★OPERATION★
BLOCKBUSTER

TOOLS REPORT

NOVETTA

NOVETTA

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 line.

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

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

www.OperationBlockbuster.com

TABLE OF CONTENTS

1. Introduction

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 report prior to the reverse engineering reports for more details on the Operation and the Lazarus Group. This reverse engineering report looks at several tools found within the Lazarus Group's collection.

Not all of the families within the Lazarus Group's collection are inherently malicious. There are samples that perform somewhat legitimate functions. These samples are tools, usually one-off tools, that the Lazarus Group may use for a specific purpose and do not gain a wide distribution in the wild. Understanding the function of these tools adds additional insight into the group's methodologies.

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, 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, TangoAlfa specifies a tool family identified as Alfa.

FIRST LEVEL IDENTIFIER	GENERAL CLASSIFICATION
Delta	DDoS
Hotel	HTTP Server
India	Installer
Lima	Loader
Kilo	Keylogger
Papa	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.

2. [Keylogger] KiloAlfa

A rudimentary but somewhat clever design, KiloAlfa provides keylogging capability for the Lazarus Group's collection of malicious tools. The design of KiloAlfa is broken down into two basic components: the persistence functionality and the keylogging functionality. The persistence functionality of KiloAlfa allows the malware to self-install on a victim's machine when activated (described below). The keylogging functionality, as the name would imply, provides keystroke capturing capabilities by continually monitoring the foreground window and recording the current keyboard's state to a log file for later retrieval. KiloAlfa establishes a keystroke log file for each user that accesses the infected machine. KiloAlfa has no capabilities to transmit the log files to an external server thereby requiring an additional companion malware family be utilized by attackers for the retrieval of the key log files.

KiloAlfa is a self-installing binary. When activated without any command line arguments or with the command line argument **`-reg`**, KiloAlfa activates its persistence functionality. In order to maintain persistence on a victim's machine, KiloAlfa copies itself to either **`%ALLUSERSPROFILE%\Intel\mscorsw.exe`** (if the victim is running Windows Vista or newer) or **`%ALLUSERSPROFILE%\Application Data\Intel\mscorsw.exe`** (if the victim is running Windows XP or older). A registry key for **`HKLM\Software\Microsoft\CurrentVersion\Run\Mozilla Update`** is added and set to the path of the newly copied KiloAlfa binary with the **`-run`** command line argument added. If the registry key insertion is unsuccessful, KiloAlfa attempts to insert the same key and value within the current user hive (HKCU).

If KiloAlfa is activated without any arguments, the persistence functionality attempts to run KiloAlfa under all users logged into the victim's machine through terminal services. By calling **`WTSEnumerateSessionsA`** to get a list of all currently active Windows Terminal Services sessions, KiloAlfa uses the list of active terminal sessions to determine if **`explorer.exe`** is running under the session (to eliminate non-interactive user sessions); KiloAlfa then obtains the token of the user of the session and calls **`CreateProcessAsUserA`** to activate the newly installed KiloAlfa binary (**`mscorsw.exe`**) with the **`-run`** command line argument. KiloAlfa then activates **`mscorsw.exe`** with the **`-run`** argument for the console session.

After installing KiloAlfa as **`mscorsw.exe`** and activating the new binary for all active users, KiloAlfa generates and executes a suicide script to remove the initial instance of itself. The suicide script is saved to **`%TEMP%\AUTOEXEO.bat`** and takes the form presented in Figure 2-1. The script is constructed one line at a time using string concatenation and saved via **`WriteFile`** before **`CreateProcess`** is called to execute the script. It is worth noting that KiloAlfa and IndiaFoxtrot share the same suicide generation script code in both functionality and the script itself. Additionally, KiloAlfa and IndiaFoxtrot also use **`AUTOEXEO.bat`** as the suicide script's filename.

```
@echo off
:R1
del /a "{KiloAlfa's filename}"
if exist "{KiloAlfa's filename}" goto R1
del /a "{suicide script filename}"
```

