

# Reversing Elf

<Write up>

## Crackme1:

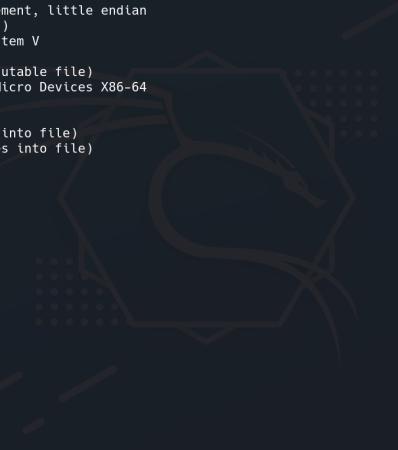
Para resolver este desafio, comecei usando o comando “**readelf**” do linux para ver algumas informações sobre o arquivo, usei a flag “**-h**” para obter seu cabeçalho.



```
Activities Terminal Apr 23 17:02
[dermes@dermes:~]
$ readelf -h index.crackme1
ELF Header:
  Magic: 7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
        ELF64
  Class: 2's complement, little endian
  Version: 1 (current)
  OS/ABI: UNIX - System V
  ABI Version: 0
  Type: EXEC (Executable file)
  Machine: Advanced Micro Devices X86-64
  Version: 0x1
  Entry point address: 0x400450
  Start of program headers: 64 (bytes into file)
  Start of section headers: 5208 (bytes into file)
  Flags: 0x0
  Size of this header: 64 (bytes)
  Size of program headers: 56 (bytes)
  Number of program headers: 8
  Size of section headers: 64 (bytes)
  Number of section headers: 31
  Section header string table index: 28
[dermes@dermes:~]
$
```

Como a propriedade “**Type**” nos mostra, o arquivo é um executável, logo dei o comando “**chmod +x**” para mudar a permissão do arquivo para executá-lo.

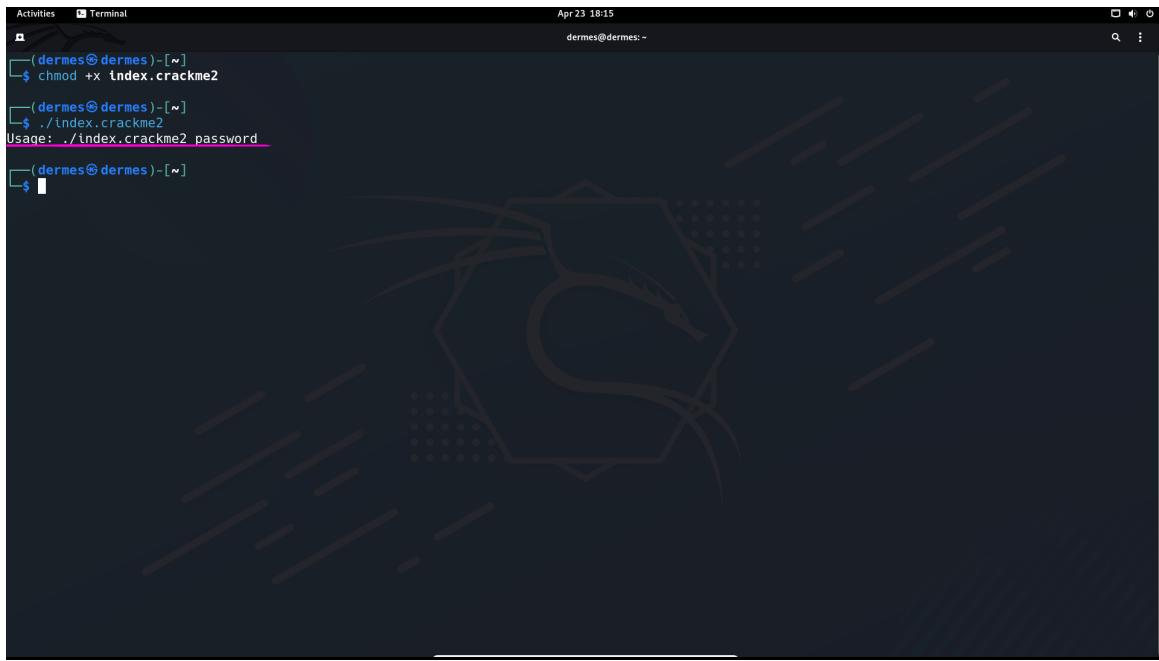
Ao executar o arquivo obtemos nossa flag: **flag{not\_that\_kind\_of\_elf}**



```
Activities Terminal Apr 23 18:10
[dermes@dermes:~]
$ readelf -h index.crackme1
ELF Header:
  Magic: 7f 45 4c 46 02 01 00 00 00 00 00 00 00 00 00 00
        ELF64
  Class: 2's complement, little endian
  Version: 1 (current)
  OS/ABI: UNIX - System V
  ABI Version: 0
  Type: EXEC (Executable file)
  Machine: Advanced Micro Devices X86-64
  Version: 0x1
  Entry point address: 0x400450
  Start of program headers: 64 (bytes into file)
  Start of section headers: 5208 (bytes into file)
  Flags: 0x0
  Size of this header: 64 (bytes)
  Size of program headers: 56 (bytes)
  Number of program headers: 8
  Size of section headers: 64 (bytes)
  Number of section headers: 31
  Section header string table index: 28
[dermes@dermes:~]
$ chmod +x index.crackme1
[dermes@dermes:~]
$ ./index.crackme1
flag{not_that_kind_of_elf}
[dermes@dermes:~]
$
```

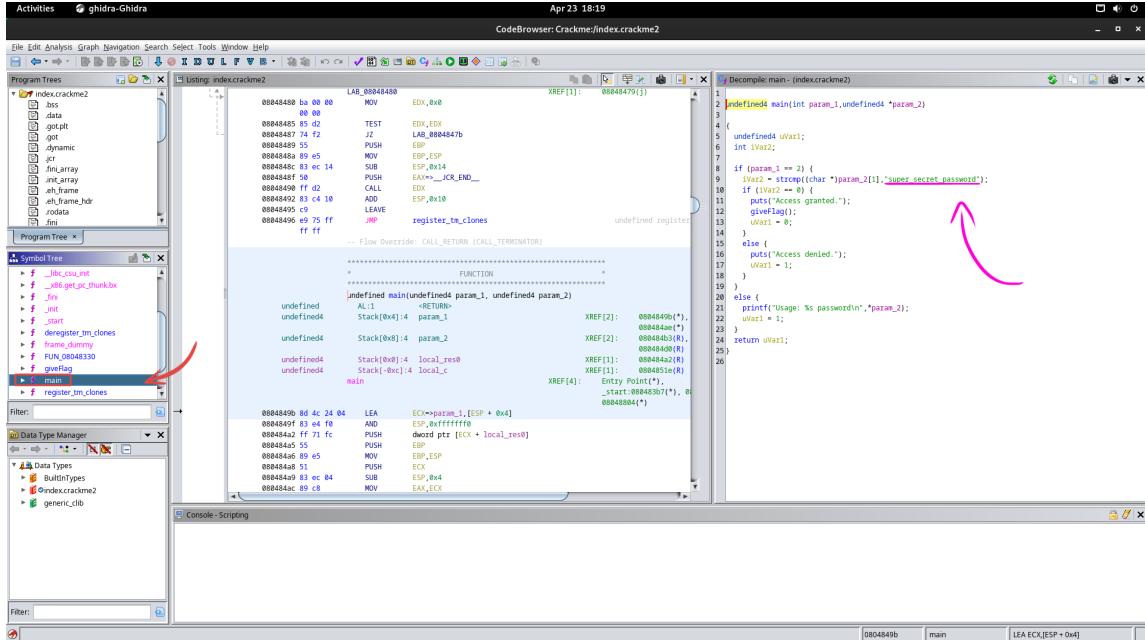
## Crackme2:

No segundo desafio, o programa exige uma senha como podemos ver:



```
[dermes@dermes: ~]
$ chmod +x index.crackme2
[dermes@dermes: ~]
$ ./index.crackme2
Usage: ./index.crackme2 password
[dermes@dermes: ~]
$
```

