CrackMe#00

I do not have a source site for downloading this crackme because I do not remember how it came to me. Probably in one of these many searches I did online.

Crackme by b1h0 https://crackmes.one/user/b1h0

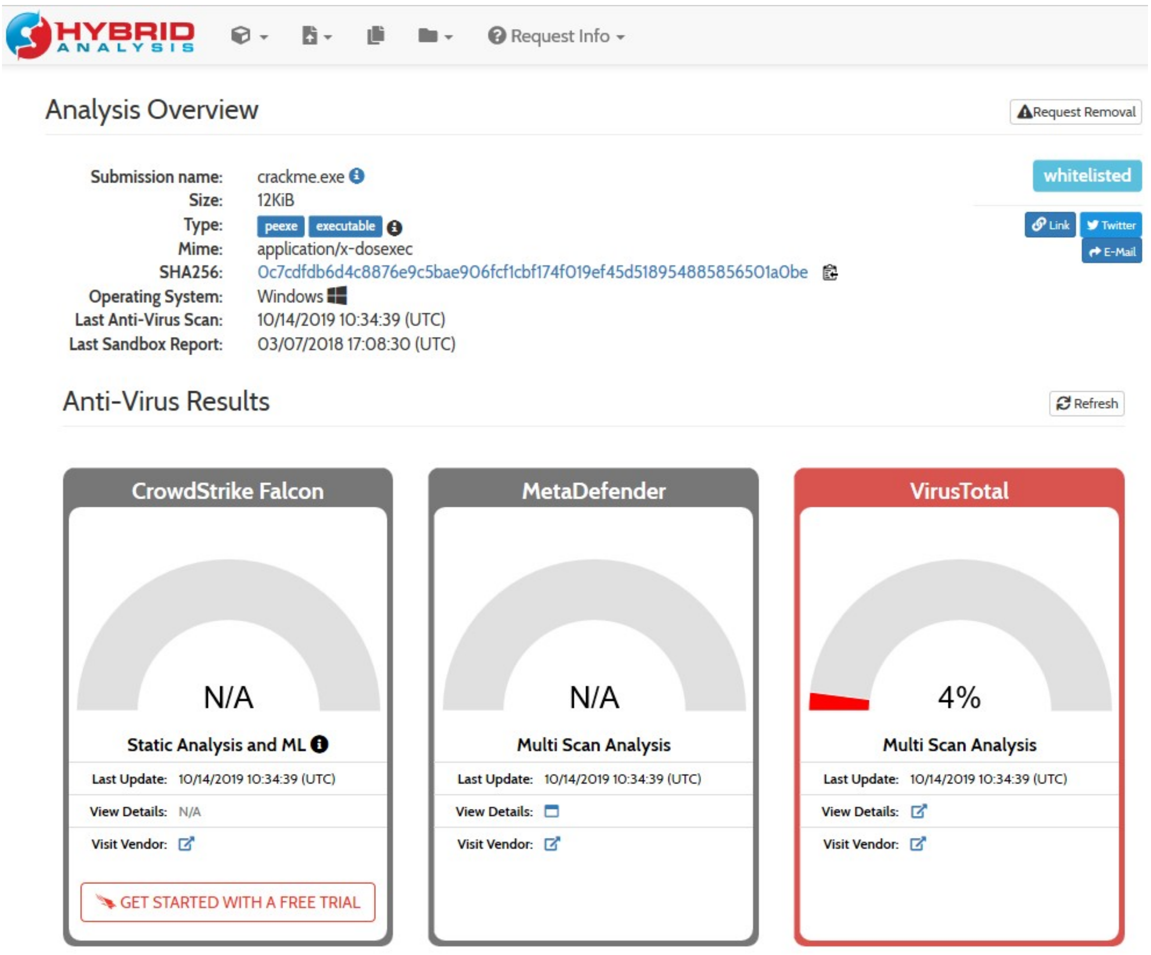
Date: 03/nov/2019

You can download it from this link . The password is **1337**.

Attention!!!! Maybe it's a false positive. But it should be alert. **Hybrid Analysis** whitelisted this file, but can be a Trojan.

Post-note: It is a false positive. The file is safe. But you should always be alert and analyze the binaries with different sources and never try them on a machine in production. Always use a Virtual Machine.

