### **Taking Back Netcat**

Ever since Symantec added Netcat's signature to their virus database, there have been repeated outcries against the detection of Netcat as a "Hack Tool"; while Netcat is a very useful networking utility, like many useful tools, it can be used for both good an evil. This paper will show how to locate the signature used to identify Netcat, and modify it so that the executable no longer matches Symantec's signature, without interfering with any of the program's functionality. This is not an act of defiance against Symantec, but rather an exercise in identifying and modifying sections of code (aka, signatures) that are used by anti virus programs to identify malicious code; the tools and techniques used here can be applied to any program that is marked as malicious by AV applications.

### **Changing File Signatures**

While there are some easy ways of changing the signature of a program (packers, encryptors, etc), they may not always be viable options for those wishing to bypass anti virus applications. Additionally, it would be easy for anti virus companies to run a program such as Netcat through a few popular packers/encryptors and add signatures for the resulting binaries to their virus databases as well. As such, we will be manually examining and editing the Netcat program in order to create a custom "version" of the Netcat utility.

There are two basic ways of editing a program: either by changing the original source code and recompiling it, or by using a hex editor to change various bytes in the compiled binary; in either case, in order to invalidate an AV signature, you must know what parts of the code are used to identify the signature. Thus, regardless of the availability of source code, one still needs to know how to find an AV signature in order to thwart it. While Netcat source code is available, we will be using the hex editor approach; this allows us the opportunity to see how closed-source programs are changed in order to slip past anti virus protection (i.e., when creating new virus strains, modifying shellcode, etc).

### **Tools Needed**

The following tools will be used in this paper. Note that only the first three tools are required. If you have a preference for different programs, feel free to use them instead of the listed applications, just as long as they can provide the same basic functionality:

- 1. Netcat for Windows v1.11 (<u>http://www.vulnwatch.org/netcat</u>)
- 2. Norton Anti-Virus 2006 (http://www.symantecstore.com)
- 3. HexWorkshop v4.23 (http://www.bpsoft.com/downloads)
- 4. Olly Debugger (<u>http://www.ollydbg.de</u>)

In addition, a familiarity with disassembly, ASM, and PE editing will help you to further obfuscate an AV signature, but they are not necessarily prerequisites for this paper. You will, however, be able to follow along much better if you have at least some understanding of ASM, hex editing, and PE headers.

## **Identifying a Signature**

Once your AV agent has identified a file as a virus, trojan, worm, or other malicious program, you need to locate the section of the file that is used by the AV application to identify the program as malicious. The easiest way to do this is through bracketing (aka, brute force, trial and error, or whatever you'd like to call it). If you open the program in a hex editor and replace, say, the second half of the file with a string of 0's, and the AV application no longer identifies the file as a malicious program, then the AV signature (or at least part of it) is located in the second half of the file. Conversely, if it is still identified as a malicious program, then the signature is located in the first half of the file. This halving technique can be used again and again to narrow down the location of the

signature, until you can positively identify the exact location and length of the signature.

To do this, first open up Netcat with Hex Workshop (right click nc.exe and select 'Hex edit with Hex Workshop'). If you scroll to the bottom of the hex dump, you see the last byte is located at offset EFFF. Divide EFFF in half and you get 77FF; open up a goto box (Ctl+G) and go to the offset 77FF from the beginning of the file:

Goto	
<u>O</u> ffset: 000077f <b>f</b> ○ <u>D</u> ec ● He <u>x</u>	From Where • <u>B</u> eginning of File • Current <u>P</u> osition
Go Cancel Help Right click in your docume	C End of File (back from)

Select everything from 77FF to the end of the file (EFFF), right click, select 'Fill', and fill the selected section with 0s:

Fill Bytes					
Number of bytes: 1423	O Hex € Dec	ОК			
Fill with the following hex byte:		Cancel			
Fills the Current Selection					

Save your changes; when prompted to make a backup, say yes. Now, have Norton scan nc.exe again, and it should still detect the file as a "Hack Tool":

😣 Norton AntiVirus	- Scan			
Manual Repa	ir - Risks			more info
1 Remaining Risk				
These risks have not to take the recommer			the Symantec website, selec	t a risk name, Click OK
Name	Risk	Warning	Details	Action
Hacktool.NetCat	Low		<u>Hack tool</u>	Exclude*
Removed spywar	e and adware can b	e retrieved using <u>No</u>	rton AntiVirus Quarantine an	d Restore
			OK	Cancel

