Cracking a simple CD check

Well this is a little tiny paper about cracking a cd check nothing fancy just a old cd check. Since this is fairly new for me to I will try to explain this the best I can. I know there are probably better methods and things and there is also a official solution on the website of the challenge but I am just sharing my knowledge since it was a bit different from the original solution.

So here we go. Tools needed:

- W32Dasm
- HIEW
- win32.hlp or http://www.msdn.microsoft.com

The challenge: http://biw.rult.at/crackmes/findcd.zip
Official BIW solution: http://biw.rult.at/tuts/cdpatching.htm

The yellow block in the code are the code blocks where the text is talking about.
So let's fire up the application. You probably will see this:

Now let's fire up W32Dasm and hiew with the application opened in both of them.

Then push the "code start" button in W32Dasm so you are put at the beginning of the code. It will look like this:

++++++++++++++++++++ ASSEMBLY CODE LISTING ++++++++++++++++++++
*//********** Start of Code in Object .text **********
Program Entry Point = 00401000 (C:\Documents and Settings\Francisco\Bureaublad\findcd.exe File Offset:00001600)*

*//********** Program Entry Point ********
:00401000 6A00 push 00000000*

** Reference To: KERNEL32.GetModuleHandleA, Ord:0111h*
:00401002 E80B010000 Call 00401112
:00401007 A39E314000 mov dword ptr [0040319E], eax

So now that we know where to start let's try and understand the code that we need. Since I am no asm expert I will only comment the code that I understand.

```
* Possible StringData Ref from Data Obj ->"A:\"
                        |
:0040100C 6836304000              push 00403036
```

Well wth is happening here? Is the first thing I thought. So go to hiew press enter twice(or just F4 and then select hex mode). Now press F5 and enter .00403036
Only thing we have done is use hiew to search for the address to see what is at that address. As you will see it points to A:\

```
* Reference To: KERNEL32.GetDriveTypeA, Ord:00F0h
                        |
:00401011 E8F6000000              Call 0040110C
:00401016 83F805                  cmp eax, 00000005
:00401019 7411                    je 0040102C
:0040101B 803D363040005A          cmp byte ptr [00403036], 5A
:00401022 7440                    je 00401064
```

Here you see a Call 0040110C and since we want to know what it means we can do 2 things.

1) use hiew to find it out
2) in this case just look above and you see it means KERNEL.GetDriveTypeA

Now that looks a lot like a function doesn't it? So fire up msdn or win32.hlp and search for: GetDriveType. When found you will see that it's a function that: *The GetDriveType function determines whether a disk drive is a removable, fixed, CD-ROM, RAM disk, or network drive.* Ah that's good news the function has something to do with drives including cd-rom and this entire challenge is about a cd check. Now look at the arguments it accepts.
LPCTSTR lpRootPathName // address of root path

So it needs something like A:\ C:\ E:\ etc. Hmmm isn't that exactly what we searched up earlier? Now let's see what kinda return values the function does have.:

Return Values
The return value specifies the type of drive. It can be one of the following values:

Value	*Meaning*
0	*The drive type cannot be determined.*
1	*The root directory does not exist.*
DRIVE_REMOVABLE	*The drive can be removed from the drive.*
DRIVE_FIXED	*The disk cannot be removed from the drive.*
DRIVE_REMOTE	*The drive is a remote (network) drive.*
DRIVE_CDROM	*The drive is a CD-ROM drive.*
DRIVE_RAMDISK	*The drive is a RAM disk.*

Well after the 1 it goes into letters but you can just keep counting. Now if we look back at the asm code you see that eax gets compared to 5. So eax holds the return value of the function and compares it to 5. Looking at the little table above you will see that 5 = DRIVE_CDROM and A:\ defiantly isn't a cd-rom (unless you got some wicked comp).