Hybrid Analysis Overview



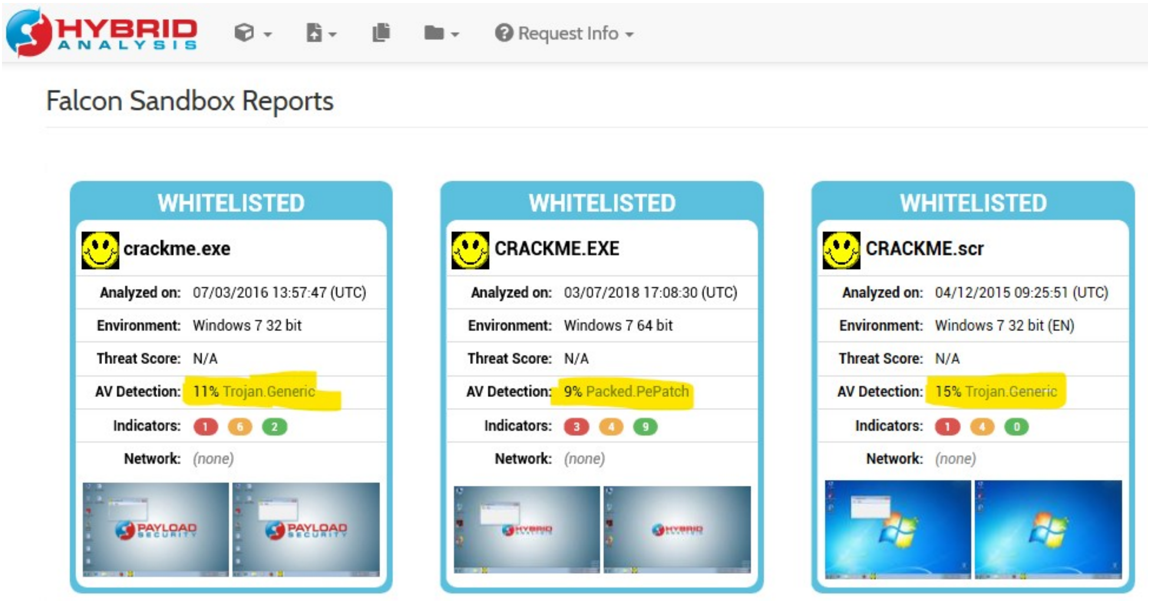
Analysis Overview

Submission name:	crackme.exe
Size:	12KiB
Type:	peexe executable
Mime:	application/x-dosexec
SHA256:	0c7cdfdb6d4c8876e9c5bae906fcf1cbf174f019ef45d518954885856501a0be
Operating System:	Windows
Last Anti-Virus Scan:	10/14/2019 10:34:39 (UTC)
Last Sandbox Report:	03/07/2018 17:08:30 (UTC)

Anti-Virus Results

CrowdStrike Falcon	MetaDefender	VirusTotal
N/A	N/A	4%
Static Analysis and ML	Multi Scan Analysis	Multi Scan Analysis
Last Update: 10/14/2019 10:34:39 (UTC)	Last Update: 10/14/2019 10:34:39 (UTC)	Last Update: 10/14/2019 10:34:39 (UTC)
View Details: N/A	View Details:	View Details:
Visit Vendor:	Visit Vendor:	Visit Vendor:
GET STARTED WITH A FREE TRIAL		