Figure 2-1: KiloAlfa's Suicide Script

When KiloAlfa is activated with the **`-run`** command line argument, KiloAlfa activates its keylogger functionality. This functionality begins by loading the configuration file **`%ALLUSERSPROFILE%\Application Data\Intel\AdobeUpdaterPrefs.dat`** (on Windows XP and later) or **`%ALLUSERSPROFILE%\Intel\AdobeUpdaterPrefs.dat`** (on Windows Vista or newer). The configuration file contains a list of usernames and their respective keylogger filenames and paths. The structure of the configuration file (Table 2-1) consists of a magic value followed by one or more per-user configuration records (Table 2-2). The configuration is encrypted with RC4 using the hardcoded password `C!@I#%VJSIEOTQWPVz034vuA`. The magic value allows KiloAlfa to determine if the configuration file has been successful decrypted prior to attempting to read the per-user configuration records.

OFFSET	SIZE	DESCRIPTION
0	4 Bytes	Magic Value (`0xAABBCCDD`)
4	360 Bytes	First per-user configuration
4+360n	360 Bytes	nth per-user configuration

Table 2-1: Structure of KiloAlfa's Configuration File

OFFSET	SIZE	DESCRIPTION
0	100 Bytes	Username for whom the record pertains
100	260 Bytes	Full path and filename of the user's keystroke log file

Table 2-2: Per-User Configuration Record's Structure

KiloAlfa parses all of the per-user configuration records looking for one that belongs to the username under which the KiloAlfa executable is running. If the configuration file is corrupt or missing, or no entry is found for the current user, KiloAlfa generates a new record for the user and specifies the keystroke log file's path and name as `%LOCALAPPDATA%\IconCache.dat`. The current state of the configuration file is written back to disk.

The keylogging functionality of KiloAlfa works by establishing an invisible window on the victim's computer and then spawning a thread that monitors both the current foreground window's title and the current state of the keyboard every 10ms. The invisible window that KiloAlfa registers with the Windows GUI specifies its class name as "Windows IME Update" and provides a very simple window procedure function. The windows procedure function (a.k.a. `WinProc`) only responds to three types of messages from the GUI: `WM_CREATE`, `WM_DESTROY`, and `WM_ENDSESSIONS`. The `WM_CREATE` handler activates the keylogger thread after a 10 second delay, the `WM_DESTROY` handler simply causes the KiloAlfa binary to terminate, and the `WM_ENDSESSIONS` handler causes KiloAlfa to flush any buffered keystroke data to the keystroke log file.

Once activated, the keylogger thread generates an ASCII string with the current time and date to form a header and sends the header to disk after encrypting the data using the DNSCALC-style encoding. After writing the header to disk, the thread enters an infinite loop. At the top of the loop, the keylogger thread waits 10ms before calling `GetForegroundWindow` to get a handle to the current foreground window. If `GetForegroundWindow` does not return a valid handle, the keylogger thread continues a sequence of 10ms delays followed by a call to `GetForegroundWindow` until a valid handle is obtained.

The handle of the current foreground window is compared to the handle of the foreground window from the last iteration through the keylogger thread's loop. If the foreground window has changed since the last iteration, the class name of the foreground window is obtained and compared against `Window IME Update`; this determines if a window besides KiloAlfa's window is currently in the foreground. If KiloAlfa's window is in the foreground, the keylogger thread returns to the top of the loop and enters another 10ms sleep. Otherwise, the title of the foreground window is obtained, and a new ASCII string is generated containing a separator made up of 72 hyphens followed by a line with the current time stamp and the title of the foreground window. The string is then sent to the keylogger thread's keystroke log data queue.

The keylogger thread does not write entries immediately to disk. Instead, data is sent to a buffer in KiloAlfa's memory until the buffer contains at least 2048 bytes. Once the buffer has reached the 2KB threshold, the buffer is encrypted using the DNSCALC-style encoding and then written to disk. After the buffer is written to disk, the keylogger thread will clear the buffer by zeroing its memory.

The keylogger thread calls `GetAsyncKeyState` 74 different times (for 74 different virtual keys) to determine the state of each key. For any key that indicates it has been pressed, the keylogger thread will record the key to the buffer. Once the keylogger thread has checked all 74 different keys and their current state, the thread returns to the top of the loop and begins another iteration with a 10ms delay.

