Author : NaTzGUL [REVOLT´97]
Released on Nov.18.1997
Email : natzgul@hotmail.com

InstallSHIELD Script Cracking (best viewed under 800x600 with WordPad)

CONTENTS:

A) INTRODUCTION

I welcome you to my first Cracking Tutorial and I will try to write more Tutorials in the Future.
I could have made more in the past, but i was afraid if

anybody could read my **BAD English ;)**
so please excuse me and just try to follow me.

LEVEL : Well, I will try to give you all Informations and document all my Steps and Listings, so maybe also
a Beginner will understand this Tutorial (maybe ;).
As I told you the only Problem you will maybe have is my bad bad English ,hehe.

TARGET : Our Target is **Cakewalk HomeStudio from Twelve Tone Systems** ,
I have got it from Kirk_Hamm in #Cracking(EFNET)
THANX !!! =)
- a Person I dont really know ,he was just req the Crack.
The File contains not the whole App by the way, just all the neccessary Files to get the Installation running.
The compressed File size is only 536 KB, so if you want it just msg me on Efnet or Email me and i will
send ya the File if iam not busy =).

PROTECTION : This App has 3 Protections.

1.CD-CHECK
2.CD-KEY
3.SERIAL

B) TOOLS YOU NEED

You will need the following Tools:

- **SoftICE 3.x** from Numega (The best Debugger. Big Thanx to Numega)
- **W32Dasm 8.9** from URSoft (I love References)
- **Hex-Workshop** or any other **Hex-Editor** (Yeah, gimme the Bytes location)

- **Icompx** the **InstallSHIELD de/compressor** (Thanx to Lord Caligio that he has put it on his Page)

- A Martini and/or a cigarette if ur a +Cracker ;)

You can get all these Tools from **Lord Carligo´s Web-Page**. One of the best Cracking Resource i ever have seen before by the way !!!

http://cracking.home.ml.org/

C) WHAT WE ARE DEALING WITH

After unzip´ping the File into **C:/TEMP** there are the following files:

_SETUP.LIB	**151**	**KB**
SETUP.EXE	**659**	**KB**
_SETUP.DLL	**5,98**	**KB**
SETUP.INS	**89,5**	**KB**
SETUP.PKG	**Not important**	

(There are a lot more files in the complete App)

Let me first explain what we got here.

These are the typical Files from a InstallSHIELD Installation. **_SETUP.LIB** is a **compressed Data-Base** from InstallSHIELD. It can contain **exe´s and dll´s** supporting the Installation.
Sometimes these Support Files are in the same dir like SETUP.EXE (unlikely), but in our case they are compressed into _SETUP.LIB (You will see later).
What that person from #Cracking didn´t send me was the compressed Data-Base Files (xxx.1-x,xxx.z) containing the App Files and so they can be very big ;).
Don´t mind it , because we dont need them anyway for cracking.
A **compressed Data-Base File** allways begins with **"13 5D 65 8C 3A 01 02 00",**
so if you cant find any xxx.z or xxx.1-x then just look for these bytes.
At the End of every compressed Data-Base File you can see all the
File Names by the way.
SETUP.PKG contains all the **File-Names** in the App Data-Base which we

dont need and so we dont need SETUP.PKG either.
InstallSHIELD uses SETUP.PKG to refer the Files in the App Data-Base
in the copying process i believe.
Anyway, we dont need it, so lets go on.
_SETUP.DLL is a **InstallSHIELD Resource DLL** and its not important for us,
because its only a Support File which is supplied with any InstallSHIELD Installation.
SETUP.INS is the **compiled Installation Script** and its the most important Part in a
InstallSHILED Installation Process !!!.
In Win95 it has got a globe connected to a phone as icon.

This File Controls any Action and has got most of the messages of the
Installation and it will play a major Role in our SECOND APPRAOCH.
SETUP.EXE is the head of all , its the **Installation Engine** and **executes the Script** and does all
calls to DLL´s and Disk-Access (32 Bit !!!).
So far so good, now we know much more about InstallSHIELD =)

Lets start with the....

D) FIRST APPROACH

(CD-CHECK)

ASSUMPTION : I assume the following things **under SoftICE** :

F5="^x;"
F7="^here;"
F8="^t;"
F9="^bpx;"
F10="^p;"
F11="^G @SS:ESP;"
F12="^p ret;"

Also the **winice.dat** File in your SoftICE dir should contain :

EXP=c:\windows\system\kernel32.dll
EXP=c:\windows\system\user32.dll

HINT : **"*"** in Front of the Text coming up means, that the **text into brackets** must be **typed under SoftICE!**

START : Ok, now lets get to business and start cracking.
First we just start the Istallation (SETUP.EXE) and see whats happening.
Well, a **MessageBox** tells us, that **"Setup must be run from the original CD".**
Our next logical step now should be setting a **Breakpoint on**
GetDriveTypeA ("A" coz SETUP.EXE is a 32 Bit App).
Have a look at part **G) WIN32.HLP** of this tutorial to get more info about **GetDriveType** !!!

* We press Crtl+D and SoftICE pops up and then we type in "BPX GetDriveTypeA"
* Pressing "Crtl+D" ("F5") gets us back to Windows, where we start Setup.exe again.

Ok, we are in SoftICE before the MessageBox appears.
We are in the Kernel32 at GetDriveTypeA, so lets get out of here

* by pressing "F11" one time. And now we are in **INSHELP**, damn !!! whats that ? it wasnt in our dir !!
* Well i typed in "MOD INSHELP" to get more info about this file

and SoftICE shows me, that its located in :

C:\TEMP_ISTMP0.DIR\INSHELP.DLL

Now we see that it´s a DLL and that IstallSHIELD has created a **Temporary directory** called **_ISTMP0.DIR** and then it puts the file INSHELP.DLL in there. But where this File comes from ?
Ok, maybe you dont have forgotten what i told you in **C)** about
compressed Data-Bases ? Yes ? Then you should read it again now !!!!
So this DLL must be in **_SETUP.LIB**, but how should we patch it ?
Well we got **ICOMPX** the **InstallSHIELD de/compressor** ;)
Let´s decompress _SETUP.LIB **("ICOMP _SETUP.LIB *.* -d -i")**
These Files we will get :

INSHELP.DLL
UNINST.EXE
_ISRES.DLL

The **last two files** are only **support Files** and not important for us.

What we know now is that **INSHELP.DLL** makes the **CD-CHECK** and that it is in **_SETUP.LIB** which we can decompress and then compress again.

By the way you may just type in **"ICOMP"** to get the **full usage.**

Now that we got all infos about this File and how to patch it lets go on with SoftICE´ing ;).

We are still in INSHELP.DLL, so let me give you the listing first :

Your adresses may differ in the first four diggits ! (relocation)

And SoftICE pops up at 100011A0 (0) , so go there now !!!!

DWORD TABLE:

```
:10001308 BA120010          DWORD 100012BA          These are the DWORDS for the indirect jmps
:1000130C C7120010          DWORD 100012C7          I have place them here coz it will be
:10001310 D4120010          DWORD 100012D4
        easier for you to follow me ;)
:10001314 E1120010          DWORD 100012E1
:10001318 EE120010          DWORD 100012EE
:1000131C B0110010          DWORD 100011B0
:10001320 FB120010          DWORD 100012FB
```

Start of this routine:

```
:10001160 81ECE8020000      sub esp, 000002E8
        Create a tempprary Stack-Frame
:10001166 B9FFFFFFFF        mov ecx, FFFFFFFF
        ecx=FFFFFFFF (counter)
:1000116B 2BC0              sub eax, eax                eax=0
:1000116D 56                    push esi                Save esi
:1000116E 57                    push edi                Save edi
:1000116F 8BBC24F4020000 mov edi, [esp + 000002F4]      edi points
```

to **"C:\TEMP\"**
:10001176 F2 repnz
:10001177 AE scasb Scan String for 0 (end)
:10001178 F7D1 not ecx
ecx=lenght+1=9
:1000117A 2BF9 sub edi, ecx Adjust edi back
:1000117C 8BC1 mov eax, ecx Save lenght in eax
:1000117E C1E902 shr ecx, 02 Divide lenght by 4 =2
:10001181 8BF7 mov esi, edi
esi=edi=ptr to **"C:\TEMP\"**
:10001183 8D7C2448 lea edi, [esp + 48] <------|
edi=ptr to [esp+48]
:10001187 F3 repz |
:10001188 A5 movsd | Copy "C:\TEMP\" to *edi
:10001189 8BC8 mov ecx, eax |
ecx=eax=lenght
:1000118B 83E103 and ecx, 00000003 |
ecx=mod 9/4=1
:1000118E F3 repz |
:1000118F A4 movsb | Copy last byte(s)
:10001190 C644244B00 mov [esp + 4B], 00 |------- "C:\TEMP\"
:10001195 8D4C2448 lea ecx, [esp + 48] <------ **"C:\" = RootPathName**
:10001199 51 push ecx **Handle it to GetDriveTypeA**

