Introducere în Inginerie inversă pentru aplicații executabile, scripturi sau documente

Resurse utile pentru incepatori din UNbreakable România

unbreakable.ro

Declinarea responsabilității

Aceste materiale și resurse sunt destinate exclusiv informării și discuțiilor, avand ca obiectiv conștientizarea riscurilor si amenintarilor informatice dar și pregatirea unor noi generatii de specialisti in securitate informatica.

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 recomanda 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 considerata infractiune in diverse tari.

În România, accesul ilegal la un sistem informatic este considerată infracţiune contra siguranţei şi integrităţii sistemelor şi datelor informatice si 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 sa anonimeze serviciile cu care comunica, sau capabilitatile 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 vulnerabilitati** într-o aplicație ce a fost compilată (de eg. C/C++, C#, Java etc) prin recuperarea partiala sau integrala a codului sursa initial folosind unelte de compilare, aplicații de debugging samd

Adesea, hackerii dezvolta metode de a induce în eroare un analist ce încearcă aplicarea unor tehnici de inginerie inversa 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-debugge**r 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 in limbajul de programare Java.

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

- **Ida Pro**
- **Ghidra**
- **jd-gui**
- **radar2 sau r2**
- **dex2jar**
- **apktool**

Introducere în dezasamblarea codului sursa.

Vom începe prin a vizualiza în Ghidra cum se initializeaza o variabilă sau mai exact cum arată variabilele dupa 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):

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

int main(){

      char sir_de_caractere[20]="Salut tuturor";
      int variabila_int=200;
      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 hexadecimal, dacă convertim acea valoare în ascii vom avea (“ut tulaS” = “Salut tu”) adresele de memorie se

citest de la dreapta la stanga. În continuare registrul RDX va avea următoarele caractere stocate (“rorut”=turor). Mai jos am convertit valorile din hex in 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
```

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
```

Următoarle 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.

```
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>:    movsd  xmm0,QWORD PTR [rip+0xcf]        # 0x790
   0x00000000000006c1 <+87>:    movsd  QWORD PTR [rbp-0x30],xmm0
   0x00000000000006c6 <+92>:    movss  xmm0,DWORD PTR [rip+0xca]        # 0x798
   0x00000000000006ce <+100>:   movss  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:

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

```c
void functii_matematice(){

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.