Figure 2-2 illustrates a keystroke log file (after decryption). The data in the log file was observed in a memory-dumped KiloAlfa binary.

```
----------------------------------------------------------------------
[2015.01.23 16:35:28] - "Windows Explorer"

----------------------------------------------------------------------
[2015.01.23 16:35:32] - "Administrator: C:\Windows\System32\cmd.exe"
d:
twi[BS][BS][BS]cd te[BS]wi5
twi5 D:\Backup



----------------------------------------------------------------------
[2015.01.23 16:35:57] - "Start menu

----------------------------------------------------------------------
[2015.01.23 16:36:00] - ""

----------------------------------------------------------------------
[2015.01.23 16:36:00] - "Windows Explorer"

----------------------------------------------------------------------
[2015.01.23 16:36:11] - ""

----------------------------------------------------------------------
[2015.01.23 16:36:12] - "Administrator: C:\Windows\System32\cmd.exe -
twi5  D:\Backup"
n
n
y

----------------------------------------------------------------------
[2015.01.23 16:36:22] - "Start menu"

----------------------------------------------------------------------
[2015.01.23 16:36:24] - ""

----------------------------------------------------------------------
[2015.01.23 16:36:24] - "Windows Explorer"

----------------------------------------------------------------------
[2015.01.23 16:36:30] - ""

----------------------------------------------------------------------
[2015.01.23 16:36:31] - "Administrator: C:\Windows\System32\cmd.exe -
twi5  D:\Backup"
y
```

Figure 2-2: Snippet of a KiloAlfa Keystroke Log (After Decryption)

For some samples of KiloAlfa, the developer(s) used the version information resource entry of Windows XP's **cmd.exe** for KiloAlfa's version information. Yet for one sample, the developer(s) took the strings, dialog, menu, and version information from **mspaint.exe** and imported them into the resource section of KiloAlfa. The sample that uses **mspaint.exe**'s resources is particularly unique, as it deviates from the normal KiloAlfa binary by the following key points:

- The sample does not use the configuration file (**AdobeUpdaterPrefs.dat**) nor does it support per-user keystroke log files
- The sample does not activate itself under the accounts of users logged in through the terminal services
- The suicide script is named **PRE**{random value}**.ba**t and changed to

```
@echo off
:Hello
del /a %1
if exist %1 goto Hello
del /a %0
```

Figure 2-3: Alternative KiloAlfa Suicide Script

Beyond the three listed changes and the inclusion of a large number **mspaint.exe**'s resources, the basic functionality and design of the errant sample is identical to the standard KiloAlfa form.

3. [Proxy] PapaAlfa

Evidence suggest that the Lazarus Group uses compromised infrastructure as the public-facing touchpoint for the majority of their malware samples. PapaAlfa is believed to be one of the proxy malware components that the Lazarus Group uses to hide the true command and control server(s) for operations. PapaAlfa is not a simple proxy or relay. Rather, PapaAlfa could be considered a smart proxy due in part to the fact that the attacker can easily switch the backend destination address and port without having to reestablish control over the infected machine hosting the PapaAlfa malware. PapaAlfa contains an administrative control channel (or "admin channel") separate from its proxy channel. PapaAlfa is malware-traffic specific: PapaAlfa does not proxy general traffic, but rather traffic that is specific to the Romeo-CoreOne based families; these RAT families are discussed in Novetta's other report on the Lazarus Group's RAT and Staging capabilities.[1]

In terms of form factor, PapaAlfa comes in two flavors: service DLL and standalone executable. The primary difference between the two is that the executable variants require the admin channel's listening port number to be specified on the command line, otherwise it defaults to 443; meanwhile, the service DLL variants receive their admin channel listening port from a configuration file (`pmsconfig.msi`). The IndiaBravo-PapaAlfa installer is responsible for installing the service DLL variant.

Regardless of the form factor, the core of PapaAlfa is the same. Once the core component of PapaAlfa activates (which occurs after the service DLL initializes or after the standalone executable reaches the `main` function), the following tasks occur:

1. Dynamically load API functions
2. Determine the admin channel's listening port number
3. Establish the admin channel's listening port and open a hole in the Windows Firewall to facilitate incoming connections
4. For each new incoming admin channel connection, generate a new thread to handle the communication.
5. At the start of each new thread, perform a handshake between PapaAlfa and the client
6. Determine if the client is requesting access to the admin channel or initiating the relay mode

PapaAlfa uses the Space-Dot obfuscation method, found in multiple other Lazarus Group malware families, within the dynamic API loading portion of its code. Four different sets of API functions are loaded from `kernel32.dll`, `ws2_32.dll` (or `wsock32.dll`, depending on the Windows OS version), `advapi32.dll`, and `iphapi.dll`, but only a fraction of the loaded API functions are actually used by PapaAlfa.