*** Reference To: KERNEL32.GetDriveTypeA, Ord:00CEh**
|
:1000119A FF15E0900010 Call dword ptr [100090E0] This calls GetDriveTypeA (return: eax=Type)
:100011A0 83F806 cmp eax, 00000006 <------
(0) SoftICE breaks in here !!!
:100011A3 0F8704010000 ja 100012AD **(1)**
:100011A9 FF248508130010 jmp dword ptr [4*eax + 10001308] **(2)**
:100011B0 8D442414 lea eax, [esp + 14] **(3)**
:100011B4 6A32 push 00000032
FileSystemNameSize
:100011B6 8D4C2414 lea ecx, [esp + 14]
:100011BA 50 push eax

lpFileSystemNameBuffer
:100011BB 8D542414 lea edx, [esp + 14]
:100011BF 51 push ecx
lpFileSystemFlags
:100011C0 8D442414 lea eax, [esp + 14]
:100011C4 52 push edx
lpMaximumComponentLength
:100011C5 8D8C2420010000 lea ecx, [esp + 00000120]
:100011CC 50 push eax
lpVolumeSerialNumber
:100011CD 8D54245C lea edx, [esp + 5C]
:100011D1 68C8000000 push 000000C8
VolumeNameSize
:100011D6 51 push ecx
lpVolumeNameBuffer
:100011D7 52 push edx
lpRootPathName ("C:\")

Ok, we are right after the GetDrivetypeA call.
Let us first figure out what will happen if we trace further.
(1) This conditional jmp will never happen if i can trust on the
Disscription of GetDriveType.
(2) My eax is 3 (Hard-Disk) so this ptr will be 3*4+10001308=10001314
so this jmp would lead us to **100012E1 (see the DWORD TABLE above !)**

:100012E1 33C0 xor eax, eax Set eax to 0
:100012E3 5F pop edi
Restore edi from stack
:100012E4 5E pop esi
Restore esi from stack
:100012E5 81C4E8020000 add esp, 000002E8
Delete temporary Stack-Frame
:100012EB C20400 ret 0004 return

Well it seems that **EAX=0** stands for **BAD BOY ;)**
Cracking this CD-CHECK could end here just by patching the instructions
at the Start of this routine (10001160)...

Original:
:10001160 81ECE8020000 sub esp, 000002E8
Create a temporary Stack-Frame

```
:10001166 B9FFFFFFFF        mov ecx, FFFFFFFF
        ecx=FFFFFFFF
:1000116B 2BC0              sub eax, eax                    eax=0
:1000116D 56                    push esi                    Save esi
:1000116E 57                    push edi                    Save edi
```

Change to:

```
:10001160 33C0              xor eax,eax                 eax=0
:10001162 40                    inc eax
        eax=eax+1=1 GOOD BOY
:10001163 C20400                ret 0004                    Return
```

Search for "81ECE8020000" in INSHELP.DLL with your Hex-Editor.
You will only find one location (Offset 560). **Replace the bytes with "33C040C20400"** and **save it**.
Ok, and now **compress it back into _SETUP.LIB**.
Just type in **"icomp inshelp.dll _setup.lib"** and **dont delete INSHELP.DLL**,
because we will need it again later ;)

Do you want to know what this CD-CHECK would do further on ?
If not just go over to the (CD-KEY) Section below !!!

Hmmm, so you wanna learn more about CD-CHECKS ;) OK
* What we do now is setting eax to 5 by typping in "r eax=5"
then the jmp will bring us to
dptr[5*4+10001308]=dptr[1000131C]=100011B0
which means we are right after the jmp itself ! at **(3)**
The instructions after **(3)** just pushes all the infos for the GetVolumeInformationA call at 100011D8.

*** Reference To: KERNEL32.GetVolumeInformationA, Ord:013Ah**
|

```
:100011D8 FF15DC900010          Call dword ptr [100090DC]       This calls GetVolumeInformation
:100011DE 85C0              test eax, eax                   Do we got all infos?
:100011E0 0F8481000000          je 10001267                 (4)  if yes goto 10001267
:100011E6 8D842410010000        lea eax, [esp + 00000110]
        Volume Name ("HD_C")
```

*** Possible StringData Ref from Data Obj ->"CWHS_601"**

:100011ED B938600010 mov ecx, 10006038

*** Referenced by a (U)nconditional or (C)onditional Jump at Address:**
|:1000120C(C)

:100011F2 8A10 mov dl, [eax] Here it **compares** my Volume Name **"HD_C"**
:100011F4 3A11 cmp dl, [ecx] with **"CWHS_601"**
:100011F6 751A jne 10001212 **(5)** Bad jmp !
:100011F8 0AD2 or dl, dl
:100011FA 7412 je 1000120E
:100011FC 8A5001 mov dl, [eax+01]
:100011FF 3A5101 cmp dl, [ecx+01]
:10001202 750E jne 10001212 **(5)** Bad jmp !
:10001204 83C002 add eax, 00000002
:10001207 83C102 add ecx, 00000002
:1000120A 0AD2 or dl, dl
:1000120C 75E4 jne 100011F2

*** Referenced by a (U)nconditional or (C)onditional Jump at Address:**
|:100011FA(C)

:1000120E 33C0 xor eax, eax All OK !
:10001210 EB05 jmp 10001217

To continue our tracing session you have to **nop out the Bad jmps !**

* Trace to the jmps "F10" and then "a" with two "nop"´s.

(4) This jmp will only occure if Setup is running from the original CD-Rom.

It then just bypasses the Volume and Filetype Check.

I also suggest that you read part F) of this Tutorial to get more and
detailed infos about GetVolumeInformation (FileSytemFlags) !!

Ok, now comes the part the **(5)** Bad jmps will jump to....

*** Referenced by a (U)nconditional or (C)onditional Jump at Addresses:**
|:100011F6(C), :10001202(C)

```
|
:10001212 1BC0              sbb eax, eax                        eax=0
:10001214 83D8FF                sbb eax, FFFFFFFF
        eax=1

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:10001210(U)
|
:10001217 85C0              test eax, eax                       if eax=0
then
:10001219 740D              je 10001228                         goto
10001228 GOOD BOY !
:1000121B 33C0              xor eax, eax                        otherwise
return
:1000121D 5F                    pop edi                             with
eax=0 BAD BOY !
:1000121E 5E                    pop esi
:1000121F 81C4E8020000          add esp, 000002E8
:10001225 C20400                ret 0004



* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:10001219(C)
|
:10001228 8D4C2414              lea ecx, [esp + 14]                 ecx
points to my File System Name "FAT"

* Possible StringData Ref from Data Obj ->"CDFS"
                        |
:1000122C B848600010            mov eax, 10006048

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:1000124B(C)
|
:10001231 8A11              mov dl, [ecx]                       here my
File System Name "FAT"
:10001233 3A10              cmp dl, [eax]                       will be
compared with "CDFS" !
:10001235 751A              jne 10001251                (6)     Bad jmp !
:10001237 0AD2              or dl, dl
:10001239 7412              je 1000124D
:1000123B 8A5101                mov dl, [ecx+01]
:1000123E 3A5001                cmp dl, [eax+01]
:10001241 750E              jne 10001251                (6)     Bad jmp !
:10001243 83C102                add ecx, 00000002
```

:10001246 83C002 add eax, 00000002
:10001249 0AD2 or dl, dl
:1000124B 75E4 jne 10001231

*** Referenced by a (U)nconditional or (C)onditional Jump at Address:**
|:10001239(C)
|
:1000124D 33C0 xor eax, eax **All OK !**
:1000124F EB05 jmp 10001256

Again we have to nop out the (6) Bad jmps to continue !!
Otherwise we will land here...(10001251) BAD BOY

*** Referenced by a (U)nconditional or (C)onditional Jump at Addresses:**
|:10001235(C), :10001241(C)
|
:10001251 1BC0 sbb eax, eax **Old soup, look back (10001212) !**
:10001253 83D8FF sbb eax, FFFFFFFF

*** Referenced by a (U)nconditional or (C)onditional Jump at Address:**
|:1000124F(U)
|
:10001256 85C0 test eax, eax
:10001258 740D je 10001267 **GOOD BOYS jmps to 10001267**
:1000125A 33C0 xor eax, eax
:1000125C 5F pop edi
:1000125D 5E pop esi
:1000125E 81C4E8020000 add esp, 000002E8
:10001264 C20400 ret 0004

*** Referenced by a (U)nconditional or (C)onditional Jump at Addresses:**
|:100011E0(C), :10001258(C)
|
:10001267 8A442448 mov al , [esp + 48]
al=**Drive Letter "C" 43h**
:1000126B 8D8C24D8010000 lea ecx, [esp + 000001D8]
:10001272 51 push ecx
:10001273 A250600010 mov [10006050], al

^----------**"X:\Cakewalk_setup.lib"**

*** Possible StringData Ref from Data Obj ->"C:\Cakewalk_setup.lib"**
|
:10001278 6850600010 push 10006050
:1000127D E8EE010000 call 10001470 <----------In this Sub it will call **FindFirstFileA**
:10001282 83C408 add esp, 00000008 to **look for "_setup.lib"**
:10001285 83F8FF cmp eax, FFFFFFFF in **"C:\Cakewalk\"** directory
:10001288 750D jne 10001297 **(7)** Well it wont find it there and so it will
:1000128A 33C0 xor eax, eax **return with eax=FFFFFFFF**
:1000128C 5F pop edi
:1000128D 5E pop esi
:1000128E 81C4E8020000 add esp, 000002E8
:10001294 C20400 ret 0004

(7) Change it to "jmp 10001297" or "je 10001297" to continue !!!