Hybrid Analysis Sandbox



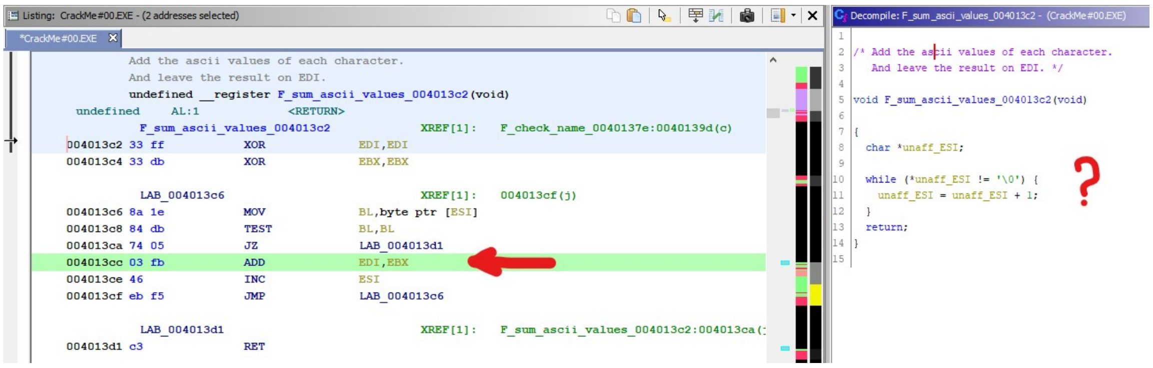
Falcon Sandbox Reports

WHITELISTED	WHITELISTED	WHITELISTED
crackme.exe	CRACKME.EXE	CRACKME.scr
Analyzed on: 07/03/2016 13:57:47 (UTC)	Analyzed on: 03/07/2018 17:08:30 (UTC)	Analyzed on: 04/12/2015 09:25:51 (UTC)
Environment: Windows 7 32 bit	Environment: Windows 7 64 bit	Environment: Windows 7 32 bit (EN)
Threat Score: N/A	Threat Score: N/A	Threat Score: N/A
AV Detection: 11% Trojan.Generic	AV Detection: 9% Packed.PePatch	AV Detection: 15% Trojan.Generic
Indicators: 1 6 2	Indicators: 3 4 9	Indicators: 1 4 0
Network: *(none)*	Network: *(none)*	Network: *(none)*

To analyze this file I have used **Ghidra** and **x64dbg**, simply by trying both tools and seeing the differences. It is good to contrast and train in both dynamic and static.

Ghidra - notes

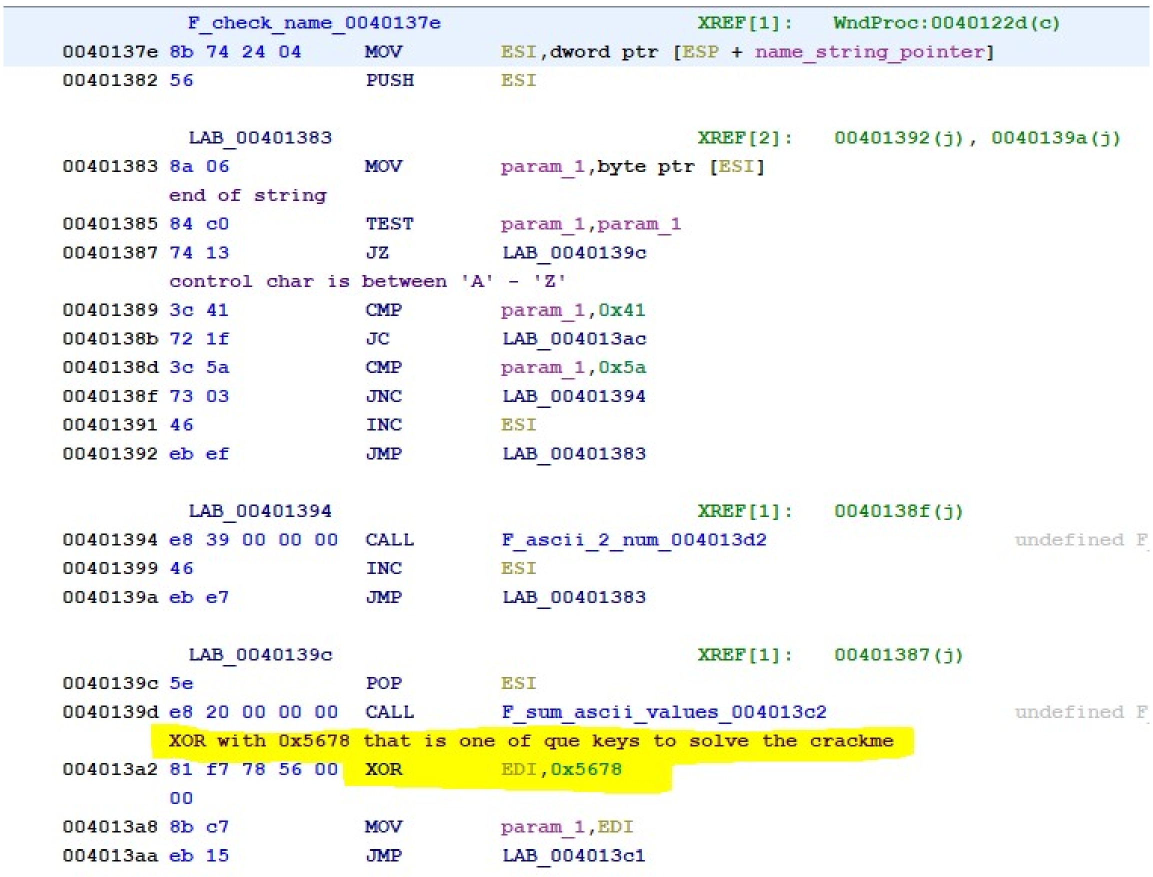
Ghidra is not very effective decompiling the code when it comes to a binary created directly in assembler and not a language like C. In this case you can see how part of the code that accumulates the ascii value of the characters in a string has been skipped. Probably for using the EDI registry since it is not normal.