The selection of the port to use for PapaAlfa's admin channel is made either on the command line (for standalone executable versions of PapaAlfa) or from a configuration file (`pmsconfig.msi`) that PapaAlfa loads. Establishing the admin port begins by first opening a hole within the Windows Firewall using the `netsh firewall` command. PapaAlfa uses the following code to construct that command:

1 http://www.operationblockbuster.com/wp-content/uploads/2016/02/Operation-Blockbuster-RAT-and-Staging-Report.pdf

```
sprintf(szCommandLine, "%sd.e%sc n%ssh%srewa%s ad%s po%sop%sing T%s %d \"%s\"",
"cm", "xe /", "et", " fi", "ll",  "d", "rt", "en", "CP", wPort, "adp")
```

The method of constructing the `netsh firewall` command is very specific and can be found in both RomeoCharlie and TangoAlfa, allowing for a direct code-level connection between the three malware families. After opening the firewall to allow for incoming connections (a critical operation given that PapaAlfa operates as a server and not a client), the listening port is established by calling Winsock API functions `socket`, `bind`, and `listen`.

Outside of the two form factors that PapaAlfa can have, there are two distinct variants of PapaAlfa that have emerged over time. While both perform the equivalent set of operations for Tasks #5 and #6, the underlying communication subsystem and methodology changes. PapaAlfa-One uses a very simple, unencrypted communication subsystem and a straight-forward command processing system, while PapaAlfa-Two employs the fake TLS communication scheme to encrypt communication between the client and the admin channel and uses a more robust command processing system.

Despite their communication subsystem differences, the protocol of the handshake for establishing a connection between PapaAlfa and a client is the same. Once an external client connects to the admin channel of PapaAlfa, a new thread is spun up to handle the communication between itself and the client. The initial communication with the client consists of the client sending 12 bytes to PapaAlfa and PapaAlfa responding with 12 bytes. The 12 bytes that the client sends to PapaAlfa take the following format:

OFFSET	SIZE	DESCRIPTION
0	4 bytes (DWORD)	Channel ID Request (`dwChannel`)
4	4 bytes (DWORD)	Ignored, typically set to `0` (`dwIPAddress`)
8	4 bytes (DWORD)	Client’s Internal State (`dwState`)

Table 3-1: PapaAlfa Client’s Initial Data Burst Format

The 12 bytes that PapaAlfa responds with take the following format:

OFFSET	SIZE	DESCRIPTION
0	4 bytes (DWORD)	Channel ID Response Code (`dwChannel`)
4	4 bytes (DWORD)	Client’s IP Address (`dwIPAddress`)
8	4 bytes (DWORD)	PapaAlfa’s Current State (`dwState`)

Table 3-2: PapaAlfa-One’s Response Format to the Client

The Channel ID Request (`dwChannel`) value specified by the client indicates the type of channel that the client wishes to activate. There are two supported channels: proxy activation and proxy configuration. The proxy activation channel, as the name implies, is responsible for the activation of the proxy (or relay) between PapaAlfa and another endpoint, and the proxy configuration channel allows the connecting client to configure or reconfigure the PapaAlfa proxy.

The proxy activation channel establishes a connection between PapaAlfa and a predetermined endpoint. Establishing a connection between PapaAlfa and the configured endpoint requires another set of 12-byte data transfers between PapaAlfa, as the client, and the endpoint, as the server. Since the endpoint retains the same handshake protocol as the clients connecting to PapaAlfa, it is theoretically possible to chain multiple PapaAlfa-infected hosts together to generate a web of misdirection and obfuscation.

