse utile pentru incepatori din UNbreakable România

Și acum obținem flag-ul corect:

```
test.py:15: SyntaxWarning: "is" with a literal. Did you mean "=="?  
    if 'aaaaaaaaaaaaaaaaaaaaa' is 'aaaaaaaaaaaaaaaaaaaaa':  
test.py:16: SyntaxWarning: "is" with a literal. Did you mean "=="?
```

```
    if 'a' * 21 is 'aaaaaaaaaaaaaaaaaaaaa':  
Enter password to get the correct flag: w3yxWjoyL3Hu259ABvSfmVeZ  
w3yxWjoyL3Hu259ABvSfmVeZ  
CTF{b5858f16d9e3174a367ad5beecb171dcd8e2494d6edcc7a8caa7be2082a2a31f}
```

Contribuitori

- Mihai Dancaescu (yakuhto)
- Octavian Purcaru
- Andrei Albișoru
- Teodor Duțu
- Adina Smeu
- Niță Horia
- Moldovan Darius (T3jv1l)
- Netejoru Bogdan (trollzorftw)
- Emanuel Strugaru (edmund/Th3R4nd0m)

Resurse utile pentru incepatori din UNbreakable România