*** Referenced by a (U)nconditional or (C)onditional Jump at Address: |:10001288(C)**
|
:10001297 E894FDFFFF call 10001030 **(8)** In this Sub **eax** will just be
:1000129C 5F pop edi **set to 1 GOOD BOY ;)**
:1000129D 83F801 cmp eax, 00000001 if eax was wasnt 1 this
:100012A0 1BC0 sbb eax, eax sub will turn eax to
:100012A2 5E pop esi
FFFFFFFF and the inc
:100012A3 40 inc eax
finally will make it 0 BAD BOY !
:100012A4 81C4E8020000 add esp, 000002E8
:100012AA C20400 ret 0004

(8) Here the call will go to...

:10001030 A130600010 mov eax, [10006030]
eax=dword at [10006030]

:10001035 C3 ret Return

Setup calls a **Sub in INSHELP** while **initialisation**, which sets the **dword [10006030] to 1** !!!

OK, we just have learned something more about CD-Protections under Windows95 ;)

- INSHELP first checks if setup is running from a CD-ROM.
- Then it checks the Volume Name and the File System.
- And at least it just checks for a specific File "setup.lib".
- After all INSHELP will return "1" for OK and "0" for Error !!!

This CD-CHECK is defeated, now lets face the....

(CD-KEY)

Ok, the MessageBox never appears now, we get a Welcome Window instead ;)
We get an Edit area and a Text telling us to enter the **13 diggit CD-KEY** ,brbrb.
We type in "1234567890123" and then i set a **Breakpoint on**
* **GetWindowTextA** : "BPX GetWindowTextA".
After pressing the **NEXT-> Button** SoftICE pops up, this is easy hehe ;)
* We are in GetWindowTextA so lets get back to the App and press "F11".
I looked at **EAX**, because it allways **contains the Text lenght**
GetWindowTextA returns,
but hell !!!! this isnt the lengh of my Text and so this cant be
my Text =(, brb.

Dont worry, this is just a little trick to prevent Beginners to crack it.
There are lotta other App out there using this trick btw !

Setup uses GetWindowTextA to retrieves our input, but

it dont wait for
the user pressing NEXT->, it just gets the text **anytime we type in a single letter,**

* so lets first disable our Breakpoint : "BD 0",
and then **we type in "12345678901234"** and then we enable our Breakpoint :

* "BE 0".(dont forget to leave SoftICE)
So, now comes the truth. I just deleted the last number with back-space
and BOOM !!! yeah we are in GetWindowTextA again so lets leave here

* again by pressing "F11".
Well, this looks much better, because **EAX=0D=13, yeah our Key-lenght ;)**
We are **in Setup** by the way. Right after the Call GetWindowTextA
there is a **"LEA EAX,[EBP+FFFFFBF4]"** which will let **EAX points to our Text,**

* so trace over it with "F8" or "F10".
* Do a "D EAX" and you will see our text "1234567890123" !!
* ok lets delete our Breakpoint, because we got what we wanted : "BC *".

And now we set a **Breakpoint on Memory Access on our text loaction** :

* "BPM EAX". Ok, exit SoftICE and it will fast pop up again.
SoftICE will break into different locations, but the one that is important for us is the **lstrcpyA.**
You will land in there **at the following instructions** :

```
...     ............
REPNZ       SCASB               <----------**SoftICE will break in here !!!**
NOT   ECX
MOV  ESI,[EBP+0C]                                   This is our old location
MOV  EDI,[EBP+08]                                   This will be our new location
...     ............
```

So, **if you see these instructions** you can **delete your old breakpoint**,

* **trace over the 2 MOVS** with "F8" and then **set a new Breakpoint on EDI**:
* "BPM EDI". **Otherwise just leave SoftICE** until you are **back in the**

Installation Window. Press **NEXT->** and you will break into lstrcpyA
several times again, but now **dont delete the old Breakpoints**,
just **set the new ones on EDI** after the 2 MOVS like before,
until you are in **INSHELP !!!! yeah its the same dll ;).**
Let me give you the listing first and consider again that the first
four digits of the adresses may differ from yours under SoftICE (relocation).

SoftICE will break in at 10001377 !!!

Start of this routine:

```
:10001350 83EC34          sub esp, 00000034
        Create a temporary Stack-Frame
:10001353 53              push ebx              Save ebx
:10001354 56              push esi              Save esi
:10001355 57              push edi              Save edi
:10001356 E8D5FCFFFF      call 10001030         Was this routine initialysed ?
:1000135B 85C0            test eax, eax         Check ok ? (It will be)
:1000135D 750B            jne 1000136A          then goto 1000136A, else
:1000135F 33C0            xor eax, eax          Set eax=0 BAD BOY !!!
:10001361 5F              pop edi
        Restore edi
:10001362 5E              pop esi
        Restore esi
:10001363 5B              pop ebx               Restore ebx
:10001364 83C434          add esp, 00000034
        Delete temporary Stack-Frame
:10001367 C20400          ret 0004              Return
```

Well it seems that **EAX=0** stands for **BAD BOY again like in the CD-Check !!**
Cracking this CD-KEY could end here just by patching the instructions
at the Start of this routine (10001350)...
Dont patch it yet, if you wanna learn how to reverse ingineer this KEY-Protection !!!!

Original:
:10001350 83EC34 sub esp, 00000034
Create a temporary Stack-Frame
:10001353 53 push ebx Save ebx
:10001354 56 push esi Save esi
:10001355 57 push edi Save edi
:10001356 E8D5FCFFFF call 10001030 **Was this routine initialysed ?**

Change to:
:10001350 33C0 xor eax,eax eax=0
:10001352 40 inc eax
eax=eax+1=1 GOOD BOY
:10001353 C20400 ret 0004 Return

Search for "83EC34535657" in INSHELP.DLL with your Hex-Editor.
You will only find one location (Offset 750). **Replace the bytes with "33C040C20400"** and **save it**.
Ok, and now **compress it back into _SETUP.LIB**.
Just type in **"icomp inshelp.dll _setup.lib"** and **dont delete INSHELP.DLL**,
because we will need it again later ;)
And now any KEY you type in will be valid, cool heh =)

Do you wanna learn how to reverse this CD-KEY Protection ?
If not just go over to the (SERIAL) Section below !!!

Ok, lets go on with this routine...

*** Referenced by a (U)nconditional or (C)onditional Jump at Address:**
|:1000135D(C)
|
:1000136A 8B5C2444 mov ebx, [esp + 44] **ebx will point to our KEY !**
:1000136E 8D4C240C lea ecx, [esp + 0C] **ecx will be the new location**
:10001372 8BC3 mov eax, ebx
eax=ebx=pointer to our KEY
:10001374 803B00 cmp byte ptr [ebx], 00 **(9)**
KEY=NULL ?
:10001377 741B je 10001394 <----------**SoftICE will break in here !!!!**

(9) Check if our KEY is emty, if yes goto 10001394

*** Referenced by a (U)nconditional or (C)onditional Jump at Address: |:10001392(C)**
|
:10001379 8A10 mov dl, [eax] **(10) Get a char from our KEY**
:1000137B 0FBEF2 movsx byte ptr esi, edx
esi=dl=the char
:1000137E 83FE30 cmp esi, 00000030
Compare char with "0"
:10001381 7C05 jl 10001388 If lower goto 10001388, else
:10001383 83FE39 cmp esi, 00000039
Compare char with "9"
:10001386 7E03 jle 1000138B If lower,equal then goto 1000138B

*** Referenced by a (U)nconditional or (C)onditional Jump at Address: |:10001381(C)**
|
:10001388 40 inc eax
Increment char pointer
:10001389 EB04 jmp 1000138F goto 1000138F

*** Referenced by a (U)nconditional or (C)onditional Jump at Address: |:10001386(C)**
|
:1000138B 8811 mov [ecx], dl **(11) Store number in new location**
:1000138D 40 inc eax
Increment char pointer
:1000138E 41 inc ecx
Increment loacation pointer

*** Referenced by a (U)nconditional or (C)onditional Jump at Address: |:10001389(U)**
|
:1000138F 803800 cmp byte ptr [eax], 00 End of KEY ?
:10001392 75E5 jne 10001379 If not then goto 10001379

(10) This pice of code will retrieve **only numbers** from our

KEY and then it

(11) stores them at the new location, so if you typed in "1234a67b89" the new

location will contain only "12346789" consider this !!

*** Referenced by a (U)nconditional or (C)onditional Jump at Address: |:10001377(C)**

|

```
:10001394 8D7C240C      lea edi, [esp + 0C]          (12) edi will point to our KEY
:10001398 2BC0          sub eax, eax                      Set eax=0
:1000139A C60100            mov byte ptr [ecx], 00
        Teminate KEY with 0
:1000139D B9FFFFFFFF    mov ecx, FFFFFFFF                 Set counter ecx to FFFFFFFF
:100013A2 F2                repnz
:100013A3 AE                scasb                     Scan KEY for "0" = End
:100013A4 F7D1          not ecx
:100013A6 49                dec ecx                        ecx = KEY length
:100013A7 83F90D            cmp ecx, 0000000D         (13) KEY length = 13 diggits ?
:100013AA 740B          je 100013B7                       If yes goto 100013B7, else
:100013AC 33C0          xor eax, eax                      BAD BOY !!!
:100013AE 5F                pop edi
:100013AF 5E                pop esi
:100013B0 5B                pop ebx
:100013B1 83C434            add esp, 00000034
:100013B4 C20400            ret 0004
```

(12) This part calculates our **KEY length** and then it checks if it is

(13) 13 (0Dh) diggits long. If not it will return with eax=0 BAD BOY !!!