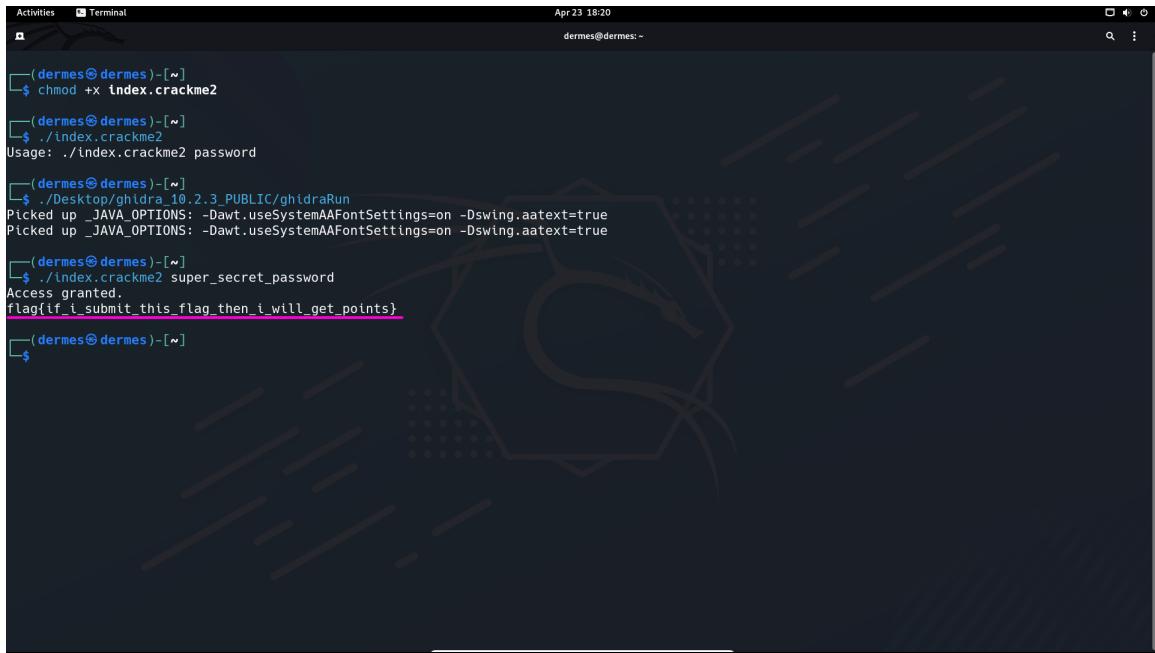
Logo, para poder analisar melhor o que o programa fazia e ver seu source-code usei o Ghidra... olhando para as funções do programa, já fui direto na “main”, já que era familiar por conta de todo código em c/c++ precisar dela para começar



```
1 | #include <stdio.h>
2 | #include <string.h>
3 |
4 | int main(int param_1,undefined4 *param_2)
5 | {
6 |     undefined4 iVar1;
7 |
8 |     if (param_1 == 2) {
9 |         iVar1 = strcmp((char *)param_2[1],"super_secret_password");
10 |         if (iVar1 == 0) {
11 |             puts("Access granted.");
12 |             iVar1 = 1;
13 |         }
14 |         else {
15 |             puts("Access denied.");
16 |             iVar1 = 1;
17 |         }
18 |     }
19 |     else {
20 |         printf("Usage: %s password\n",param_2);
21 |         iVar1 = 1;
22 |     }
23 |     return iVar1;
24 }
```

No código há um “**strcmp**” ou “**string compare**” que é a função para comparar se o parâmetro digitado pelo usuário era o mesmo que o requisitado pelo sistema como podemos ver ali no “**super\_secret\_password**”

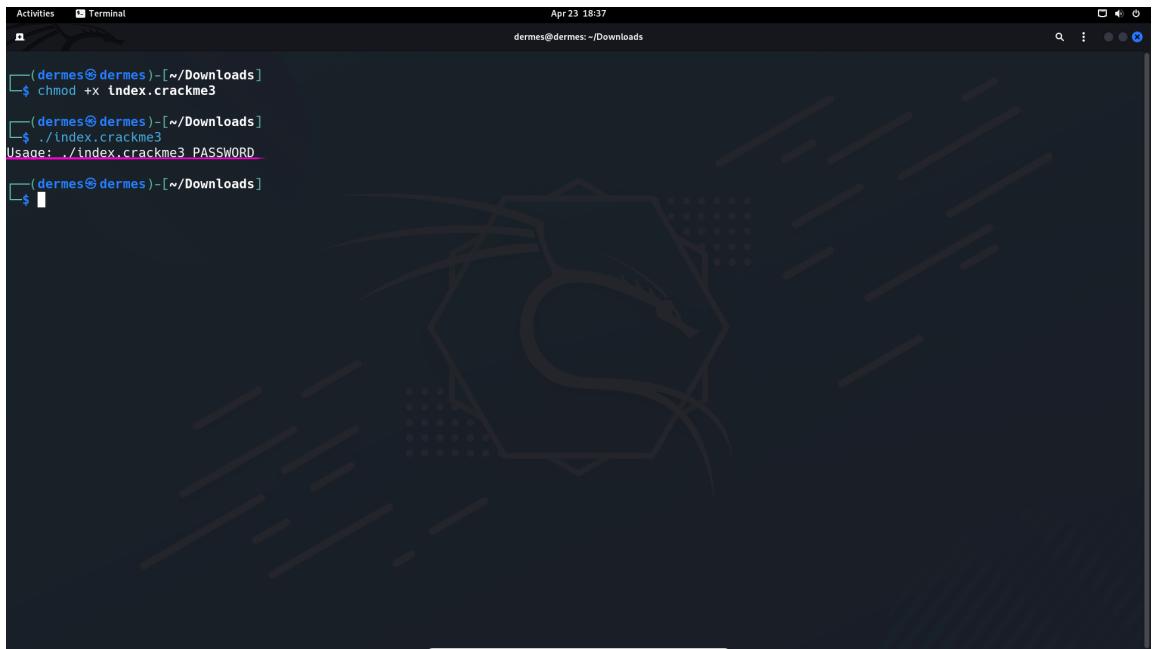
Assim, passei ela como parâmetro para o programa ao executar e obtive a flag:  
**flag{if\_i\_submit\_this\_flag\_then\_i\_will\_get\_points}**



```
Activities Terminal Apr 23 18:20 dermes@dermes: ~
└─[dermes@dermes: ~]
$ chmod +x index.crackme2
└─[dermes@dermes: ~]
$ ./index.crackme2
Usage: ./index.crackme2 password
└─[dermes@dermes: ~]
$ ./Desktop/ghidra_10.2.3_PUBLIC/ghidraRun
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
└─[dermes@dermes: ~]
$ ./Index.crackme2 super_secret_password
Access granted.
flag{if_i_submit_this_flag_then_i_will_get_points}
└─[dermes@dermes: ~]
$
```

## Crackme3:

O sistema, nesse terceiro desafio, também precisa de uma senha para obtermos a flag:



```
Activities Terminal Apr 23 18:37 dermes@dermes: ~/Downloads
└─[dermes@dermes: ~/Downloads]
$ chmod +x index.crackme3
└─[dermes@dermes: ~/Downloads]
$ ./index.crackme3
Usage: ./index.crackme3 PASSWORD
└─[dermes@dermes: ~/Downloads]
$
```

Usando o Ghidra, assim como no desafio anterior, fui logo atrás da função “**main**”, porém nas funções só haviam números na base hexadecimal, então fui vasculhando uma a uma até encontrar a função “**FUN\_080484f4**” que me pareceu muito com a função “**main**” por conta de:

1. Seus parâmetros que o primeiro era do tipo int assim como o “**argc**” (**argument counter**) e o segundo que era um ponteiro igual o “**argv**” (**argument value**)
2. retornava um valor de resposta para caso a função seja executada com êxito ou não

```

080484df b8 00 00 MOV EAX,0x0
080484e4 85 c9 TEST EAX,EAX
080484e6 74 09 JZ LAB_080484f1
080484e8 c7 04 24 MOV dword ptr [ESP]++[local_1c],DAT_0804af24
080484ef 11 00 CALL EAX
LAB_080484f1
080484f1 c9 LEAVE
080484f2 c3 RET
080484f3 90 ?? 90h

***** FUNCTION *****
undefined4 FUN_080484f4(undefined4 param_1, undefined4 pa...
undefined4 AL:1 <RETURN>
Stack[0x14]:4 param_1 XREF[1]: 080484d4(j), 080484e4(j)
Stack[0x14]:4 param_2 XREF[2]: 080484d4(j), 080484e4(j)
Stack[-0x10]:4 local_10 XREF[3]: 080484d4(j), 080484e4(j)

Stack[-0x14]:4 local_14 XREF[4]: 080484d4(j), 080484e4(j)
Stack[-0x18]:4 local_18 XREF[5]: 080484d4(j), 080484e4(j)
Stack[-0x1c]:4 local_1c XREF[10]: 080484d4(j), 080484e4(j)

Decompile: FUN_080484f4 - (index.crackme3)
1 undefined4 FUN_080484f4(int param_1,undefined4 *param_2)
2 {
3     char *_s;
4     size_t sVar1;
5     char *_s_00;
6     int iVar2;
7
8     if (param_1 == 2) {
9         _s = *(char *)param_2[1];
10        iVar2 = strlen(_s);
11        sVar1 = strlen(_s_00);
12        if ((iVar2 == 0x40) && (sVar1 == 0x40)) {
13            if (_s_00 == *(char *)malloc((iVar1 * 2))) {
14                if (_s_00 == *(char *)malloc((iVar1 * 2))) {
15                    fwrite("Hello! failed\n",0xe,1,stderr);
16                }
17            }
18        }
19        iVar2 = strlen(_s_00);
20        sVar1 = strlen(_s_00);
21        if ((iVar2 == 0x40) && (iVar1 == 0x40)) {
22            if ((iVar1 == 0x40) && (iVar2 == 0x40)) {
23                iVar2 = 0x0;
24                iVar1 = 0x0;
25                puts("Correct password!");
26            }
27            puts("Come on, even my aunt Mildred got this one!");
28        }
29    }
30    else {
31        fprintf(stderr,"Usage: %s PASSWORD\n",param_2);
32    }
33    return 0xffffffff;
34}
35

