White Paper

The Lazarus Constellation

A study on North Korean malware

19/02/2020

LEXFO

avisa partners

TABLE OF CONTENTS

I. RISE OF LAZARUS

INTRODUCING THE LAZARUS APT GROUP

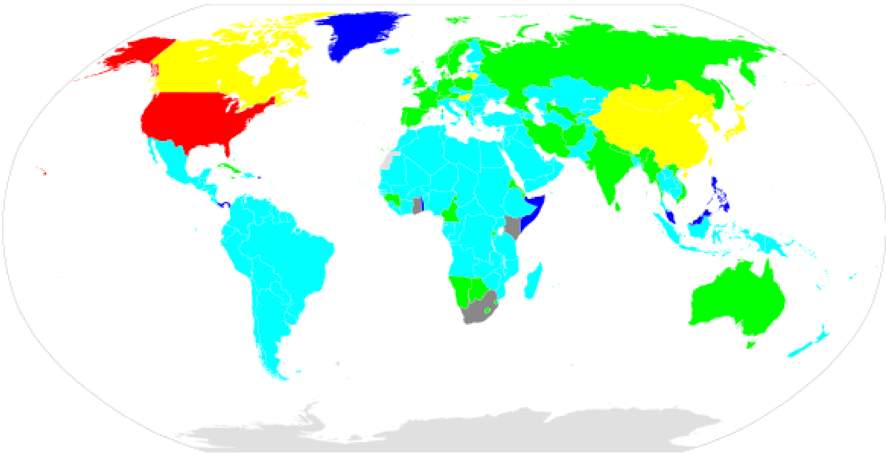
Lazarus activities have been retroactively tracked back to 2007, under various names. For years, these activities were seen as acts of cyberterrorism and vandalism, since most of them systematically involved destruction of data and / or distributed denial of service attacks.

The Lazarus group was clearly identified and named in the 2016 Novetta report "Operation Blockbuster"[1]. This report uncovered and attributed a large set of malware based on the analysis of the Sony Pictures Entertainment targeted attack. Attribution and tracking was made possible due to the group's habits of reusing huge chunks of code in most of their malware.
This report showcased how active and diverse the group is: using more than 45 different home-developed malware families, Lazarus has been conducting destructive attacks but also advanced and persistent spying campaigns all over the world, making it worthy of the "APT" designation. TTP, arsenal and targets reveal that Lazarus is composed of at least three different subgroups: the Lazarus "core", aiming at disrupting activities and causing damage, Andariel, hacking for profit and intelligence, and Bluenoroff, motivated by financial gains.
Uncovering its malware and activities didn't stop the Lazarus group from continuing its operations or renewing its arsenal, as the rest of this report will show.

The U.S. Government, mostly through its CERT, is referring to Lazarus as Hidden Cobra[2].

ATTRIBUTION: LINKS WITH NORTH KOREA

Lazarus activities have often been wrongly attributed to China or to unknown cyberterrorist groups. After identifying the Lazarus arsenal in 2016, researchers were able to track and attribute the group's attacks, as well as monitoring their command & control servers. During an investigation, Group-IB discovered that Lazarus operators connected to a C&C using two IP addresses from North Korea (`210.52.109.22` and `175.45.178.222`). Moreover, analyses of compilation timestamps of the binaries used by the group in their attacks were consistent with North Korean working hours (see our analysis below). Other artefacts can be mentioned as well, such as the YMD date format found in Lazarus log files, which is used almost exclusively in the Korean region.



Date formats by country. Yellow = YMD format

It is believed that Lazarus operators are linked to Bureau 121, a division of the Reconnaissance General Bureau intelligence agency (Group-IB). This attribution to North Korea was confirmed by FBI and NSA investigations, based on internal sources and the technical elements previously mentioned [3].

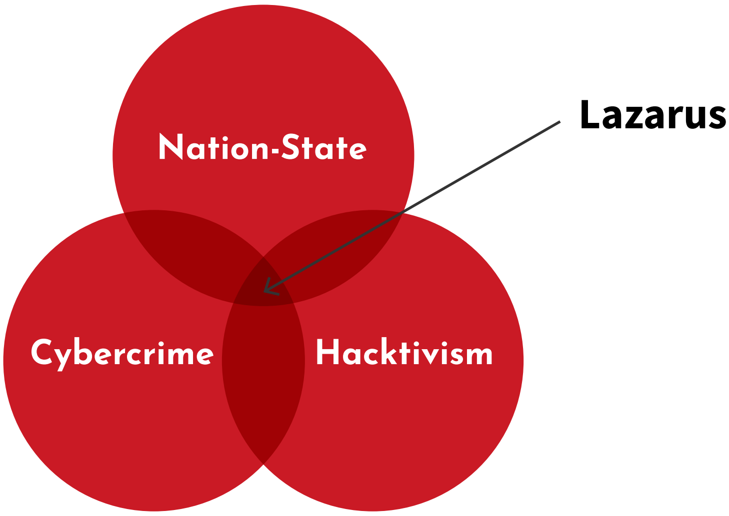
TARGETS & CAPABILITIES

Lazarus targets are very disparate, as the group has very diverse motives: intelligence, financial gains and disruption. Lazarus and its subgroups have been focusing on attacking governments, financial institutions, defense industry actors, IT and videogame companies. Geographically, most targets are located in South Korea and in South America.

Despite operator mistakes and the fact that their attacks are most of the time technically simple, Lazarus and its subgroups are well-funded and able to discretely maintain persistence in networks for years. They were seen adapting very fast, fighting against forensic investigators in real-time by repacking malware, erasing files or modifying encryption keys and algorithms in less than an hour after being discovered.

Furthermore, they have been leveraging many 0day vulnerabilities they bought or developed on their own throughout the years.

All of these operations come at a cost. The Bluenoroff subgroup is supposedly in charge of financing the whole ecosystem through big money heists.



CLARIFYING LINKS WITH OTHER ATTACKER GROUPS

Lazarus shares some TTP with other North Korean APT groups and has been using crimeware malware.

– APT37 (Reaper)

Other names:

- → Reaper (FireEye)
- → Ricochet Chollima (CrowdStrike) ScarCruft
- → Red Eyes

APT37 is another North Korean attacker group focusing on the Middle East and South Korea. Reaper uses its own set of malware and infrastructure, and its activities don't overlap with Lazarus'. The first known attack attributed to APT37 was traced back to 2014. They rely strongly on known or 0day exploits and spear phishing to infect their victims.

The group was publicly exposed by FireEye [4].

– APT38 (Bluenoroff)

APT38 targets financial companies mostly in Asia. The first known operation took place in 2014 according to FireEye. The group was publicly exposed by FireEye [5]. This report doesn't clearly draw a link between APT38 and Lazarus subgroup Bluenoroff, which comes from the fact that FireEye classify APT groups following its own strict rules and criteria. To remove any confusion, we will be less rigorous than FireEye and consider APT38 to be Bluenoroff, based on malware code overlaps and TTPs. See the "Classification" part of this report for technical links with Lazarus.

APT38 TTP resemble those of Lazarus subgroups, especially how they carry out their attacks and chose their targets. They have been focusing on attacking banks connected to the SWIFT network. They will most of the time infiltrate a bank network through vulnerable exposed servers, spend months gathering information, doing reconnaissance and moving laterally in the network until they find a way to steal money. Once the theft is complete, they will try to destroy all evidence by deploying crimeware ransomware or wipers.

APT38 has its own toolset to maintain persistence, move laterally and manipulate SWIFT transactions. Their targets are diverse and worldwide: Russia, Turkey, USA, Uruguay, Brazil, Vietnam, etc. This group has shown some amateurism and carelessness despite being quite sophisticated, which is a common trait amongst North Korean APT groups.

– Clarifying links with TA505 (Emotet, TrickBot & Dridex)

TA505 is a financially-motivated threat actor mostly operating from Russia. This actor is known for phishing campaigns using banking trojans such as Dridex and TrickBot, ransomware campaigns deploying Locky and the wide use of the Emotet loader.

– McAfee's mistake

Since early 2019, some reports mentioning links between Emotet/TrickBot and Lazarus were published. It appears, however, that these reports were filled with misconceptions and faulty logic, which led to misattributions.

Emotet is one of the most common malware loaders in the wild. It has been used by the TrickBot gang to install their eponymous banking trojan. Both Emotet and TrickBot are believed to come from the Russian cybercrime.

In late 2018, Emotet and TrickBot were seen deploying a ransomware called Ryuk in well-funded companies' infected networks. Contrary to most ransomware, Ryuk asks for a huge amount of money to decrypt files, sometimes more than $100,000 (see paid ransoms [6]). Analysis of this malware revealed that it shared most of its code with another crimeware ransomware named Hermes. Hermes was sold on underground hacker forums for as little as $300 in 2017/2018 and was quite popular during those years. Lazarus has been buying and using Hermes to cover their tracks by encrypting disks after a completed operation multiple times. Given these facts, some hasty researchers spread the idea that Ryuk and Lazarus were tied due to Hermes. This was also supported by the fact that researchers reported that they saw previous Lazarus infections cohabit with Emotet and TrickBot, which can also be observed during a forensic mission.

McAfee, in charge of investigating a Ryuk outbreak at that time, published a blogpost to clarify the situation and reveal some findings supporting that Ryuk was in fact coming from a Russian-speaking country and probably linked to the TrickBot gang.

– Latest proof of actual links

In mid-2019, what were initially seen as coincidences became more and more suspicious and some strong links were found during incident response missions, with Lazarus samples being dropped shortly after TA505 malware infected the network. TA505 and Lazarus IOCs were found altogether in bank networks and PowerShell post-intrusion scripts attributed to TA505 and Lazarus appeared to be very similar [7] [8]. From there, it is hard not to consider the fact that the TA505 attackers seem to be selling accesses to bank networks to Lazarus. LEXFO also found TA505 malware (TrickBot and Emotet) during its incident response involving Lazarus, which corroborates these assertions.

MAIN OPERATIONS (2007 - 2015)

Lazarus operations have been traced back to 2007. The first attack attributed to Lazarus was a DDoS against South Korean and U.S. websites leveraging the MyDoom botnet. The group has been very active ever since, conducting the operations below (Intezer [9]):

Year	Lazarus campaign	Year	Lazarus campaign
2007	Silent Chollima	2015	Tdrop
2009	MYDOOM	2016	Bangladesh Bank Heist
2011	10 Days of Rain	2017	WannaCry
2011	Operation Troy	2017	Hidden Cobra
2011	SierraBravo	2017	Polish Attacks
2011	Blockbuster	2017	Ratankba
2011	Joanap	2017	RokRAT
2011	KorDLLBot	2018	South Korean Power Grid
2011	Brambul	2018	GoldDragon
2013	KorHigh	2018	NavRAT
2013	DarkSeoul	2018	Lazarus Bitcoin
2013	KimSuky	2018	NK Gambling
2014	Destover	2018	RedGambler
2015	Duuzer	2018	LEXFO's incident response

II. LAZARUS' NEW MOTIVES

(2016 - 2019)

FIGHTING SANCTIONS IN THE CYBER SPACE

North Korea has been targeted by multiple rounds of financial sanctions and restrictions. In 2017, the UN and the United States issued many resolutions and orders that had heavy negative impact on North Korea exchanges [10].

To compensate, we have seen the Lazarus group focus on hacking financial institutions all around the world to steal money. Even though disruptive attacks keep being conducted, it is clear that Lazarus prefer heists involving big sums of money. Likewise, spying operations are still being conducted by North Korea but are usually attributed to the fast-expanding APT37 [11].

The Andariel subgroup illustrates how Lazarus changed its focus from information gathering to financial gains. Precisely, Andariel was actively targeting the defense industry until the end of 2016, when they switched to attacking financial institutions, as showed by the timeline of the main Andariel attacks below: (AhnLab [12])

Date	Target	Purpose
November 2015	Defense	Intelligence
February 2016	Security company	Intelligence
April 2016	Defense	Intelligence
June 2016	Defense	Intelligence
August 2016	Military	Intelligence
October 2016	Gambling	Financial gains
January 2017	Gambling	Financial gains
March 2017	ATM	Financial gains
April 2017	Energy	Intelligence
May 2017	Financial industry	Financial gains
June 2017	Financial industry	Financial gains
October 2017	Travel agency	Financial gains
December 2017	Travel agency	Financial gains
December 2017	Telecommunications	Spying
December 2017	Cryptocurrency exchange	Financial gains
February 2018	Cryptocurrency exchange	Financial gains
February 2018	Politics	Spying
October 2018	ATM (FastCash)	Financial gains

BANKS & ATM

Most bank attacks are carried out by the Bluenoroff subgroup, while ATM attacks are usually attributed to Andariel. In both cases, two methods were leveraged:

→ **Spear phishing Watering hole**
→ **Vulnerabilities in specific and targeted software directly to perform supply chain attacks**

One of largest attacks occurred in early 2017, when it was discovered that more than tweny Polish banks were infiltrated by Lazarus. The financial loss is unknown but the scale of the attack and its success is a testament to how capable the attackers are. Bank employees were targeted by several watering holes [13] delivering a payload through a known Silverlight exploit (CVE-2016-0034).

