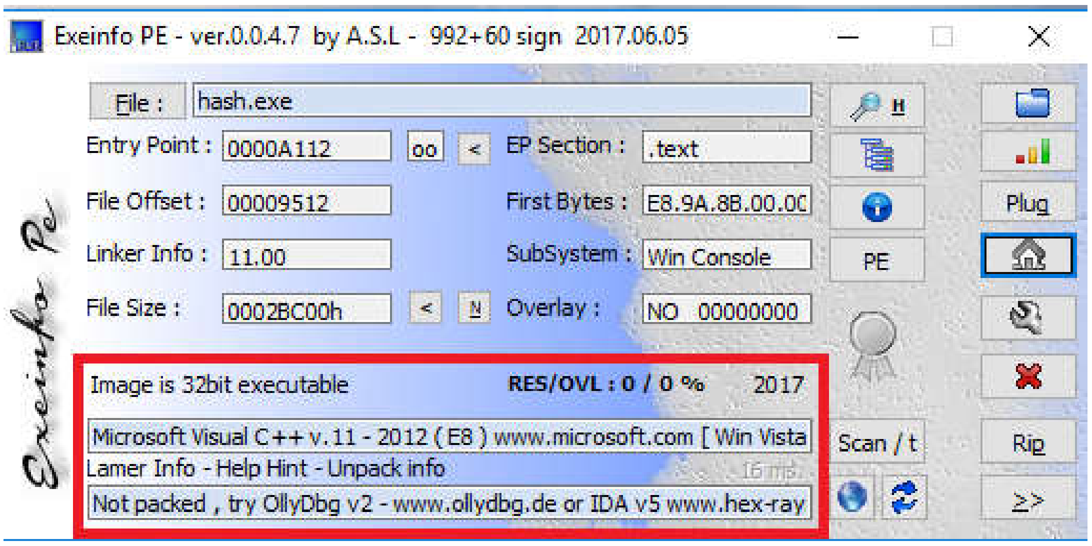
Lets starting looking the crackme maked by Hasherezade some days ago, a easy crackme but interesting, ;)

The crackme checks some hashes that i dont know the string or values that make this hash, i can make a brute force to make some collision because the algo is very easy and only is a hash of 32bits but i used a more quick way patching the crackme (the window problem for example).

Stage 1:

The crackme comes as a binary executable of 179.200 bytes (175k). Looking in ExeInfo can look that not is protected for any packer:


Exeinfo PE - ver.0.0.4.7 by A.S.L - 992+60 sign 2017.06.05

File : hash.exe
Entry Point : 0000A112
EP Section : .text
File Offset : 00009512
First Bytes : E8.9A.8B.00.0C
Linker Info : 11.00
SubSystem : Win Console
File Size : 0002BC00h
Overlay : NO 00000000

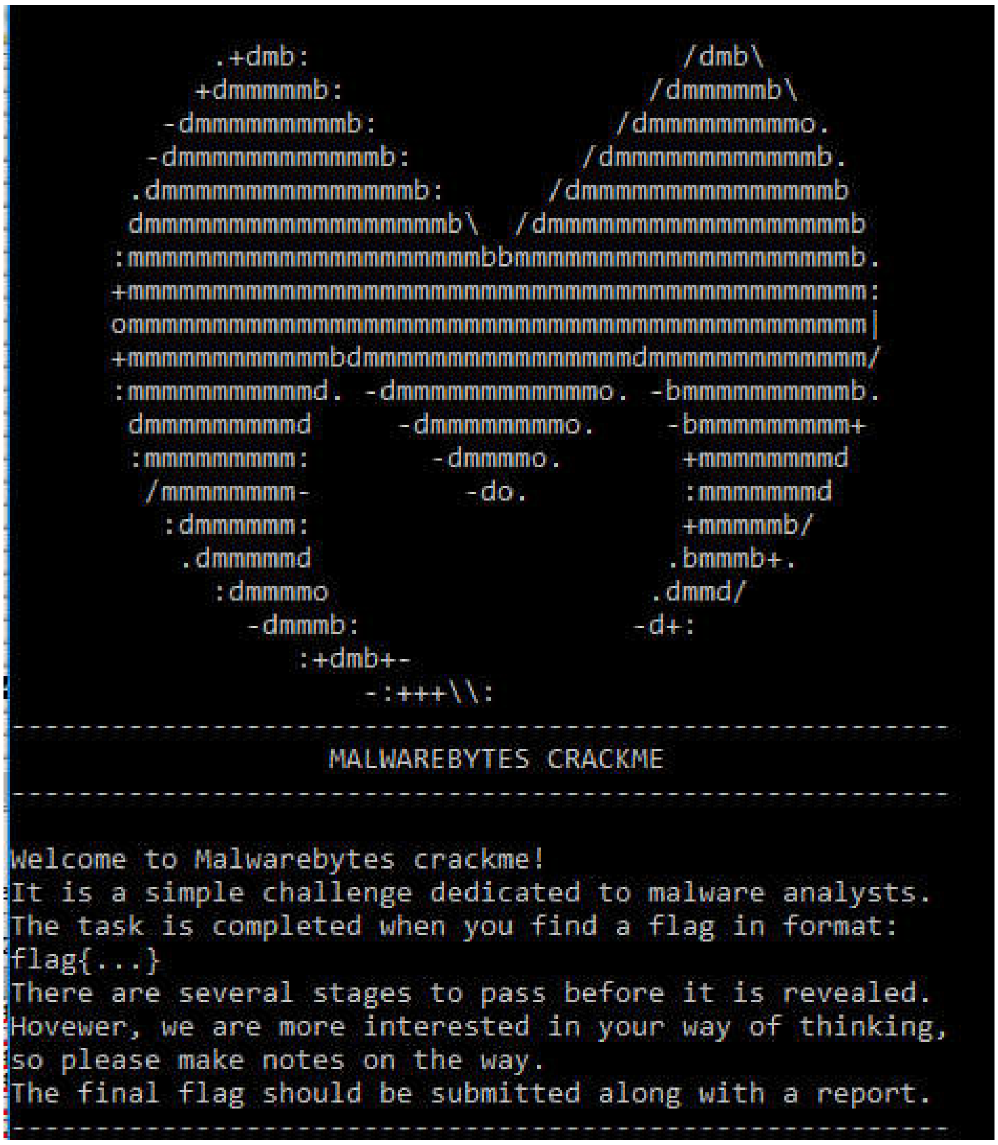
Image is 32bit executable RES/OVL : 0 / 0 % 2017
Microsoft Visual C++ v.11 - 2012 (E8) www.microsoft.com [Win Vista
Lamer Info - Help Hint - Unpack info 16 ms.
Not packed , try OllyDbg v2 - www.ollydbg.de or IDA v5 www.hex-ray