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

După cum putem observa avem 0x0C si 0x0F in 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 simpla, i se aloca valuarea lui A adică 12 în registrul edx (edx=12) și lui eax valoarea 15 (eax=15) apoi cu ajutorul instrucțiuni 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 intructiuni **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
```

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**.

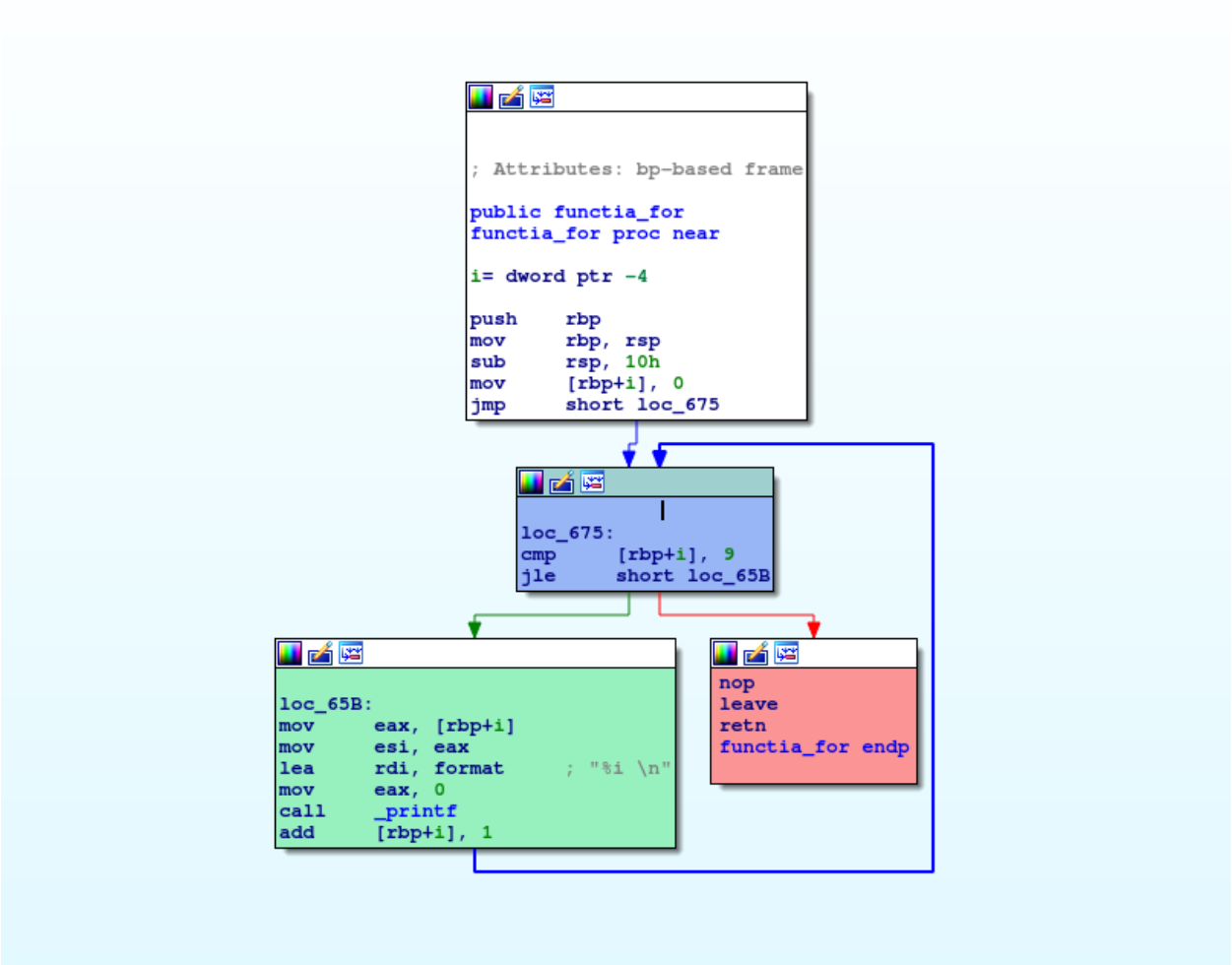
```
mov     eax, [rbp+B]
mov     edx, [rbp+A]
mov     ecx, eax
shl     edx, cl
mov     eax, edx
mov     [rbp+lshift], eax
```

Bucle infinite (for si while).

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

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

void functia_for(){

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

int main(){

   functia_for();
   return 0;
}
```

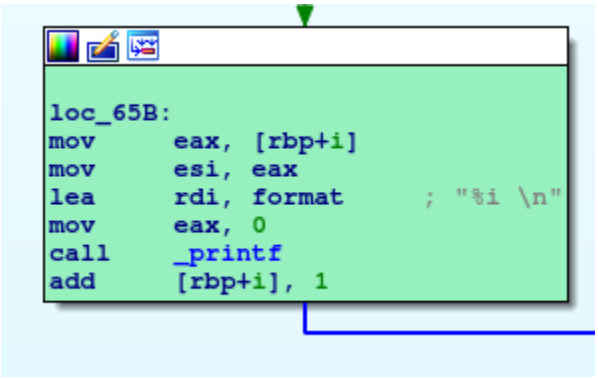
Pe scurt acest program va afișa cifrele de la 0 la 9. Reprezentarea in cod de asamblare a funcției **for** va arata ceva de genul.