Lazarus also unsurprisingly targets ATM to steal credit card information. Lazarus targeted the ATM operator VANXATM in February 2015. The attack was sophisticated and leveraged a 0day in the antivirus software as well as a bad configuration of the update server allowing the attackers to install their malware on more than 60 connected ATM. It was reported that 230,000 unique credit card information numbers were exfiltrated to Lazarus C&C. The attack was attributed to the Andariel subgroup [14].

Another example of a successful ATM attack by Lazarus was uncovered by US-CERT [15] and Symantec [16] and was named “FASTCash campaign”. This attack successfully targeted banks in Asia and Africa, and forced issuing banks to accept fraudulent withdrawal requests. Different tailor-made malware were used in each attack. Such an attack involving ATM jackpotting requires physical presence and a mule network, showing how experienced Lazarus attackers are in carrying out advanced cybercriminal operations. Tens of millions of dollar were successfully stolen from banks.

Lazarus has also been targeting Point-of-Sale businesses with the Ratankba malware family they developed, showing that they don’t miss any opportunity to make quick money using custom tools [17].

TARGETING CRYPTOCURRENCY BUSINESSES

Lazarus attackers have recently been focusing on hacking cryptocurrency businesses, with a particular emphasis on South Korean exchanges. These attacks are very profitable and most of the time quite unsophisticated, making them the perfect way for stealing money [18]. The most significant attack was against Coincheck and ended up with Lazarus stealing about $534 million [19].

In 2018, Kaspersky uncovered a Lazarus attack they called “Operation AppleJeus”. The attack was sophisticated and targeted cryptocurrency users and exchanges. Victims were infected by a backdoored MacOS cryptocurrency trading software. Most samples used were compiled in 2017 [20].

In the end of 2017, ProofPoint uncovered a new implant named PowerRatbanka. This malware was developed using PowerShell, which shows that Lazarus attackers are following the trends and their arsenal is in constant development [21].

Other Lazarus attacks were reported by Group-IB in 2018 against YouBit, Coinis and Yapizon with millions of dollars stolen in each case. All of the exchanges are located in South Korea, and spear phishing was the main intrusion vector.

NEW TOOLSET

Being exhaustive in the description of the Lazarus toolset would be a trite task, as the group is able to quickly develop custom malware for each target. They have also been seen using malware from other criminal groups, particularly ransomware, to make attribution harder and cover their tracks.

For instance, some Lazarus malware were found alongside Emotet and Trickbot, and the attackers will execute ransomware such as Hermes to hide their activities and fingerprints after a successful operation.

Recently, a new specific malware toolset was used by Lazarus in different attacks. LEXFO investigated such an attack involving malware from this set and will describe its findings in the next part.

LEXFO also noticed that the attackers were no longer using the VisualStudio C++ v6 compiler, and the most recent samples found were compiled using VisualStudio C++ v8.

– MacOS malware

Kaspersky uncovered an attack attributed to Lazarus leveraging a trojanized cryptocurrency trading application for MacOS. This discovery showed that the North Korean group is not slowing down and keeps improving its technical capabilities [22].
The malware could be attributed to Lazarus mostly because of a hardcoded RC4 key found in other Lazarus malware and a reused C&C domain.

In the fall of 2019, TrendMicro also published a blog article where they uncovered a MacOS variant of the Nukesped trojan found in the wild, attributed to Lazarus [23].

– Mobile malware

Lazarus expanded their capabilities and developed their first mobile malware in 2017, by adding malicious code to a legitimate APK. This malware was discovered and analyzed by McAfee in a blogpost [24]. The trojanized Android application was not spread through Google Play.
Attribution to Lazarus is based on the communication protocol which was made to hide packets in the legitimate flow of TLS/SSL traffic, and some hardcoded values found in other Lazarus samples.

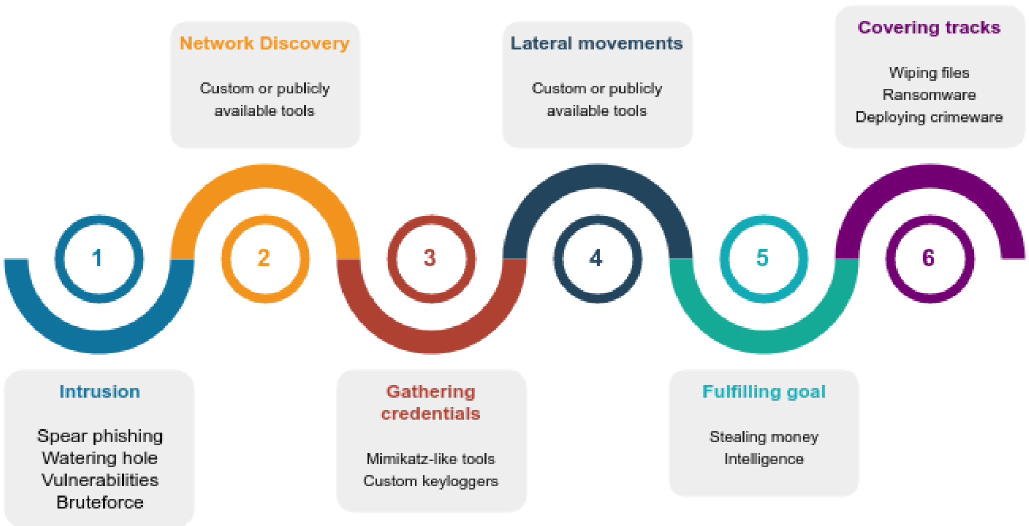
III. TECHNICAL ANALYSIS OF KEY LAZARUS ATTACKS

LAZARUS TTP

– Attack scheme

Considering the vast amount of attacks carried out by Lazarus throughout the years, it is possible to notice some recurring patterns in the way the group operates. These patterns have not changed much since their first attacks.

- → Intrusion through spear phishing, watering hole, bruteforce or web vulnerabilities Network discovery using custom or publicly-available tools
- → Gathering credentials through Mimikatz-like tools and keyloggers Lateral movements using custom or publicly-available tools Fulfilling the attack goal: stealing money and/or information
- → Covering tracks by wiping systems or infecting the victims with crimeware malware or ransomware



Lazarus attack pattern

Intrusion

Lazarus operators use a wide range of tricks to try and infect their victims. Their main vector is spear phishing, sometimes using 0day or known vulnerabilities. They also perform watering hole attacks and RDP password bruteforce [25].

Furthermore, they often exploit bad network isolation by hacking into webservers in order to try and access the internal network of a targeted organization. In this way,they were able to reach the server connected to the SWIFT network in the case of the Bangladesh Central Bank attack.

In an attempt at attacking a Chilean bank, the Lazarus operators targeted an employee with a fake job offer. They set up an interview via a Skype call where the targeted employee was tricked into downloading and executing a payload. This shows that the attackers are becoming more and more aggressive [26].

Attempts to confuse attribution

Lazarus malware developers have been trying to fool researchers by introducing some "false flag" Russian strings as command names. The attempt was not convincing as it was obvious for native speakers that names were lazily translated to Russian. The Russian command names are still used to this day and can be used as a signature [27] [28].
Here are some of them that LEXFO found in a very recent sample:

```
Poluchit
Nachalo
ssylka
ustanavlivat
kliyent2podklyuchit
```

These strings were used in combination with commercial Russian packers to try and fool researchers and journalists, at a time where they are often too quick to attribute attacks to Russian groups.

Malware design

Lazarus malware usually have the following patterns:

- Multistaged
- Command-line malware and tools
- Designed to be run as Svchost services (for persistence) API are loaded dynamically

Lazarus developers usually forget to strip the PDB path from compiled binaries, even when they disclose valuable information such as what the malware does, its goal, or even the developer's name.

Communications

Lazarus malware often use a communication protocol that has been named "Fake TLS" [29] [30] for communications. This protocol makes malicious packets look like legitimate TLS handshakes and communications might stay under the radar due to heavy TLS traffic on port 443.

This protocol can be found in most Lazarus malware. It is however hard to detect with Snort and Suricata rules considering the huge stream of TLS/SSL packets to monitor, which explains why it has been consistently used for years by the attackers.

Example of a Lazarus Fake-TLS packet:

0000	17 03 01 00 30 5d 15 3d a2 40 ef d2 01 25 ca 54	....0].=.@...%.T
0010	26 5f 5d b0 d2 2f 2f 6d 2d ec 56 85 b0 4c a9 bf	&_]..//m-.V..L..
0020	eb 97 be 31 ad cd de 3a b4 71 1e c8 53 96 0b 2d	...1...:.q..S..-
0030	c3 91 3d a2 15	..=..

A legitimate TLS packet would be structured this way:

Bytes			Meaning
17			ApplicationData protocol type
03	01		SSL version (TLS 1.0)
00	30		Message length (48 bytes)
5d	...	15	Encrypted application data

In case of a Lazarus fake-TLS packet, the structure is:

Bytes			Meaning
17	03	01	Fake TLS header
00	10		Size of next packet < 0x4000

The first packet is a fake-TLS handshake sent to the C&C server:

```
0000  17 03 01 00 04 .....
```

Data are then encrypted using algorithms and/or keys different for each malware, usually relying on XOR operations or standard algorithms such as RC4.

Different and more standard communication protocols have been used by Lazarus. Simple HTTP requests with hardcoded URLs were implemented in some cases where attackers didn't care too much about detection.

Here is an example of a Lazarus HTTP request:

```
GET /sub/lib/lib.asp?id=dn678 HTTP/1.1 Accept: */*
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.1; WOW64; Trident/4.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0)
Host: www.secuvision.co.kr Connection: Keep-Alive
```

– Filenames

For payloads and modules, filenames are generally inspired by legitimate Windows services and end with "xxxsvc(.dll|.exe)":

```
swpsvc.dll
sppsvc.dll
sqcsvc.dll
gpsvc.exe
uploadmgrsvc.dll
wmisecsvc.dll
...
```

Lazarus has been using `[filename].tmp` and `[filename].dat` filenames for configurations or to store data to be sent to the C&C. Recently, they started using configuration files named `[filename].dll.mui` .

– Persistence

Persistence is usually achieved by setting the main payload as an `AUTO_START` svchost service, which means the malware will be loaded each time the user session starts through the command `svchost -k [service]` .

Packers

As Lazarus reuse a lot of code in their malware, they manage to evade detection by signature using free and commercial packers. Here is a list of the main packers encountered:

→ UPX
→ VMProtect
→ Themida
→ Armadillo
→ ASPack
→ Enigma
→ Protector

Third-party libraries

Lazarus uses statically linked third party libraries in their malware for communications and TLS/SSL implementation. The following libraries were seen:

→ Libcurl (version 7.49.1)
→ mbedTLS / PolarSSL
→ wolfSSL

The Libcurl library with the same exact version is still being used in the most recent Lazarus samples. To compress data, Lazarus developers usually use inflate/deflate lib versions 1.1.3 and 1.1.4 as well as Zlib version 1.0.1 and 1.2.7.

Third-party tools

Lazarus has its own toolbox, but operators will also use third-party legitimate tools when necessary. They mostly include credential-gathering tools and software allowing lateral movements. Attackers will pack tools that are widely flagged by anti-virus, such as Mimikatz, to evade signature-based detection.

The list of third-party tools includes:

→ PsExec
→ Mimikatz
→ FreeRDP
→ SC.exe
→ Net.exe
→ ...

Encryption

The Lazarus group uses standard and custom encryption algorithms. Custom algorithms are usually based on several XOR operations with constant values, while standard ones are common such as RC4, AES and DES.

They will sometimes use exotic ciphers like Spritz, an RC4-like algorithm they implemented in a set of malware described by Kaspersky. They have the bad habit of reusing encryption algorithms and keys in different malware, which helps detection and attribution.

Lazarus uses encryption for communications, hiding dynamically-imported API names to avoid heuristics and to encrypt their payloads. For the latter, they also use less sophisticated ways of hiding strings, such as Base64 encoding and alphabet substitution.

We will review some encryption algorithms found in samples below.