Once PapaAlfa establishes a channel between itself and the endpoint, data transmitted to and from the client is proxied to the established endpoint. As mentioned previously, the PapaAlfa-One variants do not introduce any obfuscation or encryption to the network traffic as it traverses the PapaAlfa-One node, while PapaAlfa-Two uses the fake TLS encryption scheme found in other Lazarus Group malware families such as RomeoAlfa, RomeoBravo, and RomeoHotel, among others. Connections between a client and PapaAlfa are recorded as a 24-byte structure, described in Table 3-3, within a log file named **`pmslog.msi`**.

OFFSET	SIZE	DESCRIPTION
0	4 bytes (DWORD)	IP address of client (**`dwIPAddress`**)
4	4 bytes (DWORD)	Reported state of the client (**`dwState`**)
8	16 bytes (SYSTEMTIME)	Time of the connection

Table 3-3: PapaAlfa's Connection Log Entry Structure

If the client accesses the administrative channel of PapaAlfa, the client, without authentication, can perform one of the following functions:

- Receive information about the PapaAlfa-infected host and change the proxy endpoint
- Receive the connection log (**`pmslog.msi`**)
- Set the flag to allow PapaAlfa to proxy incoming connections

The act of configuring the endpoint that PapaAlfa will connect to is done by first requesting, via the administrative channel, the infected host's system information and then supplying the port number of the endpoint. PapaAlfa will use the client's own IP address as the IP address of the proxy's endpoint.

4. [Network Tester] TangoAlfa

Consisting of a single known sample, TangoAlfa is a tool used to determine a victim machine's external capabilities, namely whether or not the victim machine is capable of receiving incoming connections (presumably from the Internet, but possibly from the local network). In order to make such a determination, the application performs the following tasks:

1. Determine the target end point from the command line or use the default end point of `69.90.29.43:8080` if no valid command line input is supplied
2. Test if the end point will respond to a TCP socket `connect` request and will accept incoming data ("writable").
3. Generate a listening socket on a randomly selected port between 400 and 499 or, if unsuccessful, generate a listening port between 8000 and 8999.
4. Open a hole on the victim machine's firewall for the listening port
5. Transmit the port number to the end point as a two-byte value
6. Read a four-byte value from the end point to determine the success or failure of the test.

If any of the above steps fail, TangoAlfa will report an error, and the test will fail. To that end, TangoAlfa is very verbose about which stage resulted in a failed test as well as which stage TangoAlfa is executing at any given time.

TangoAlfa can be instructed to connect to any testing server on any port by supplying the IP address and port on the command line (separated into individual command line arguments). If the user of TangoAlfa does not supply a testing server (or "end point"), then TangoAlfa will default to `69.90.29.43:8080`. If TangoAlfa is unable to connect to the testing server, it will display the following message:

```
*****[Start Test -> 69.90.29.43:8080]
*****[Relay Connect Fail]
```

TangoAlfa tries up to three times to generate a listening socket on the victim's machine. The first attempt tries for a listening port randomly selected in the range of 400 to 499, while the second and third attempt tries the range of 8000 to 8999. If after three attempts to generate a listening port TangoAlfa is unable to establish a listening port, it reports a failure and terminates.

After establishing a listening port, TangoAlfa attempts to open the Windows Firewall on the victim's machine in order to allow incoming connections. TangoAlfa does this by constructing the following command and executing it via `CreateProcess`:

```
cmd.exe /c netsh firewall add portopening TCP <listening port number> "adp"
```

On Windows XP computers, the above command will result in the line "Ok." being displayed, whereas on newer versions such as Windows 7, the following text will be displayed:

```
IMPORTANT: Command executed successfully.
However, "netsh firewall" is deprecated;
use "netsh advfirewall firewall" instead.
For more information on using "netsh advfirewall firewall" commands
instead of "netsh firewall", see KB article 947709
at http://go.microsoft.com/fwlink/?linkid=121488.

Ok.
```

With the firewall reconfigured to allow incoming connections to the listening port, TangoAlfa displays the listening port number to the user before sending the value to the test server as a two-byte value. After sending the port number, TangoAlfa waits for the test server to respond with a four-byte value or a timeout of 60 seconds, whichever comes first. If the test server sends back `0` (as a 32-bit number), then TangoAlfa indicates the test was a failure as seen below:

```
*****[Start Test -> X.X.X.X:80]
*****[Relay Connect Success]
Ok.

*****[Listen Port 483] - [Fail]
*****[End Test]
```

Otherwise, TangoAlfa indicates that the victim's machine is capable of receiving incoming network connections:

```
*****[Start Test -> X.X.X.X:80]
*****[Relay Connect Success]
Ok.

*****[Listen Port 483] - [Success]
*****[End Test]
```

The structure of the test implies that the test server ultimately determines success or failure. Simply establishing a listening port and accepting incoming connection does not mean a test is successful; rather, the test server must decide the victim's machine's suitability.