```

Podemos notar a função “**strcmp**” aparecendo novamente, mas agora comparando com uma string codificada em base64 pois possui o sufixo “**==**” o que explicita tal método utilizado

```

080484df b8 00 00 MOV EAX,0x0
080484e4 85 c9 TEST EAX,EAX
080484e6 74 09 JZ LAB_080484f1
080484e8 c7 04 24 MOV dword ptr [ESP]++[local_1c],DAT_0804af24
080484ef ff d8 CALL EAX
LAB_080484f1
080484f1 c9 LEAVE
080484f2 c3 RET
080484f3 90 ?? 90h

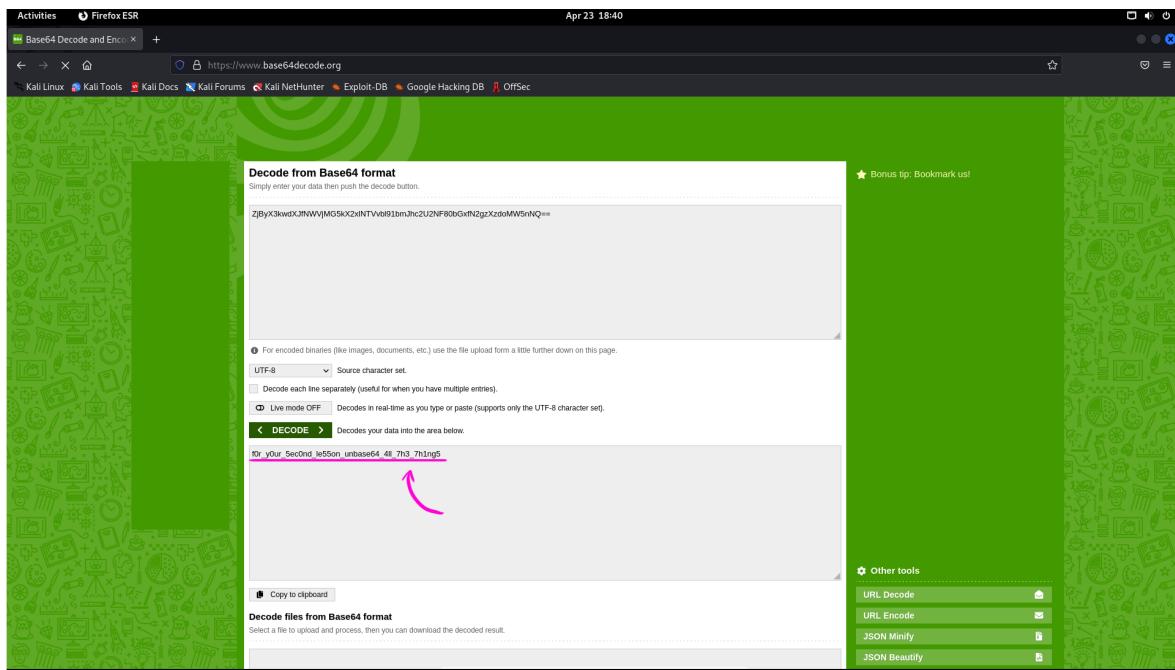
***** FUNCTION *****
undefined4 FUN_080484f4(undefined4 param_1, undefined4 pa...
undefined4 AL:1 <RETURN>
Stack[0x14]:4 param_1 XREF[1]: 080484d4(j), 080484e4(j)
Stack[0x14]:4 param_2 XREF[2]: 080484d4(j), 080484e4(j)
Stack[-0x10]:4 local_10 XREF[3]: 080484d4(j), 080484e4(j)

Stack[-0x14]:4 local_14 XREF[4]: 080484d4(j), 080484e4(j)
Stack[-0x18]:4 local_18 XREF[5]: 080484d4(j), 080484e4(j)
Stack[-0x1c]:4 local_1c XREF[10]: 080484d4(j), 080484e4(j)

Decompile: FUN_080484f4 - (index.crackme3)
1 undefined4 FUN_080484f4(int param_1,undefined4 *param_2)
2 {
3     char *_s;
4     size_t sVar1;
5     char *_s_00;
6     int iVar2;
7
8     if (param_1 == 2) {
9         _s = *(char *)param_2[1];
10        iVar2 = strlen(_s);
11        sVar1 = strlen(_s_00);
12        if ((iVar2 == 0x40) && (sVar1 == 0x40)) {
13            if (_s_00 == *(char *)malloc((iVar1 * 2))) {
14                if (_s_00 == *(char *)malloc((iVar1 * 2))) {
15                    fwrite("Hello! failed\n",0xe,1,stderr);
16                }
17            }
18        }
19        iVar2 = strlen(_s_00);
20        sVar1 = strlen(_s_00);
21        if ((iVar2 == 0x40) && (iVar1 == 0x40)) {
22            if ((iVar1 == 0x40) && (iVar2 == 0x40)) {
23                iVar2 = 0x0;
24                iVar1 = 0x0;
25                puts("Correct password!");
26            }
27            puts("Come on, even my aunt Mildred got this one!");
28        }
29    }
30    else {
31        fprintf(stderr,"Usage: %s PASSWORD\n",param_2);
32    }
33    return 0xffffffff;
34}
35

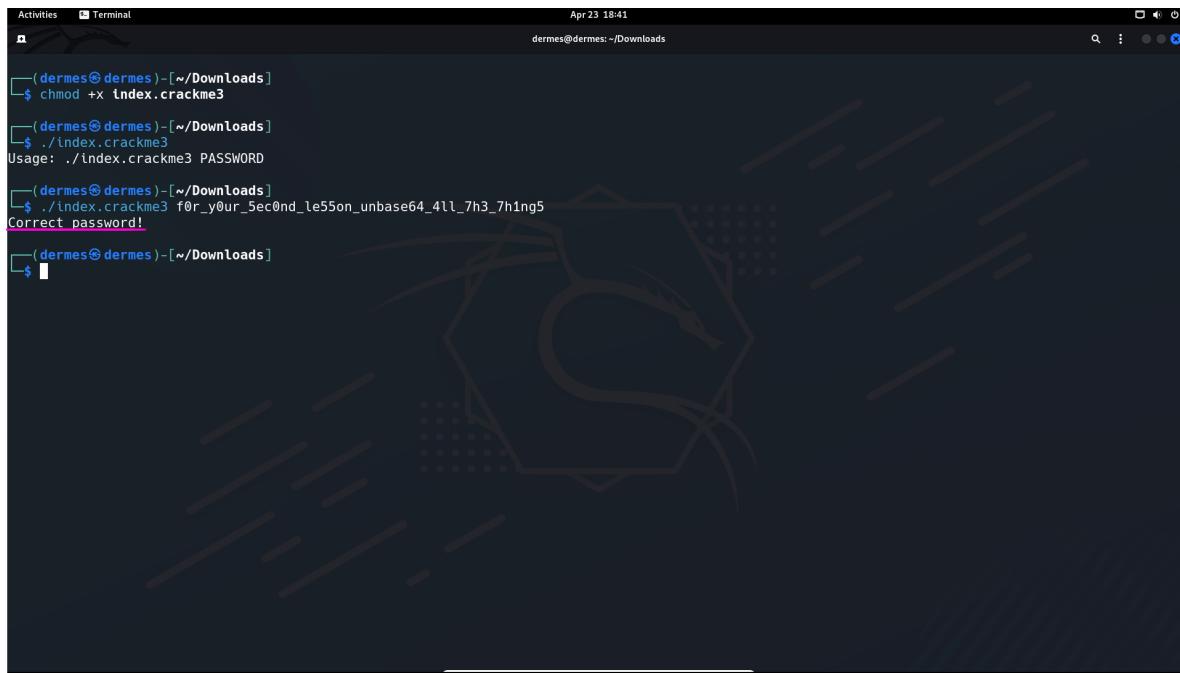
```

Indo no google e procurando por um decodificador de base64, coloquei o código e obtive a senha do programa



Testando isso no programa, percebemos que obtivemos a flag correta:

**f0r\_y0ur\_5ec0nd\_le55on\_unbase64\_4ll\_7h3\_7h1ng5**



## Crackme4:

Para o quarto desafio, assim como os demais também pede uma senha, porém dessa vez ele já nos dá o aviso:

```

Activities Terminal Apr 23 19:19
dermes@dermes:[~/Downloads]
$ chmod +x index.crackme4
[dermes@dermes:[~/Downloads]
$ ./index.crackme4
Usage : ./index.crackme4 password
This time the string is hidden and we used strcmp
[dermes@dermes:[~/Downloads]
$ 

```

Indo na função “**main**” do programa, vemos que ao enviar um argumento, ele executa a função “**compare\_pwd()**”:

Program Tree

Symbol Tree

Data Type Manager

Listing: index.crackme4

Decompile: main - (index.crackme4)

```

1 undefined main(int param_1,undefined *param_2)
2 {
3     if (param_1 == 2) {
4         compare_pwd(param_2[1]);
5     }
6     else {
7         printf("Usage : %s password\nThis time the string is hidden and we used strcmp\n",param_2);
8     }
9 }
10 return 0;
11
12
13

```

Console - Scripting

Nesta função podemos ver a função “**strcmp**” só que dessa vez ela compara nosso input com um ponteiro neste endereço na memória:

The screenshot shows the Ghidra interface with the following details:

- Top Bar:** Activities, ghidra-Ghidra, April 23 2000, CodeBrowser: Crackme/Index.crackme4
- Left Sidebar:** Program Tree (showing files like index.crackme4, bss, .data, got.plt, dynamic, jcr, fin\_array, int\_array, string, eh\_frame\_hdr, rodata, fns), Symbol Tree (showing functions like \_start, compare\_pwd, deregister\_tm\_clones, frame\_dummy, FUN\_00400440, main, register\_tm\_clones, main), and Data Type Manager.
- Middle Area:** Listing window for index.crackme4 showing assembly code for the compare\_pwd function. The assembly code includes instructions like PUSH RBP, MOV RBP, SUB RSP, XOR EAX, MOV QWORD PTR [RBP + local\_10], MOV RAX, and CMP RAX, RDX. The XREF table shows multiple references to this function.
- Right Area:** Decompile window for compare\_pwd showing C-like pseudocode. The code includes variable declarations (char param\_1), function definitions (void compare\_pwd(char \*param\_1)), and logic for password comparison and printing.
- Bottom Area:** Console - Scripting window.