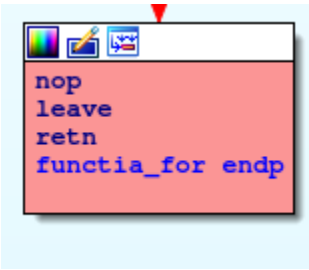
Pentru început lui “**i**” i se atribuie valoarea 0 apoi face un jump către blocul cu culoare albastra.

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

În blocul albastru se compara valoarea lui “**i**” cu valoarea 9, daca “**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 iara în blocul albastru să compare valorile. Va face acest lucru pana "**i**" va fi egal cu 9, apoi va ieși din bulca și va merge la blocul roșu unde se termina execuția programului.

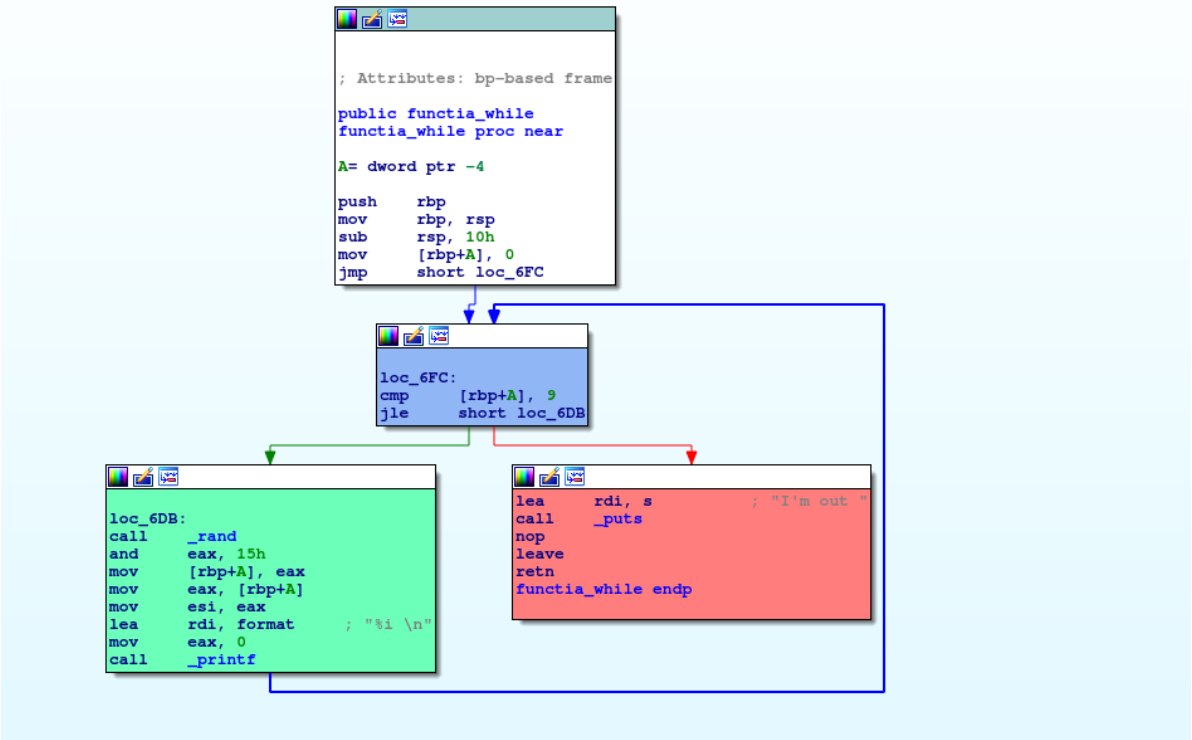


Exemplul 4, bucla while.

```c
#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");
}
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 obfuscat.

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

Acesta este unul dintre cele mai ușoare exemple de deobfuscat un code de javascript. În primul rand 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 data 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 Piscel**
- **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.

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
```

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 _imagine
drwxr-xr-x 2 darius darius    4096 mar 11 10:55 _imagine_secreta.png.extracted
darius@bit-sentinel:~/Desktop/unbreakable/inginerie_inversa$
```

Acesta contine arhiva zip și conținutul arhivei.