– XOR-based algorithms

Most custom algorithms implemented by Lazarus are based on XOR operations with hardcoded keys. While most of them are pretty straightforward to understand, some are quite imaginative. Here are some algorithms and keys found in multiple Lazarus samples:

Algorithm	Key	Campaign / Malware
XOR	0xA7	Blockbuster
XOR	0x9E	Lazarus downloader
XOR	0x23	FASTCash
XOR	QzEc , wPof	Attack on Taiwanese banks + LEXFO incident response
XOR-based	0xF4F29E1B	Lazarus under the hood
XOR-based	0xCBF9A345	Lazarus under the hood
XOR-based	0x4F833D5B	Lazarus under the hood
XOR-S^	/	Phandoor (Troy)
XOR-1FE	/	Phandoor
XOR-7F8	/	asdfdoor, FBIRat, Passive backdoor
XOR-FFFFFFF0	/	Rifle

Below is an example of a custom XOR-based encryption algorithm using hardcoded keys and constants found in several Andariel samples.

```
lpBuffer = buff;
LOBYTE(key4) = 0x82u;
v13 = buff;
key3 = 5;
key1 = 0x556F9482;
key2 = 0xAFC12058;
if ( (signed int)dwSize > 0 ) {
  offset = encryptedBuffer - (char *)lpBuffer; i = dwSize;
  do {
    *lpBuffer = key3 ^ key2 ^ key4 ^ lpBuffer[offset]; key3
    = key3 & key2 ^ key4 & (key3 ^ key2);
    key4 = ((((unsigned_int16)key1 ^ (unsigned_int16)(8 * key1)) & 0x7F8) << 20) | (key1 >> 8); key2
    = (((key2 << 7) ^ (key2 ^ 16 * (key2 ^ 2 * key2)) & 0xFFFFFF80) << 17) | (key2 >> 8);
    ++lpBuffer;  nbBytesLeft
    = i-- == 1;
    key1 = ((((unsigned_int16)key1 ^ (unsigned_int16)(8 * key1)) & 0x7F8) << 20) | (key1 >> 8);
  }
  while ( !nbBytesLeft );
  lpBuffer = v13;
  size = dwSize;
}
```

A good example of code reuse is the “S^” algorithm (S-hat) recently seen in many Andariel malware compiled in 2016/2017. We found traces of the same algorithm in samples used in Operation Troy, compiled in 2010 and 2011 in a payload named bs.dll [31].

– RC4

The RC4 algorithm is found in a number of malware, as it is easy and quick to implement. Lazarus developers will sometimes modify it lightly and double the PRGA part of the algorithm to confuse analysts.

Below are some of the hardcoded keys found in samples [32].

Hardcoded key

```
4E 38 1F A7 7F 08 CC AA 0D 56 EF F9 ED 08 EF
E2 A4 85 92
f9 65 8b c9 ec 12 f9 ae 50 e6 26 d7 70 77 ac 1e
53 87 F2 11 30 3D B5 52 AD C8 28 09 E0 52 60 D0 6C C5 68 E2 70 77 3C 8F 12 C0 7B 13 D7 B3 9F 15
```

AES

The AES algorithm was found in many Lazarus samples: Electricfish, backdoors involved in India attacks, Joanap, various Bluenoroff samples...

Spritz

The Spirtz encryption algorithm is not as common as the others but was used by Lazarus by one of their loaders to decrypt payloads. The key found was:

Hardcoded key

```
6B EA F5 11 DF 18 6D 74 AF F2 D9 30 8D 17 72 E4 BD A1 45 2D 3F 91 EB DE DC F6 FA 4C 9E 3A 8F 98
```

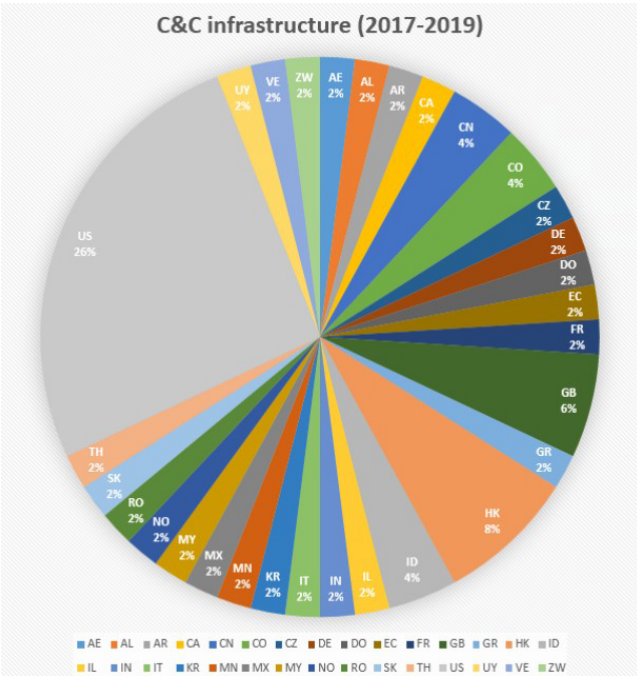
C&C Architecture

Lazarus uses a standard C&C architecture with several layers of proxy servers. These proxies will relay packets from the operators to the implants or the other way around through fake TLS packets.

According to a Group-IB investigation [33], operators set up a three-layer architecture using non standard ports.

Domains and servers are usually leased in Asian countries and paid with bitcoins or other cryptocurrencies for anonymity. Lazarus used to leverage hacked servers for their C&C infrastructure but recent attacks show that they have moved away from it.

From a geographic point of view, most C&C appear to be hosted in the US and in Asian countries. The diagram below shows locations of more than 50 C&C that have been used by Lazarus in different attacks the past two years.



Lazarus C&C by country

MITRE ATT&CK MATRIX

– Techniques used

The ATT&CK matrix [34] related to Lazarus clearly shows how active and diverse the group is.

Initial Access	Execution	Persistence	Privilege Escalation	Defense Evasion	Credential Access
Drive-by Compromise	Command-Line Interface	Account Manipulation	Access Token Manipulation	Access Token Manipulation	Account Manipulation
Spearphishing Attachment	Compiled HTML File	Bootkit	New Service	Compiled HTML File	Brute Force
	Exploitation for Client Execution	Hidden Files and Directories	Process Injection	Connection Proxy	Credential Dumping
	Scripting	New Service		Disabling Security Tools	Input Capture
	User Execution	Registry Run Keys / Startup Folder		File Deletion	
	Windows Management Instrumentation	Shortcut Modification		Hidden Files and Directories	
				Obfuscated Files or Information	
				Process Injection	
				Scripting	
				Timestomp	

Discovery	Lateral Movement	Collection	Command And Control	Exfiltration	Impact
Application Window Discovery	Remote Desktop Protocol	Data from Local System	Commonly Used Port	Data Compressed	Data Destruction
File and Directory Discovery	Remote File Copy	Data Staged	Connection Proxy	Data Encrypted	Disk Content Wipe
Process Discovery	Windows Admin Shares	Input Capture	Custom Cryptographic Protocol	Exfiltration Over Alternative Protocol	Disk Structure Wipe
Query Registry			Data Encoding	Exfiltration Over Command and Control Channel	Resource Hijacking
System Information Discovery			Fallback Channels		Service Stop
System Network Configuration Discovery			Multiband Communication		System Shutdown/ Reboot
System Owner/ User Discovery			Remote File Copy		
System Time Discovery			Standard Application Layer Protocol		
			Standard Cryptographic Protocol		
			Uncommonly Used Port		

– Software

Similary, they have been using the set of software below (list is not exhaustive) [35]:

ID	Name
S0347	AuditCred
S0245	BADCALL
S0239	Bankshot
S0181	FALLCHILL
S0246	HARDRAIN
S0376	HOPLIGHT
S0271	KEYMARBLE
S0002	Mimikatz
S0108	netsh
S0238	Proxysvc
S0241	RATANKBA
S0364	RawDisk
S0263	TYPEFRAME
S0180	Volgmer
S0366	WannaCry

IV. INCIDENT RESPONSE: HOW TO UNCOVER AN ONGOING LAZARUS ATTACK

CONTEXT

In late December 2018, LEXFO was contacted by a company following multiple infections. The company was alerted of outgoing malicious traffic to a known Lazarus C&C that was being monitored.

About 5 machines were identified as infected in the network at the time. LEXFO immediately asked for RAM and disk dumps of the infected systems, as well as all captured encrypted traffic and began investigating.

FIRST ASSESSMENT

LEXFO was provided with several RAM and disk dumps of the infected machines and three binaries as well as a configuration file and a batch installer, found on the computers and believed to have been used by the attackers.

Filename	Type	Size
`igfx.exe`	PE32+ executable (GUI) x86-64	260K
`sqcsvc.dll`	PE32+ executable (DLL) (GUI) x86-64	2,6M
`sqcsvc.dll.mui`	Data	236
`svc.bat`	Batch script	643

A quick look at the batch script revealed that its purpose was to deploy and install the RAT payload `sqcsvc.dll` and its encrypted configuration. The script also takes care of installing a persistent service named `sqcsvc`.

`svc.bat` installer script content:

```
mkdir "c:\programdata\microsoft\sqcsvc"
move "c:\perflogs\1.dat" "c:\programdata\microsoft\sqcsvc\sqcsvc6.ldx"
move "c:\perflogs\1.dll" "c:\windows\system32\sqcsvc.dll"
move "c:\perflogs\1.dll.mui" "c:\windows\system32\sqcsvc.dll.mui"
sc create sqcsvc binPath= "%SystemRoot%\System32\svchost.exe -k sqcsvc" start= auto reg add "HKLM\SYSTEM\ControlSet001\Services\sqcsvc\Parameters"
reg add "HKLM\SYSTEM\ControlSet001\Services\sqcsvc\Parameters" /v ServiceDll /t REG_EXPAND_SZ /d "%SystemRoot%\System32\sqcsvc.dll"
reg add "HKLM\Software\Microsoft\Windows NT\CurrentVersion\Svchost" /v sqcsvc /t REG_MULTI_SZ /d sqcsvc
```

Then, LEXFO started reverse-engineering the `sqcsvc` payload that was found in the RAM and disk dumps in order to assess the attackers' capabilities and find the decryption algorithm for communications.

ATTRIBUTING THE ATTACK

Our classifier tool didn't show any strong link with other Lazarus samples, as the payloads we found were part of the new arsenal of Lazarus at the time. The only link found is a Yara rule match between IGFX and a sample involved in a Lazarus heist in Taiwan (`9a776b895e93926e2a758c09e341accb9333edc1243d216a5e53f47c6043c852`). The rule matched strings from the static library libcurl with the specific version `7.49.1` . We had to investigate further to confirm.

Filenames match Lazarus' habits, as we have the payload named "*svc.dll", its encrypted configuration file as a MUI-disguised file and a batch script to install the malware. The payload is also made persistent by registering it as a service, which is how Lazarus usually operate.

Looking closely at the SQCSVC payload metadata, we can see that its original name was `sock_64.dll` , the compilation timestamp is `Sat, 03 Nov 2018 00:47:21 UTC` which is consistent with North Korea working hours (UTC+9) and that it was packed using Themida Code-Virtualizer.
At that point, Lazarus can already be considered the #1 suspect.

Filename	Compilation timestamp
sqcsvc.dll	Sat, 03 Nov 2018 00:47:21 UTC
igfx.exe	Mon, 02 May 2016 03:24:39 UTC
hs.exe	Mon, 01 Oct 2018 10:30:58 UTC
iehelp.exe	Mon, 24 Sep 2018 11:12:22 UTC
iehelp2.exe	Wed, 14 Nov 2018 14:02:19 UTC
swpsvc.dll	Sat, 11 Aug 2018 14:14:54 UTC

UNCOVERING ATTACKERS' ACTIVITIES

Having reverse-engineered the communication protocol and the encryption algorithm, LEXFO started developing a Python implementation to decrypt packets.