Listing: CrackMe#00.EXE - (2 addresses selected)
*CrackMe#00.EXE
Decompile: F_sum_ascii_values_004013c2 - (CrackMe#00.EXE)
?

```
                 Add the ascii values of each character.
                 And leave the result on EDI.
                 undefined __register F_sum_ascii_values_004013c2(void)
     undefined      AL:1           <RETURN>
                  F_sum_ascii_values_004013c2                 XREF[1]:     F_check_name_0040137e:0040139d(c)
        004013c2 33 ff          XOR          EDI,EDI
        004013c4 33 db          XOR          EBX,EBX

                  LAB_004013c6                                XREF[1]:     004013cf(j)
        004013c6 8a 1e          MOV          BL,byte ptr [ESI]
        004013c8 84 db          TEST         BL,BL
        004013ca 74 05          JZ           LAB_004013d1
        004013cc 03 fb          ADD          EDI,EBX
        004013ce 46             INC          ESI
        004013cf eb f5          JMP          LAB_004013c6

                  LAB_004013d1                                XREF[1]:     F_sum_ascii_values_004013c2:004013ca(
        004013d1 c3             RET
```

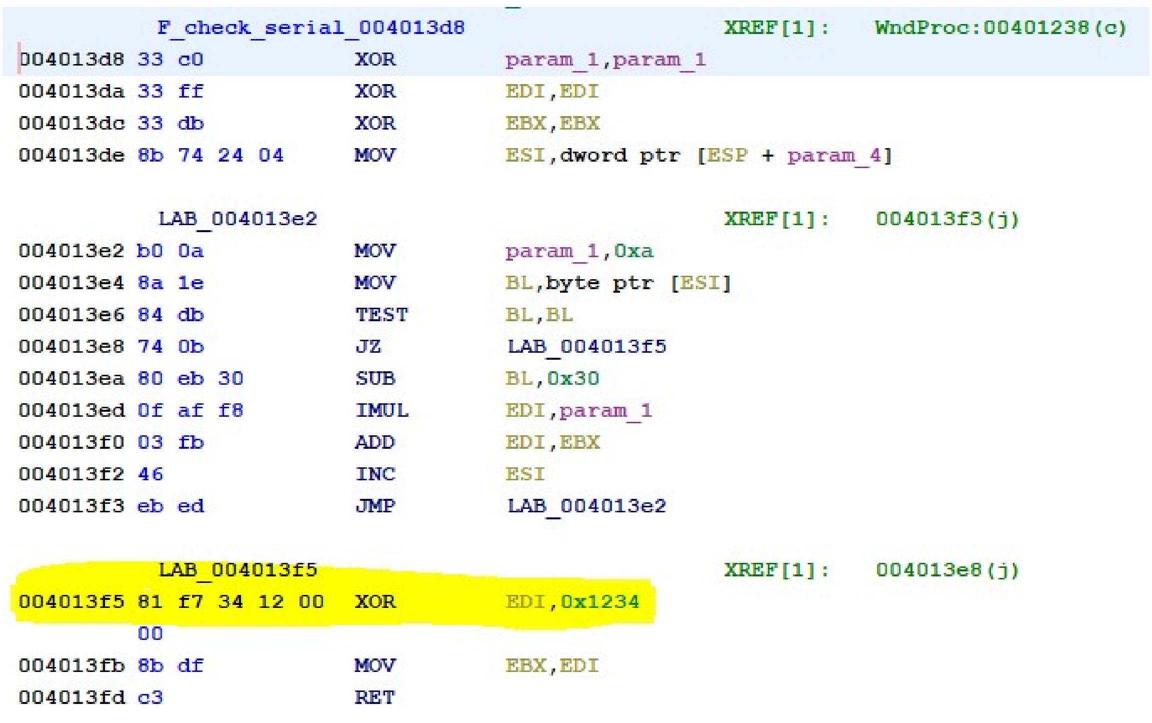
```

/* Add the ascii values of each character.
   And leave the result on EDI. */

void F_sum_ascii_values_004013c2(void)

{
  char *unaff_ESI;

  while (*unaff_ESI != '\0') {
    unaff_ESI = unaff_ESI + 1;
  }
  return;
}