```
darius@bit-sentinel:~/Desktop/unbreakable/inginerie_inversa/_imagine_secreta.png.extracted$ ls
180  180.zlib  7E3A4.zip  secret.txt
darius@bit-sentinel:~/Desktop/unbreakable/inginerie_inversa/_imagine_secreta.png.extracted$ cat secret.txt
Mesaj Secret
darius@bit-sentinel:~/Desktop/unbreakable/inginerie_inversa/_imagine_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 printeaza 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 intr-un fisier in care sa cautam ulterior flag-ul. Putem face asta cu ajutorul debugger-ului **gdb**:

```
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
0x00007ffff7f5a000 0x00007ffff7ffa000 rw-p      mapped
0x00007ffff7ffa000 0x00007ffff7ffd000 r--p      [vvar]
0x00007ffff7ffd000 0x00007ffff7fff000 r-xp      [vdso]
0x00007ffffffde000 0x00007fffffff000 rw-p      [stack]
0xffffffffff600000 0xffffffffff601000 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[redactat]533397b985c890749cbfc7e306}
ctf{1fe6954870babd55ba6e5d[redactat]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 rulam programul, vom vedea ca acesta cripteaza un string pe care-l controlăm, însă nu știm algoritmul folosit. Pentru a descoperi mai multe informații, deschidem fișierul în IDA Pro și decompilam funcția care criptează input-ul:

```
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 si o cheie din memorie, apoi reprezinta output-ul in 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'),
```

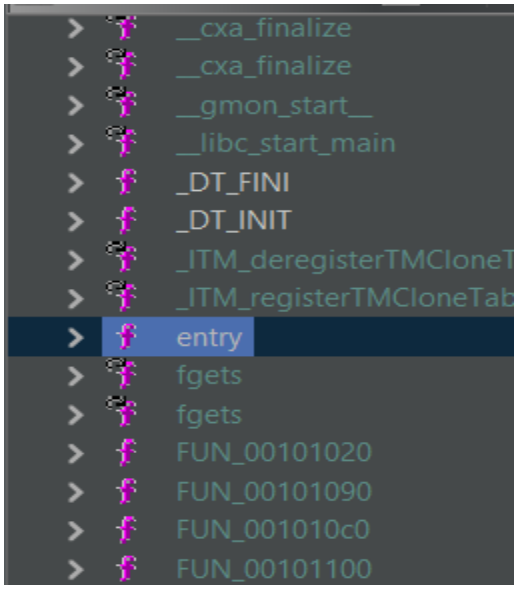
```
b'dsfd;kfoA,.iyewrkldJKDHSUBsgvca69834ncxv')
b'eCTF{Br3ak_th3_Cisc0_B0x}CCUT#H"e\x18tEsr.^'
>>>
```

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.

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 );
}
```

Funcția **entry** apelează funcția **__libc_start_main**, funcție importantă (face parte din LIBC, împreuna 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.