*** Referenced by a (U)nconditional or (C)onditional Jump at Address: |:100013AA(C)**

|

```
:100013B7 8D44240C      lea eax, [esp + 0C]               eax points to the KEY at [esp+0C]
:100013BB 50                push eax                  Handle it to Sub
:100013BC E87F000000        call 10001440             (14)
```

Generate code

:100013C1 3D377B0E00 cmp eax, 000E7B37 **(15)**

Compare code with E7B37

:100013C6 7565 jne 1000142D If not equal then goto 1000142D BAD BOY !

:100013C8 0FBE4C240C movsx byte ptr ecx, [esp + 0C] **(16)**

ecx= 1. number from KEY

:100013CD 8D1489 lea edx, [ecx + 4*ecx]

edx=ecx*5

:100013D0 0FBE44240F movsx byte ptr eax, [esp + 0F]

eax= 4. number from KEY

:100013D5 8D0C50 lea ecx, [eax + 2*edx]

ecx=edx*2+eax

:100013D8 8D1489 lea edx, [ecx + 4*ecx]

edx=ecx*5

:100013DB 0FBE442410 movsx byte ptr eax, [esp + 10]

eax= 5. number from KEY

:100013E0 8D0C50 lea ecx, [eax + 2*edx]

ecx=edx*2+eax

:100013E3 8D1489 lea edx, [ecx + 4*ecx]

edx=ecx*5

:100013E6 0FBE442411 movsx byte ptr eax, [esp + 11]

eax= 6. number from KEY

:100013EB 8D0C50 lea ecx, [eax + 2*edx]

ecx=edx*2+eax

:100013EE 2B0D54610010 sub ecx, [10006154] **(17) Sub App-ID (E11)**

:100013F4 81F950D00000 cmp ecx, 0000D050 **(18)**

Compare with D050

:100013FA 7531 jne 1000142D If not equal then goto 1000142D BAD BOY !

:100013FC 8D7C240C lea edi, [esp + 0C] **(19)** edi points to the KEY

:10001400 B9FFFFFFFF mov ecx, FFFFFFFF Set counter to FFFFFFFF

:10001405 2BC0 sub eax, eax Set eax=0

:10001407 F2 repnz

:10001408 AE scasb Scan KEY for "0"=End

:10001409 F7D1 not ecx ecx = KEY length+1

:1000140B 2BF9 sub edi, ecx Adjust edi back

:1000140D 8BC1 mov eax, ecx eax= ecx

:1000140F C1E902 shr ecx, 02

ecx=ecx/4=3

```
:10001412 8BF7          mov esi, edi                    esi points
to the KEY
:10001414 8BFB          mov edi, ebx                    edi=old
location of KEY
:10001416 F3                repz
:10001417 A5                movsd                           Copy
KEY to old location
:10001418 8BC8          mov ecx, eax                    ecx = KEY
length
:1000141A 83E103            and ecx, 00000003               ecx =
mod ecx/4=1
:1000141D F3                repz
:1000141E A4                movsb                           Copy
last byte(s)
:1000141F B801000000        mov eax, 00000001
        eax=1 GOOD BOY !!!
:10001424 5F                pop edi
:10001425 5E                pop esi
:10001426 5B                pop ebx
:10001427 83C434            add esp, 00000034
:1000142A C20400            ret 0004

* Referenced by a (U)nconditional or (C)onditional Jump at
Addresses:
|:100013C6(C), :100013FA(C)
|
:1000142D 33C0          xor eax, eax                (20)  eax=0
BAD BOY !!!
:1000142F 5F                pop edi
:10001430 5E                pop esi
:10001431 5B                pop ebx
:10001432 83C434            add esp, 00000034
:10001435 C20400            ret 0004
```

To reverse engineer a KEY-Check i start at the end of the routine.

I mean where the final check occures !!!. This will happen at line **(18) 100013F4.**

Here **ecx** must be **D050.** Now lets go back to the previous line.

Here ecx will be **subtracted by E11 the App-ID**, this means **ecx must be D050+E11=DE61**

at this point !!!!

Now let us see what the instructions at **(16)** does !

Well, let me first extract the few lines from 100013C8 - 100013EB into a more comfortable format

for you :
(1000,100 and 10 are in decimal ; numbers are in asc-II !!!)

ecx=(((1. number) * 10 + 4. number) * 10) + 5. number) * 10) + 6. number

After simplification we get :

ecx=1. number * 1000 + 4. number *100 + 5. number *10 + 6. number

Hmm, now we know how ecx is calculated, but **whats D050 ?**
Well, if we typed in **"0"=48=30h** as our 1.,4.,5. and 6. number, then we will get :

ecx=30h * 1000d + 30h * 100d + 30h * 10d + 30h = D050 !!! =)

And now consider that **E11 h=3601 d= 3 * 1000d + 6 * 100d + 0 * 10d + 1** !!!
Now guess what our 4 numbers are ;) !!!
Yes, thats right...

the 1. number must be 3 !!!
the 4. number must be 6 !!!
the 5. number must be 0 !!!
and the 6. number must be 1 !!!

So our KEY is build like this "3xx601xxxxxxx" ,hehe !!!

Ok, lets look back before **(16)**
(14) This will call a sub at **10001440** which will **calculate a code** with our KEY.
(15) This code will be **compared with E7B37 !!!**
If this compare fails we will land at **(20) 1000142D BAD BOY !!!**

Let us first examine the sub which generates the code...

```
:10001440 56          push esi          Save esi
:10001441 33D2        xor edx, edx      edx=0
:10001443 57          push edi          Save edi
:10001444 33C9        xor ecx, ecx      ecx=0, this
```

will be our char position counter
:10001446 8B74240C mov esi, [esp + 0C] **esi will point to our KEY**
:1000144A 380E cmp [esi], cl Is the KEY emty ?
:1000144C 7419 je 10001467 If yes goto 10001467 and return with code=0

*** Referenced by a (U)nconditional or (C)onditional Jump at Address: |:10001465(C)**
|
:1000144E C1E206 shl edx, 06 **(21)**
edx=edx*2^6=edx*64d=edx*40h
:10001451 BFE1D61200 mov edi, 0012D6E1
edi=12D6E1
:10001456 0FBE040E movsx byte ptr eax, [esi + ecx] **(22)**
get next number from our KEY
:1000145A 03C2 add eax, edx
eax=eax+edx
:1000145C 41 inc ecx
ecx=ecx+1, counter +1
:1000145D 2BD2 sub edx, edx **edx=0**
:1000145F F7F7 div edi **(23)**
eax=eax/edi, edx=mod (eax/edi)
:10001461 803C0E00 cmp byte ptr [esi + ecx], 00
Reach end of KEY ?
:10001465 75E7 jne 1000144E If not goto 1000144E

*** Referenced by a (U)nconditional or (C)onditional Jump at Address: |:1000144C(C)**
|
:10001467 8BC2 mov eax, edx **(24) eax=edx, the code !!!**
:10001469 5F pop edi
Restore edi
:1000146A 5E pop esi
Restore esi
:1000146B C20400 ret 0004 return

To reverse this sub we must start at the end of it at line **10001467 (24) !!!**
eax=edx is the code and it must be E7B37 (15) !!!
(23) Here we see that E7B37 is mod (eax/edi) = mod (eax/12D6E1)
(22) Well, this is shit !!!, because we will loose information

(eax) by each loop.

What we know is that eax will be clipped after every 4 number, because...

30*40*40*40+30*40*40+30*40+30=C30C30 > 12D6E1

Thus we can set a **seed KEY "3xx6x1yyyyyyy",** where x can be any number and

y will be the corrections.First go back to Setup and choice a **seed KEY** !!!

I used for example **"3006010000000".**

To get a valid KEY let us **Brute-Force-Crack** this babe =)

Its not the best way, but this code generating part is short, thus it will be executed fast.

Trace to the location at line **100013C1 (15)** where the code will be compared with E7B37.

Trace over it to the next line **100013C6** and then we have to code a little procedure.

* EBX is unused, so we will use it **as counter**. **Type in "r ebx=0".**
* Now type in "a" and let us add a little procedure, which will find a valid KEY for us.

Please adjust the adresses yourself, since this will be typed directly into memory !!!