Its said "Visual Studio 2012", and its have the usual way to start with executables compiled with Visual Studio.

The goal of the crackme is get the good boy message saying something as "flag{...", so, lets try get it!

Open it in IDA Pro and in the debugger (x32dbg in my case) you can see without any problem the code.

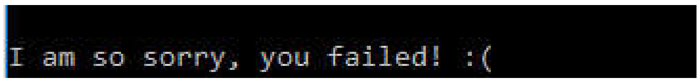
The first action of the crackme is show in the console the MalwareBytes logo and info about the goal of the crackme:



```
            .+dmb:                            /dmb\
           +dmmmmmb:                        /dmmmmmb\
         -dmmmmmmmmmb:                    /dmmmmmmmmmo.
        -dmmmmmmmmmmmmmb:               /dmmmmmmmmmmmmb.
       .dmmmmmmmmmmmmmmmmb:           /dmmmmmmmmmmmmmmmmb
       dmmmmmmmmmmmmmmmmmmmb\     /dmmmmmmmmmmmmmmmmmmmb
      :mmmmmmmmmmmmmmmmmmmmmmbbmmmmmmmmmmmmmmmmmmmmmmmb.
      +mmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmm:
      ommmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmm|
      +mmmmmmmmmmmmmbdmmmmmmmmmmmmmmmmmdmmmmmmmmmmmmmm/
      :mmmmmmmmmmmd. -dmmmmmmmmmmmmo. -bmmmmmmmmmmb.
       dmmmmmmmmmmd     -dmmmmmmmmmo.     -bmmmmmmmmmm+
       :mmmmmmmmmm:        -dmmmmo.        +mmmmmmmmmd
        /mmmmmmmmm-          -do.          :mmmmmmmmd
         :dmmmmmm:                         +mmmmmb/
          .dmmmmmd                        .bmmmb+.
            :dmmmmo                     .dmmd/
              -dmmmb:                  -d+:
                 :+dmb+-
                     -:+++\\:
---------------------------------------------------------
                   MALWAREBYTES CRACKME
---------------------------------------------------------

Welcome to Malwarebytes crackme!
It is a simple challenge dedicated to malware analysts.
The task is completed when you find a flag in format:
flag{...}
There are several stages to pass before it is revealed.
Hovewer, we are more interested in your way of thinking,
so please make notes on the way.
The final flag should be submitted along with a report.
---------------------------------------------------------
```

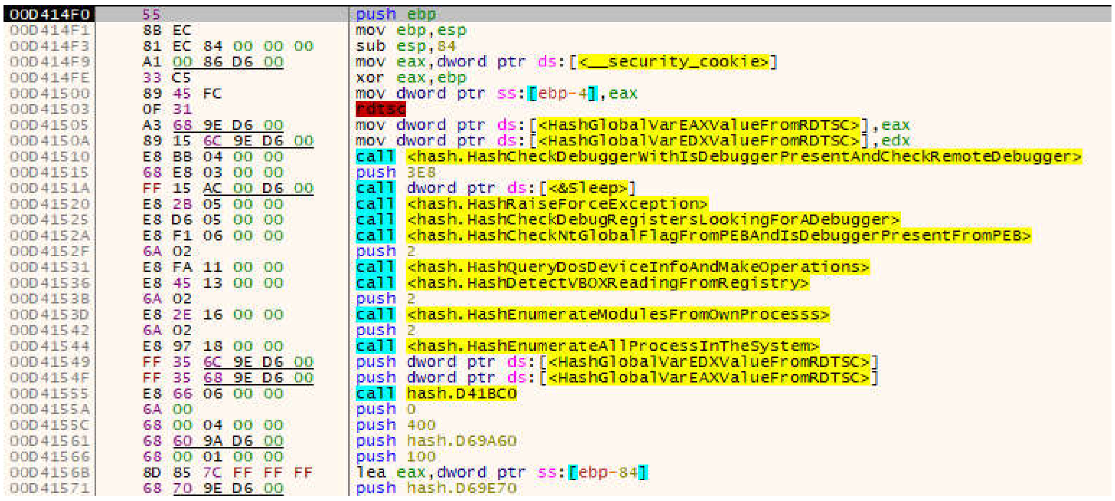
If run the crackme without any debugger you get a bad boy message:



```
I am so sorry, you failed! :(
```

So, something wrong happens here. Usually malwares try to avoid be detected, so, what happens here in a normal machine (not a virtual machine) without any debugger or tool running? Why get this bad boy message?