TangoAlfa does not use the more common features found in the Lazarus Group's collection of malware such as the various encryption/obfuscation methods seen with Space-Dot, DNSCALC, or dynamic API loading. However, it is possible to directly tie TangoAlfa to the rest of the Lazarus Group's malware based on the method that the developer(s) use for opening a hole in the victim's Windows Firewall. When constructing the command, TangoAlfa uses a very specific `sprintf` statement:

```
sprintf(szCommandLine, "%sd.e%sc n%ssh%srewa%s ad%s po%sop%sing T%s %d \"%s\"", "cm", "xe
/", "et", " fi", "ll",  "d", "rt", "en", "CP", wPort, "adp")
```

The same format string is found in both PapaAlfa (see Section 3) and RomeoCharlie, which have more concrete ties to the larger Lazarus Group collection of malware families.

Given the purpose of TangoAlfa—the determination of the viability of a victim machine for incoming connections—and the fact that a large number of server mode families are found within the Lazarus Group's collection, this particular tool could be extremely useful during an operation for finding suitable proxy hosts.

5. [Domain Redirector] TangoBravo

TangoBravo is a simple piece of malware, consisting of only a handful of non-library functions. Using a technique commonly found in commodity malware versus APT-style malware, TangoBravo's sole purpose is to prevent a victim's machine from accessing specific domains by modifying the victim's `%SYSDIR%\drivers\etc\hosts` file. To that end, the first task that TangoBravo performs is to scan the hosts file line by line and determine if one of the following domains is present; if so, the line is modified to replace the existing IP address with the loopback address `127.0.0.1`:

- www.alyac.co.kr
- www.boho.or.kr
- download.boho.or.kr
- www.ahnlab.com
- explicitupdate.alyac.co.kr
- gms.ahnlab.com
- ko-kr.albn.altools.com
- ko-kr.alupdatealyac.altools.com
- su.ahnlab.com
- su3.ahnlab.com
- update.ahnlab.com
- ahnlab.nefficient.co.kr

To further ensure that the above domains are inaccessible to the victim, any of the domains that were not explicitly modified are appended to the end of the `hosts` file with the `127.0.0.1` loopback address. Note that not all variants of TangoBravo overwrite all of the domains; specifically, alyac.co.kr, boho.or.kr, download.boho.or.kr, and ahnlab.com are not found in all samples.

Upon completion of the hosts file modification, some variants of TangoBravo generate and execute a suicide script. It is this suicide script generation and execution that ties TangoBravo into the larger Lazarus Group's malware collection, as the suicide script generator and suicide script content is also found in IndiaCharlie. The difference, as seen below, is that TangoBravo uses `CreateProcess` instead of `WinExec` to activate the script.

```
fp = fopen("d.bat", "w+");
if ( fp )
{
 fprintf(fp, ":R\nIF NOT EXIST %s GOTO E\ndel /a %s\nGOTO R\n:E\ndel /a d.bat", "%1", 
"%1");
 fclose(fp);
 sprintf(szCommandLine, "d.bat \"%s\"", szTangoBravoFilename);
 CreateProcessA(0, &szCommandLine, 0, 0, 0, 0x8000000u, 0, 0, &StartupInfo, 
&ProcessInformation);
}
```

Additionally, the core component of TangoBravo has been found integrated in another malware family within the Lazarus Group's collection: IndiaGolf.

6. [Windows Firewall Disabler] TangoCharlie

Observed as a payload of SierraJuliett-MikeOne, TangoCharlie is a singularly focused tool whose one and only purpose is to disable the Windows Firewall. Depending on the operating system, TangoCharlie will use either `netsh firewall` (for Windows XP or older) or `netsh advfirewall` (for Windows Vista or newer) to disable the firewall on a victim's machine. Table 6-1 identifies the full command that TangoCharlie passes to `WinExec` in order to open the Windows Firewall depending on the victim's Windows version.