*		**"JNZ**	**GO_ON"**	**Not a valid KEY, goto GO_ON**
*	**FOUND&FAIL:**	**"NOP"**		**This will be our Stop Point**
*	**GO_ON:**	**"CMP**	**EBX,1312CFF"**	**Check only numbers from 0-19999999 !!!**
*		**"JZ**	**FAIL"**	**Yes, goto FAIL**
*		**"MOV**	**ESI,[ESP+C]"**	**ESI points to our KEY**
*		**"MOV**	**EAX,EBX"**	**EAX=EBX**
*		**"MOV**	**ECX,A"**	**ECX=A=10d**
*	**CONVERT_DEC:**	**"XOR**	**EDX,EDX"**	**EDX=0**
*		**"DIV**	**ECX"**	**EAX=EAX/ECX, EDX=MOD (EAX/ECX)**
*		**"ADD**	**DL,30"**	**EDX=EDX+"0"**
*		**"MOV**	**[ESI+C],DL"**	**STORE NUMBER INTO KEY**
*		**"DEC**	**ESI"**	**ESI will point to the previous number**

*** "CMP EAX,0" Conversion completed ?**
*** "JNZ CONVERT_DEC" If not goto CONVERT_DEC**
*** "JMP 100013B7" Check this KEY !**

The comparision at GO_ON makes sure that the App-ID will not be manipulated !!

* Ok, you typed in all this mess ;) Now you must **clear all Break-Points** "BC *"

* and then set a **Break-Point on execution** on line **FOUND&FAIL !!!! "BPX <your adress>".**

Now leave SoftICE and wait.....

SoftICE will pop up at **FOUND&FAIL**, so first **check EAX**, it should be **E7B37** !!!

* If yes, you can get your KEY with **"D [ESP+C]".**
I have found **"3006010147046"** for my seed KEY ,btw =)

* **To get out of this Loop set your EIP to 1000142D "r eip=1000142D" and clear all**
Break-Points !!!
Then leave SoftICE, and you will be back in Setup. Cancel it and then start it again
and use your valid KEY !!!

Summarize:
- KEY must contain 13 numbers.
- KEY has got 4 fixed numbers "**3**xx**601**yyyyyyy". Its the App-ID (3601), which may differ in other
App from Twelve Tone Systems. Setup handles this App-ID to INSHELP before he calls it.
- yyyyyyy can be found with Brute-Force-Cracking.

This Protection is defeated, lets go over to the...

(SERIAL) Well, the KEY was a little bit tricky, heh ? Anyway you are here now to face the Serial !!!

Setup asks for a **User-Name, Company and Serial**, so lets type in sum crap.

I typed in "NaTzGUL" as User-Name, "REVOLT" as Company and "1234567890" as Serial.

Please procced with the Serial like in the KEY Section !!!!

You will land into Setup !!!, damn the Script is doing the Check, brbrb.

I gave up !!! There are just too many push,pop and calls, believe me, try it out !!!

To defeat this Protection we need a new method !!!

E) SECOND APPROACH

ASSUMPTION: I assume that you have partialy read the first Approach and that the App (INSHELP)
is unpatched in any way !!!! (Original state !!! you may uncompress the whole App again !).

INTRO: **Zen** !!! yeah, thats what we need =)
As i told you in our first approach **SETUP.INS** is the main part of a InstallSHIELD Installation !!!
SETUP.INS is a **compiled Script**, this means before compilation it may have the following
basic instructions :

- "IF,THEN,(ELSE)"
- "GOTO"
- "CALL"
- "RETURN()"
- "LOAD","OPEN","CLOSE"
- "MESSAGEBOX"
- etc.

To decrypt the whole mnemonic back to its instructions is not necessary to crack this app,
so i though that the most **important** instruction should be the **"IF,THEN"** one. It should occure very often in the Script and it may have the following syntax :

IF cmp THEN....

cmp = (arg1) compare_type (arg2)

arg1 is a variable, **arg2** can be a variable or a constant (two constants makes no sense ,of coz !).
the **compare_type** can only be one of these six types :

Type:	**Coresponding jmp:**

LOWER-EQUAL	JLE
GREATER-EQUAL	JGE
LOWER	JL
GREATER	JG
NOT-EQUAL	JNE
EQUAL	JE

A compiled COMPARE instruction could look like this :

Compare_mnemonic,result,Byte_A, arg1 , Byte_B, compare_type, Byte_C, arg2

Byte_A is refering arg1, Byte_B gets the compare_type and Byte_C is refering arg2 and also says if arg2 is a variable or constant.

You maybe have realised , that there are some mnemonic ´s are missing.

As i mentioned this instruction should **occure very often in SETUP.INS,** so i examined the file for this **byte structure** and me found out :

>>>>>> COMPARE mnemonic (actualy 128) !!!
| | |
28,01,32,result_var,Byte_A, arg1 , Byte_B, compare_type, Byte_C, arg2

Byte_A="B"=0x42 means variable_index(word) is following
Byte_B="A"=0x41 means constant (dword) is following

Byte_C="A"=0x41 if comparing with a constant
Byte_C="B"=0x42 if comparing two viriables

result_var = type of word (variable_index)
arg1 = type of word (variable_index)
compare_type = type of dword (1-6)
arg2 = type of word (variable_index) or dword (constant)

Example : lets say we have found the following bytes .

28,01,32, 03,00, **42,** 01,00, **41,** compare_type, **42,** 02,00

This will compare a variable with index 0x0001 and a varible with index 0x002
with the specific compare_type and then stores the result (0/1) of this comparision
into the variable with index 0x003.

Now what we need are the type of comparisions, hmm how should we obtain them ?
Setup is executing this Script, so there is the place we have to search for them !!!
I **W32dasm** Setup.exe and searched for the place where **compare_type gets compared with 1-6** and
i found them at line **0043C89B.**

*** Referenced by a (U)nconditional or (C)onditional Jump at Address:**
|:0043C89F(C)
|
:0043C7B2 8B45F4 mov eax, [ebp-0C]
eax=arg1
:0043C7B5 3945F8 cmp [ebp-08], eax
compare arg2 with arg1
:0043C7B8 0F8E0C000000 jle 0043C7CA
lower-equal? compare_type_1 !!!
:0043C7BE C745FC01000000 mov [ebp-04], 00000001 return result 1 in [ebp-4]
:0043C7C5 E907000000 jmp 0043C7D1 jmp to end

*** Referenced by a (U)nconditional or (C)onditional Jump at Address:**
|:0043C7B8(C)
|
:0043C7CA C745FC00000000 mov [ebp-04], 00000000 return result 1 in [ebp-4]

*** Referenced by a (U)nconditional or (C)onditional Jump at Address:**
|:0043C7C5(U)
|
:0043C7D1 E906010000 jmp 0043C8DC jmp to end

*** Referenced by a (U)nconditional or (C)onditional Jump at Address:**
|:0043C8A9(C)
|
:0043C7D6 8B45F4 mov eax, [ebp-0C]
:0043C7D9 3945F8 cmp [ebp-08], eax

:0043C7DC 0F8D0C000000 jnl 0043C7EE
greater-equal? compare_type_2 !!!
:0043C7E2 C745FC01000000 mov [ebp-04], 00000001
:0043C7E9 E907000000 jmp 0043C7F5

*** Referenced by a (U)nconditional or (C)onditional Jump at Address:**
|:0043C7DC(C)
|
:0043C7EE C745FC00000000 mov [ebp-04], 00000000

*** Referenced by a (U)nconditional or (C)onditional Jump at Address:**
|:0043C7E9(U)
|
:0043C7F5 E9E2000000 jmp 0043C8DC

*** Referenced by a (U)nconditional or (C)onditional Jump at Address:**
|:0043C8B3(C)
|
:0043C7FA 8B45F4 mov eax, [ebp-0C]
:0043C7FD 3945F8 cmp [ebp-08], eax
:0043C800 0F8C0C000000 jl 0043C812
lower? compare_type_3 !!!
:0043C806 C745FC01000000 mov [ebp-04], 00000001
:0043C80D E907000000 jmp 0043C819

*** Referenced by a (U)nconditional or (C)onditional Jump at Address:**
|:0043C800(C)
|
:0043C812 C745FC00000000 mov [ebp-04], 00000000

*** Referenced by a (U)nconditional or (C)onditional Jump at Address:**
|:0043C80D(U)
|
:0043C819 E9BE000000 jmp 0043C8DC

*** Referenced by a (U)nconditional or (C)onditional Jump at Address:**
|:0043C8BD(C)
|
:0043C81E 8B45F4 mov eax, [ebp-0C]
:0043C821 3945F8 cmp [ebp-08], eax
:0043C824 0F8F0C000000 jg 0043C836
greater ? compare_type_4 !!!
:0043C82A C745FC01000000 mov [ebp-04], 00000001
:0043C831 E907000000 jmp 0043C83D

*** Referenced by a (U)nconditional or (C)onditional Jump at Address:**

|:0043C824(C)
|
:0043C836 C745FC00000000 mov [ebp-04], 00000000

*** Referenced by a (U)nconditional or (C)onditional Jump at Address:**
|:0043C831(U)
|
:0043C83D E99A000000 jmp 0043C8DC

*** Referenced by a (U)nconditional or (C)onditional Jump at Address:**
|:0043C8C7(C)
|
:0043C842 8B45F4 mov eax, [ebp-0C]
:0043C845 3945F8 cmp [ebp-08], eax
:0043C848 0F850C000000 jne 0043C85A **not-equal ? compare_type_5 !!!**
:0043C84E C745FC01000000 mov [ebp-04], 00000001
:0043C855 E907000000 jmp 0043C861