So, lets start the crackme in the debugger and after the intro message you arrive to the main function all of this mess:

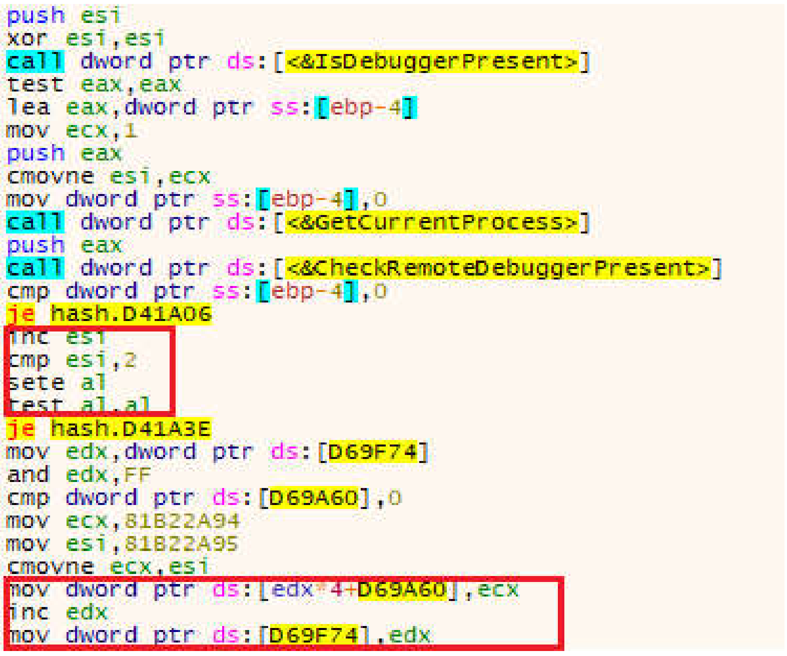


```
00D414F0    55                   push ebp
00D414F1    8B EC                mov ebp,esp
00D414F3    81 EC 84 00 00 00    sub esp,84
00D414F9    A1 00 86 D6 00       mov eax,dword ptr ds:[<__security_cookie>]
00D414FE    33 C5                xor eax,ebp
00D41500    89 45 FC             mov dword ptr ss:[ebp-4],eax
00D41503    0F 31                rdtsc
00D41505    A3 68 9E D6 00       mov dword ptr ds:[<HashGlobalVarEAXValueFromRDTSC>],eax
00D4150A    89 15 6C 9E D6 00    mov dword ptr ds:[<HashGlobalVarEDXValueFromRDTSC>],edx
00D41510    E8 BB 04 00 00       call <hash.HashCheckDebuggerWithIsDebuggerPresentAndCheckRemoteDebugger>
00D41515    68 E8 03 00 00       push 3E8
00D4151A    FF 15 AC 00 D6 00    call dword ptr ds:[<&Sleep>]
00D41520    E8 2B 05 00 00       call <hash.HashRaiseForceException>
00D41525    E8 D6 05 00 00       call <hash.HashCheckDebugRegistersLookingForADebugger>
00D4152A    E8 F1 06 00 00       call <hash.HashCheckNtGlobalFlagFromPEBAndIsDebuggerPresentFromPEB>
00D4152F    6A 02                push 2
00D41531    E8 FA 11 00 00       call <hash.HashQueryDosDeviceInfoAndMakeOperations>
00D41536    E8 45 13 00 00       call <hash.HashDetectVBOXReadingFromRegistry>
00D4153B    6A 02                push 2
00D4153D    E8 2E 16 00 00       call <hash.HashEnumerateModulesFromOwnProcesss>
00D41542    6A 02                push 2
00D41544    E8 97 18 00 00       call <hash.HashEnumerateAllProcessInTheSystem>
00D41549    FF 35 6C 9E D6 00    push dword ptr ds:[<HashGlobalVarEDXValueFromRDTSC>]
00D4154F    FF 35 68 9E D6 00    push dword ptr ds:[<HashGlobalVarEAXValueFromRDTSC>]
00D41555    E8 66 06 00 00       call hash.D41BC0
00D4155A    6A 00                push 0
00D4155C    68 00 04 00 00       push 400
00D41561    68 60 9A D6 00       push hash.D69A60
00D41566    68 00 01 00 00       push 100
00D4156B    8D 85 7C FF FF FF    lea eax,dword ptr ss:[ebp-84]
00D41571    68 70 9E D6 00       push hash.D69E70
```

Hasherezade try make a trick here, instead try that we can detect her crackme, she checks that if its debugged or not and SHE WANTS that its been debugged. It is the main reason the crackme show the bad boy message in a normal execution.

The first trick is get use the "rdtsc" opcode instruction and save in two global vars the values stored in EAX and EDX registers. (She will check later).

After this she uses the Windows API "IsDebuggerPresent" and "CheckRemoteDebuggerPresent". Both functions returns boolean values, and the crackme checks it and increment a counter for each one, finally if the counter have at least number 1 the crackme will write in the offset the first value that later will use to make a hash.