```

At **0040137e** we have a function that checks the name after entering it, which actually makes a sum of the **ASCII** values of each character and then makes an **XOR** with the value **0x5678** which is one of the keys to solve this crackme.



```
                  F_check_name_0040137e                         XREF[1]:     WndProc:0040122d(c)
        0040137e 8b 74 24 04     MOV          ESI,dword ptr [ESP + name_string_pointer]
        00401382 56              PUSH         ESI

                  LAB_00401383                                  XREF[2]:     00401392(j), 0040139a(j)
        00401383 8a 06           MOV          param_1,byte ptr [ESI]
                  end of string
        00401385 84 c0           TEST         param_1,param_1
        00401387 74 13           JZ           LAB_0040139c
                  control char is between 'A' - 'Z'
        00401389 3c 41           CMP          param_1,0x41
        0040138b 72 1f           JC           LAB_004013ac
        0040138d 3c 5a           CMP          param_1,0x5a
        0040138f 73 03           JNC          LAB_00401394
        00401391 46              INC          ESI
        00401392 eb ef           JMP          LAB_00401383

                  LAB_00401394                                  XREF[1]:     0040138f(j)
        00401394 e8 39 00 00 00  CALL         F_ascii_2_num_004013d2                      undefined F
        00401399 46              INC          ESI
        0040139a eb e7           JMP          LAB_00401383

                  LAB_0040139c                                  XREF[1]:     00401387(j)
        0040139c 5e              POP          ESI
        0040139d e8 20 00 00 00  CALL         F_sum_ascii_values_004013c2                 undefined F
                  XOR with 0x5678 that is one of que keys to solve the crackme
        004013a2 81 f7 78 56 00  XOR          EDI,0x5678
                  00
        004013a8 8b c7           MOV          param_1,EDI
        004013aa eb 15           JMP          LAB_004013c1
```

From address **004013d8** we have the function that I call **check_serial**, which checks the serial number, converting the string value to numerical and making an **XOR** with the value **0x1234**, which is the second key to solve this crackme.

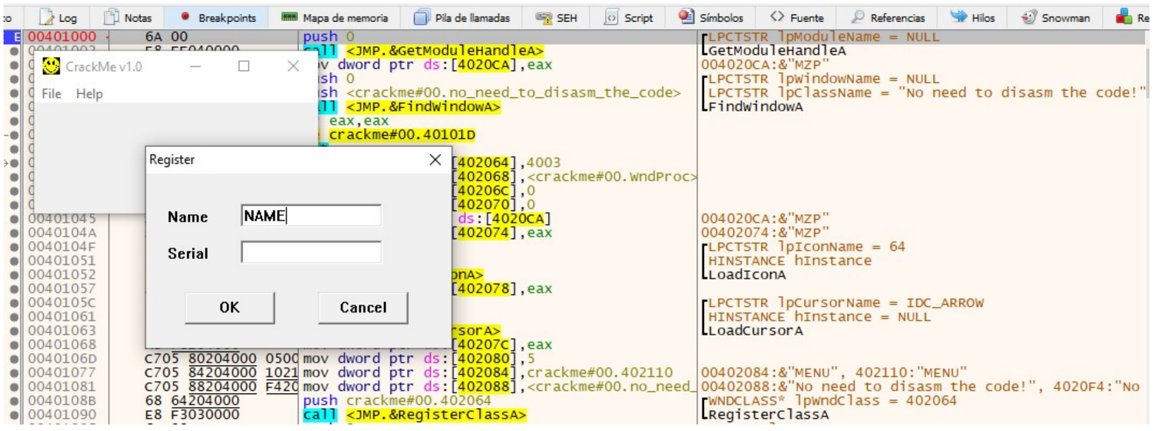


```
                  F_check_serial_004013d8                      XREF[1]:     WndProc:00401238(c)
004013d8 33 c0              XOR          param_1,param_1
004013da 33 ff              XOR          EDI,EDI
004013dc 33 db              XOR          EBX,EBX
004013de 8b 74 24 04        MOV          ESI,dword ptr [ESP + param_4]

                  LAB_004013e2                                 XREF[1]:     004013f3(j)
004013e2 b0 0a              MOV          param_1,0xa
004013e4 8a 1e              MOV          BL,byte ptr [ESI]
004013e6 84 db              TEST         BL,BL
004013e8 74 0b              JZ           LAB_004013f5
004013ea 80 eb 30           SUB          BL,0x30
004013ed 0f af f8           IMUL         EDI,param_1
004013f0 03 fb              ADD          EDI,EBX
004013f2 46                 INC          ESI
004013f3 eb ed              JMP          LAB_004013e2

                  LAB_004013f5                                 XREF[1]:     004013e8(j)
004013f5 81 f7 34 12 00     XOR          EDI,0x1234
         00
004013fb 8b df              MOV          EBX,EDI
004013fd c3                 RET
```