Tentei então obter o que a função “`strcmp`” fazia, porém ao abri-lá obtive só uma chamada para uma outra função “`halt_baddata`”:

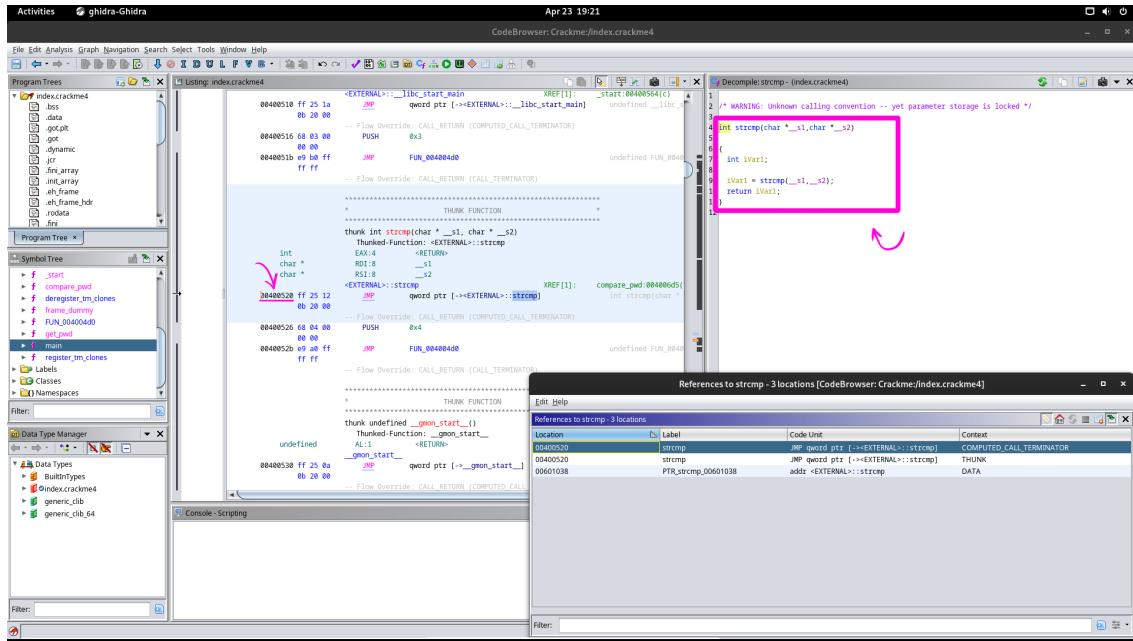
The screenshot shows the Ghidra debugger interface with the following details:

- Program Tree:** Shows the file structure of the crackme binary.
- Symbol Tree:** Shows symbols defined in the binary, including `_start`, `compare_strdup`, `deregister_tm_clones`, `frame_dummy`, `FUN_00400440`, and `get_pwi`.
- Data Type Manager:** Shows data types like `__Data`, `__BSS`, `__DATA`, `__DYNAMIC`, `__got`, `__gotplt`, `__TSD`, `__eh_frame`, `__eh_frame_hdr`, `__rodata`, and `__text`.
- CodeBrowser:** The main workspace displays assembly code for the `_libc_start_main` function. It highlights a call to `strcmp@@GLIBC_2.2.5`. To the right, a decompiled C version of the strcmp logic is shown, with several lines annotated with pink arrows pointing to specific instructions or labels.
- Console - Scripting:** The bottom window shows the command history and output of any scripts run in the debugger.

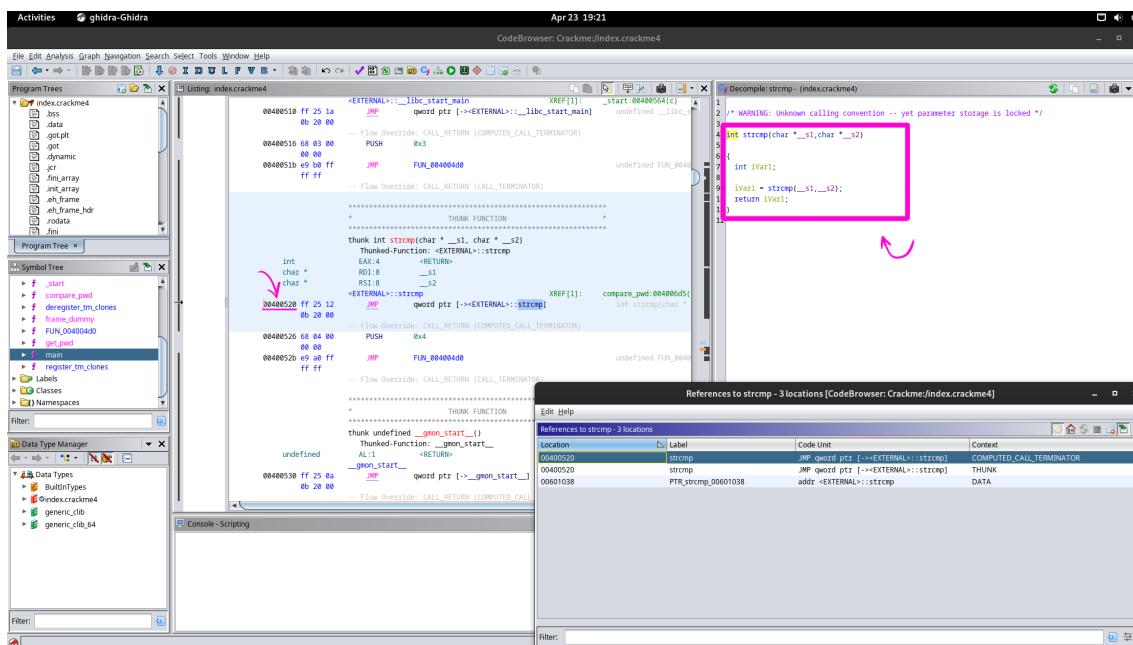
Fui atrás então do que ela fazia e descobri que o Ghidra não pode ler essas instruções pois não suporta elas, porém podemos perceber no comentário acima do código em assembly onde diz “**“THUNK FUNCTION”**”

Ainda atrás de respostas, e muito tempo gasto, vi esse comentário e fui atrás do que significava: quando o Ghidra prefixa thunk, significa que a função passa o controle para outra função de destino ou a chamada para a função API system.

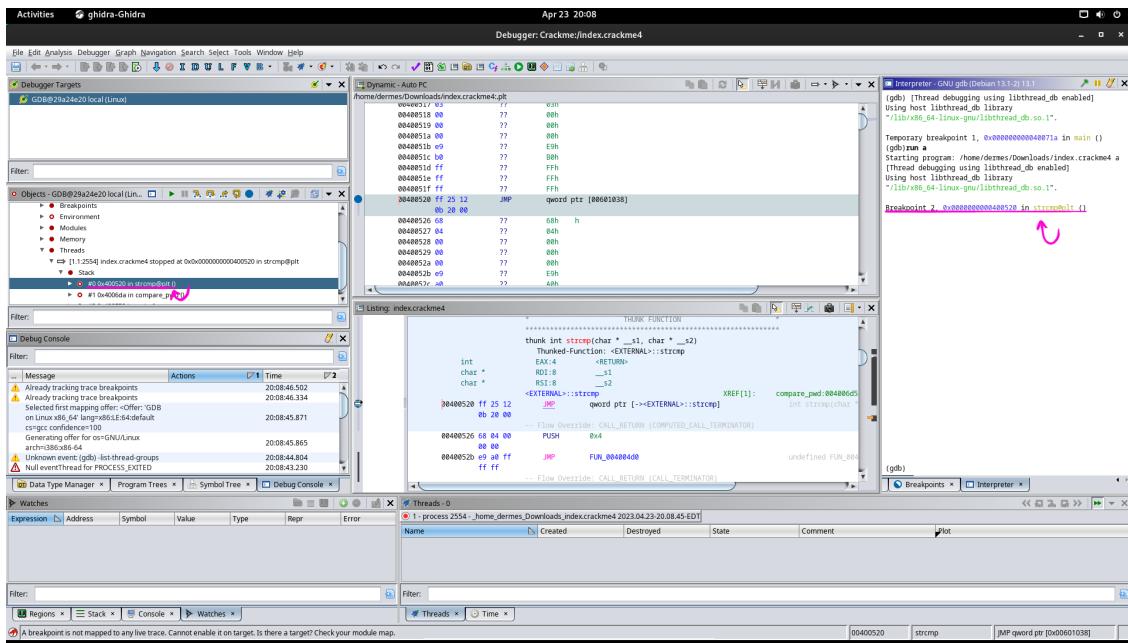
Na minha cabeça fez o seguinte sentido e o que me deu a virada de chave, se funções referenciam outras funções, porque não pegar os endereços que a referenciam?!