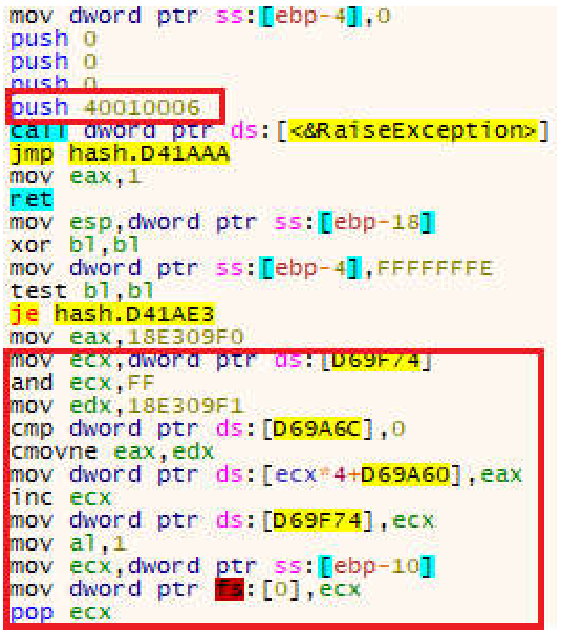


```
push esi
xor esi,esi
call dword ptr ds:[<&IsDebuggerPresent>]
test eax,eax
lea eax,dword ptr ss:[ebp-4]
mov ecx,1
push eax
cmovne esi,ecx
mov dword ptr ss:[ebp-4],0
call dword ptr ds:[<&GetCurrentProcess>]
push eax
call dword ptr ds:[<&CheckRemoteDebuggerPresent>]
cmp dword ptr ss:[ebp-4],0
je hash.D41A06
inc esi
cmp esi,2
sete al
test al,al
je hash.D41A3E
mov edx,dword ptr ds:[D69F74]
and edx,FF
cmp dword ptr ds:[D69A60],0
mov ecx,81B22A94
mov esi,81B22A95
cmovne ecx,esi
mov dword ptr ds:[edx*4+D69A60],ecx
inc edx
mov dword ptr ds:[D69F74],edx
```

The most important thing here is the global var (D69F74) that keep the counter to put the new value for the future hash.

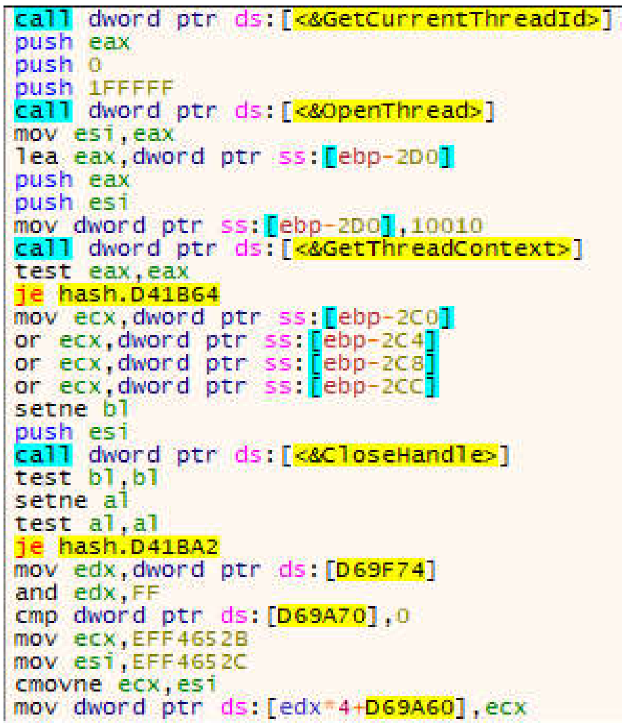
The second check is make a forced exception, but dont have any problem to avoid this exception or put a breakpoint in the SEH installed before of the RaiseException API call.

Again put another value in one position.



```
mov dword ptr ss:[ebp-4],0
push 0
push 0
push 0
push 40010006
call dword ptr ds:[<&RaiseException>]
jmp hash.D41AAA
mov eax,1
ret
mov esp,dword ptr ss:[ebp-18]
xor bl,bl
mov dword ptr ss:[ebp-4],FFFFFFFE
test bl,bl
je hash.D41AE3
mov eax,18E309F0
mov ecx,dword ptr ds:[D69F74]
and ecx,FF
mov edx,18E309F1
cmp dword ptr ds:[D69A6C],0
cmovne eax,edx
mov dword ptr ds:[ecx*4+D69A60],eax
inc ecx
mov dword ptr ds:[D69F74],ecx
mov al,1
mov ecx,dword ptr ss:[ebp-10]
mov dword ptr fs:[0],ecx
pop ecx
```

The third check is detect is some "Dx" register have some value or not. For this the crackme open its own thread and look in the context and check the 4 registers. If some one of the registers have some value will put the third value in the array, if not will avoid write nothing.

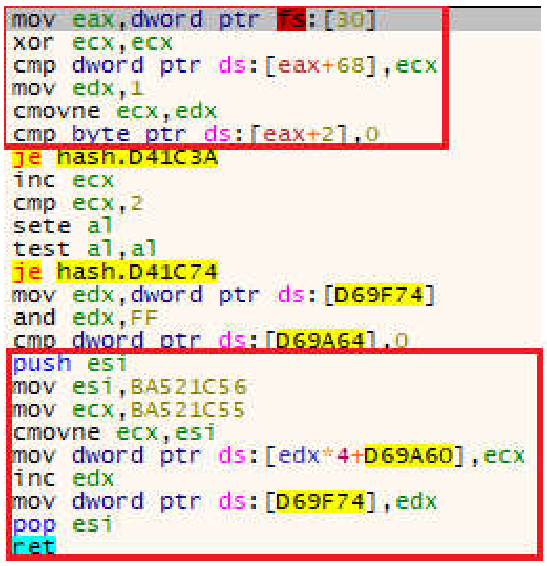


```
call dword ptr ds:[<&GetCurrentThreadId>]
push eax
push 0
push 1FFFFF
call dword ptr ds:[<&OpenThread>]
mov esi,eax
lea eax,dword ptr ss:[ebp-2D0]
push eax
push esi
mov dword ptr ss:[ebp-2D0],10010
call dword ptr ds:[<&GetThreadContext>]
test eax,eax
je hash.D41B64
mov ecx,dword ptr ss:[ebp-2C0]
or ecx,dword ptr ss:[ebp-2C4]
or ecx,dword ptr ss:[ebp-2C8]
or ecx,dword ptr ss:[ebp-2CC]
setne bl
push esi
call dword ptr ds:[<&CloseHandle>]
test bl,bl
setne al
test al,al
je hash.D41BA2
mov edx,dword ptr ds:[D69F74]
and edx,FF
cmp dword ptr ds:[D69A70],0
mov ecx,EFF4652B
mov esi,EFF4652C
cmovne ecx,esi
mov dword ptr ds:[edx*4+D69A60],ecx
```

The fourth check look into the PEB (Process Environment Block) in 2 fields:

IsDebuggerPresent (PEB + 0x2) (is the same that the Microsoft API makes) and check the NtGlobalFlag value (PEB+0x68), this value is filled from the kernel and if the process is debugged have the 0x70 value, if not have 0.