This indicates that the signature is somewhere in the first half of the binary file. Click Cancel on the Norton repair window, delete the nc.exe file and rename nc.BAK (the backup made by Hex Workshop) to nc.exe. Open up nc.exe with Hex Workshop and let's examine the first half of the hex dump. You'll notice a few text strings such as "This program cannot be run in DOS mode", ".text", ".data" and ".rdata", followed by several hundred bytes of 0s. This is the PE header section of the program; the actual program instructions begin immediately after the section of 0s at offset 1000:

Hex Worksho File Edit D			ls Winde	ow Help															_ 8 _ 8
☞ 🖬 🚭 🐰 🖻 🖻 그 으 📎 🥳 📵 🔚 🔞 B S L Q F D 😥 🗉 🛛 (ト チ → →)																			
≒~ ‹‹ ›	> 🎸 🏹	2 🗠 2	<u>} ^  </u>	<b>&amp;</b>	<del>†∕</del> - +	- +	1 %	<li>[x] [x]</li>	A† a	↓ąA	<b>1</b>	R	<b>%</b>	<b>H</b>					
00000F3C	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000						-
00000F56		0000		0000		0000	0000		0000	0000	0000	0000							
00000F70	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000						
00000F8A	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000						
00000FA4	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000						
OOOOOFBE	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000						
00000FD8	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000						
00000FF2			0000	0000	0000	0000	0000	83EC	5453	8B1D	6CB0	4000	5633				.TS	1.@.V3	
0000100C			6A02	8D4C	2464	5107	4424	2844	0000	0089	7424	2C89						t\$,.t\$	
00001026			3089		4489		4089	7424			3866							8f.t\$X	
00001040		245C	6689		5AC7	4424	5401	0100		4424	6089							`. \$d.	
0000105A		57FF	D350		68B0		8BOD	COE7	4000	8D54								\$.R.D\$	
00001074		5656	566A		5651			B040		C074	178B		OC8B					T\$	
)000108E		0852	FF15	60B0		8BC6	5E5B	83C4	54C3	5656								VVVj.h	
000010A8			FF15		4000	50E8	2E8F	0000		0050		B140						'h@	
00001002		0000	83C4		C65E	5883	C454			CCCC			cccc						
		CCCC	81EC			A190					8C02		8B17			-			
000010F6		6A00		248C							008D							L\$.Q	
00001110				0085														@.S.D\$	
0000112A																		L\$ Q	
)0001144 )000115E		D58B				0D40												u A:r	
00001156																			
	DAUD	41.00	0400	3000	0424	5400	0000	2021	FF 15	0401	4000	0300	7626		1.5		<u>v</u>	e	_
nc.exe																			
offset: 4096 [(	)×000010	00]									are Result	s					All		•

We can safely eliminate the PE header, as it would not be used as part of a virus signature. Thus, we know that the signature must be somewhere between offsets 1000 and 77FF; this can be confirmed by zeroing out all the file contents between these two addresses as we did with the second half of the file, and running a virus scan on nc.exe again:

😵 Norton AntiVirus - Scan	
Scan Complete	more info
No viruses, spyware, or oth	er risks were found.
Total files scanned	0
Viruses, spyware and other security risks	
Detected	0
Resolved	0
Remaining	0
	Finish

We can continue this process of elimination by systematically modifying sections of code between 1000 and 77FF and testing the resulting file against Norton. If the file is still detected as the Netcat "Hack Tool", then we know that the section that was modified was not used as part of the signature; if it is not detected, then we know that the modified section was used as part of the signature. Of course, it is important to delete the modified file after each modification and make any new changes to a copy of the original file (just as we did in the previous example), or else your results may be skewed and you will have a non-functioning program to boot. For brevity, I will simply list the sections that I zeroed out and the results that each modification had on the AV detection:

Action Taken	Result
Zero out bytes 1000 through 29FE	Not detected
Zero out bytes 1000 through 1CF7	Not detected
Zero out bytes 1734 through 1CF7	Detected
Zero out bytes 12AC through 1734	Detected
Zero out bytes 1000 through 111C	Not detected
Zero out bytes 1000 through 10CC	Not detected
Zero out byte 1130	Not detected
Zero out byte 1220	Detected
Zero out byte 120C	Detected
Zero out byte 11F8	Not detected
Zero out byte 1210	Detected
Zero out byte 120F	Detected
Zero out byte 1203	Detected
Zero out byte 1202	Detected
Zero out byte 11FE	Not detected
Zero out byte 1201	Detected
Change byte 1200 from 0 to 1	Detected
Zero out byte 11FF	Not detected <this end="" is="" of="" signature<="" td="" the=""></this>

Now that we've found the end of the signature, we need to identify where it begins:

Action Taken	Result
Zero out byte 10B8	Not detected
Zero out byte 1040	Not detected
Zero out byte 0FFF	Detected
Zero out byte 1000	Not detected <beginning of="" signature<="" td="" the=""></beginning>

As you can see, the code located between offsets 1000 and 11FF are used by Norton as a signature to uniquely identify this program as Netcat. Modifying any given byte in between these addresses will

cause the signature in Norton's database to not match the code in the program. The problem is, if we modify the wrong byte, or modify it the wrong way, the program won't work anymore. The easiest way to identify where and how to change the program code is to open it up in a disassembler/debugger and analyze the resulting assembly code; for this we will use OllyDbg. Open up the original copy of nc.exe in Olly, and scroll up to the top of the code window. Notice that the code starts at offset 1000, and that the hex dump of the code located there matches the hex dump at offset 1000 in Hex Workshop. This confirms our previous assumption that this offset was the beginning of the actual program instructions:

00401000	t\$ 83EC 54	SUB ESP,54	
00401003	. 53	PUSH EBX	
00401004	. 8B1D 6CB04000	MOV EBX,DWORD PTR DS:[<&KERNEL32.GetCur:	kernel32.GetCurrentProcess
0040100A	. 56	PUSH ESI	
0040100B	. 33F6	XOR ESI,ESI	
0040100D	. 56	PUSH ESI	COptions => 0
0040100E	. 6A 01	PUSH 1	Inheritable = TRUE
00401010	. 6A 02	PUSH 2	Access = 2
00401012	. 8D4C24 64	LEA ECX,DWORD PTR SS:[ESP+64]	
	<b>E1</b>		phTarget
00401017	. ČŽ4424 28 440	MOU DWORD PTR SS: [ESP+28],44 MOU DWORD PTR SS: [ESP+2C],ESI MOU DWORD PTR SS: [ESP+21,ESI MOU DWORD PTR SS: [ESP+30],ESI MOU DWORD PTR SS: [ESP+40],ESI MOU DWORD PTR SS: [ESP+40],ESI MOU DWORD PTR SS: [ESP+40],ESI	prior Beo
	997424 20 440	MOU DWORD PTR SS [ESP+201 EST	
00401023	997424 34	MOU DWORD PTR SS [ESP+341 EST	
00401027	897424 30	MOU DWORD PTR SS [ESP+30] EST	
0040102B	997424 44	MOU DWORD PTP SS FEST	
0040102F	997424 40	MOULDWORD PTR 33.123P+441,231	
00401033	· 07/727 70	MOU DWOND DTD CC.FECD_201 ECT	
00401037	007424 30	MOULDWORD FIR 33:LESF+3CJ,ESI	
0040103B	- 071424 30 - 22.007434 EQ	MOU WORD PTR SSILESFTSSI,ESI	
00401040	. 00:07/424 00 007404 EC	MOU DWORD DID CC.[ECD.EC] ECI	
00401044	· 07(424 SC 66.007434 ED	MOU WORD FIR SOLLESFTSCI,ESI	
00401049	- 00:07/424 OH	MOU DWORD FIR SSILESFISHI,SI	
00401051	. 074424 54 010	MOU DWORD FIR 33:LE3F7343,101	
00401051	. 894424 60	MOU DWORD FIR SSILESF+601,EHA	
00401059	. 897624 64 EEDO	MOV DWORD PTR SS: LESP+2C1, ESI MOV DWORD PTR SS: LESP+341, ESI MOV DWORD PTR SS: LESP+341, ESI MOV DWORD PTR SS: LESP+441, ESI MOV DWORD PTR SS: LESP+441, ESI MOV DWORD PTR SS: LESP+381, ESI MOV DWORD PTR SS: LESP+581, SI MOV WORD PTR SS: LESP+521, ESI MOV WORD PTR SS: LESP+541, I01 MOV DWORD PTR SS: LESP+541, I01 MOV DWORD PTR SS: LESP+541, EDI CALL EBX	CGetCurrentProcess
00401059 0040105B		CALL EBX PUSH EAX	
0040105C	. 50 . 57	PUSH EDI	hTargetProcess hSource
0040105D	. FFD3	CALL EBX	CGetCurrentProcess
0040105F		PUSH EAX	hSourceProcess
	. 50 EE1E (OD04000	CALL DWORD PTR DS:[<&KERNEL32.Duplicate(	
00401066	. FF15 66604000	MOV ECX.DWORD PTR DS:1(@REnneLS2.Dup110a(e)	• Dup ( reacenance
0040106C	. 8D5424 08	LEA EDX.DWORD PTR DS: [402700]	
00401060		PUSH EDX	- Durana Ta Ca
00401070 00401071	· 52	LEA EAX,DWORD PTR SS:[ESP+1C]	pProcessInfo
	. 8D4424 1C	DUCH ENA,DWUKU MIK SSILESMTICJ	=Ctautus Isfa
00401075	. 50	PUSH EAX	pStartupInfo
00401076	. 56	PUSH ESI	CurrentDir => NULL
00401077	. 56	PUSH ESI	pEnvironment => NULL
00401078	. 56	PUSH ESI	CreationFlags => 0
00401079	. <u>6</u> A 01	PUSH 1	InheritHandles = TRUE
0040107B	. 56 . 56	PUSH ESI	pThreadSecurity => NULL pProcessSecurity => NULL
0040107C		PUSH ESI	processecurity => MULL
0040107D	. 51	PUSH ECX	CommandLine => NULL
0040107E	. 56	PUSH ESI	ModuleFileName => NULL
0040107F		CALL DWORD PTR DS: [<&KERNEL32.CreatePro	■UreateProcessH
00401085	. 8500	TEST EAX, EAX	
00401087	.~74 17	JE SHORT nc.004010A0	
00401089	. 885424 0C	MOV EDX, DWORD PTR SS: [ESP+C]	-
0040108D	. 8B7424 08	MOV ESI, DWORD PTR SS: [ESP+8]	