Here is the identified decryption function in the SQCSVC payload:

```
charpos = 0i64;
if ( datalen > 0 )
{
  do
  {
    car = data[charpos];
    i = 1870;
    do
    {
      k = i % 256;
      i += 187;
      car = (k ^ car) - k;
    }
    while ( i < 5610 );
    data[charpos++] = car;
  }
  while ( charpos < datalen );
}
```

XOR decryption stub

A Python implementation is quite straightforward:

```
def decryptTCPData(data):
    output = ''
    i = 0
    j = 0

    while j < len(data):
        i = 5423
        car = ord(data[j])
        while i >= 1870:
            k = i % 256 i
            -= 187
            car = (k ^ (car + k)) &
        0xFF j += 1
        output += chr(car)

    return output
```

From there, we were able to write a script to automatically decrypt all traffic in the PCAP files exchanged between the implant and the Lazarus C&C.

```
---- 03:34:44.251294 ethertype IPv4, IP ddd.ddd.ddd.ddd.443 > sss.sss.sss.sss.53477: Flags [P.], seq 1459:1557, ack 1459, win 511, length 98
'0000000B\x0230\x02"cmd.exe" /c "ping -n 1 XXXROOM0099"\x02'
---- 03:34:47.465025 ethertype IPv4, IP sss.sss.sss.sss.53477 > ddd.ddd.ddd.ddd.443: Flags [P.], seq 1477:1943, ack 1557, win 256, length 466
'458\x02\r\n
Pinging XXX.xxx.org [10.xxx.xxx.xxx] with 32 bytes of data:\r\n Reply from 10.xxx.xxx.xxx: Destination host unreachable.\r\n
\r\n
Ping statistics for 10.xxx.xxx.xxx:\r\n
Packets: Sent = 1, Received = 1, Lost = 0 (0% loss),\r\n
'
```

Other manually executed commands:

```
"cmd.exe" /c "time /t"
"cmd.exe" /c "echo 1000 > c:\\windows\\temp\\tmp1105.tmp"
"cmd.exe" /c "type "c:\\windows\\temp\\tmp1105.tmp""
"cmd.exe" /c "type "C:\\Windows\\Temp\\temp0917.tmp""
"cmd.exe" /c "type C:\\Windows\\Temp\\TMP0389A.tmp"
"cmd.exe" /c "dir "c:\\windows\\temp\\tmp1105.tmp""
"cmd.exe" /c "echo 1000 > c:\\windows\\temp\\tmp1105.tmp"
"cmd.exe" /c "dir c:\\windows\\temp\\tmp1105.tmp" "cmd.exe"
/c "type c:\\windows\\temp\\tmp1105.tmp" "cmd.exe" /c "type
C:\\Windows\\Temp\\TMP0389A.tmp" "cmd.exe" /c "type
C:\\Windows\\Temp\\temp0917.tmp" "cmd.exe" /c "type
c:\\windows\\temp\\tmp1105.tmp" "cmd.exe" /c "dir
c:\\windows\\temp\\tmp1105.tmp" "cmd.exe" /c "type
C:\\Windows\\Temp\\tmp1105.tmp" "cmd.exe" /c "type
C:\\Windows\\Temp\\temp0917.tmp" "cmd.exe" /c "type
C:\\Windows\\Temp\\temp0917.tmp" "cmd.exe" /c "ping -n 1 XXXROOM0099"
"cmd.exe" /c "ping -n 1 XXXROOM0099"
"cmd.exe" /c "time /t"
"cmd.exe" /c "type "C:\\Windows\\Temp\\TMP0389A.tmp""
"cmd.exe" /c "query user"
"cmd.exe" /c "query user"
```

The Lazarus operators also leveraged the RAT to get information on the infected machines, using the directory and process listing feature of `SQCSVC` . We decrypted many fragmented packets exfiltrating folders and files as well as running processes.
In some other captures, we saw that the attackers were checking the state of a service named `swpsvc` . This name if consistent with other Lazarus malware such as the first payload `sqcsvc` , makes it very suspicious.

```
"[SC] EnumQueryServicesStatus:OpenService \x1a chec(s) 1060 :\r\n
\r\n
Le service sp\x1a cifi\x1a n'existe pas en tant que service install\x1a .\r\n
\r\n "
'\r\n
SERVICE_NAME: swpsvc \r\n
TYPE                      : 30  WIN32  \r\n
STATE                     : 4  RUNNING \r\n
                          (STOPPABLE, PAUSABLE, ACCEPTS_SHUTDOWN)\r\n
WIN32_EXIT_CODE           : 0  (0x0)\r\n
SERVICE_EXIT_CODE         : 0  (0x0)\r\n
CHECKPOINT                : 0x0\r\n
WAIT_HINT                 : 0x0\r\n
'
'[SC] DeleteService r\x1a ussite(s)\r\n'
```

The Lazarus operators deleted this file when they realized that the company security team was investigating. Fortunately, they failed to delete it safely and LEXFO managed to recover the `swpsvc.dll` using carving tools.

This payload appeared to be a stage 1 RAT with a similar communication protocol.

Further investigations of the decrypted PCAP files also revealed two other DLL plugins that were sent and written to disk by the attackers: an injector performing payload injection in the `explorer.exe` process, and a keylogger / screencapper. Both these plugins were unknown at the time.
We provided the client with newly-made YARA rules to detect all discovered payloads as well as a PowerShell script to automate the deployment process. We also implemented Suricata rules to detect the Lazarus fake- TLS and custom protocol traffic that can be used along with our Python script to decrypt the packets. This successfully stopped the attack and helped identify all infected machines.

PAYLOAD ANALYSIS

– IGFX tool

This binary was compiled on `Monday, May 02 05:24:39 2016 UTC`. This sample appeared to be a version of the Lazarus tool `Client_TrafficForwarder` described by Group-IB !REF.

This tool's purpose is to forward traffic to another infected host in order to relay operators' commands.

One interesting particularity of this tool is that the Lazarus developers used non-native Russian strings for command names, trying to confuse attribution:

```
13F428D00 aKliyent2podkly db 'kliyent2podklyuchit',0
13F428D00                                        ; DATA XREF: cnc_sendCommand+2E↑r
13F428D14 aSsylka         db 'ssylka',0          ; DATA XREF: thread_comm_ssylka+6↑o
13F428D1B                 align 20h
13F428D20 ; char Str2[]
13F428D20 Str2            db 'ustanavlivat',0    ; DATA XREF: cnc_comm_loop+180↑o
13F428D2D                 align 10h
13F428D30 ; char aPoluchit[]
13F428D30 aPoluchit       db 'poluchit',0        ; DATA XREF: cnc_comm_loop:loc_13F2773↑o
13F428D39                 align 20h
13F428D40 ; char aPereslat[]
13F428D40 aPereslat       db 'pereslat',0        ; DATA XREF: cnc_comm_loop:loc_13F2807↑o
13F428D49                 align 10h
13F428D50 ; char aDerzhat[]
13F428D50 aDerzhat        db 'derzhat',0         ; DATA XREF: cnc_comm_loop:loc_13F2906↑o
13F428D58 ; char aVykhodit[]
13F428D58 aVykhodit       db 'vykhodit',0        ; DATA XREF: cnc_comm_loop+4C1↑o
13F428D58                                        ; cnc_comm_loop+4DC↑r ...
13F428D61                 align 8
13F428D68 aNachalo        db 'Nachalo',0         ; DATA XREF: cnc_comm_start:loc_13F29E0↑o
13F428D70 asc_13F428D70   db ' ',0               ; DATA XREF: XOR_cEzQfoPw+21↑o
```

Translated Russian strings to mess with attribution

This binary was compiled with a static version of libcurl v7.49.1, which is common amongst Lazarus' samples.

– SQCSVC RAT

This binary was compiled on `Saturday Nov 03 01:47:21 2018 UTC`.

```
// strConfFile = "C:\Windows\system32\sqcsvc.dll.mui"
file_read((__int64)strConfFile, (__int64)lpBuffer, filesize);
v38 = 0i64;
v39 = 0;
v40 = 0;
v41 = 0;
v37 = 0;
if ( filesize )
{
  i = filesize - 1;
  if ( filesize != 1 )
  {
    lpByte = &lpBuffer_[i];
    do
    {
      byte = *(lpByte-- - 1);
      lpByte[1] ^= byte;
      --i;
    }
    while ( i );
  }
}
sizetodecrypt = filesize - 32;
RC4((__int64)lpBuffer_, (__int64)(lpBuffer_ + 32), sizetodecrypt);
MD5_hash((__int64)&v37, (__int64)(lpBuffer_ + 32), sizetodecrypt);
```

SQCSVC configuration decryption

`sqcsvc.dll.mui` decrypted configuration:

0000	31	00	38	00	38	00	30	00	30	00	35	00	37	00	35	00	1.8.8.0.0.5.7.5.
0010	34	00	32	00	02	00	36	00	35	00	35	00	32	00	30	00	4.2...6.5.5.2.0.
0020	02	00	31	00	35	00	31	00	31	00	31	00	36	00	31	00	..1.5.1.1.1.6.1.
0030	30	00	35	00	37	00	30	00	37	00	38	00	32	00	39	00	0.5.7.0.7.8.2.9.
0040	36	00	37	00	39	00	32	00	02	00	6d	00	65	00	6d	00	6.7.9.2...m.e.m.
0050	62	00	65	00	72	00	2e	00	69	00	74	00	65	00	6d	00	b.e.r...i.t.e.m.
0060	64	00	62	00	2e	00	63	00	6f	00	6d	00	3a	00	34	00	d.b...c.o.m.:.4.
0070	34	00	33	00	02	00	31	00	38	00	30	00	2e	00	32	00	4.3...1.8.0...2.
0080	33	00	35	00	2e	00	31	00	33	00	32	00	2e	00	32	00	3.5...1.3.2...2.
0090	30	00	36	00	3a	00	34	00	34	00	33	00	02	00	20	00	0.6.:.4.4.3... .
00a0	02	00	20	00	02	00	20	00	02	00	20	00	02	00	20	00	.. ..
00b0	02	00	30	00	02	00	36	00	30	00	02	00	30	00	02	00	..0...6.0...0...
00c0	36	00	35	00	35	00	32	00	31	00	02	00					6.5.5.2.1...

This configuration contains two C&C addresses: `member.itemdb.com` and `180.235.132.206`, both to be contacted on port 443, which is consistent with the Fake-TLS protocol implemented.
The payload is packed using a powerful virtualization-based packer called Themida Code-Virtualizer. However, the attackers did not use the packer correctly and the non-obfuscated payload code can be dumped easily from memory.

According to BinDiff, the non-obfuscated payload code is up to 65% similar to the code of the `IGFX.exe` tool used by the attackers, compiled two years prior to `SQCSVC`, proving that they probably come from the same developer team or the same code base.
The `SQCSVC` payload is able to:

- Download and write files on disks
- Execute files or bash commands
- Inject code in a running process
- Listen to commands on a specified port (server mode)
- Rewrite the configuration file with new values

The payload was similar to the one described by TrendMicro after a Lazarus bank heist in Latin America in November, 2018 [36].

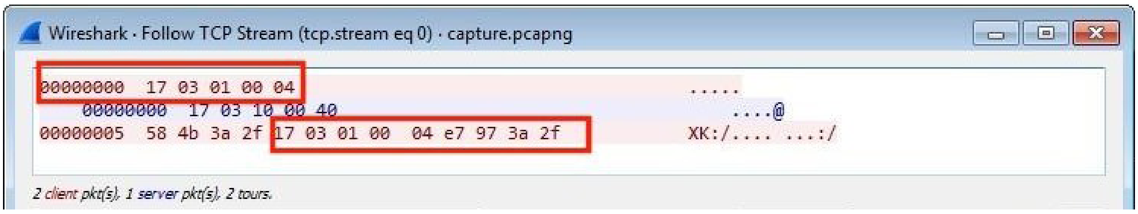
– SWPSVC (Stage 1)

The analyzed malware sample of the group Lazarus is a “stage 1” reconnaissance malware which implements Remote Administration Tool features.

The analyzed sample is a DLL library which is loaded by the `svchost` service, as it is registered as an `AUTO_START` service for persistence. The delivery method is most likely manual. In such case, the attacker drops the malware on an already compromised machine.
The malware configuration is stored encrypted in the registry, unlike most Lazarus malware that come with an encrypted file as configuration. In our case, the configuration data could not be retrieved as it was fully erased before the investigation began by the attackers that didn’t need this component anymore since the `SQCSVC` RAT was installed.

The malware uses different kind of encryption for different kind of purposes. The first substitution-based encryption is used for decrypting encrypted strings in the static binary. The XOR-based encryption is used to obfuscate communications between the server and the client and to decrypt configuration content such as the Command-and-Control (C&C) server name and port number stored in the Windows registry.

This RAT uses the already mentioned Fake-TLS protocol for communications:



Fake-TLS handshake sent by the RAT

The following commands are implemented in the RAT:

Command ID	Description
`0x19283746`	Get information on the infected system (processor architecture, network interfaces...)
`0x1928374C`	Write file on system
`0x1928374A`	Read file on system
`0x1928374F`	Delete file
`0x1928374F`	Get process info
`0x19283753`	Kill process
`0x1928374D`	Create process
`0x19283756`	Execute process as a given user
`0x19283748`	List files in a directory
`0x19283755`	Modify C&C configuration by changing the value in Windows registry
`0x19283747`	List local drives and network shares
`0x19283750`	Move file

– Downloaded modules

LEXFO found two downloaded modules in the decrypted packets that were deployed on specific targets.

The first one is an injector that takes a file path as a parameter and injects it in an `explorer.exe` process. The injected file is executed in a new thread. This injector uses RC4 encryption with the hardcoded key `key` to hide suspicious strings that are decrypted at runtime, and will write some log data to the file `C:\windows\temp\temp0917.tmp`.

The second module is a keylogger and screencapper. This file is a DLL originally named `capture_x64.dll` by the attackers. The keylogging and screencapping features are implemented standardly

V. CLASSIFYING NORTH KOREAN MALWARE AND INTERPRETING LINKS

DATASET

We gathered more than 290 malware attributed to North Korea from various sources:

- → Twitter
- → Various RE and malware forums
- → VirusTotal (Hunting)
- → Online sandboxes (HybridAnalysis, Any.RUN...)
- → Malware repository (VirusBay, VirusShare, Malshare)
- → U.S. Cyber command malware uploads
- → Threat intelligence reports
- → LEXFO's own incident responses

We ended up with the following families:

Malware family	
apt38_contopee	polishbanks
powerratankba	joanapbrambul
nukesped	bankshot
killdisk	mydoom

cybercom	karbarcobra
apt37_summit	hoplight
apt37_humanrights	bitcoin
apt37	fastcash
blockbuster_sequel	golddragon
redbanc	kimsuky_shark2
keymarble	sony
darkhotel	redgambler
apt37_rocketman	typeframe
safebank	troydarkseoul
fallchill	kimsuky
electricfish	dtrack
hermesryuk	sharpshooter
ratankba	intezer
apt37_evilnewyear	bangladesh_swift
volgmer	backswap
wipall	wannacry
hiddencobra	duuzer
kimsuky_mysterybaby	deltacharlie

Malware family	
andariel_rifle	apt38_dyepack
sony_sierraalfa	ghostsecret
apt38	blockbuster_continues
sony_kordllbot	vietnam
kimsuky_stolenpencil	taiwan
applejeus_loader	Lazarus under the hood (Kaspersky)

Samples were compiled from 2004-05-23 to 2019-10-22 according to compilation timestamps that seemed legitimate.