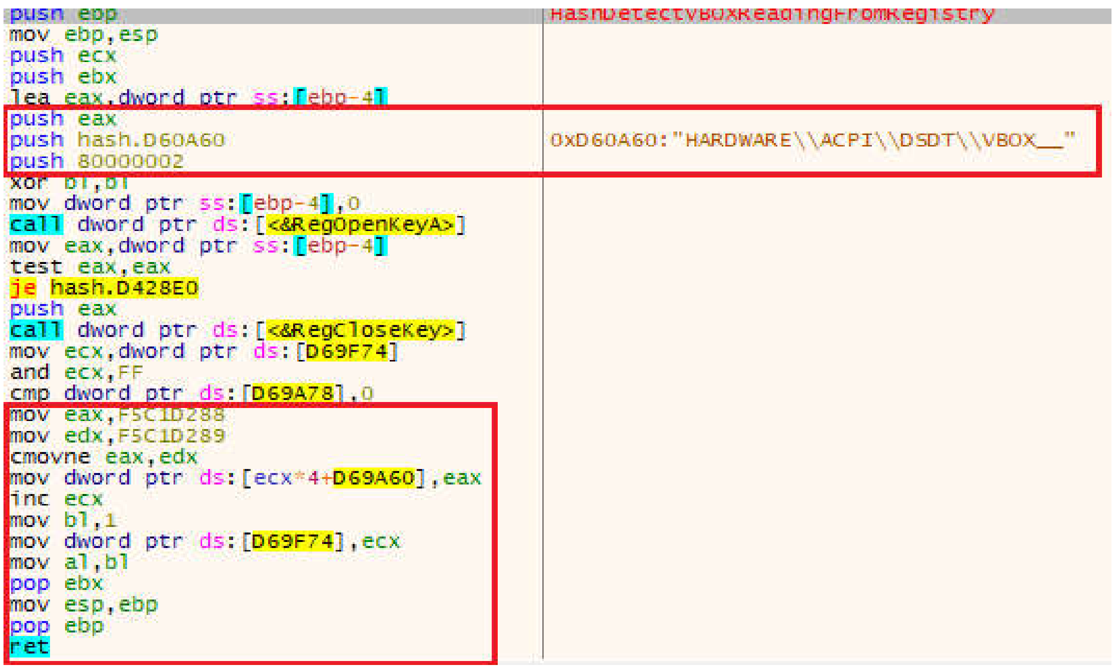
Again if both things are ok, the crackme write the next value in the array.



```
mov eax,dword ptr fs:[30]
xor ecx,ecx
cmp dword ptr ds:[eax+68],ecx
mov edx,1
cmovne ecx,edx
cmp byte ptr ds:[eax+2],0
je hash.D41C3A
inc ecx
cmp ecx,2
sete al
test al,al
je hash.D41C74
mov edx,dword ptr ds:[D69F74]
and edx,FF
cmp dword ptr ds:[D69A64],0
push esi
mov esi,BA521C56
mov ecx,BA521C55
cmovne ecx,esi
mov dword ptr ds:[edx*4+D69A60],ecx
inc edx
mov dword ptr ds:[D69F74],edx
pop esi
ret
```

The next check uses the API "QueryDosDevice" to get all devices in the machine and use a counter that need be 2 after enumerate all devices in the machine. In a virtual machine (VirtualBox one) its run ok, but in a normal machine not, so i need patch here the check to put the next value into the array.

Next check read from the registry the usual VirtualBox entry, in a virtual box machine not hardened its will be ok but in a normal machine or in a vm virtualbox hardened machine its will fail. In my case i patch to get the value in the array again.



```
push ebp                                  HasDetectVBoxReadingFromRegistry
mov ebp,esp
push ecx
push ebx
lea eax,dword ptr ss:[ebp-4]
push eax
push hash.D60A60                          0xD60A60:"HARDWARE\\ACPI\\DSDT\\VBOX__"
push 80000002
xor bl,bl
mov dword ptr ss:[ebp-4],0
call dword ptr ds:[<&RegOpenKeyA>]
mov eax,dword ptr ss:[ebp-4]
test eax,eax
je hash.D428E0
push eax
call dword ptr ds:[<&RegCloseKey>]
mov ecx,dword ptr ds:[D69F74]
and ecx,FF
cmp dword ptr ds:[D69A78],0
mov eax,F5C1D288
mov edx,F5C1D289
cmovne eax,edx
mov dword ptr ds:[ecx*4+D69A60],eax
inc ecx
mov bl,1
mov dword ptr ds:[D69F74],ecx
mov al,bl
pop ebx
mov esp,ebp
pop ebp
ret
```