If we scroll down the code a little bit, we find a string of INT3 instructions starting at offset 10D1:

004010A1 004010A3 004010A3 004010A4 004010A4 004010A5 004010A5 004010B2 004010B2 004010B3 004010B3 004010B3 004010B3 004010B3 004010C5 004010C5 004010C5 004010C5	. 56 . 56 . 56 . 68 60EA4000 . FF15 5CB04000 . 50 . 83C4 0C . 50 . 68 ACB14000 . 83C4 0C . 800F0000 . 83C4 1C . 8BC6 . 5E . 5B . 83C4 54 . C3	PUSH ESI PUSH ESI PUSH ESI PUSH 0A PUSH 0.0040EA60 PUSH 0A PUSH 0A PUSH EX PUSH EX CALL nc.00409FE6 ADD ESP.0C PUSH EAX PUSH EAX PUSH EAX PUSH EAX PUSH EAX PUSH EAX PUSH EAX PUSH EX PUSH EX PUSH EX PUSH EX ADD ESP.1C MOV EAX,ESI POP ESI POP ESX ADD ESP.54 RETN	Arg3 = 0000000A Arg2 = 0040EA60 CGetLastError Arg1 no.00409FE6 ASCII "Failed to execute shell, error = %s"
00401001 00401002 00401003 00401004 00401005 00401005 00401007 00401007 00401007 00401008 00401009 00401000 00401000 00401000 00401005	22222222222222222222222222222222222222	INT3 INT3 INT3 INT3 INT3 INT3 INT3 INT3	
004010E0 004010E6 004010E0 004010E0 004010F0 004010F4 004010F6 004010F6 004010F8 004010F8 004010F8 00401105 00401105 00401105 00401105	. 81EC 80020000 . A1 90E04000 . 56 . 57 . 8BBC24 8C02000 . 8B17 . 6A 00 . 6A 00	SUB ESP,280 MOU EAX,DWORD PTR DS:[40E090] PUSH ESI PUSH EDI MOV EDI,DWORD PTR SS:[ESP+28C] MOV EDX,DWORD PTR DS:[EDI] PUSH 0 PUSH 0 MOV DWORD PTR SS:[ESP+28C],EAX LEA EAX,DWORD PTR SS:[ESP+10] PUSH EAX PUSH 0C8 LEA ECX,DWORD PTR SS:[ESP+1C] PUSH ECX	PRemains = NULL pAvailable = NULL pRead BufSize = C8 (200.) Buffer ▼