*** Referenced by a (U)nconditional or (C)onditional Jump at Address:**
|:0043C848(C)
|
:0043C85A C745FC00000000 mov [ebp-04], 00000000

*** Referenced by a (U)nconditional or (C)onditional Jump at Address:**
|:0043C855(U)
|
:0043C861 E976000000 jmp 0043C8DC

*** Referenced by a (U)nconditional or (C)onditional Jump at Address:**
|:0043C8D1(C)
|
:0043C866 8B45F4 mov eax, [ebp-0C]
:0043C869 3945F8 cmp [ebp-08], eax
:0043C86C 0F840C000000 je 0043C87E
equal ? compare_type_6 !!!
:0043C872 C745FC01000000 mov [ebp-04], 00000001
:0043C879 E907000000 jmp 0043C885

*** Referenced by a (U)nconditional or (C)onditional Jump at Address:**
|:0043C86C(C)
|
:0043C87E C745FC00000000 mov [ebp-04], 00000000

*** Referenced by a (U)nconditional or (C)onditional Jump at Address:**
|:0043C879(U)

```
|
:0043C885 E952000000            jmp 0043C8DC

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0043C8D7(U)
|
:0043C88A C745FC00000000 mov [ebp-04], 00000000
:0043C891 E946000000            jmp 0043C8DC
:0043C896 E941000000            jmp 0043C8DC

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0043C7AD(U)
|
:0043C89B 837DEC01          cmp [ebp-14], 00000001      <------      This
is the entry point of the compare-part
:0043C89F 0F840DFFFFFF          je 0043C7B2                              and
[ebp-14] will be the compare_type !!!
:0043C8A5 837DEC02          cmp [ebp-14], 00000002
:0043C8A9 0F8427FFFFFF          je 0043C7D6
:0043C8AF 837DEC03          cmp [ebp-14], 00000003
:0043C8B3 0F8441FFFFFF          je 0043C7FA
:0043C8B9 837DEC04          cmp [ebp-14], 00000004
:0043C8BD 0F845BFFFFFF          je 0043C81E
:0043C8C3 837DEC05          cmp [ebp-14], 00000005
:0043C8C7 0F8475FFFFFF          je 0043C842
:0043C8CD 837DEC06          cmp [ebp-14], 00000006
:0043C8D1 0F848FFFFFFF          je 0043C866
:0043C8D7 E9AEFFFFFF        jmp 0043C88A
```

Ok, let us summerize the compare_types :

Compare Type (dword):	Type:	math.exp.:	Coresponding jmp:
1	LOWER-EQUAL	<=	JLE
2	GREATER-EQUAL	>=	JGE
3	LOWER	<	JL
4	GREATER	>	JG
5	NOT-EQUAL	!=	JNE
	EQUAL	=	JE

MESSAGEBOX byte structure :

2A,0,61,length(word),text will show a **messagebox with the specific text** !!!

Since the compare part of an IF-THEN instruction is what we really need for our interest
you could now go directly to the START further below !!!

Otherwise learn more about other instructions and how they are build up =)

The structure of a compiled IF-THEN instruction may look like this :

COMPARE , BRANCH_TO location IF !(result - arg_x)

(result - arg_x) will be zero if they are equal else it will be not zero.
The result comes from the comparision and arg_x can be a varible or a constant.

Now we come to the **IF-THEN** byte structure :

COMPARE-structure,BRANCH_TO_mnemonic,l_index, SUB, Byte_A,result,Byte_C,arg_x

BRANCH_TO_mnemonic = 22,0,70
SUB = 95 (in an IF-THEN instruction !!!)

Byte_A="B"=0x42 result of comparision will allways be a variable_index
Byte_C="A"=0x41 arg_x allways will be a constant in an IF-THEN instruction !!!

l_index = type of word (index)
result = type of word (variable_index)
arg_x = will be a dword (constant) =0x00000000 in an IF-THEN instruction !!!

The branch location will be an offset into the script

and it is calculated like this :

location = dword [l_index* 6 + Branch-Table-Offset+2]
Location-Table-Offset = Offset "_EWQ"
;in this script it was 14546 !!!

Just search for "_EWQ" and you will find it (Its linked at the end of the script)!!!

GOTO byte stucture :

2C,00,70,l_index

There are more instructions i have decrypted, but we dont need them for this tutorial.
Its quite easy to write a Decompiler with this information and if you have found out
the location where Setup is executing the script then its not that hard to see what
it is doing depending on the mnemonic, but thats another story and this tutorial
is damn big enough !!!

Now we can try out our first Script-Cracking attempt =)...

START:

(CD-CHECK) First think about how this check was written with the Script instructions !!
The easiest way may be done like this :
(Assume: Return_of_INSHELP=0/1 (BAD/GOOD) !!!)

```
arg1=CALL(INSHELP,CD-CHECK)
IF arg1 = 0 THEN MESSAGEBOX "Setup must be run from the original CD":END
        ELSE RETURN(1)
```

or this...

```
arg1=CALL(INSHELP,CD-CHECK)
IF arg1 != 0 THEN RETURN(1)
        ELSE MESSAGEBOX "Setup must be run from the original CD":RETURN(0)
```

After compiling this pice of code, the bytes would look

like this :

28,01,32,"B",arg1 (word),"A",6 (dword),"A",0 (dword),...,2A,0,61,27 (word),"**Setup must be..."**

or this...

28,01,32,"B",arg1 (word),"A",5 (dword),"A",0 (dword),...,2A,0,61,27 (word),"**Setup must be..."**

_____I have retrieved this part of **SETUP.INS** for you....**(Offset 8D70)**

arg1_Variable_index (word) < < > **compare_type_5 !!!**
result_Variable_index (word) <<<| | |
IF mnemonic <<<<< | | | | |
| | | | | | | |

00008D70 9A FF 42 2D 00 **28 01 32** 2D 00 **42** 9B FF **41** 05 00 ..B-.**(.2**-.**B**..**A**..
00008D80 00 00 **41** *00 00 00 00* 22 00 70 53 01 95 42 2D 00 ..**A**....".pS..B-.
00008D90 41 00 00 00 00 **2A 00 61** *27 00* 53 65 74 75 70 20 A....*.**a**'.Setup
00008DA0 6D 75 73 74 20 62 65 20 72 75 6E 20 66 72 6F 6D must be run from
00008DB0 20 74 68 65 20 6F 72 69 67 69 6E 61 6C 20 43 44 the original CD

We see that its **compare_type_5 (!=),** so we just have to **change it into 6 (=)**
at Offset 8D7E to defeat this CD-CHECK, isnt it easy !!!

BTW, if you are using the patched INSHELP, this change will reverse
the result from INSHELP, so dont use the patched INSHELP !!!!!!

(CD-KEY) I **seeked SETUP.INS** for the bytes **2A,0,61** and found the **CD-KEY** notification part
at **Offset 8FD0**

00008FD0 42 00 00 **28 01 32** 2E 00 **42** 2D 00 **41** 02 00 00 00 B..

(.2..**B**-.**A**.... KEY-length !<0 ?

00008FE0 **41** *00 00 00 00* 22 00 70 5A 01 95 42 2E 00 41 00 **A**....".pZ.•B..A.

00008FF0 00 00 00 21 00 32 99 FF 41 01 00 00 00 2C 00 70 ...!.2™ÿA....,.p

00009000 5C 01 00 00 01 00 3A 00 41 00 00 00 00 00 00 00 \.....:.A.......

00009010 00 00 00 01 00 2C 00 70 59 01 00 00 0B 00 19 01 .....,.pY.......

00009020 32 97 FF 42 97 FF 41 01 00 00 00 B4 00 80 6D 00 2—ÿB—ÿA....´.€m.

00009030 42 9A FF 21 00 32 2D 00 42 00 00 21 00 32 9B FF Bšÿ!.2-.B..!.2›ÿ

00009040 42 2D 00 **28 01 32** 2D 00 **42** 9B FF **41** 05 00 00 00 B-.**(.2**-.**B**›ÿ**A**.... **KEY-CHECK here !!**

00009050 **41** *00 00 00 00* 22 00 70 61 01 95 42 2D 00 41 00 **A**....".pa.•B-.A.

00009060 00 00 00 **28 01 32** 2E 00 **42** 97 FF **41** 01 00 00 00 ...**(.2**..**B**—ÿ**A**.... Tries <= 6 times ?

00009070 **41** *06 00 00 00* 22 00 70 5E 01 95 42 2E 00 41 00 **A**....".p^.•B..A. -- if not display this and

00009080 00 00 00 3A 00 41 00 00 00 00 **2A 00 61** *2B 00* 50 ...:.A....*.**a**+.P |End.

00009090 6C 65 61 73 65 20 65 6E 74 65 72 20 79 6F 75 72 lease enter your |

000090A0 20 43 44 2D 4B 65 79 20 74 6F 20 63 6F 6E 74 69 CD-Key to conti |

000090B0 6E 75 65 20 73 65 74 75 70 2E 41 01 00 FF FF 2C nue setup.A..ÿÿ, |

000090C0 00 70 60 01 00 00 05 00 **2A 00 61** *38 00* 59 6F 75 .p`.....*.**a**8.You <---|

000090D0 20 6D 75 73 74 20 65 6E 74 65 72 20 74 68 65 20 must enter the

000090E0 70 72 6F 70 65 72 20 43 44 2D 4B 65 79 20 74 6F proper CD-Key to

000090F0 20 69 6E 73 74 61 6C 6C 20 74 68 65 20 70 72 6F install the pro

00009100 64 75 63 74 2E 41 03 00 FF FF B3 00 62 9B FF 21 duct.A..ÿÿ³.b›ÿ!