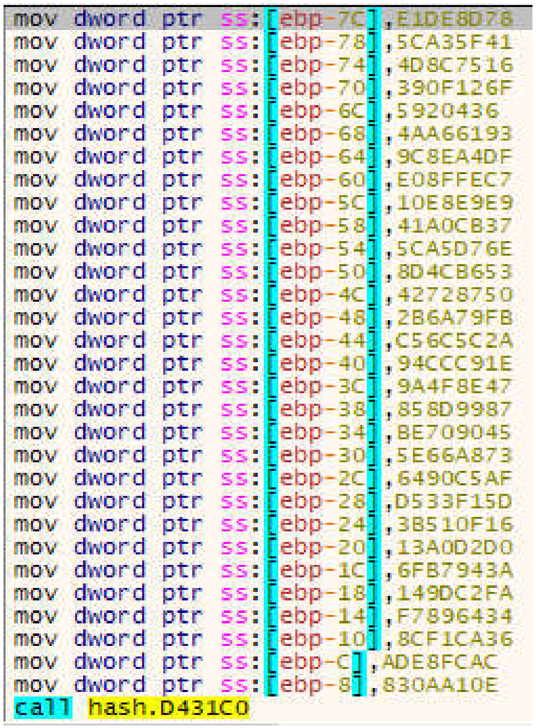
The next check enumerate all modules of the own process and increment a counter for later check, in my case its dont detect any module so i need patch again to force the write of the value.

The next one is the same but enumerating all process in the system, i guess hasherezade try detect some tools or debugging programs, etc, but in my case i need patch again the check to force write the value.

The last check is use again the opcode "rdtsc" and check with the previous values stored in the global vars, in a normal case a malware can use this trick to detect the slowly execution from the ticks because a debugger but here hasherezade checks in the opposite way. So lets patch again and put the last value.

The crackme puts in the stack some hardcoded values that are a block crypted with the URL where its will download the second binary.

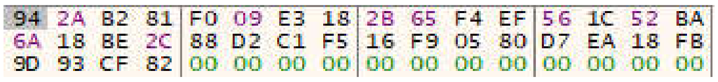
Hasherezade puts here the values to avoid that somebody can look in the data section.



```
mov dword ptr ss:[ebp-7C],E1DE8D78
mov dword ptr ss:[ebp-78],5CA35F41
mov dword ptr ss:[ebp-74],4D8C7516
mov dword ptr ss:[ebp-70],390F126F
mov dword ptr ss:[ebp-6C],5920436
mov dword ptr ss:[ebp-68],4AA66193
mov dword ptr ss:[ebp-64],9C8EA4DF
mov dword ptr ss:[ebp-60],E08FFEC7
mov dword ptr ss:[ebp-5C],10E8E9E9
mov dword ptr ss:[ebp-58],41A0CB37
mov dword ptr ss:[ebp-54],5CA5D76E
mov dword ptr ss:[ebp-50],8D4CB653
mov dword ptr ss:[ebp-4C],42728750
mov dword ptr ss:[ebp-48],2B6A79FB
mov dword ptr ss:[ebp-44],C56C5C2A
mov dword ptr ss:[ebp-40],94CCC91E
mov dword ptr ss:[ebp-3C],9A4F8E47
mov dword ptr ss:[ebp-38],858D9987
mov dword ptr ss:[ebp-34],BE709045
mov dword ptr ss:[ebp-30],5E66A873
mov dword ptr ss:[ebp-2C],6490C5AF
mov dword ptr ss:[ebp-28],D533F15D
mov dword ptr ss:[ebp-24],3B510F16
mov dword ptr ss:[ebp-20],13A0D2D0
mov dword ptr ss:[ebp-1C],6FB7943A
mov dword ptr ss:[ebp-18],149DC2FA
mov dword ptr ss:[ebp-14],F7896434
mov dword ptr ss:[ebp-10],8CF1CA36
mov dword ptr ss:[ebp-C],ADE8FCAC
mov dword ptr ss:[ebp-8],830AA10E
call hash.D431C0
```

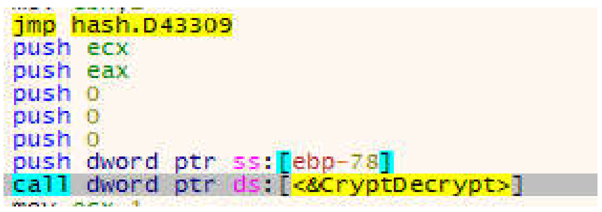
The next step is get a context for the CryptoAPI to use AES to decrypt this ciphered block.

After get the context, make a hash using the function "CryptCreateHash" and "CryptHashData". The value used for get the hash is the values that are inserted in the array with the previous checks.

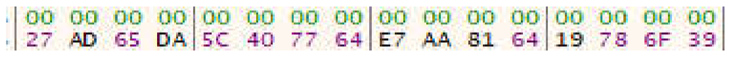


```
94 2A B2 81 F0 09 E3 18 2B 65 F4 EF 56 1C 52 BA
6A 18 BE 2C 88 D2 C1 F5 16 F9 05 80 D7 EA 18 FB
9D 93 CF 82 00 00 00 00 00 00 00 00 00 00 00 00
```

Finally derive a key from this hash to decrypt the block. The derive is maked with the API "CryptDeriveKey". After get the key its decrypt the block with the API "CryptDecrypt" (here we can see code used by hasherezade to crypt the block using "CryptEncrypt").



```
jmp hash.D43309
push ecx
push eax
push 0
push 0
push 0
push dword ptr ss:[ebp-78]
call dword ptr ds:[<&CryptDecrypt>]
```



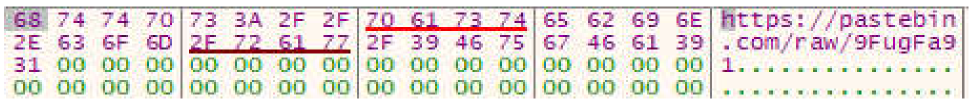
```
00 00 00 00|00 00 00 00|00 00 00 00|00 00 00 00
27 AD 65 DA|5C 40 77 64|E7 AA 81 64|19 78 6F 39
```

The key using AES in CBC mode is "27 AD 65 DA 5C 40 77 64 E7 AA 81 64 19 78 6F 39", using a IV of 0.