INT3 is a software interrupt that is used by debuggers to pause program execution. Since Netcat obviously doesn't pause indefinitely during execution, these bytes are just filler and can be modified without worrying about affecting the program execution flow. Additionally, they are within the signature code which we need to modify. If you are using Olly, select one of these INT3 instructions (I chose the first one at 10D1), and press the space bar. In the 'Assemble' text box that appears, enter 'nop' (no quotes) and click assemble:

Assemble at 004010D1		×
nop		•
Fill with NOP's	Assemble	Cancel

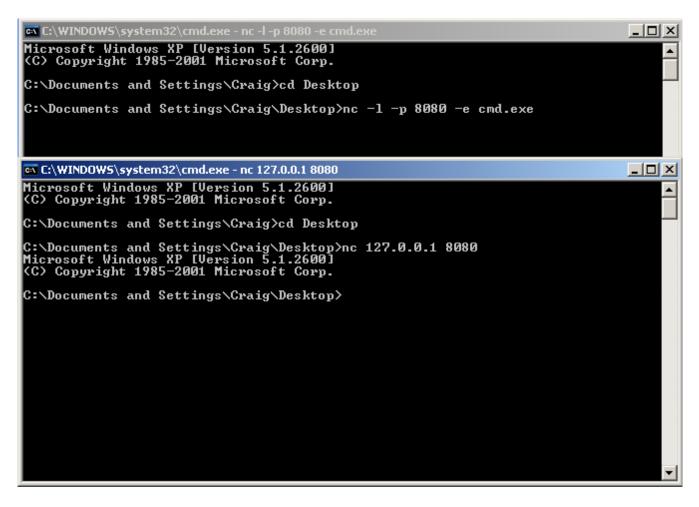
Now right click the code window, and select 'Copy to executable -> All modifications' (if prompted to "Copy selection to executable?", select 'Copy'). Right click the new window that appears and select 'Save file', then save the program as an executable file.

If you do not have OllyDbg, you can still make the above observations and modifications. The hexadecimal equivalent of the INT3 instruction is CC and the hex equivalent of the NOP instruction is 90. Using Hex Workshop, you can go to offset 10D1 and see that the same string of INT3 instructions (they are just displayed in their hexadecimal format of 'CC'); change one of those CC's to 90 and save the changes.

The final test of course is to see if Netcat does indeed evade detection by Norton, and still functions properly. First, let's scan the file with Norton:

😵 Norton AntiVirus - Scan	
Scan Complete	more info
No viruses, spyware, o	or other risks were found.
Total files scanned	0
Viruses, spyware and other security risks	
Detected	0
Resolved	0
Remaining	0
	Finish

So far so good, now let's create one instance of Netcat to listen for connections and spawn a command shell when one is received ('nc.exe -l -p 8080 -e cmd.exe'), and a second instance of Netcat to connect to the first ('nc.exe 127.0.0.1 8080'):



# **Closing Thoughts**

As this paper has demonstrated, it is quite simple for someone with even a rudimentary understanding of assembly code to take a potentially malicious program and modify it to bypass anti virus protection mechanisms, using nothing more than a hex editor. This is by no means a groundbreaking conclusion, and has been going on for many years. However, it is my hope that this paper has achieved at least one of three things:

- 1. Shown to the average user, even if they were not able to follow the paper in detail, that anti virus is not an end-all security solution, as seems to be the common belief among both consumers and vendors (or, at the very least, vendor marketing).
- 2. Provided, in detail, an example of how easy it is to invalidate an AV signature to those with little or no experience with binary file modification.
- 3. Proven to all the script kiddies who keep whining about Symantec, that unfortunately, their "133t h4x0r" days with Netcat are not over...