After analyzing different parts of the code I prefer to go directly to the analysis with **x64dbg**.

x64dbg - notes

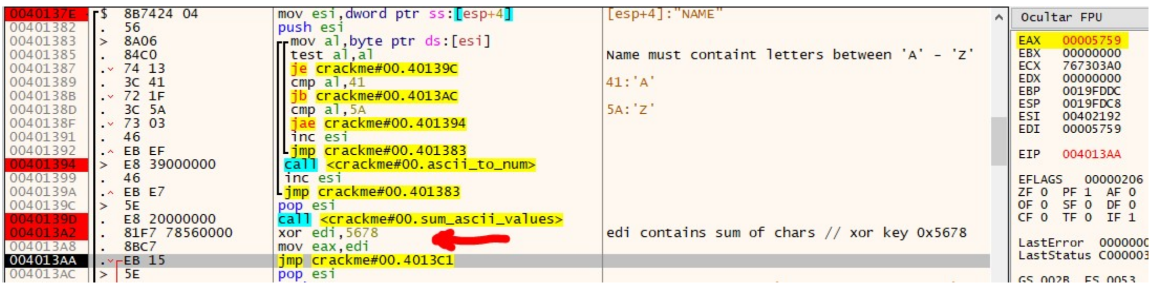
The form field asks for **NAME** and **SERIAL**. We will first focus on the NAME where we will put the text "NAME" and see what we get. In the SERIAL for the moment we will put "12345" although the value does not matter.



Name: NAME
Serial:

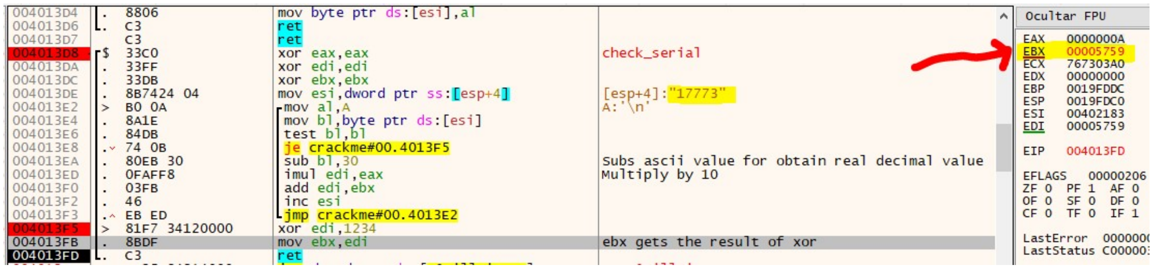
```
00401000   6A 00                push 0                                          LPCTSTR lpModuleName = NULL
                                call <JMP.&GetModuleHandleA>                    GetModuleHandleA
                                dword ptr ds:[4020CA],eax                       004020CA:&"MZP"
                                push 0                                          LPCTSTR lpWindowName = NULL
                                push <crackme#00.no_need_to_disasm_the_code>    LPCTSTR lpClassName = "No need to disasm the code!"
                                call <JMP.&FindWindowA>                         FindWindowA
                                eax,eax
                                crackme#00.40101D
                                [402064],4003
                                [402068],<crackme#00.WndProc>
                                [40206C],0
                                [402070],0
00401045                        ds:[4020CA]                                     004020CA:&"MZP"
0040104A                        [402074],eax                                    00402074:&"MZP"
0040104F                                                                        LPCTSTR lpIconName = 64
00401051                                                                        HINSTANCE hInstance
00401052                        nA>                                             LoadIconA
00401057                        [402078],eax
0040105C                                                                        LPCTSTR lpCursorName = IDC_ARROW
00401061                                                                        HINSTANCE hInstance = NULL
00401063                        rsorA>                                          LoadCursorA
00401068                        [40207C],eax
0040106D   C705 80204000 0500   mov dword ptr ds:[402080],5
00401077   C705 84204000 1021   mov dword ptr ds:[402084],crackme#00.402110     00402084:&"MENU", 402110:"MENU"
00401081   C705 88204000 F420   mov dword ptr ds:[402088],<crackme#00.no_need_  00402088:&"No need to disasm the code!", 4020F4:"No
0040108B   68 64204000          push crackme#00.402064                          WNDCLASS* lpWndClass = 402064
00401090   E8 F3030000          call <JMP.&RegisterClassA>                      RegisterClassA
```