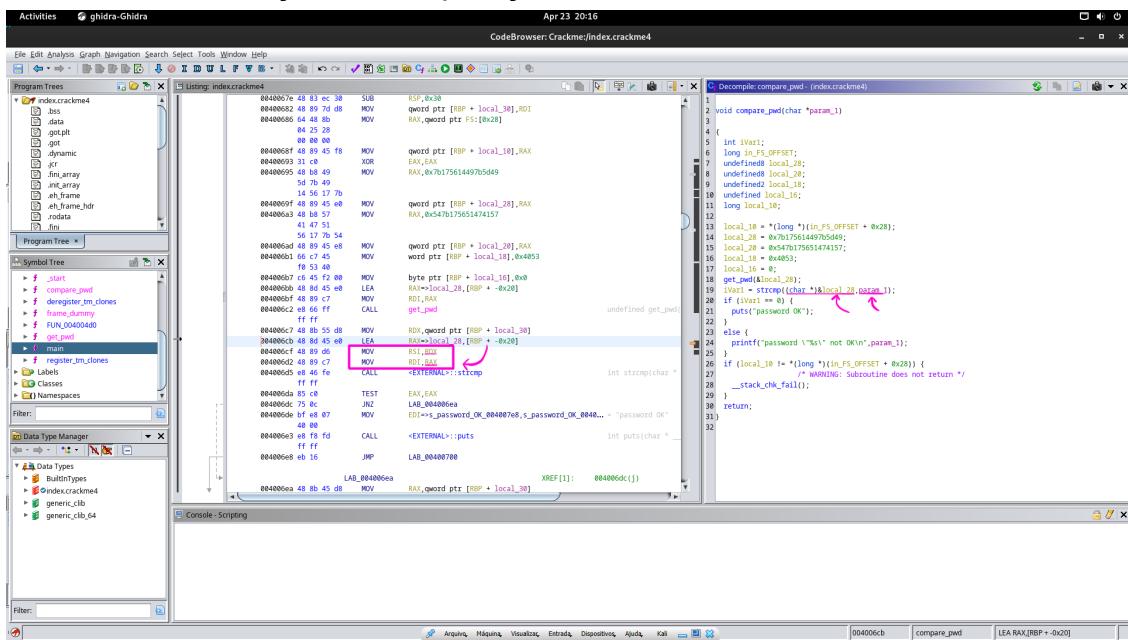
E lá estava ela, o endereço **“0x00400520”** que fazia a chamada da função e retornava se seu valor era 0 ou não



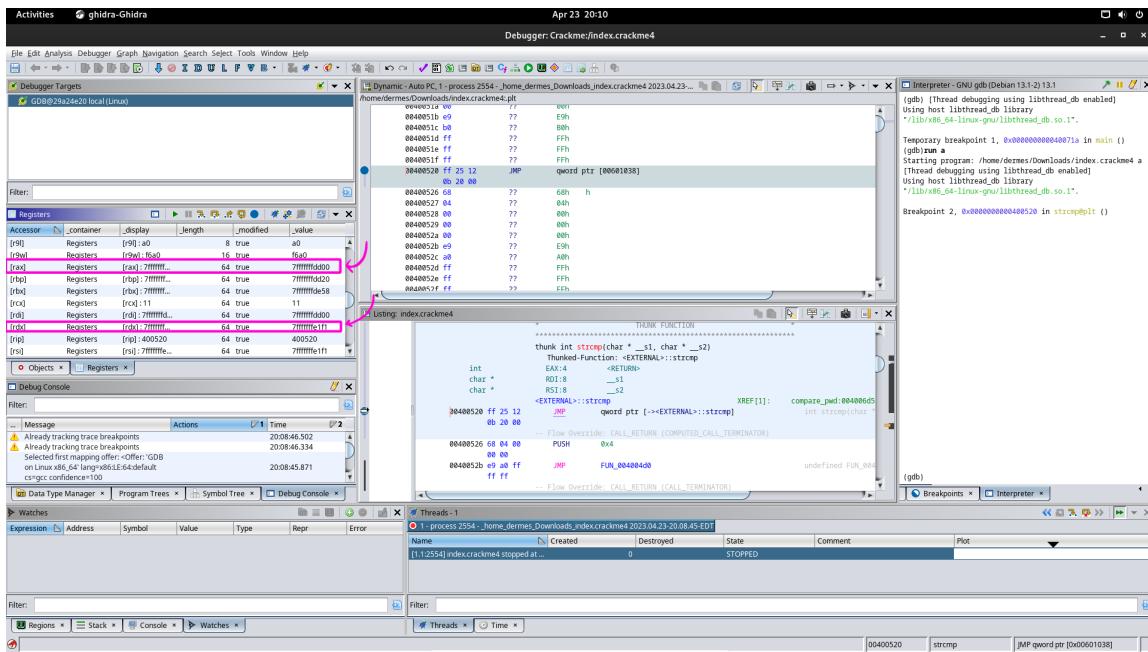
Decidi usar um debugger para poder analisar o que o código fazia enquanto rodava, para isso usei o GDB dentro do Ghidra e coloquei um Breakpoint no endereço que obtive:



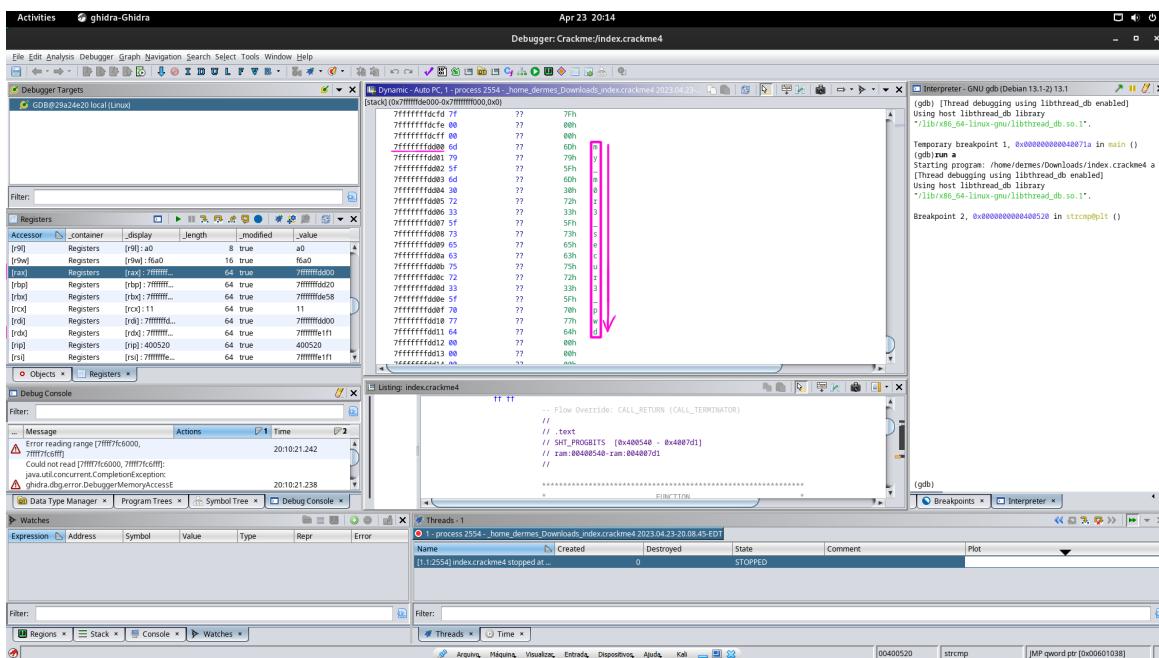
Com o código parado nesse Breakpoint, e a função já na pilha, fui atrás de onde a string de comparação estava armazenada, logo retornei na função “**compare\_pwd**” no disassembler e no código assembly vi a movimentação desses dois registradores na chamada da função de comparação: **RAX** e **RDX**



Voltando ao debugger, analisei os dois e obtive os seguintes endereços:  
**0x7fffffffdd00** e **0x7fffffff1f1**:



Fui então no menu de análise dinâmica do debugger até o endereço do registrador **EAX** e obtive a senha:



Colocando isso no programa para verificar se está correta, vemos a prova que a senha é: **my\_m0r3\_secur3\_pwd**

```
Activities Terminal Apr 24 18:23
dermes@dermes: ~/Downloads
└─$ ./index.crackme4 my_m0r3_secur3_pwd
password OK
└─$
```

A screenshot of a terminal window titled "Terminal". The window shows a command-line interface with a dark background featuring a stylized shield logo. The text in the terminal reads: "Activities Terminal Apr 24 18:23", "dermes@dermes: ~/Downloads", "└─\$ ./index.crackme4 my\_m0r3\_secur3\_pwd", "password OK", and "└─\$". A pink arrow points from the text "password OK" back up towards the command line.

## Crackme5:

No quinto desafio, o programa pede um input da senha ao executar:

```
Activities Terminal Apr 24 18:26
dermes@dermes: ~/Downloads
└─$ chmod +x index.crackme5
└─$ ./index.crackme5
Enter your input: ↗
pass
Always dig deeper
└─$
```

A screenshot of a terminal window titled "Terminal". The window shows a command-line interface with a dark background featuring a stylized shield logo. The text in the terminal reads: "Activities Terminal Apr 24 18:26", "dermes@dermes: ~/Downloads", "└─\$ chmod +x index.crackme5", "└─\$ ./index.crackme5", "Enter your input: ↗", "pass", and "Always dig deeper", followed by the command line prompt "└─\$". A pink arrow points from the text "Enter your input:" back up towards the command line.