_____Change **Offset(904C) to 6** and this KEY-Protection will be history,hehe !!!

You can now type in anything you want and it will be valid.

BTW, if you also change Offset(8FDC) to 4 it will also accept an emty KEY !!!

(Serial) Ok, now we will see if this Script-Cracking will defeat this damn Serial-Check !

This **Check dont use INSHELP** or any other DLL. It **strickly uses the Script !!**

This means we cant espect a simple compare_type_5 or 6 before its messagebox !

There is no other way than using our beloved **SoftICE a bit !**

To see what Setup is comparing when he checks the Serial we must first **type in**

User-Name,(Company) and a Serial. I used **"123456789" as Serial**.

Now invoke SoftICE with its hotkey (Strg+D) and make sure you are in Setup´s

Adress-Context ("Setup" in the right, bottom egde) , otherwise leave SoftICE

and invoke it again until you are there. If you are in the Kernel or User API just

* trace back with "F12" until you are in Setup !!!
* Set **BPX on 0043C89B** "bpx 0043C89B" the **entry point of the compare part** !!!

Now leave SoftICE and press **NEXT-> .**

SoftICE will pop up at 0043C89B several times and Setup will perform comparisions !

Here is my history of the comparisions :

	Comparisions:			***Compare_type:***	
(1)	0	!=	1	5	Not important
(2)	0	>=	3	2	Not important (chr-position counter?)
(3)	9	<=	0	1	This looks like our Serial-length !!!
(4)	61	>	31	4	Well, its the first char of our Serial !!!
(5)	7A	<	31	3	and it setup is checking if it is
(6)	41	>	31	4	between "a"-"z","A"-"Z","0"-"9"
(7)	5A	<	31	3	
(8)	30	>	31	4	
(9)	39	<	31	3	

(10) 3 <= 0 1 Not important(chr-position counter?)
BREAK.

It seems that it checks every char from our serial seperately.
Since our Serial is not valid lets fake this check !!!
(3) This really looks like a char position pointer, which is compared to our serial length.
We have to reverse this compare to get out of this check !!!
Here is the hex dump...

00006240 00 **28 01 32** 2E 00 **42** 2D 00 **41** 02 00 00 00 **41** *00* .
(.2..**B**-.**A**....**A**. This only checks if our Serial
00006250 *00 00 00* 22 00 70 D7 00 95 42 2E 00 41 00 00
00 ...".p×.•B..A... is emty !!!
00006260 00 B5 00 80 66 00 70 DB 00 62 26 00 21 00 32 2D .µ.
€f.pÛ.b&.!.2-
00006270 00 42 00 00 22 00 70 D4 00 95 42 2D 00 41 00
00 .B..".pÔ.•B-.A..
00006280 00 00 21 00 32 9B FF 41 01 00 00 00 2C 00 70
D6 ..!.2›ÿA....,.pÖ
00006290 00 00 00 02 00 3A 00 41 00 00 00 00 **2A 00 61** *37* .....:.A....*.a7
000062A0 *00* 50 6C 65 61 73 65 20 65 6E 74 65 72 20 79 6F .Please enter
yo
000062B0 75 72 20 73 65 72 69 61 6C 20 6E 75 6D 62 65 72 ur serial
number
000062C0 20 74 6F 20 63 6F 6E 74 69 6E 75 65 20 77 69 74 to continue
wit
000062D0 68 20 73 65 74 75 70 2E 41 01 00 FF FF 00 00 00 h
setup.A..ÿÿ...
000062E0 00 00 00 01 00 2C 00 70 D9 00 00 00 06 00 2F 00 .....,.pÙ...../.
000062F0 62 24 00 21 00 32 2D 00 42 00 00 **28 01 32** 2E 00 b$.!.2-.B**..**
(.2..
00006300 **42** 2D 00 **41** 03 00 00 00 **41** 00 00 00 00 22 00 70
B-.**A**....**A**....".p This checks if our Name
00006310 D8 00 95 42 2E 00 41 00 00 00 00 3A 00 41 00 00
Ø.•B..A....:.A.. is emty !!!
00006320 00 00 **2A 00 61** *2E 00* 50 6C 65 61 73 65 20 65 6E ..*.a..Please
en
00006330 74 65 72 20 79 6F 75 72 20 6E 61 6D 65 20 74 6F ter your
name to
00006340 20 63 6F 6E 74 69 6E 75 65 20 77 69 74 68 20 73 continue
with s
00006350 65 74 75 70 2E 41 01 00 FF FF 00 00 00 00 00 00 etup.A..ÿÿ.....
00006360 01 00 2C 00 70 D3 00 00 00 02 00 01 00 41 32 00 ..,.pÓ.....A2.

00006370 00 00 B8 00 00 00 06 00 B6 00 10 00 01 00 02 02 ..¸.....¶.....
00006380 00 00 05 00 00 00 2F 00 62 9B FF 21 00 32 2D 00 ....../.b›ÿ!.2-.
00006390 42 00 00 21 00 32 9A FF 42 2D 00 21 00 32 99 FF
B..!.2šÿB-.!.2™ÿ
000063A0 41 00 00 00 00 21 00 32 98 FF 41 00 00 00 00 00 A....!.2˜ÿA.....
000063B0 00 10 00 29 01 **28 01 32** 2D 00 **42** 99 FF **41** 01 00 ...).
(.2-.**B**™ÿ**A**.. **(3) obviously !!!**
000063C0 00 00 **42** *9A FF* 22 00 70 E5 00 95 42 2D 00 41
00 ..**B**šÿ".på.•B-.A.
000063D0 00 00 00 7A 00 32 97 FF 52 9B FF 42 99 FF **28 01** ...z.2—
ÿR›ÿB™ÿ**(.**
000063E0 **32** 2D 00 **42** 97 FF **41** 04 00 00 00 **41** *61 00 00 00* **2**-.**B**—
ÿ**A**....**Aa**... **(4)**
000063F0 **28 01 32** 2E 00 **42** 97 FF **41** 03 00 00 00 **41** *7A 00* **(.2**..**B**—
ÿ**A**....**Az**. **(5)**
00006400 *00 00* 27 01 32 2F 00 42 2D 00 42 2E 00 **28 01 32** ..'.2/.B-.B**..**
(.2
00006410 2D 00 **42** 97 FF **41** 04 00 00 00 **41** *41 00 00 00* **28** -.**B**—
ÿ**A**....**AA**...**(** **(6)**
00006420 **01 32** 2E 00 **42** 97 FF **41** 03 00 00 00 **41** *5A 00 00* .**2**..**B**—
ÿ**A**....**AZ**.. **(7)**
00006430 *00* 27 01 32 30 00 42 2D 00 42 2E 00 26 01 32
2D .'.20.B-.B..&.2-
00006440 00 42 2F 00 42 30 00 22 00 70 DF 00 95 42 2D
00 .B/.B0.".pß.•B-.
00006450 41 00 00 00 00 28 01 32 2E 00 42 99 FF 41 02 00 A....
(.2..B™ÿA..
00006460 00 00 41 03 00 00 00 22 00 70 DD 00 95 42 2E
00 ..A...."."pÝ.•B..
00006470 41 00 00 00 00 2F 01 B7 00 41 00 00 00 00 00 00 A..../.·.A.....
00006480 00 00 00 00 01 00 19 01 32 98 FF 42 98 FF 41 01 ....2˜ÿB˜ÿA.
00006490 00 00 00 00 00 00 00 00 00 08 00 **28 01 32** 2D 00 ....**(.2**-.
000064A0 **42** 97 FF **41** 04 00 00 00 **41** *30 00 00 00* **28 01 32** **B**—
ÿ**A**....**A0...(.2** **(8)**
000064B0 2E 00 **42** 97 FF **41** 03 00 00 00 **41** *39 00 00 00* 27 ..**B**—
ÿ**A**....**A9**...' **(9)**
000064C0 01 32 2F 00 42 2D 00 42 2E 00 22 00 70 E3 00
95 .2/.B-.B..".pã.•
000064D0 42 2F 00 41 00 00 00 00 28 01 32 2D 00 42 99 FF B/.A....
(.2-.B™ÿ
000064E0 41 01 00 00 00 41 03 00 00 00 22 00 70 E1 00 95 A....A...."."pá.•
000064F0 42 2D 00 41 00 00 00 00 2F 01 B7 00 41 00 00 00 B-.A..../.·.A...
00006500 00 00 00 00 00 00 00 01 00 19 01 32 98 FF 42 98 ....2˜ÿB˜
00006510 FF 41 01 00 00 00 00 00 00 00 00 00 02 00 19 01 ÿA.............
00006520 32 99 FF 42 99 FF 41 01 00 00 00 2C 00 70 DC 00
2™ÿB™ÿA....,.pÜ.