We arrive at the point where we are interested, where after entering the name, we see that the **XOR** performs and we obtain the resulting value that is **0x5759** that is in the **EAX** register.



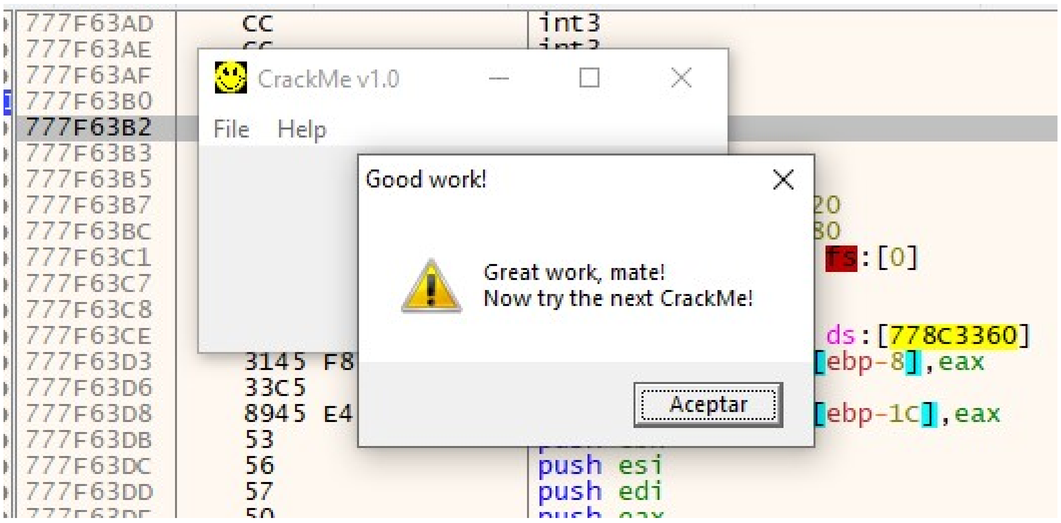
```
0040137E  ┌$ 8B7424 04    mov esi,dword ptr ss:[esp+4]      [esp+4]:"NAME"
00401382  .  56           push esi
00401383  >  8A06         ┌mov al,byte ptr ds:[esi]
00401385  .  84C0         │test al,al                         Name must containt letters between 'A' - 'Z'
00401387  .v 74 13        │je crackme#00.40139C
00401389  .  3C 41        │cmp al,41                          41:'A'
0040138B  .v 72 1F        │jb crackme#00.4013AC
0040138D  .  3C 5A        │cmp al,5A                          5A:'Z'
0040138F  .v 73 03        │jae crackme#00.401394
00401391  .  46           │inc esi
00401392  .^ EB EF        └jmp crackme#00.401383
00401394  >  E8 39000000   call <crackme#00.ascii_to_num>
00401399  .  46            inc esi
0040139A  .^ EB E7        └jmp crackme#00.401383
0040139C  >  5E           pop esi
0040139D  .  E8 20000000  call <crackme#00.sum_ascii_values>
004013A2  .  81F7 78560000 xor edi,5678                     edi contains sum of chars // xor key 0x5678
004013A8  .  8BC7         mov eax,edi
004013AA  .v┌EB 15        jmp crackme#00.4013C1
004013AC  > │5E           pop esi

Ocultar FPU
EAX  00005759
EBX  00000000
ECX  767303A0
EDX  00000000
EBP  0019FDDC
ESP  0019FDC8
ESI  00402192
EDI  00005759

EIP  004013AA

EFLAGS   00000206
ZF 0  PF 1  AF 0
OF 0  SF 0  DF 0
CF 0  TF 0  IF 1

LastError  0000000
LastStatus C000003
```