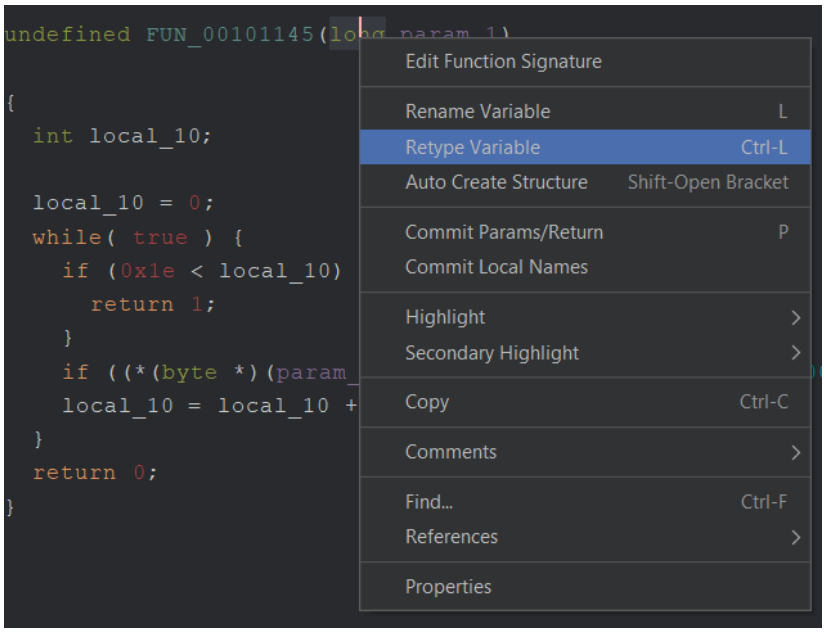
Funcția **__libc_start_main**, pe langa altele, apelează funcția **main**, primind ca prim parametru un pointer către funcția **main**. Deci, FUN_00101199 este **main**. Redenumim funcția, si o accesam.

```
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 in variabila local_38, si apoi apeleaza functia **FUN_00101145** cu parametrul local_38. Apoi, in funcție de valoarea returnată de funcție, hotărăște dacă parola este corecta sau nu. Inspectam funcția **FUN_00101145**:

```
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 daca **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 daca 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(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 0x3f, și verifica dacă este diferit de ce se afla in 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:

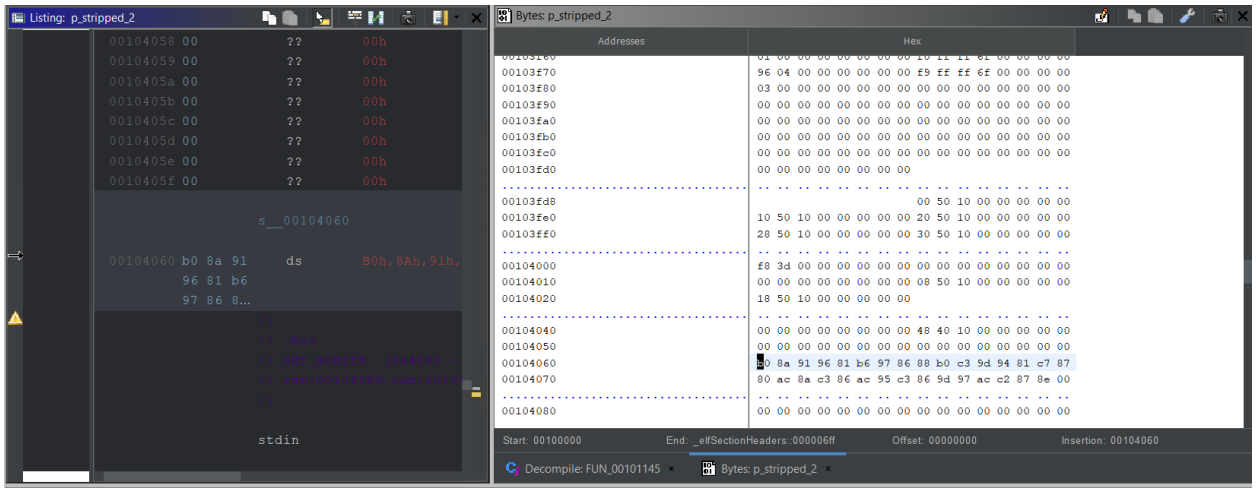
A ^ B = C ⇔ B ^ C = A

În cazul nostru, **A** = caracterele din input, **B** = 0x3f, 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 sa xoram **B**-ul cu **C**-ul.

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

```
                    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 ca la 31(0x1f), while se oprește), ii copiem, si ii 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:

```
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.
```

Contribuitori

- Mihai Dancaescu (yakuhito)
- Moldovan Darius (T3jv1l)
- Emanuel Strugaru (edmund/Th3R4pond0m)
- Andrei Avadanei