METHODOLOGY

After several manual analyses of Lazarus samples, we concluded that the following links where relevant:

– Idenfifying links

After several manual analysis of Lazarus samples, we figured that the following links where relevant:

– Standard links:

→ Code reuse (Fuzzy hashes SSDEEP + MACHOKE)
→ Import hashing
→ Timestamps PDB

– Advanced links:

→ Rich Headers
→ Yara signatures (see next part)

A word on Rich headers

Rich headers are added to standard PE headers in executables compiled using VisualStudio. It is a fingerprint of the compilation environment that can be easily decrypted and decoded. It can then be used to identify if binaries were compiled in the same environment, which is a strong relation. As we empirically saw that North Korean groups have been using VisualStudio almost exclusively, and there is a high chance that their malware-building infrastructure is quite conservative, we chose to develop a script to parse rich headers from samples and included it as a relation link in our classifier.

Building Yara rules

For each North-Korean malware family we identified, we built Yara rules in order to keep signatures of the following implementations that are likely to be reused by Lazarus:

- Specific strings
- Cryptographic algorithms
- Cryptographic keys
- Unique implementations of features:
  - mapping of files
  - lateral movement
  - installing service
  - wiper implementation
  - handling logs
  - ...
- Way of dynamically loading API
- Obfuscation
- ...

We also built rules for statically linked library like OpenSSL, libcurl, ZIP etc. of specific versions, as Lazarus was seen to be pretty conservative in using the same versions over the years. Those rules were named `lib_static_[lib name]_[version]` and we attributed them a lesser weight than implementation rules as it doesn't illustrate a strong enough link between two samples.

We built a set of about 100 rules that we ran on our sample dataset. To our own Yara rule set, we added auto-generated rules from Malpedia [37] when they showed accurate results.
Before adding them to our ruleset, we ran tests on a huge malware set to make sure that the rules were accurate and there were close to zero false positive.

Building similarity profiles

We produced a profile for each sample with fuzzy hashes, decoded rich header, compilation timestamps and matching Yara rules. We then compared profiles using nearest neighbor algorithms with weight we empirically tested to get the best results. Jaccard distance was used to compare fuzzy hashes. We attributed heavy weights for identical rich headers encountered in different samples and for every non `lib_static_*` Yara rule matches. Weights (W) were roughly according to this order relation:

```
W(Exact same Rich header) > W(Yara match (non lib_static_*)) > W(Machoke code
reuse) > W(Compilation timestamp) > W(Yara match (lib_static_*)) > W(Rich header
similarity) > W(Imphash) > W(Various metadata)
```

Handling packers

Non-specific packers like UPX are handled separately: as fuzzy hashes become irrelevant, we dismissed them when computing weights. For more specific packers (Themida, Enigma...), we built Yara rules to identify them and considered them as a valid relation of similarity between samples since Lazarus uses specific versions of those packers.

When possible, we reversed the packed samples and tried to get clean unpacked executables so our tool could classify them indiscriminately and accurately.

Result review and improvements

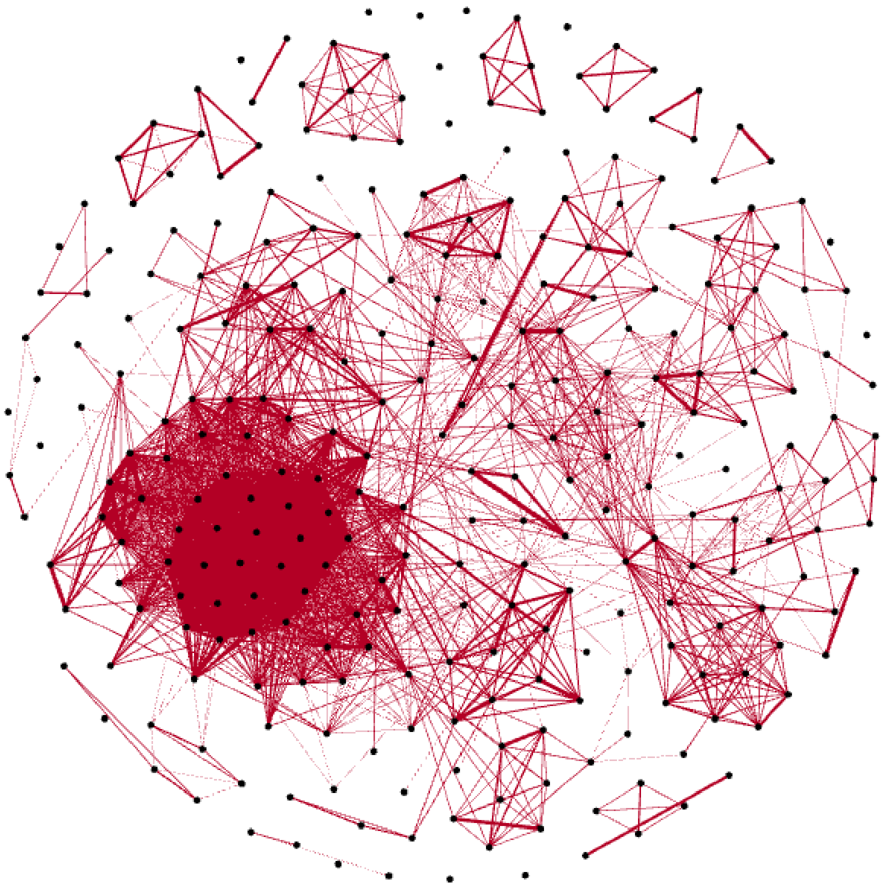
We ran our tool multiple times and tried to analyze samples that seemed to be oddly placed or unique. We reversed each of them and adapted our classification methodology and criteria according to our findings, and ran the test again and iteratively re-applied this process until the classification was accurate enough.

Building the graph

From there, we created a graph with samples as nodes and weighted links as relations between them.

VISUALIZATION

We used the Fruchterman-Reingold spacialisation to visualize links and identify clusters. We ended up with the following constellation, where each dot is a sample and each link represents the strength of a relation between two samples:



Fruchterman-Reingold spacialisation applied to our relation graph

REVIEWING RESULTS

Our tool revealed more than 2500 actual links between around 290 samples, which showcases that such a classification is relevant as Lazarus samples are rarely unique in a 10 years span period. We see clear clusters and many overlaps. This will shed some light on North Korean malware and groups, as the number of reports and campaign names grows and it can be hard to keep track and attribution is often confusing or unsure.

Kimsuky

The Kimsuky group has its own cluster but we see Rich header and compilation timestamp overlaps with other Lazarus samples. Kimsuky and Lazarus are therefore likely to be working together, which is confirmed by the fact that Kimsuky malware were found on Lazarus targets several times.

Moreover, our tool revealed links between DTrack samples from the Kudankulam Nuclear Power Plant (KNPP) and Kimsuky samples: both use SQlite as a statically-linked library, but a different version (compiled on `2017- 10-24 18:55:49` for the latter vs `2017-02-13 16:02:40` for the former). Looking at compilation timestamps, we can see that some DTrack and Kimsuky samples were also compiled the same day (or close to) as other Lazarus malware used in campaigns:

```
(2019-07-29 13:36:26) ./dtrack/npp_
bfb39f486372a509f307cde3361795a2f9f759cbeb4cac07562dcbaebc070364
<- Timestamp -> (2019-07-29 07:08:01) ./andariel_rifle/javaupdatemain_unpack.
exe
(2019-03-01 00:07:25) ./dtrack/
npp_3cc9d9a12f3b884582e5c4daf7d83c4a510172a836de90b87439388e3cde3682
<- Timestamp -> (2019-03-01 09:08:44)
./kimsuky_
shark2/4b3416fb6d1ed1f762772b4dd4f4f652e63ba41f7809b25c5fa0ee9010f7dae7.bin
```

This could mean that the groups are working together for some operations, with Lazarus doing the intrusion and handing the exfiltrating part to Kimsuky when the target matches their interest.

Finally, interesting findings stand out when looking at Rich header similarities. The Kimsuky stolen pencil sample has the exact same Rich header as samples found in Lazarus campaigns such as DarkSeoul and GoldDragon.

```
(2018-12-21 00:34:35) ./kimsuky_stolenpencil/1.bin
```

<-	Rich	->	(2012-07-06	12:24:18)	./troydarkseoul/DarkSeoul/DarkSeoul_50E-03200C3A0BECBF33B3788DAC8CD46
<-	Rich	->	(2012-07-06	12:24:18)	./troydarkseoul/DarkSeoul/DarkSeoul_E4F66C-3CD27B97649976F6F0DAAD9032
<-	Rich	->	(2013-01-31	10:27:18)	./troydarkseoul/DarkSeoul/DarkSeoul_5FCD6E-1DACE6B0599429D913850F0364
<-	Rich	->	(2013-01-31	10:27:18)	./troydarkseoul/DarkSeoul/DarkSeoul_0A8032CD-6B4A710B1771A080FA09FB87
<-	Rich	->	(2013-01-31	10:27:18)	./troydarkseoul/DarkSeoul/DarkSeoul_DB4BBD-C36A78A8807AD9B15A562515C4
<-	Rich	->	(2013-01-31	10:27:18)	./troydarkseoul/DarkSeoul/DarkSeoul_F0E045210E3258DAD91D7B6B4D64E7F3
<-	Rich	->	(2017-12-24	08:16:57)	./golddragon/e68f43ecb03330ff0420047b61933583b4144585
<-	Rich	->	(2017-12-24	08:47:21)	./golddragon/4f58e6a7a04be2b2ecbcdc-bae6f281778fdbd9f9
<-	Rich	->	(2017-12-24	08:29:04)	./golddragon/11a38a9d23193d9582d02ab0eae-767c3933066ec
<-	Rich	->	(2017-12-24	08:37:57)	./golddragon/3a0c617d17e7f819775e48f7ede-fe9af84a1446b
<-	Rich	->	(2017-12-24	08:44:08)	./golddragon/bf21667e4b48b8857020ba455531c-9c4f2560740

– DarkHotel

Samples attributed to the DarkHotel group have identical Rich header as a lot of APT38 Nukesped samples, which is a strong link:

```
(2011-04-07 06:58:03) ./
darkhotel/2b6288bbd81bb9d666c3a0372f26ede47c8c9ff11c604307982d51654fb9e850.ViR
<- Rich -> (2017-07-14 22:40:25) ./cybercom/d2da675a8adfef9d0c146154084fff62.bin
<- Rich -> (2017-07-11 18:26:59) ./nukesped/3EDCE4D49A2F31B8BA9BAD0B8EF54963
<- Rich -> (2017-08-11 05:03:45) ./cybercom/2a791769aa73ac757f210f8546125b57.bin
<- Rich -> (2017-08-01 16:39:36) ./ghostsecret/Sample_5ae56e2077d7dc0d380c3bfd_
exe
...
```

V. Classifying North Korean malware and interpreting links

Though DarkHotel TTPs and malware are different from Lazarus, those groups seem to be working in tandem.