As we know that compares the results of the two XOR operations, and in the case of the number it makes an XOR with **0x1234** (and it is a symmetric function), if we perform an XOR with the value obtained from the NAME chain that is **0x5759** we get the value **0x456D** that passed to the decimal value corresponds to the number **17773**, which is the SERIAL number that would correspond to the name "NAME".



```
004013D4  │.  8806         mov byte ptr ds:[esi],al
004013D6  └.  C3           ret
004013D7      C3           ret
004013D8  ┌$  33C0         xor eax,eax                      check_serial
004013DA  │.  33FF         xor edi,edi
004013DC  │.  33DB         xor ebx,ebx
004013DE  │.  8B7424 04    mov esi,dword ptr ss:[esp+4]     [esp+4]:"17773"
004013E2  │>  B0 0A        ┌mov al,A                        A:'\n'
004013E4  │.  8A1E         │mov bl,byte ptr ds:[esi]
004013E6  │.  84DB         │test bl,bl
004013E8  │.v 74 0B        │je crackme#00.4013F5
004013EA  │.  80EB 30      │sub bl,30                        Subs ascii value for obtain real decimal value
004013ED  │.  0FAFF8       │imul edi,eax                     Multiply by 10
004013F0  │.  03FB         │add edi,ebx
004013F2  │.  46           │inc esi
004013F3  │.^ EB ED        └jmp crackme#00.4013E2
004013F5  │>  81F7 34120000 xor edi,1234
004013FB  │.  8BDF         mov ebx,edi                      ebx gets the result of xor
004013FD  └.  C3           ret

Ocultar FPU
EAX  0000000A
EBX  00005759
ECX  767303A0
EDX  00000000
EBP  0019FDDC
ESP  0019FDC0
ESI  00402183
EDI  00005759

EIP  004013FD

EFLAGS   00000206
ZF 0  PF 1  AF 0
OF 0  SF 0  DF 0
CF 0  TF 0  IF 1

LastError  0000000
LastStatus C000003
```

And here is the result.



CrackMe v1.0

File Help

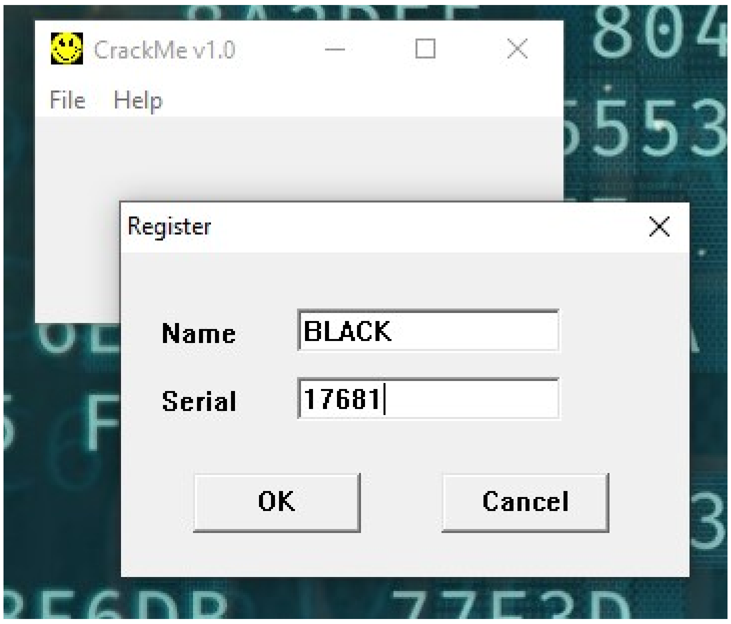
Good work!

Great work, mate!
Now try the next CrackMe!

Aceptar

Thus, it is about obtaining the value of the XOR from the name chain, and to this value apply an XOR with the value 0x1234, which will give us the serial number that we must enter.

For example, for the name **"BLACK"** we obtain the value **0x5725** to which if we apply the XOR with **0x1234** it gives us the value **0x4511** that corresponds to the decimal number **17681** which is the serial number.



Name BLACK

Serial 17681

That's all folks!