So now that we know all this let's look at the jmp

```
:00401016 83F805                cmp eax, 00000005
:00401019 7411                  je 0040102C
:0040101B 803D363040005A        cmp byte ptr [00403036], 5A
:00401022 7440                  je 00401064
```

You see 2 jumps and 1 more compare. (je stands for jump if equal) So the first jump jumps to 0040102C IF the comparison is equal. So let's see what that address is, it lands us at the beginning of the following code

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00401019(C)
|
:0040102C 6A00                  push 00000000
:0040102E 6A00                  push 00000000
:00401030 6A00                  push 00000000
:00401032 6A00                  push 00000000
:00401034 6A00                  push 00000000
:00401036 6A14                  push 00000014
:00401038 68AA314000            push 004031AA

* Possible StringData Ref from Data Obj ->"A:\"
                        |
:0040103D 6836304000            push 00403036

* Reference To: KERNEL32.GetVolumeInformationA, Ord:0162h
                        |
:00401042 E8D1000000            Call 00401118
:00401047 51                    push ecx
:00401048 33C9                  xor ecx, ecx

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00401060(U)
|
:0040104A 8A8100304000          mov al, byte ptr [ecx+00403000]
:00401050 3881AA314000          cmp byte ptr [ecx+004031AA], al
:00401056 750A                  jne 00401062
:00401058 41                    inc ecx
:00401059 83F90A                cmp ecx, 0000000A
:0040105C 7202                  jb 00401060
:0040105E EB1E                  jmp 0040107E
```

As said before it lands us in a row of pushes and it keeps going until it encounters a function Call 00401118 now if we look this up or just look above to see what function it is. WE see that it is GetVolumeInformation so let's search this up on msdn. Now when reading this you see that the function return the name of the volume. Meaning that if you insert a disc it return the name of it. You can conclude that the name of the inserted disk will be used as a extra check. So the following lines after that are all to make that extra check. Now if you look carefully you see that ECX is pushed which mostly is used as a counter register. Then it's

xored out leaving it at a value of zero. Then you see that the actual volume name check is started. If you go to the first address you see that it's THE MATRIX and in the compare that follows it gets compared to the same text to know if you entered the cd with the correct name. After the compare there is a JNE (jump if not equal) so if the name is not equal to THE MATRIX then it jump and if you jump along with W32DASM you will see it lands you at the complete beginning again. Well that isn't what we want is it? Supposing it's equal you that the counter gets updated with 1 and the compared to a value of 10 (since A in hex is the equivalent of 10 in decimal) if the result of that comparison is below 10 the whole thing starts over again.

Now let's think. And try to patch this nifty crackme. This crackme doesn't only have a cd check but also a volume name check and a length check. Making it harder for us to crack. Now here comes the different part of this tut compared to others. I don't want to patch all 3 checks. So let's start with the first check the cd check. Why start with this one? Well cause this one has to be done else you never reach the other 2 checks.

We have 2 options either we change the name of the drive or we change the number compared to. We will go for the easiest We change the number since most computers use the letter A to indicate a floppy drive.

So this line:

:00401016 83F805 *cmp eax, 00000005*

Becomes:

:00401016 83F805 *cmp eax, 00000002*

So now it's always true. Unless it isn't a floppy drive.

Now for the second and third checks let's patch only the second one in such a way that the third check is never reached and we immediate go to the cd found messagebox.

:0040104A 8A8100304000 *mov al, byte ptr [ecx+00403000]*
:00401050 3881AA314000 *cmp byte ptr [ecx+004031AA], al*
:00401056 750A *jne 00401062*
:00401058 41 *inc ecx*
:00401059 83F90A *cmp ecx, 0000000A*

As you can see the line:

:00401056 750A *jne 00401062*

Is the critical line that decides if we enter the third check or not. So assuming that we never are going to enter the correct cd we can change the jump to address to the beginning of the correct messagebox. So the line becomes:

:00401056 750A *jne 004010DB OR*
:00401056 750A *jne 004010DD*

Now you wonder from where did he fucking get that new address? Well just walk along the code until you see where the good messagebox starts. You will encounter this:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:004010D2(U), :004010D4(U)
|
:004010DB 6A00                  push 00000000

* Possible StringData Ref from Data Obj ->"Great!"
                        |
:004010DD 6843304000            push 00403043

* Possible StringData Ref from Data Obj ->"CD Found"
                        |
:004010E2 683A304000            push 0040303A
:004010E7 6A00                  push 00000000

* Reference To: USER32.MessageBoxA, Ord:01BBh
                        |
:004010E9 E806000000            Call 004010F4
:004010EE 50                    push eax
```

As you see both address land us in the good messagebox code.

With this we have successfully cracked the exe. Now as a little resume the 2 things you have to change in this exe.

```
:00401016 83F805                cmp eax, 00000005
                    &
:00401056 750A                  jne 00401062
```

The final screen when running it after patching will look like this:

So this concludes this little tut. If you have questions ask them on our forum:

http://forum.kd-team.com