After decrypt have in memory the URL to download the second binary, the crackme releases the handles to the key, hash and context BUT the key remains in the memory because this buffer never is nullified or erased (so, you can get the key in a dump always that it works).

The next step is make a hash of the decrypted block and compare with a hardcoded value "3B47B2E6", if its the same the crackme follow and if not its show the bad boy message with "I am so sorry, you failed, :(".

The URL decoded is:



```
68 74 74 70|73 3A 2F 2F|70 61 73 74|65 62 69 6E|https://pastebin
2E 63 6F 6D|2F 72 61 77|2F 39 46 75|67 46 61 39|.com/raw/9FugFa9
31 00 00 00|00 00 00 00|00 00 00 00|00 00 00 00|1...............
00 00 00 00|00 00 00 00|00 00 00 00|00 00 00 00|................
```

The crackme checks if have internet connection, and if its have download the new file in parts (0x400 bytes each one). If not have internet the crackme reports that and finish.

After download all file the crackme decoded it from Base64, and put a hint in the console saying the address that keep the uncompress size of the new file. If you look in this address you can get the final size of the second file (0xE400 bytes).

After it the crackme gets the function "RtlDecompressBuffer" from the module "ntdll.dll" using "GetProcAddress", and calling it in a dinamic way. After decompress access to the clipboard, checks his content and use it to decrypt in a XOR loop for get the final file.

This part of the clipboard is tricky, but if you look the buffer after decompression you can see a lot of times the string "malwarebytes" and if you go to the beginning of the buffer + size of the file decompressed (0xE400) you can get the final position and in a executable, usually, the final bytes are null, and in this case have the same string again and again. A value xored with his own value gives null, so, the key that need put in the clipboard is "malwarebytes".

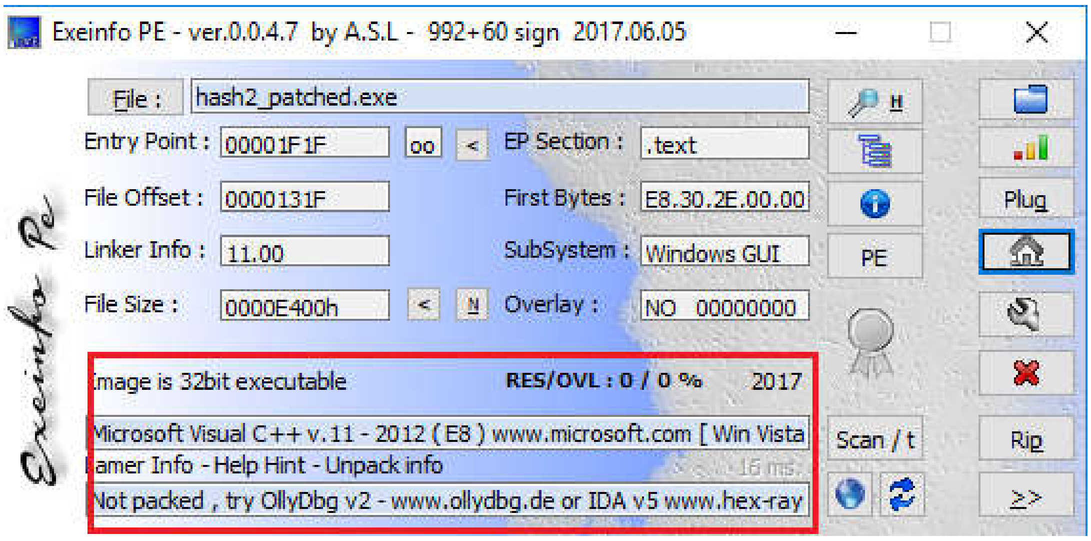
After decrypt it, checks his header to check if is "MZ", the crackme will launch in a suspended state the official binary "rundll32.exe" with a dummy argument "secret.dll, #1" (trying fooling you as a loading a dll and calling the export with the number 1 ordinal", but the crackme wants make a process hollowing with the new binary, and after it resume the main thread and launch the second binary.

In my case i dont need the crackme makes the process hollowing because i have the binary in the memory decompressed and decrypted so only need dump it from the memory to thr disk.

In this moment i can close the debugger with the first binary and open the second.

State 2:

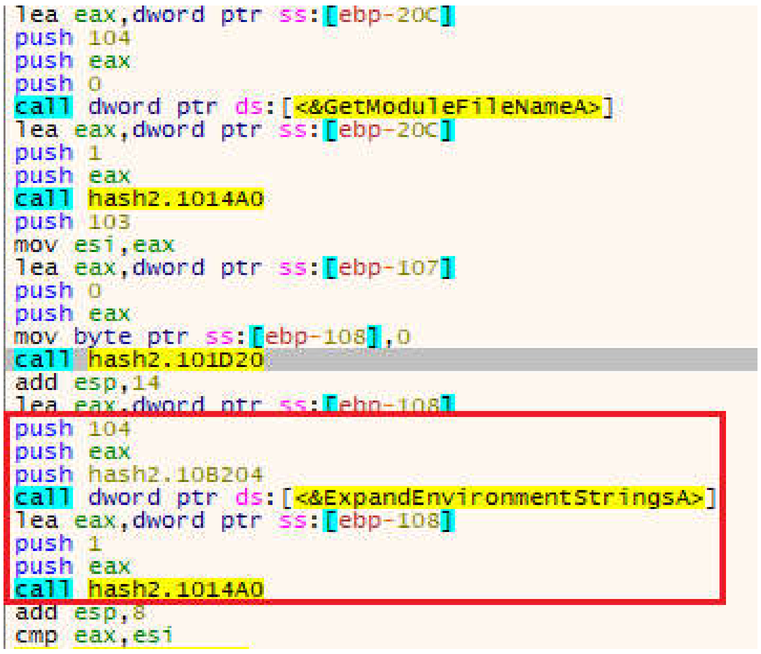
The second binary is a executable with 58.368 bytes (57k) and not is compressed or packed.



```
File : hash2_patched.exe
Entry Point : 00001F1F    EP Section : .text
File Offset : 0000131F    First Bytes : E8.30.2E.00.00
Linker Info : 11.00       SubSystem : Windows GUI
File Size : 0000E400h     Overlay : NO 00000000
Image is 32bit executable    RES/OVL : 0 / 0 %    2017
Microsoft Visual C++ v.11 - 2012 ( E8 ) www.microsoft.com [ Win Vista
Lamer Info - Help Hint - Unpack info    16 ms.
Not packed , try OllyDbg v2 - www.ollydbg.de or IDA v5 www.hex-ray
```