Andariel subgroup

Clear Andariel clusters stand out, with malware involved in operations Red Gambler and Rifle. Those samples are closely linked with each other by specific and custom cryptographic algorithms found in malware from both operations. For instance:

```
yara_andariel_7F8: (2016-04-21 10:41:15)
./andariel_rifle/
d246669cf1e25860f8601e456edd7156aa7304026ff4eadac18a2a82a18fabbf
yara_andariel_7F8: (2016-12-01 13:56:28) ./
redgambler/9a50be3def3681242f35d3c0911e2e70
yara_andariel_7F8: (2017-03-21 16:05:58) ./
redgambler/2573d0ad00f4ba8ee86d7fce7454d963_unpack
```

```
00405523                      loc_405523:
00405523 8A 1C 37             mov     bl, [edi+esi]
00405526 32 DA                xor     bl, dl
00405528 32 D8                xor     bl, al
0040552A 32 D9                xor     bl, cl
0040552C 88 1E                mov     [esi], bl
0040552E 8A D8                mov     bl, al
00405530 32 D9                xor     bl, cl
00405532 22 DA                and     bl, dl
00405534 8B 55 FC             mov     edx, [ebp+var_4]
00405537 8D 3C D5 00 00 00 00 lea     edi, ds:0[edx*8]
0040553E 33 FA                xor     edi, edx
00405540 81 E7 F8 07 00 00    and     edi, 7F8h
00405546 C1 E7 14             shl     edi, 14h
00405549 C1 EA 08             shr     edx, 8
0040554C 0B D7                or      edx, edi
0040554E 8D 3C 00             lea     edi, [eax+eax]
00405551 33 F8                xor     edi, eax
00405553 22 C8                and     cl, al
00405555 C1 E7 04             shl     edi, 4
00405558 33 F8                xor     edi, eax
0040555A 32 CB                xor     cl, bl
0040555C 8B D8                mov     ebx, eax
0040555E 83 E7 80             and     edi, 0FFFFFF80h
00405561 C1 E3 07             shl     ebx, 7
00405564 33 FB                xor     edi, ebx
00405566 C1 E7 11             shl     edi, 11h
00405569 C1 E8 08             shr     eax, 8
0040556C 0B C7                or      eax, edi
0040556E 46                   inc     esi
0040556F FF 4D 0C             dec     [ebp+arg_4]
00405572 89 55 FC             mov     [ebp+var_4], edx
00405575 75 A9                jnz     short loc_405520
```

```
00403F00                      loc_403F00:
00403F00 8A 1C 37             mov     bl, [edi+esi]
00403F03 32 DA                xor     bl, dl
00403F05 32 D8                xor     bl, al
00403F07 32 D9                xor     bl, cl
00403F09 88 1E                mov     [esi], bl
00403F0B 8A D8                mov     bl, al
00403F0D 32 D9                xor     bl, cl
00403F0F 22 DA                and     bl, dl
00403F11 8A D0                mov     dl, al
00403F13 22 D1                and     dl, cl
00403F15 32 DA                xor     bl, dl
00403F17 8B 54 24 10          mov     edx, [esp+18h+var_8]
00403F1B 8A CB                mov     cl, bl
00403F1D 8D 1C D5 00 00 00 00 lea     ebx, ds:0[edx*8]
00403F24 33 DA                xor     ebx, edx
00403F26 81 E3 F8 07 00 00    and     ebx, 7F8h
00403F2C C1 E3 14             shl     ebx, 14h
00403F2F C1 EA 08             shr     edx, 8
00403F32 0B D3                or      edx, ebx
00403F34 8D 1C 00             lea     ebx, [eax+eax]
00403F37 33 D8                xor     ebx, eax
00403F39 C1 E3 04             shl     ebx, 4
00403F3C 33 D8                xor     ebx, eax
00403F3E 8B E8                mov     ebp, eax
00403F40 83 E3 80             and     ebx, 0FFFFFF80h
00403F43 C1 E5 07             shl     ebp, 7
00403F46 33 DD                xor     ebx, ebp
00403F48 C1 E3 11             shl     ebx, 11h
00403F4B C1 E8 08             shr     eax, 8
00403F4E 0B C3                or      eax, ebx
00403F50 46                   inc     esi
00403F51 83 6C 24 14 01       sub     [esp+18h+var_4], 1
00403F56 89 54 24 10          mov     [esp+18h+var_8], edx
00403F5A 75 A4                jnz     short loc_403F00
```

Same cryptographic algorithm found in RedGambler and Rifle samples

RedGambler samples seem to be related to older samples from the Troy/DarkSeoul operation as well as an APT37 Navrat sample by their Rich headers which are identical:

```
(2016-12-01 13:56:28) ./redgambler/9a50be3def3681242f35d3c0911e2e70
<- Rich -> (2016-05-01 05:53:43)
./apt37_summit/navrat_old_
e0257d187be69b9bee0a731437bf050d56d213b50a6fd29dd6664e7969f286ef.bin
<- Rich -> (2013-03-20 04:07:02)
./troydarkseoul/DarkSeoul/DarkSeoulDropper/
DarkSeoulDropper_9263E40D9823AECF9388B64DE34EAE54_unpack
<- Rich -> (2017-03-21 16:05:58) ./
redgambler/2573d0ad00f4ba8ee86d7fce7454d963_unpack
```

Other Andariel samples are linked with various Lazarus samples. For example, Andariel uses inflate library version 1.1.4 which is the same version found in several other North Korean samples (GoldDragon, Fallchill, Dtrack...).

– APT38/Bluenoroff

Bluenoroff clusters are linked by Rich headers, timestamps and code similarity. Most of the links are quite strong and make APT38 clusters the most distinguishable ones, meaning that the group doesn't think it's necessary to be sneaky and reinvent itself, but will reuse a lot of elements, from architecture to malware implementations. These clusters are mainly composed of the following malware families:

→ Nukesped
→ Fallchill
→ Volgmer
→ Electricfish
→ Dyepack
→ SWIFT-related malware
→ Hoplight
→ Some Sony / Blockbuster samples
→ Malware from bank attacks (Poland, Vietnam...)
→ Destover
→ Bankshot
→ Fastcash
→ ...

Looking at the links, we can see that a Yara rule we built is matching almost 40 samples from our dataset, all of them attributed to APT38. The Yara rule was built to detect a specific RC4 implementation and called `yara_apt38_rc4`:

```
rule yara_apt38_rc4 { strings:
      $s1 = { 8A 90 01 01 00 00 // mov dl, byte [eax + 0x101]
            8A 88 00 01 00 00 // mov cl, byte [eax + 0x100] 8A 14 02 // mov dl, byte [edx + eax]
            8A 1C 01          // mov bl, byte [ecx + eax]
            02 D3             // add dl, bl
            8A 1C 2E          // mov bl, byte [esi + ebp]
            81 E2 FF 00 00 00 // and edx, 0xff
            8A 0C 02          // mov cl, byte [edx + eax]
            32 CB }           // xor cl, bl
   condition:
      uint16(0) == 0x5a4d and any of ($s*)
}
```

This rule showcases once again that Lazarus groups reuse a lot of code for their malware. Here are some of the samples using this RC4 implementation:

```
yara_apt38_rc4: ./apt38_
contopee/766d7d591b9ec1204518723a1e5940fd6ac777f606ed64e731fd91b0b4c3d9fc.bin
yara_apt38_rc4: ./nukesped/3EDCE4D49A2F31B8BA9BAD0B8EF54963
yara_apt38_rc4: ./nukesped/sample2.bin
yara_apt38_rc4: ./nukesped/34E56056E5741F33D823859E77235ED9 yara_apt38_rc4: ./
nukesped/sample (9).bin
yara_apt38_rc4: ./nukesped/sample (1).bin
yara_apt38_rc4: ./nukesped/F315BE41D9765D69AD60F0B4D29E4300
yara_apt38_rc4: ./
nukesped/32ec329301aa4547b4ef4800159940feb950785f1ab68d85a14d363e0ff2bc11
yara_apt38_rc4: ./cybercom/38fc56965dccd18f39f8a945f6ebc439.bin
yara_apt38_rc4:  ./cybercom/5c0c1b4c3b1cfd455ac05ace994aed4b.bin
yara_apt38_rc4: ./typeframe/
e69d6c2d3e9c4beebee7f3a4a3892e5fdc601beda7c3ec735f0dfba2b29418a7.
bin yara_apt38_rc4: ./fallchill/
ca70aa2f89bee0c22ebc18bd5569e542f09d3c4a060b094ec6abeeeb4768a143.
bin yara_apt38_rc4: ./
intezer/4a84452752cf8e493ae820871096044edd9f6453366842927148e7d8e218dc87.
bin yara_apt38_rc4: ./
intezer/80b5cc9feb10fac41ee2958ab0f751bf807126e34dcb5435d2869ef1cf7abc41_
z5Xv8XY4hN.bin yara_apt38_rc4: ./
intezer/7429a6b6e8518a1ec1d1c37a8786359885f2fd4abde560adaef331ca9deaeefd.
```

```
bin yara_apt38_rc4: ./intezer/
dbae68e4cab678f2678da7c48d579868e35100f3596bf3fa792ee000c952c0ed.
bin yara_apt38_rc4: ./intezer/
a4a2e47161bbf5f6c1d5b1b3fba26a19dbfcdcf4eb575b56bde05c674089ae95.
bin yara_apt38_rc4: ./bangladesh_
swift/4659dadbf5b07c8c3c36ae941f71b631737631bc3fded2fe2af250ceba98959a.bin
yara_apt38_rc4: ./bangladesh_swift/nroff_b.exe
yara_apt38_rc4:  ./bangladesh_swift/evtdiag.exe
yara_apt38_rc4: ./apt38_
dyepack/4659dadbf5b07c8c3c36ae941f71b631737631bc3fded2fe2af250ceba98959a
yara_apt38_rc4: ./apt38_
dyepack/5b7c970fee7ebe08d50665f278d47d0e34c04acc19a91838de6a3fc63a8e5630
yara_apt38_rc4: ./
ghostsecret/45e68dce0f75353c448865b9abafbef5d4ed6492cd7058f65bf6aac182a9176a.
bin yara_apt38_rc4: ./ghostsecret/Sample_5ae56e2077d7dc0d380c3bfd_exe
yara_apt38_rc4:
./blockbuster_continues/
volgmer_7429a6b6e8518a1ec1d1c37a8786359885f2fd4abde560adaef331ca9deaeefd.bin
[...]
```

Other Yara rules are matching several APT38 samples from different malware families: some related to file wiping implementations, Fallchill success codes, string decoding algorithms, inflate 1.1.3 strings... On another hand, Rich header analysis reveals that some recent malware found in India, Vietnam and Taiwan, as well as samples LEXFO found during incident responses share the same Rich headers, which are strong links.

WannaCry

WannaCry samples are timestomped, but we see that the WannaCry cluster is close to the Bluenoroff ones. In particular, we see that the `wannacry_rand` Yara rule we built from the WannaCry sample `3e6de9e2baacf930949647c399818e7a2caea2626df6a468407854aaa515eed9` matches the Contopee malware attributed to APT38 (`766d7d591b9ec1204518723a1e5940fd6ac777f606ed64e731fd91b0b4c3d9fc`).

```
yara_wannacry_rand: (2015-02-23 01:32:57)
./apt38_
contopee/766d7d591b9ec1204518723a1e5940fd6ac777f606ed64e731fd91b0b4c3d9fc.bin
(2017-02-09 09:47:27)
./wannacry/3e6de9e2baacf930949647c399818e7a2caea2626df6a468407854aaa515eed9
```

```
10004BEB FF 15 5C E0 00 10    call  ds:rand     |  004025A2 FF 15 64 F4 40 00    call  ds:rand
10004BF1 99                   cdq               |  004025A8 99                   cdq
10004BF2 B9 05 00 00 00       mov   ecx, 5      |  004025A9 B9 05 00 00 00       mov   ecx, 5
10004BF7 33 FF                xor   edi, edi    |  004025AE 33 FF                xor   edi, edi
10004BF9 F7 F9                idiv  ecx         |  004025B0 F7 F9                idiv  ecx
```

Shared code between Bluenoroff Contopee and WannaCry

Most WannaCry samples were statically linked with inflate lib version 1.1.3, which links them to some Bluenoroff samples that are using the exact same version (for instance the recent APT38 keylogger `efd470cfa90b918e5d558e5c8c3821343af06eedfd484dfeb20c4605f9bdc30e` used on Vietnamese targets).

```
yara_lib_static_inflate_113: (2010-11-20 09:05:05) ./wannacry/dropper.bin
(2018-04-28 02:53:06)
./vietnam/efd470cfa90b918e5d558e5c8c3821343af06eedfd484dfeb20c4605f9bdc30e.
bin yara_lib_static_inflate_113: (2010-11-20 09:03:08) ./wannacry/mssecsvc.bin
(2018-04-28 02:53:06)
./vietnam/efd470cfa90b918e5d558e5c8c3821343af06eedfd484dfeb20c4605f9bdc30e.bin
```

DTrack

DTrack is a malware attributed to Lazarus / APT38. Recent DTrack samples found on critical infrastructures like nuclear power plants are linked with a sample from the Troy/DarkSeoul campaign compiled in 2011. The link comes from the reuse of the specific ZIP password `dkwero38oerA^t@#` . This is surprising and could be a false flag.

```
yara_zip_password: ./troydarkseoul/Http Troy/Files inside 8FBC1F3048263AA0D4F56D119198ED04/Layer 4/DLL 1 (bs.dll).dll
yara_zip_password: ./dtrack/npp_3cc9d9a12f3b884582e5c4daf7d83c4a510172a836de90b87439388e3cde3682
yara_zip_password: ./dtrack/npp_bfb39f486372a509f307cde3361795a2f9f759cbeb4cac07562dcbaebc070364 yara_zip_password: ./dtrack/dfa984f8d6bfc4ae3920954ec8b768e3d5a9cc4349966a9d16f8bef658f83fcd.bin
```

Those DTrack samples are also weakly linked with other Lazarus samples by statically-linked libraries such as TZip and SQlite.