Indo no Ghidra para analisar o código, como de costume, analisando a função “**main**”, já notei a “**strcmp\_**” e dessa vez comparando um hexadecimal com um endereço na memória, que novamente estão guardados nos registradores **RDX** e **RAX**:

The screenshot shows the Ghidra interface with the following details:

- Project Tree:** Shows the project structure with files like index.crackme5, .bss, .data, .got.plt, .got, .dynamic, .jcr, fin, array, arrary, ah.frame, ah.frame.hdr, rodata, and .rsrc.
- Symbol Tree:** Shows symbols such as \_start, \_check, \_deregister\_tm\_clones, \_frame\_dummy, and \_FUN\_00400550.
- Data Type Manager:** Shows data types like Builtintypes, \_Dword, \_Qword, generic\_cil, and generic\_cil\_64.
- Code Browser:** The assembly view shows the main function (00400871) which reads user input from standard input (0x80000004) and stores it in memory (0x00400956). It then calls \_\_isoc99\_scanf at 00400871. The assembly code includes comments for EDI and ESI registers.
- Decompiler:** The decompiled C code for the main function is shown below:

```
43     local_31 = *local_24;
44     local_32 = *local_23;
45     local_33 = *local_22;
46     local_34 = *local_21;
47     local_35 = *local_20;
48     local_2f = *local_19;
49     local_2e = *local_18;
50     local_2d = *local_17;
51     local_2c = *local_16;
52     local_2b = *local_15;
53     local_2a = *local_14;
54     local_29 = *local_13;
55     local_28 = *local_12;
56     local_27 = *local_11;
57     local_26 = *local_10;
58     local_25 = *local_9;
59     local_24 = *local_8;
60     local_23 = *local_7;
61     local_22 = *local_6;
62     local_21 = *local_5;
63     local_20 = *local_4;
64     local_1f = *local_3;
65     local_1e = *local_2;
66     local_1d = *local_1;
67     puts("Enter your input: ");
68     iVar1 = *(int *)0x400966; /* *local_5@0x400966, local_5@ */
69     iVar1 = strmem(local_56 + local_38);
70     if ((iVar1 == 0)) {
71         puts("Good game");
72     }
73     else {
74         puts("Always dig deeper");
75     }
76     if ((local_10 != 1) && (*long *)((int_F8_OFFSET + 0x28)) {
77         /* WARNING: Subroutine does not return */
78         _stack_chk_fail();
79     }
80     return 0;
81 }
```

The assembly browser also highlights specific instructions and memory locations with pink arrows and boxes, indicating points of interest or analysis.

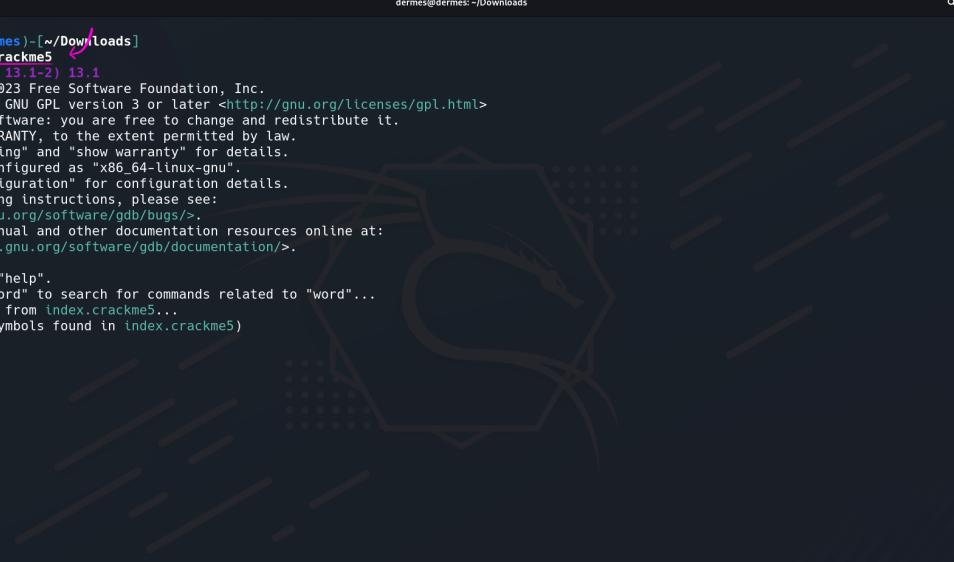
Como dentro da função tinha muitas operações com os parâmetros, ficou meio turvo entender o que o código dizia, mas como era uma função de comparação entre os valores, decidi usar o debugger ali mesmo para ver o que aparecia nos registradores:

The screenshot shows the Ghidra interface with the following details:

- File Tree:** Shows the project structure with files like index.crackme5, lsof, data, getopt, getopt, dyanmic, jcr, fms, array, int, array, eh, frame, eh, frame.hdr, zodata, fms.
- Program Tree:** Shows symbols such as \_start, \_check, \_derefer\_tm\_clones, \_frame\_dummy, \_FUN\_00400550, main, register\_tm\_clones, \_f, \_f\_labels, \_f\_classes, and \_f\_namespaces.
- Data Type Manager:** Shows data types like BuiltInTypes, index.crackme5, generic\_cdb, and generic\_cdb\_64.
- Symbol Tree:** Shows symbols like strcmp\_, strcmp, \_main, \_register\_tm\_clones, \_f, \_f\_labels, \_f\_classes, and \_f\_namespaces.
- Assembly Window:** Displays the assembly code for the strcmp\_ function, including instructions like PUSH RBP, MOV RBP, SUB RSP, MOV QWORD PTR [RBP + local\_38], MOV QWORD PTR [RBP + local\_20], MOV QWORD PTR [RBP + local\_24], XOR, ADD, and JMP.
- CodeBrowser:** Shows the C decompilation of the strcmp\_ function:

```
void strcmp_(char *param_1, char *param_2){  
    size_t svar1;  
    int local_20;  
    int local_1c;  
  
    for (local_20 = 0; local_20 + 0x10; local_20 = local_20 + 1) {  
        local_1c = 0;  
        while (true) {  
            svar1 = strlen(param_1);  
            if (svar1 <= (ulong)(long)local_1c) break;  
            param_1[local_1c] = (byte)key ^ param_1[local_1c];  
            local_1c = local_1c + 1;  
        }  
        strcmp(param_1, param_2, 0xc);  
    }  
    return;  
}
```
- Console/Scripting:** Shows the command `LA_004006F7`.

Tentei executar o GDB no Ghidra mesmo, mas por algum motivo estava dando erro na hora de passar o input, então decidi largar minha zona de conforto e aprender a mexer nele pelo terminal mesmo:



```

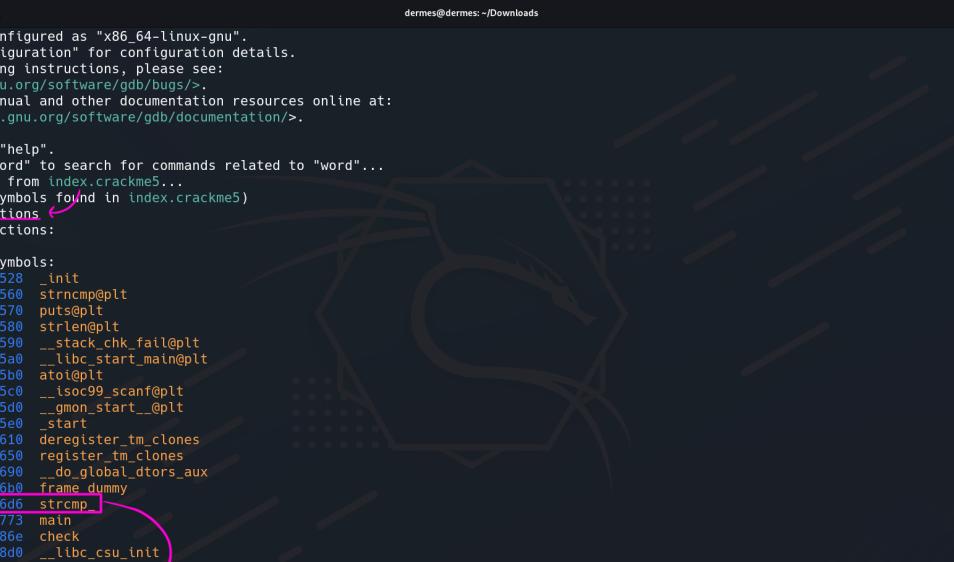
Activities Terminal Apr 24 18:58
dermes@dermes:~/Downloads

$ gdb index.crackme5
GNU gdb (Debian 13.1-2) 13.1
Copyright (C) 2023 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:
<https://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 index.crackme5...
(No debugging symbols found in index.crackme5)
(gdb)