WINDOWS VERSION	FIREWALL COMMAND
Windows XP or older	`cmd.exe /c netsh firewall set opmode mode=disable`
Windows Vista or newer	`cmd.exe /c netsh advfirewall firewall add rule name=CoreNetworkingHTTPS dir=in action=allow Protocol=TCP`

Table 6-1: TangoCharlie's Firewall Disabling Commands

7. [Antivirus Suite Killer] TangoDelta

Antivirus applications can be bothersome to malware developers, and TangoDelta is a solution to a specific antivirus problem: using the open source Process Hacker's kernel driver,[2] TangoDelta attempts to terminate various McAfee VirusScan Enterprise and McAfee Agent-specific processes listed in Table 7-1.

TARGET PROCESS NAME	DESCRIPTION[2]
McShield.exe	"The McShield On-Access Scanner Service"
UdaterUI.exe	"Provides the user interface for the updates"
McTray.exe	"The System tray icon management tool"
shstat.exe	"It is responsible for loading the McAfee VirusScan (Vshield) +icon in the Windows system tray, the VSE About screen, the On-Access Scanner statistics, and the messages window."
FrameworkService.exe	"Provides the majority of (McAfee Agent) functionality"
VsTskMgr.exe	"Facilitates janitorial duties within the" McAfee Agent
mfeann.exe	"Responsible for event creation and logging" of the McAfee Agent
naPrdMgr.exe	"The Product Manager for the McAfee Framework service"

Table 7-1: Target Processes of TangoDelta

TangoDelta requires no command line arguments; upon activation it will generate and activate a service called `KProcessHacker2` to load the Process Hacker driver into kernel memory. With the Process Hacker driver installed, TangoDelta connects to the driver and uses the Process Hacker function `TerminatorTP3` to terminate each of the McAfee processes listed in Table 7-1 in order. TangoDelta pays special attention to the `McShield.exe` process: after terminating the McAfee processes, TangoDelta continually scans for `McShield.exe`'s process in the list of active processes on the victim's machine, sleeping for 10ms between scans. Once `McShield.exe` has been successfully terminated, TangoDelta locates the path to the `McShield.exe` file by inspecting the registry key `HKLM\SYSTEM\CurrentControlSet\services\McShield\ImagePath` and then replacing the `McShield.exe` binary with itself. This ensures that upon reboot, TangoDelta will load rather than the legitimate McAfee `McShield.exe` binary.

With the McAfee processes terminated and TangoDelta replacing the legitimate `McShield.exe` file, TangoDelta disconnects from the Process Hacker driver and removes it from memory before deleting the driver's service from the Windows services database. TangoDelta then activates the new `McShield.exe` binary by reactivating the `McShield` service before generating and executing a suicide script to remove the original TangoDelta binary.

2 Wen Jia Liu. "Overview – Process Hacker" http://processhacker.sourceforge.net/. Accessed 27 November 2015.
McAfee. "Process description and load order for VirusScan Enterprise and McAfee Agent". https://kc.mcafee.com/corporate/index?page=content&id=KB65784&pmv=print 29 September 2015

The suicide script that TangoDelta generates is nearly identical to that of WhiskeyAlfa-One. The only difference between the suicide scripts is the superfluous **`strcpy(buffer, "@echo off\r\n")`** instruction at the beginning of TangoDelta's script generation. Otherwise, the suicide scripts are identical in structure, filename (**`%TEMP%\zawq.bat`**), and the APIs that are used in their generation and execution, as seen below:

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

8. Conclusion

While the tools profiled in this report are not inherently malicious, their capabilities are nonetheless integral to the Lazarus Group's cyber operations, both espionage and destructive in nature, making them inherently dangerous to potential victims. These tools often lay the groundwork for further malicious activity, such as the targeting of antivirus capabilities and the disabling of firewalls, both of which are very fundamental defensive measures. Furthermore, like many other identified Lazarus Group families, these tools showcase the group's creative solutions, such as the PapaAlfa proxy, which makes it difficult to immediately identify potentially malicious activity on a compromised network.

NOVETTA

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