The new binary starting getting her own path and making a hash from it, later make a hash with the system path to %systemroot% field (using ExpandEnvironmentStringsA api) and making the full path to "rundll32.dll", for example "C:\Windows\System32\rundll32.exe".

Compare both hashes and if they are ok, the crackme follow his execution, in another case its show a message box with the bad boy message and finish his execution.



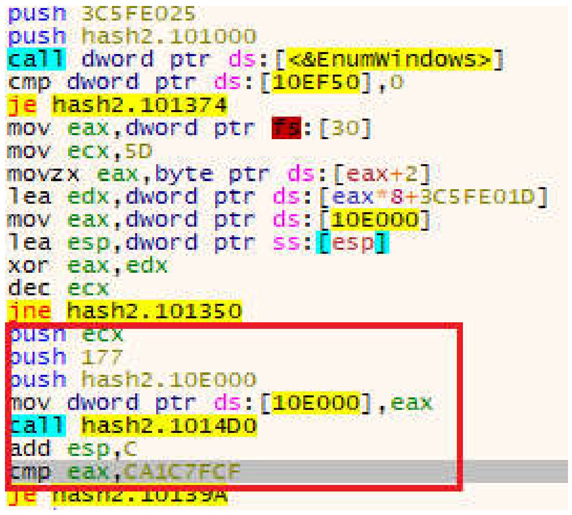
```
lea eax,dword ptr ss:[ebp-20C]
push 104
push eax
push 0
call dword ptr ds:[<&GetModuleFileNameA>]
lea eax,dword ptr ss:[ebp-20C]
push 1
push eax
call hash2.1014A0
push 103
mov esi,eax
lea eax,dword ptr ss:[ebp-107]
push 0
push eax
mov byte ptr ss:[ebp-108],0
call hash2.101D20
add esp,14
lea eax,dword ptr ss:[ebp-108]
push 104
push eax
push hash2.10B204
call dword ptr ds:[<&ExpandEnvironmentStringsA>]
lea eax,dword ptr ss:[ebp-108]
push 1
push eax
call hash2.1014A0
add esp,8
cmp eax,esi
```

In my case as i used another path, i patch this check to avoid the bad boy message. After this the crackme enumerate all windows and try found one that have the hash "3C5FE025" (i dont know what is this window), the callback function used in the api "EnumWindows" get the class of the window enumerated and make a hash, later compare with the wished value and if its ok, its gets the process id from the window, open the process, and get a handle to it.

After the enumerate all windows, the crackme checks if have a handle for the process, in case that dont have any handle this variable is 0 and the crackme shows the bad boy message and finish.

In my case i trick the crackme to its thinks that found the window that she wants, and this way open a process that i control (for example my own debugger), after this i can pass the check of the var of the handle.

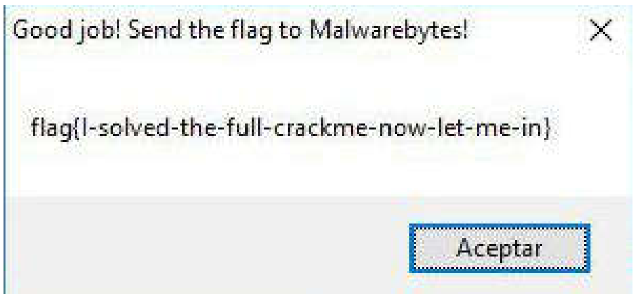
The next step of the crackme is check is being debugged looking in the PEB+0x2. If this value is 1 he will make a compute with a xor loop and later will check a block of 0x177 bytes making his hash and compare with the hardcoded value "CA1C7FCF". If the PEB value have a 0 value the first 4 bytes of the block will have a value, but the crackme waits that this value need be the value that is inserted if the PEB value is 1.



```
push 3C5FE025
push hash2.101000
call dword ptr ds:[<&EnumWindows>]
cmp dword ptr ds:[10EF50],0
je hash2.101374
mov eax,dword ptr fs:[30]
mov ecx,5D
movzx eax,byte ptr ds:[eax+2]
lea edx,dword ptr ds:[eax*8+3C5FE01D]
mov eax,dword ptr ds:[10E000]
lea esp,dword ptr ss:[esp]
xor eax,edx
dec ecx
jne hash2.101350
push ecx
push 177
push hash2.10E000
mov dword ptr ds:[10E000],eax
call hash2.1014D0
add esp,C
cmp eax,CA1C7FCF
je hash2.10139A
```

Finally the crackme choose using "GetTickCount" and the handle value a random choice to make a new thread.

Its have three options: CreateThead, CreateRemoteThread (in the open process that have the handle) or "ZwCreateThreadEx". Anyways if the thread is created its run showing the good boy message.



Good job! Send the flag to Malwarebytes!

flag{I-solved-the-full-crackme-now-let-me-in}

And crackme finish, :)

A easy crackme and funny. Good job Hasherezade.

Valthek