```

Executei o comando “**info functions**” para listar as funções do programa e seus endereços, depois usei o comando “**b**” para setar um breakpoint na função “**strcmp**”.



```

Activities Terminal Apr 24 19:00
dermes@dermes:~/Downloads

This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://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 index.crackme5...
(No debugging symbols found in index.crackme5)
(gdb) info functions
All defined functions:

Non-debugging symbols:
0x0000000000400528 _init
0x0000000000400560 strcmp@plt
0x0000000000400570 puts@plt
0x0000000000400580 strlen@plt
0x0000000000400590 __stack_chk_fail@plt
0x00000000004005a0 __libc_start_main@plt
0x00000000004005b0 atoi@plt
0x00000000004005c0 __isoc99_scanf@plt
0x00000000004005d0 __mon_start__@plt
0x00000000004005e0 __start
0x0000000000400610 deregister_tm_clones
0x0000000000400650 register_tm_clones
0x0000000000400690 __do_global_dtors_aux
0x00000000004005b0 frame_dummy
0x00000000004006d6 strcmp
0x0000000000400773 main
0x000000000040086e check
0x00000000004008d0 __libc_csu_init
0x0000000000400940 __libc_csu_fini
0x0000000000400944 fini
(gdb) b *0x00000000004006d6
Breakpoint 1 at 0x4006d6
(gdb)

```

Rodei o programa com o comando “**run**” e ao passar no breakpoint dei o comando “**info registers**” para mostrar os registradores e seus respectivos endereços. Lá estava os registradores **RAX** e **RDX**, dei então o comando “**x/s**” com o endereço dos registradores para me mostrar o valor guardado no formato de uma string, primeiro foi o **RAX** que possuía o valor que eu havia digitado, depois foi o **RDX** que possuía uma possível senha:

Activities Terminal Apr 24 19:07  
dermes@dermes: ~/Downloads

```
(gdb) run
Starting program: /home/dermes/Downloads/index.crackme5
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Enter your input:
pass

Breakpoint 1, 0x00000000004006d6 in strcmp_ ()
(gdb) info registers
rax      0x7fffffffdd00  140737488346368
rbx      0x7fffffffde68  140737488346728
rcx      0x0                0
rdx      0x7fffffffdd20  140737488346400
rsi      0x7fffffffdd20  140737488346400
rdi      0x7fffffffdd00  140737488346368
rbp      0x7fffffffdd50  0x7fffffffdd50
rsp      0x7fffffffddc8  0x7fffffffddc8
r8       0x400967        4196711
r9       0x7ffff7f98a80  140737353714304
r10      0xfffffffffffff  -1
r11      0x7ffff7f99560  140737353717088
r12      0x0                0
r13      0x7fffffffde78  140737488346744
r14      0x0                0
r15      0x7ffff7ffd020  140737354125344
rip      0x4006d6          0x4006d6 <strcmp_>
eflags   0x206           [ PF IF ]
cs       0x33            51
ss       0x2b            43
ds       0x0              0
es       0x0              0
fs       0x0              0
gs       0x0              0

(gdb) x/s 0x7fffffffdd00
0x7fffffffdd00: "pass"
(gdb) x/s 0x7fffffffdd20
0x7fffffffdd20: "OfdIDSA|3tXb32~X3tX@sX`4tXtz"
(gdb)
```

Testando, percebemos que a senha está correta: **OfdIDSA|3tXb32~X3tX@sX`4tXtz**

Activities Terminal Apr 24 19:19  
dermes@dermes: ~/Downloads

```
[dermes@dermes:~/Downloads]
$ ./index.crackme5
Enter your input:
OfdIDSA|3tXb32~X3tX@sX`4tXtz
Good game
```

## Crackme6:

Ao ser executado, o programa pede uma senha e já nos dá uma dica de ler o código fonte:

```
dermes@dermes:[~/Downloads]
$ chmod +x index.crackme6
dermes@dermes:[~/Downloads]
$ ./index.crackme6
Usage : ./index.crackme6 password
Good luck, read the source
dermes@dermes:[~/Downloads]
```

No Ghidra vemos um pouco do que já vimos anteriormente, na função “**main**” tem uma chamada para a função “**compare\_pwd**” e de diferencial que tem uma chamada pra uma função chamada “**my\_secure\_test**”:

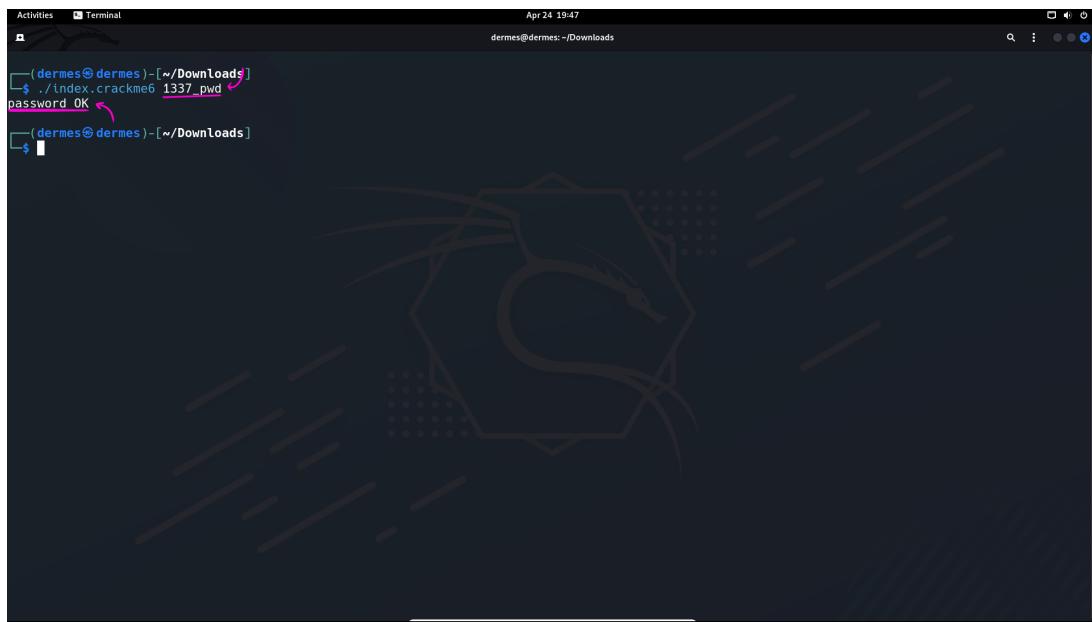
The screenshot shows the Ghidra interface with the following details:

- Program Tree:** Shows the file structure of index.crackme6.
- Symbol Tree:** Shows symbols like `_libc_csu_init`, `_fini`, `_start`, `main`, `compare_pwd`, `deregister_tm_clones`, `frame_dummy`, `FUN_00400440`, `my_secure_test`, and `printf`.
- Decompiler:** The main function (`main`) is shown with assembly code and C-like pseudocode. It includes calls to `my_secure_test` and `compare_pwd`.
- Console - Scripting:** Displays the command `00400711 main PUSH RBP`.

Nesta função, vemos um teste baseado em ponteiro de caracteres e em cada valor que compõe essa string.

A árvore condicional verifica se é diferente do que o programa pede, assim retorna o maior valor positivo no caso “**0xffffffff**”, para alcançar o resultado esperado que é 0, devemos digitar então “**1337\_pwd**”:

Colocando o valor no programa vemos que a senha está correta: **1337\_pwd**



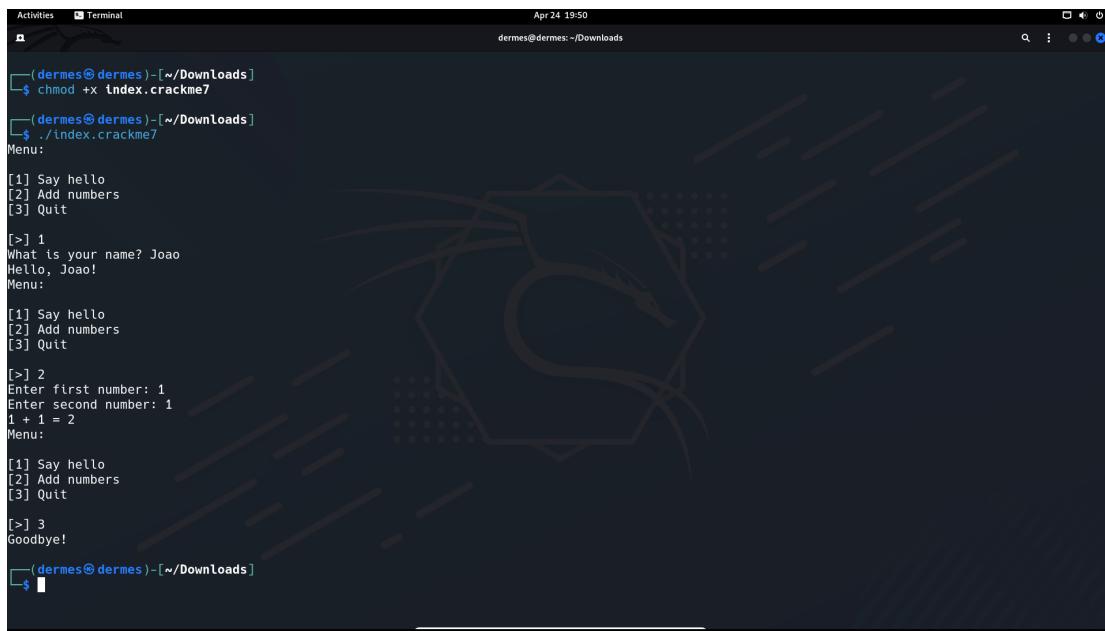
```
Activities Terminal
(dermes@dermes) [~/Downloads]
$ ./index_crackme6 1337_pwd
password OK
(dermes@dermes) [~/Downloads]
$
```