– GoldDragon campaign

GoldDragon samples are linked to Lazarus by two main features: the reuse of a specific RC4 implementation that was seen in old Joanap dropper samples and detected by our Yara rules, and the overlaps of rich headers. Here is an example of a GoldDragon sample sharing its Rich header with other known Lazarus samples (as well as other GoldDragon samples):

```
(2017-12-24 08:37:57) ./golddragon/3a0c617d17e7f819775e48f7edefe9af84a1446b
```

<-	Rich	->	(2013-01-31	10:27:18)	./troydarkseoul/DarkSeoul/DarkSeoul_0A8032C-D6B4A710B1771A080FA09FB87
<-	Rich	->	(2017-12-24	08:29:04)	./golddragon/11a38a9d23193d9582d02ab0eae-767c3933066ec
<-	Rich	->	(2012-07-06	12:24:18)	./troydarkseoul/DarkSeoul/DarkSeoul_E4F66C3C-D27B97649976F6F0DAAD9032
<-	Rich	->	(2017-12-24	08:44:08)	./golddragon/bf21667e4b48b8857020ba455531c-9c4f2560740
<-	Rich	->	(2017-12-24	08:47:21)	./golddragon/4f58e6a7a04be2b2ecbcdcbae6f-281778fdbd9f9
<-	Rich	->	(2013-01-31	10:27:18)	./troydarkseoul/DarkSeoul/DarkSeoul_DB4BBD-C36A78A8807AD9B15A562515C4
<-	Rich	->	(2013-01-31	10:27:18)	./troydarkseoul/DarkSeoul/DarkSeoul_F0E045210E-3258DAD91D7B6B4D64E7F3
<-	Rich	->	(2018-12-21	00:34:35)	./kimsuky_stolenpencil/1.bin
<-	Rich	->	(2017-12-24	08:16:57)	./golddragon/e68f43ecb03330f-f0420047b61933583b4144585
<-	Rich	->	(2012-07-06	12:24:18)	./troydarkseoul/DarkSeoul/DarkSeoul_50E03200C3A-0BECBF33B3788DAC8CD46
<-	Rich	->	(2013-01-31	10:27:18)	./troydarkseoul/DarkSeoul/DarkSeoul_5FCD6E1DACE6B-0599429D913850F0364

Another link is the statically-linked inflate v. 1.1.4 that we found in GoldDragon samples, as this version is widely used in a lot of Lazarus samples.

– APT37

Samples attributed to APT37 (Reaper) seem to be quite unique and only linked with Lazarus samples by statically-linked library or encryption algorithms, which are weak links. This confirms what FireEye stated in its report: this group needs to be tracked separately from Lazarus.

A lot of APT37 samples share the same Rich header. We also found the following identical Rich headers between an APT37 malware and a Bluenoroff Nukesped sample:

```
(2019-01-02 01:43:47)
./apt37_evilnewyear/2019_636844ce36f41641d854a1b239df91da3103873d3dfec0c25087582eec064e4d.bin
<- Rich -> (2018-02-12 20:06:28) ./cybercom/07d2b057d2385a4cdf413e8d342305df.bin
<- Rich -> (2018-02-12 20:06:28) ./nukesped/07D2B057D2385A4CDF413E8D342305DF
```

Finally, we found a Navrat sample attributed to APT37 and an Andariel sample from the RedGambler operation with the same Rich header, which connects the two groups (see the part about Andariel below).

– The OlympicDestroyer false flag

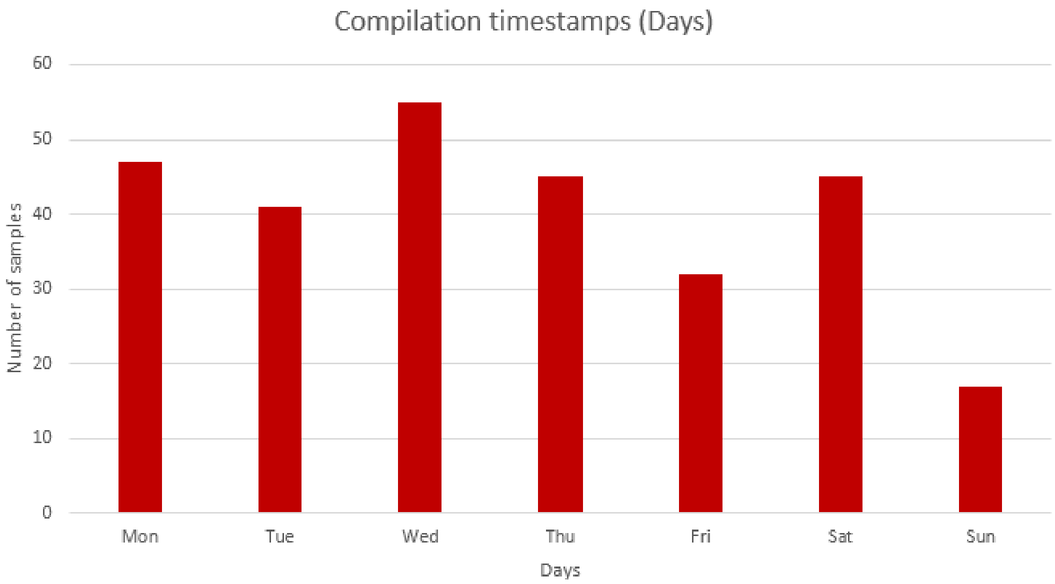
To complicate attribution, the attackers behind OlymicDestroyer copied a Rich header from Lazarus samples to replace the rich header of some of their malware. Our tool gives the following result, showing that the Rich header was taken from Bluenoroff samples (one of them from the Bangladesh SWIFT heist):

```
(2017-12-27 09:03:48) ./olympicdestroyer/3c0d740347b0362331c882c2dee96dbf
  <- Rich -> (2016-02-04 13:45:39) ./bangladesh_swift/evtsys.exe
  <- Rich -> (2017-03-02 16:46:13)
./blockbuster_sequel/032ccd6ae0a6e49ac93b7bd10c7d249f853fff3f5771a1fe3797f733f09db5a0.bin
```

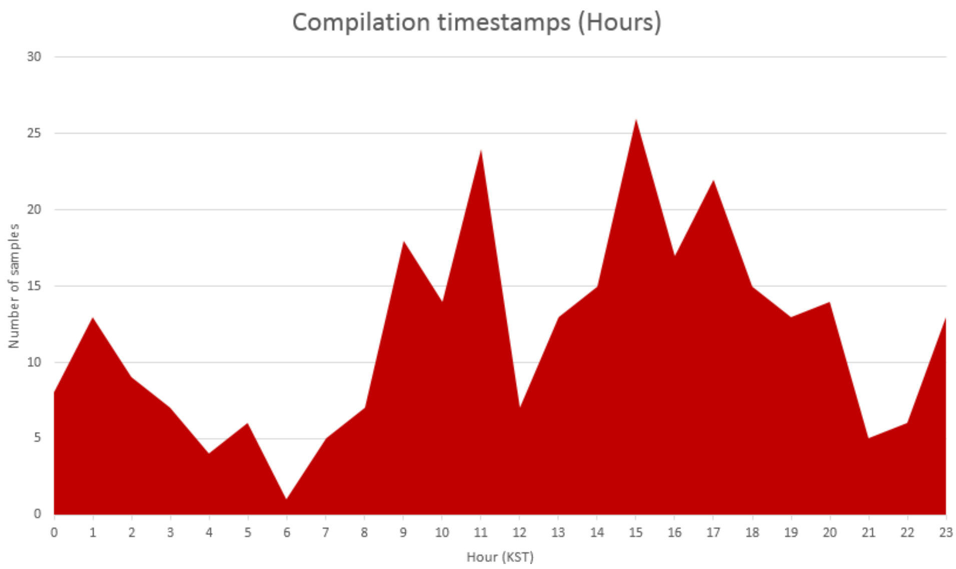
Kaspersky published an article about this false flag operation [38].

WORKING HOURS AND DAYS OF THE LAZARUS DEVELOPERS

We extracted all compilation timestamps from the samples in our dataset and removed those that were either altered or inconsistent. Some samples appeared to be legitimate software infected by Lazarus without recompiling, making the timestamps irrelevant. For instance, we ignored the sample `2223a93521b261715767f00f0d1ae4e692bd593202be40f3508cb4fd5e21712b` which turned out to be a version of the FTP tool FileZilla that the attackers altered by adding some malicious code without recompiling it, leaving its original compilation timestamp and compiler fingerprints unmodified.
Analyzing unaltered compilation timestamps, we see that the Lazarus developers are mostly working between 8AM and 8PM UTC+9 (KST). We can even notice some breaks at lunchtime, and that Lazarus developers are working overnight. Most samples were compiled from Monday to Saturday included.



Lazarus compilation timestamps by days



Lazarus compilation timestamps by hours (KST / UTC+9)

CONCLUSION

Such a classification proved to be very relevant for North Korean malware. It highlighted heavy links illustrating code and architecture reuse inside established groups, as well as relations (or lack of) between these separate groups.

Attacker groups like Lazarus are so active they struggle or are reluctant to renew their arsenal. Studying their TTPs prove to be very valuable and will greatly help properly reacting to incident. As a defender, being able to exploit Lazarus laziness and carelessness by quickly identifying their TTPs will give you some key information: you know what they want, how they plan to achieve it and with which tools.

The information given in this report, the classification LEXFO established and the associated internally developed tools helped the incident response team practically during missions involving Lazarus, as it narrowed down the analysis and gave good hints on where to look for technical clues: persistence, communications, lateral movements, exfiltration etc.

VI. DETECTION & MITIGATION

VULNERABILITY USED

North Korean groups have been exploiting a lot of vulnerabilities, such as 0days and as 1days. Most exploits target Adobe Flash Player as well as the Hangul Word Processor, though groups like Andariel have also been seen finding and exploiting vulnerabilities in specific corporate software. The list of CVE that have been exploited by DPRK groups below shows once again that keeping its software updated is crucial.

– Lazarus

Lazarus and its subgroups Andariel and Bluenoroff often rely on software vulnerabilities to infect their targets. Here are some of them:

Vulnerability	0day	Comments
CVE-2014-0497	Yes	Flash exploit
CVE-2015-6585	Yes	Vulnerability in HWP
CVE-2015-8651	No	Flash exploit
CVE-2016-0034	Yes	Silverlight exploit
CVE-2016-0189	Yes	Internet Explorer Scripting Engine Remote Memory Corruption Vulnerability
CVE-2016-1019	No	Flash expoit
CVE-2016-4117	Yes	Flash exploit used in watering hole attacks
CVE-2017-0261	Yes	EPS restore use-after-free
CVE-2018-8373	Yes	VBScript Engine vulnerability used by the DarkHotel subgroup
CVE-2018-4878	Yes	Flash exploit used by APT37 and Lazarus
CVE-2018-20250	No	WinRar exploit targeting Israeli companies
CVE-2018-8174	Yes	Internet Explorer VBS engine vulnerability

APT37 / Reaper

APT37 usually exploits 1day to target unpatched systems, mostly through Adobe vulnerabilities. The vulnerabilities below were attributed to APT37 by FireEye [39]:

Vulnerability	0day	Comments
CVE-2013-4979	No	Buffer overflow in EPS Viewer
CVE-2014-8439	No	Adobe Flash Player arbitrary code execution
CVE-2015-2387	No	Adobe Type Manager Font Driver memory corruption vulnerability

Vulnerability	0day	Comments
CVE-2015-2419	No	Internet Explorer JScript RCE
CVE-2015-2545	No	Microsoft Office Malformed EPS File Vulnerability
CVE-2015-3105	No	Adobe Flash Player arbitrary code execution
CVE-2015-5119	No	Adobe Flash Player Use-After-Free leading to code execution
CVE-2015-5122	No	Adobe Flash Player Use-After-Free leading to code execution
CVE-2015-7645	No	Adobe Flash Player vulnerability
CVE-2016-1019	No	Adobe Flash Player vulnerability
CVE-2016-4117	No	Adobe Flash Player vulnerability
CVE-2017-0199	No	Microsoft Office/WordPad Remote Code Execution Vulnerability
CVE-2018-4878	Yes	Flash exploit also used by Lazarus

DETECTING LAZARUS ACTIVITIES

Network detection rules

US-CERT Snort rules to detect Fake TLS packets [40]:

```
alert tcp any any -> any any (msg:”Malicious SSL 01 Detected”;content:”|17 03 01 00 08|”; pcre:”/\x17\x03\x01\x00\x08.{4}\x04\x88\x4d\x76/”; rev:1; sid:2;)
alert tcp any any -> any any (msg:”Malicious SSL 02 Detected”;content:”|17 03 01 00 08|”; pcre:”/\x17\x03\x01\x00\x08.{4}\x06\x88\x4d\x76/”; rev:1; sid:3;)
alert tcp any any -> any any (msg:”Malicious SSL 03 Detected”;content:”|17 03 01 00 08|”; pcre:”/\x17\x03\x01\x00\x08.{4}\xb2\x63\x70\x7b/”; rev:1; sid:4;)
alert tcp any any -> any any (msg:”Malicious SSL 04 Detected”;content:”|17 03 01 00 08|”; pcre:”/\x17\x03\x01\x00\x08.{4}\xb0\x63\x70\x7b/”; rev:1; sid:5;)
```

The following rule will specifically detect the SWPSVC RAT LEXFO discovered:

```
alert tcp any -> any (msg:”Lazarus Stage 1 SWPSVC Handshake”; dsize:5; content:”|17 03 01 00 04|”;)
```

Yara rules

LEXFO produced the following YARA rules to sign and allow detection of the latest Lazarus samples encountered during the investigation.