00006530 00 00 04 00 **28 01 32** 2D 00 **42** 98 FF **41** 06 00 00 ....
(.2-.**B**˜ÿ**A**... **(11) The Final check !!!**
00006540 00 **41** *0D 00 00 00* 22 00 70 E6 00 95 42 2D 00
41 .**A**....".pæ.•B-.A
00006550 00 00 00 00 2F 01 B7 00 41 00 00 00 00 00 00 00 ..../.·.A.....

If you have change the byte at **(3) offset (63BE) to 2** you will get to the final check.

(11) Setup will finally check if 13 chars of your serial were valid !!!

Just change byte at **(11) offset (653D) to 5** and this Serial check will be defeated !!!

Summarize:

You see now that Script Cracking is much easier than the first approach !!!

We only have to search for MessageBoxes and analyze the script.

At all we only have to edit (patch) the script and thats all =)

If i find out more instructions then you even will be able to get a valid Serial(Keymaker) !!!

A Decompiler will follow anyway. Its only a question of time when it will

be written so watch out for it,hehe.

F) ADDON

This part will disscribe the most common InstallSHIELD Installation.

If **Setup.exe (InstallSHIELD 2.x)** is a **16 Bit** executeable, then its called

The Installation launcher.

It **needs a support file** called **_inst32i.ex_** to install under a **win32 OS.**

This Installation is a bit different from the one i have cracked in this Tutorial.

inst32i.ex is compressed but not with icompx, but it dont matter !!!

and it contains the following files :

INSTALL.EXE
_INS0432._MP
LZWSERV.EXE
_INZ0432._MP
WUTL95i.DLL
_WUTL95.DLL
BOOT16.EXE
_INJ0432._MP

You can retrieve these File-Names at the beginning of _inst32i.ex_ by yourself.
Setup will do the initialization and then it **uncompresses _inst32i.ex_** into your
Windows-Temp (C:\Windows\Temp).
When ya start the Installation you will see the following in Windows\Temp:

<_ISTMP0.DIR> DIR This dir will be created by _ins0432._mp !!!
_INS0432._MP 659 KB This is exactly Setup.exe from this Tutorial !!!
_INZ0432._MP 20,1 KB This is LZWSERV.EXE (doing the de-compress.)
_WUTIL95.DLL 36,0 KB A win95 support file

_ISTMP0.DIR content :

_SETUP.LIB 151 KB This is exactly the same compressed lib file !!!
1f8584.DLL 89,0 KB Support DLL
INSHELP.DLL 23,5 KB Yup, da same DLL !!!
UNINST.EXE 292 KB Also da same one

You see now that there are the same files, but only **renamed** , thats all !!!
Copy and rename them if you wanna work with these files.

G) WIN32.HLP

These Dissciptions comes from win32.hlp

GetDriveType:

The GetDriveType function determines whether a disk drive is a removable, fixed, CD-ROM, RAM disk, or network drive.

UINT GetDriveType(

LPCTSTR lpRootPathName // address of root path

);

Parameters

lpRootPathName

Points to a null-terminated string that specifies the root directory of the disk to return information about. If lpRootPathName is NULL, the function uses the root of the current directory.

Return Value

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.
2	The drive can be removed from the drive.
3	The disk cannot be removed from the drive.
4	The drive is a remote (network) drive.
5	The drive is a CD-ROM drive.
6	The drive is a RAM disk.

GetVolumeInformation:

The GetVolumeInformation function returns information about a file system and volume whose root directory is specified.

BOOL GetVolumeInformation(

LPCTSTR lpRootPathName, // address of root directory of the file system
LPTSTR *lpVolumeNameBuffer*, // address of name of the volume
DWORD nVolumeNameSize, // length of lpVolumeNameBuffer
LPDWORD lpVolumeSerialNumber, // address of volume serial number
LPDWORD lpMaximumComponentLength, // address of system's maximum filename length
LPDWORD lpFileSystemFlags, // address of file system flags
LPTSTR *lpFileSystemNameBuffer*, // address of name of file system
DWORD nFileSystemNameSize // length of lpFileSystemNameBuffer
);

Parameters

lpRootPathName

Points to a string that contains the root directory of the volume to be described. If this parameter is NULL, the root of the current directory is used.

lpVolumeNameBuffer

Points to a buffer that receives the name of the specified volume.

nVolumeNameSize

Specifies the length, in characters, of the volume name buffer. This parameter is ignored if the volume name buffer is not supplied.

lpVolumeSerialNumber

Points to a variable that receives the volume serial number. This parameter can be NULL if the serial number is not required.

lpMaximumComponentLength

Points to a doubleword value that receives the maximum

length, in characters, of a filename component supported by the specified file system. A filename component is that portion of a filename between backslashes.

The value stored in variable pointed to by *lpMaximumComponentLength is used to indicate that long names are supported by the specified file system. For example, for a FAT file system supporting long names, the function stores the value 255, rather than the previous 8.3 indicator. Long names can also be supported on systems that use the NTFS and HPFS file systems.

lpFileSystemFlags

Points to a doubleword that receives flags associated with the specified file system. This parameter can be any combination of the following flags, with one exception: FS_FILE_COMPRESSION and FS_VOL_IS_COMPRESSED are mutually exclusive.

Value Meaning

FS_CASE_IS_PRESERVED If this flag is set, the file system preserves the case of filenames when it places a name on disk.

FS_CASE_SENSITIVE If this flag is set, the file system supports case-sensitive filenames.

FS_UNICODE_STORED_ON_DISK If this flag is set, the file system supports Unicode in filenames as they appear on disk.

FS_PERSISTENT_ACLS If this flag is set, the file system preserves and enforces ACLs. For example, NTFS preserves and enforces ACLs, HPFS and FAT do not.

FS_FILE_COMPRESSION The file system supports file-based compression.

FS_VOL_IS_COMPRESSED The specified volume is a compressed volume; for example, a DoubleSpace volume.

lpFileSystemNameBuffer

Points to a buffer that receives the name of the file system (such as FAT, HPFS, or NTFS).

nFileSystemNameSize

Specifies the length, in characters, of the file system name buffer. This parameter is ignored if the file system name buffer is not supplied.

Return Value

If all the requested information is retrieved, the return value is TRUE; otherwise, it is FALSE. To get extended error information, call GetLastError.

Remarks

The FS_VOL_IS_COMPRESSED flag is the only indicator of volume-based compression. The file system name is not altered to indicate compression. This flag comes back set on a DoubleSpace volume, for example. With volume-based compression, an entire volume is either compressed or not compressed.

The FS_FILE_COMPRESSION flag indicates whether a file system supports file-based compression. With file-based compression, individual files can be compressed or not compressed.

The FS_FILE_COMPRESSION and FS_VOL_IS_COMPRESSED flags are mutually exclusive; both bits cannot come back set.

The maximum component length value, stored in the DWORD variable pointed to by lpMaximumComponentLength, is the only indicator that a volume supports longer-than-normal FAT (or other file system) file names. The file system name is not altered to indicate support for long file names.

The GetCompressedFileSize function obtains the compressed size of a file. The GetFileAttributes function can determine whether an individual file is compressed.

GetWindowText:

The GetWindowText function copies the text of the specified window's title bar (if it has one) into a buffer. If the specified window is a control, the text of the control is copied.

int GetWindowText(

HWND hWnd, // handle of window or control with text
LPTSTR lpString, // address of buffer for text
int nMaxCount // maximum number of characters to copy

);

Parameters

hWnd

Identifies the window or control containing the text.

lpString

Points to the buffer that will receive the text.

nMaxCount

Specifies the maximum number of characters to copy to the buffer. If the text exceeds this limit, it is truncated.

Return Value

If the function succeeds, the **return value is the length**, in characters, of the copied string, not including the terminating null character. If the window has no title bar or text, if the title bar is empty, or if the window or control handle is invalid, the return value is zero. To get extended error information, call GetLastError.

This function cannot retrieve the text of an edit control in another application.

Remarks

This function causes a WM_GETTEXT message to be sent to the specified window or control.

This function cannot retrieve the text of an edit control in another application.

H) LAST WORDS

Yeah, you made it =)

This is the end of this tutorial and i hope i could teach you something , more or less.

If you have any **questions, suggestions** or just wanna gimme some **feedback**, then just email me !!!

Also plz inform me if you have find out any error - iam only

a human being =)

This Tutrorial was first written under note-pad, but it got just to big, so that i had to
continue writting it with WordPad. I hope you dont mind it ;)
The next Tutorial (natz-2) will be in html and i dont exactly know what it will discuss yet,
so just watch out for it !!!

NaTzGUL/REVOLT
natzgul@hotmail.com

I) GREETINGS

Groups:

REVOLT, #CRACKING, UCF, PC97, HERITAGE,CRC32
#CRACKING4NEWBIES, CORE, RZR, PWA, XF, DEV etc.

PERSONAL:

CoPhiber, Spanky, Doc-Man, Korak, lgb, DDensity, Krazy_N, delusion, riches, Laamaah, Darkrat, wiesel, DirHauge, GnoStiC, JosephCo, niabi, Voxel,TeRaPhY, NiTR8, Marlman, THE_OWL, razzia, K_LeCTeR, FaNt0m, zz187, HP, Johnastig, StarFury, Hero, +ORC, +Crackers, Fravia, LordCaligo, BASSMATIC, j0b ,xoanon, EDISON etc.

-EOF-