## Crackme7:

Neste programa, diferente dos demais apresenta 3 opções de operações possíveis:

1. Dizer olá
2. Somar números
3. Sair

O que nos faz pensar em como obter a flag:



```
Activities Terminal
(dermes@dermes) [~/Downloads]
$ chmod +x index_crackme7
(dermes@dermes) [~/Downloads]
$ ./index_crackme7
Menu:
[1] Say hello
[2] Add numbers
[3] Quit

[>] 1
What is your name? Joao
Hello, Joao!
Menu:

[1] Say hello
[2] Add numbers
[3] Quit

[>] 2
Enter first number: 1
Enter second number: 1
1 + 1 = 2
Menu:

[1] Say hello
[2] Add numbers
[3] Quit

[>] 3
Goodbye!
(dermes@dermes) [~/Downloads]
$
```

Indo na função “main” do programa vemos um condicional testando a variável “`local_14`”, a mesma que armazena a opção escolhida, para o valor em hexadecimal **0x7a69** que se convertermos para base decimal obteremos **31337** uma possível opção escondida no programa, que nos levará a chamada da função “`giveFlag`”:

```

Apr 24 19:55
CodeBrowser: Crackme/Index.crackme7

Decompiled: main - (index.crackme7)

19 printf("\n");
20 iVar1 = _ioc99_scanf(0x401_88848814,&local_14);
21 if (iVar1 != 1) {
22     puts("Unknown Input!");
23     return 1;
24 }
25 if (local_14 != 1) break;
26 printf("What is your name? ");
27 iVar1 = local_80;
28 for (iVar1 = 0x19; iVar1 != 0; iVar1 = iVar1 + -1) {
29     iVar2 = iVar2 + (uint)iVar3 * 2 + 1;
30 }
31 iVar1 = _ioc99_scanf(0x401_8884883a,local_80);
32 if (iVar1 != 1) {
33     iVar1 = 0;
34     iVar2 = iVar2 + (uint)iVar3 * 2 + 1;
35 }
36 iVar1 = 0;
37 printf("Hello, %s!\n",local_80);
38
39 if (local_14 != 2) {
40     if (local_14 == 3) {
41         puts("Goodbye!");
42     }
43 else_if (local_14 == 0x7a69) {
44     puts("Wow such h4x0r!");
45 }
46 else {
47     printf("Unknown choice: %d\n",local_14);
48 }
49 }
50
51 iVar1 = 0;
52 printf("Enter first number: ");
53 iVar1 = _ioc99_scanf(0x401_88848875,&local_18);
54 if (iVar1 != 1) break;
55 printf("Enter second number: ");
56 iVar1 = _ioc99_scanf(0x401_88848875,&local_1c);
57 if (iVar1 != 1) break;

```

Testando ela então obtemos nossa flag: `flag{much_reversing_very_ida_wow}`

```

Activities Terminal
dermes@dermes: ~/Downloads
$ ./Index.crackme7
Menu:
[1] Say hello
[2] Add numbers
[3] Quit
[>] 6408
Unknown choice: 6408

[1] Say hello
[2] Add numbers
[3] Quit
[>] 31337
Wow such h4x0r!
flag{much_reversing_very_ida_wow}

$ 

```

## Crackme8:

Bem simples, o programa também pede uma senha:

```
Activities Terminal Apr 24 20:08
dermes@dermes: ~/Downloads

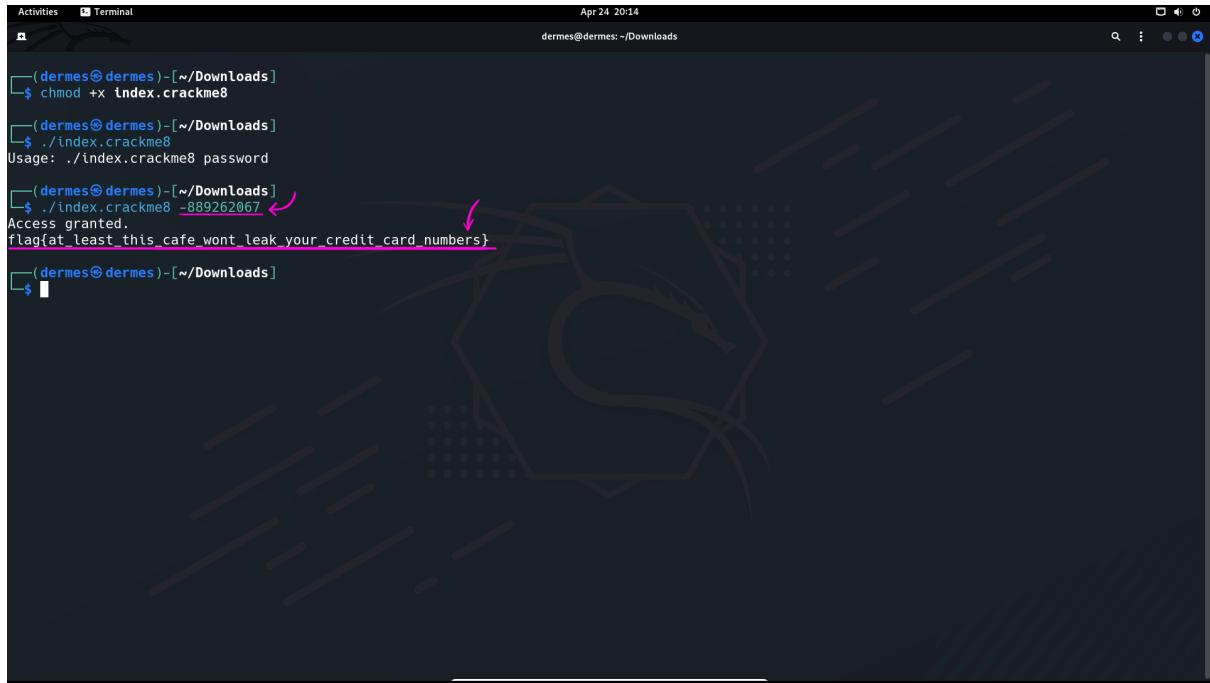
└─(dermes@dermes) [~/Downloads]
$ chmod +x index.crackme8
└─(dermes@dermes) [~/Downloads]
$ ./index.crackme8
Usage: ./index.crackme8 password
└─(dermes@dermes) [~/Downloads]
$
```

Na função “**Main**” já podemos observar a condição que nos dará a flag com a chamada da “**giveFlag**” e ela compara se o parâmetro passado é igual ao hexadecimal **-0x35010ff3**:

The screenshot shows the Ghidra debugger interface with the following details:

- Title Bar:** Activities, ghidra-Ghidra, Apr 20 2011, CodeBrowser: Crackme/index.crackme8
- Program Trees:** Shows the file structure of index.crackme8.
- Symbol Tree:** Shows symbols such as `_ZLc_main`, `_ZLc_get_pc_thunk$bx`, `_ZLc_fini`, `_ZLc_start`, `_ZLc_deregister_tm_clones`, `_ZLc_f_PUN_000405330`, `_ZLc_giveFlag`, and `_ZLc_main`.
- Data Type Manager:** Shows data types like `FileTypes`, `BuiltinTypes`, `IndexerTypes`, and `generic.cib`.
- Code View:** Displays assembly code for `main()` at address `00404480`. The code includes instructions like `MOV EDX, ECX`, `TEST EDX, EDX`, and `JZ LAB_0040447b`. A pink arrow points from the `giveFlag` symbol in the Symbol Tree to the `JMP register_tm_clones` instruction.
- Decompiler:** Shows the decompiled C-like code for `main()`:1 undefined4 main(undefined4 param\_1,undefined4 \*param\_2)
2 {
3 undefined4 iVar1;
4 int iVar2;
5
6 if (param\_1 == 2) {
7 iVar1 = \*(char \*)param\_2[1];
8 if (iVar1 == 0x30303030) {
9 giveFlag();
10 iVar1 = 0;
11 }
12 else {
13 puts("Access denied.");
14 iVar1 = 1;
15 }
16 }
17 else {
18 printf("Usage: %s password\n",\*param\_2);
19 iVar1 = 1;
20 }
21
22 return iVar1;
23 }
- Console:** Shows the command `register_tm_clones`.
- Registers:** Shows registers `ECX`, `ESP`, `EDX`, `EBP`, `ECX`, `ESI`, `EDI`, and `EBP`.
- Stack:** Shows the stack contents at `00404480` and `00404490`.

Convertendo o valor em hexadecimal para base decimal obteremos o valor **-889262067**, usando ele como senha obteremos nossa flag:  
**flag{at\_least\_this\_cafe\_wont\_leak\_your\_credit\_card\_numbers}**



```
(dermes@dermes:[~/Downloads]
$ chmod +x index.crackme8
[dermes@dermes:[~/Downloads]
$ ./index.crackme8
Usage: ./index.crackme8 password
[dermes@dermes:[~/Downloads]
$ ./index.crackme8 -889262067
Access granted.
flag{at_least_this_cafe_wont_leak_your_credit_card_numbers}
[dermes@dermes:[~/Downloads]
$
```

## Conclusão:

Adorei realizar este CTF, adquiri muito conhecimento na forma de organização dos binários de cada tipo de arquivo e muitas ferramentas para poder trabalhar com engenharia reversa, foi simplesmente incrível!!!