The rules `lazarus_forward_libcurl` and `themida_virtualizer` can produce false-positives, as they will respectively detect any file with a specific statically compiled libcurl library and files packed with Themida Code-Virtualizer, which can be legitimate in some cases. These rules will work on uncompressed disk and memory dumps, as well as network capture files.

```
rule lazarus_forward_
    strings { strings:
        $s1 = “ssylka” fullword
        $s2 = “ustanavlivat” fullword
        $s3 = “pereslat” fullword
        $s4 = “Nachalo” fullword
        $s5 = “kliyent2podklyuchit” full-
    word condition:
        (3 of ($s*))
}
rule lazarus_forward_lib-
    curl { strings:
        $s1 = “7.49.1” fullword
        $s2 = “x86_64-pc-win32” fullword
        $s3 = “libcurl/7.49.1” full-
    word condition:
        (3 of ($s*))
}
rule lazarus_forward_tcp
    { strings:
        $s1 = {b0 00 b0 00 b0 00 b0 00 b0 00 b0 00 e9 00}
    condition:
        (1 of ($s*))
}
rule lazarus_sqcsvc
    { strings:
        $s1 = “7.49.1” fullword
        $s2 = “x86_64-pc-win32” fullword
        $s3 = “libcurl/7.49.1” fullword
        $s4 = “sock_64.dll” full-
    word condition:
        (4 of ($s*))
}
rule themida_virtualizer
    { strings:
        $s1 = “v-lizer” fullword con-
    dition:
        (uint16(0) == 0x5a4d and filesize < 5MB and 1 of ($s*))
}
rule lazarus_rc4
    { strings:
        $s1 = {4E 38 1F A7 7F 08 CC AA 0D 56 ED EF F9 ED 08 EF}
        $s2 = {11 20 00 00 00 00 00 00 00 00 00 00 00 00 00 00}
        $s3 = {53 87 F2 11 30 3D B5 52 AD C8 28 09 E0 52 60 D0 6C C5 68 E2 70 77 3C 8F 12 C0 7B
        13 D7 B3 9F
7C}
}
  $s4 = {85 C0 7C 17 8B 4D F4 8B 76 20 33 C0 3B C8 77 0B}
condition:
  (1 of ($s*))
rule lazarus_svcbat
    { strings:
        $s1 = “sc create sqcsvc”
        $s2 = “sc start
    sqcsvc” condition:
        (1 of ($s*))
}
rule lazarus_capture
    { strings:
    $s2 = “[ENTER]” wide fullword
    $s3 = “SpliceImages: CreateCompatibleBitmap failed”
    fullword condition:
        (2 of ($s*))
}
rule lazarus_injector
    { strings:
        $s1 = “finding target project”
        $s2 = “delete
    ddd” condition:
        (2 of ($s*))
}
```

The full Yara ruleset we used for this report will be available to our clients.

RECOMMENDATIONS

Several Lazarus infection vectors can be severely mitigated to prevent or block an attack.

– Preventing an infection

The WannaCry incident showed how important it is keeping one's OS updated. Lazarus will certainly continue to implement and leverage such 1day vulnerabilities to target unpatched systems quickly after a fix is deployed.

As shown in this report, Lazarus leverages known vulnerabilities in webservers to try and get a first access to the internal network of a target. To mitigate this vector, it is necessary to make sure all exposed servers and their components are up-to-date and isolated from the internal networks of the organizations.

Furthermore, Lazarus leveraged several 0day and 1day vulnerabilities in popular software such as Flash Player, HWP and Silverlight. Keeping those software up-to-date is mandatory. The group is also able to quickly find and exploit vulnerabilities in custom internal software used by companies, sometimes leading to supply chain attacks. Auditing software used internally is also advised to mitigate this vector.

– Mitigating lateral movements

Lazarus uses mostly legitimate tools for lateral movements. When a form of authentication is needed, they will either reuse stolen passwords gathered with Mimikatz-like tools or keyloggers or try to bruteforce it with dictionaries.

Tools like PSExec can be monitored through log analysis. As Lazarus implants usually achieve persistent by installing services, event id `7045` and `4697` with the `Service Start Type` information set to `SERVICE_AUTO_START` must be closely monitored.

Last but not least, enforcing a strong password policy is obviously advised.

– Threat intelligence

As Lazarus activities are actively monitored by many security firms such as LEXFO, it is important for security teams to stay up-to-date and follow threat intelligence reports. As we showed in this paper, Lazarus will most of the time reuse known and easy-to-detect communication protocols and tools, and most infections can therefore be prevented.

If any indicator of a compromised system is found, it is strongly advised to quickly contact a specialized firm that knows how the attackers work and can quickly assess the impact of the attack and mitigate it.

VII. APPENDICES

APPENDIX A: ABBREVIATIONS

Abbreviation	Meaning
RAT	Remote Access Tool
PCAP	Packet Capture
MUI	MultiLanguage User Interface extension
DDoS	Distributed Denial of Service
TTP	Tactics, Techniques/Tools and Procedures
TLS	Transport Layer Security
C&C	Command & Control server
CERT	Computer Emergency Response Team
APK	Android Package Kit

APPENDIX B: LIST OF STUDIED SAMPLES

Hashes are SHA256.

```
766d7d591b9ec1204518723a1e5940fd6ac777f606ed64e731fd91b0b4c3d9fc
d4616f9706403a0d5a2f9a8726230a4693e4c95c58df5c753ccc684f1d3542e2
95c8ffe03547bcb0afd4d025fb14908f5230c6dc6fdd16686609681c7f40aca2
99017270f0af0e499cfeb19409020bfa0c2de741e5b32b9f6a01c34fe13fda7d
7646c2afbc8b9719b0295e5a880bb89fb85bdd4346603a52768b161eda12e8be
f12db45c32bda3108adb8ae7363c342fdd5f10342945b115d830701f95c54fa9
077d9e0e12357d27f7f0c336239e961a7049971446f7a3f10268d9439ef67885
a1c483b0ee740291b91b11e18dd05f0a460127acfc19d47b446d11cd0e26d717
3c1e4c334629b20e21b8ab08b8aa19db738f2ed761290ffdd26665cd61cb7807
7c73619ff8d5e4ed3b29b7ae71a69602df4071fd8c1029f9674e9978cdc03de9
6b90e2a3f0ad8819b5afe67bf13451c9782af26a9f2bdac3a0e042569054e5fd
73dcb7639c1f81d3f7c4931d32787bdf07bd98550888c4b29b1058b2d5a7ca33
c66ef8652e15b579b409170658c95d35cfd6231c7ce030b172692f911e7dcff8
f8f7720785f7e75bd6407ac2acd63f90ab6c2907d3619162dc41a8ffa40a5d03
32ec329301aa4547b4ef4800159940feb950785f1ab68d85a14d363e0ff2bc11
c66ef8652e15b579b409170658c95d35cfd6231c7ce030b172692f911e7dcff8
b05aae59b3c1d024b19c88448811debef1eada2f51761a5c41e70da3db7615a9
f8f7720785f7e75bd6407ac2acd63f90ab6c2907d3619162dc41a8ffa40a5d03
73dcb7639c1f81d3f7c4931d32787bdf07bd98550888c4b29b1058b2d5a7ca33
1fe1fa6b01166c373de68c029d8cdda60cb1599053f935e580f3f40aaf106345
fe43bc385b30796f5e2d94dfa720903c70e66bc91dfdcfb2f3986a1fea3fe8c5
0608e411348905145a267a9beaf5cd3527f11f95c4afde4c45998f066f418571
fe43bc385b30796f5e2d94dfa720903c70e66bc91dfdcfb2f3986a1fea3fe8c5
084b21bc32ee19af98f85aee8204a148032ce7eabef668481b919195dd62b319
ccafbcff1596e3dfd28dcb97a5ba85e6845e69464742edfe136fe09bbec86ba1
b9a26a569257fbe02c10d3735587f10ee58e4281dba43474dbdef4ace8ea7101
```

0608e411348905145a267a9beaf5cd3527f11f95c4afde4c45998f066f418571
8a1d57ee05d29a730864299376b830a7e127f089e500e148d96d0868b7c5b520
8a1d57ee05d29a730864299376b830a7e127f089e500e148d96d0868b7c5b520
084b21bc32ee19af98f85aee8204a148032ce7eabef668481b919195dd62b319
1a01b8a4c505db70f9e199337ce7f497b3dd42f25ad06487e29385580bca3676
32ec329301aa4547b4ef4800159940feb950785f1ab68d85a14d363e0ff2bc11
26a2fa7b45a455c311fd57875d8231c853ea4399be7b9344f2136030b2edc4aa
ec254c40abff00b104a949f07b7b64235fc395ecb9311eb4020c1c4da0e6b5c4
26a2fa7b45a455c311fd57875d8231c853ea4399be7b9344f2136030b2edc4aa
d8af45210bf931bc5b03215ed30fb731e067e91f25eda02a404bd55169e3e3c3
ec44ecd57401b3c78d849115f08ff046011b6eb933898203b7641942d4ee3af9
0753f8a7ae38fdb830484d0d737f975884499b9335e70b7d22b7d4ab149c01b5
8a81a1d0fae933862b51f63064069aa5af3854763f5edc29c997964de5e284e5
c4a07bfc37a44dc85df2c63f369abb530dc2193ab1be506fc5dd45d56a44ca76
9e4c6410ab9eda9a3d3cbf23c58215f3bc8d3e66ad55e40b4e30eb785e191bf8
1b46afe1779e897e6b9f3714e9276ccb7a4cef6865eb6a4172f0dd1ce1a46b42
2e7c052fbc08473f60d5365157b1a0952e2dddee630fe4abe827382dade23a76
f8967b814c814c36559987a5baec67ebc44e9e1031600e1cf4e0c2bdaf8c6497
fa405cd8cd8565301d138e3826bd121cc8691731b889a7503132bda6c57f4691
adafcdf7196a73a24b1e6e523b0a3dd4c62886702b45a9b29021bc961f0d5ea4
6a9c46d96f001a1a3cc47d166d6c0aabc26a5cf25610cef51d2b834526c6b596
48cf912217c1b5ef59063c7bdb93b54b9a91bb6920b63a461f8ac7fcff43e205
32ec329301aa4547b4ef4800159940feb950785f1ab68d85a14d363e0ff2bc11
0608e411348905145a267a9beaf5cd3527f11f95c4afde4c45998f066f418571
b05aae59b3c1d024b19c88448811debef1eada2f51761a5c41e70da3db7615a9
73dcb7639c1f81d3f7c4931d32787bdf07bd98550888c4b29b1058b2d5a7ca33
f8f7720785f7e75bd6407ac2acd63f90ab6c2907d3619162dc41a8ffa40a5d03
fe43bc385b30796f5e2d94dfa720903c70e66bc91dfdcfb2f3986a1fea3fe8c5
084b21bc32ee19af98f85aee8204a148032ce7eabef668481b919195dd62b319
1a01b8a4c505db70f9e199337ce7f497b3dd42f25ad06487e29385580bca3676
b9a26a569257fbe02c10d3735587f10ee58e4281dba43474dbdef4ace8ea7101
8a1d57ee05d29a730864299376b830a7e127f089e500e148d96d0868b7c5b520
c66ef8652e15b579b409170658c95d35cfd6231c7ce030b172692f911e7dcff8
c6c332ae1ccb580ac621d3cf667ce9c017be41f8ad04a94c0c0ea37c4789dd14
d62bf83fb5a7b148f326908051b149b77663149d47426ce749e944f7abf5d304
84edc9b828de54d4bd00959fabf583a1392cb4c3eab3498c52818c96dc554b90
4f06eaed3dd67ce31e7c8258741cf727964bd271c3590ded828ad7ba8d04ee57
e0257d187be69b9bee0a731437bf050d56d213b50a6fd29dd6664e7969f286ef
b5838ecaad041a033ad16ddd6644d502546bc4916cbd10636c27b3eed3214578
e7b9c37be0ffca97002294f5813405855731b37b3b4ad2f4d73d1da9b7c535e5
2db6f93e99e55ef46c5b3ca52a52e34f088b7c1cd3835938557842b71b24ef56
283d1d2efa36d31fc00425cc88dab82e426c1c51d1b4da7925c91aef56d817a3
25ea7f67638e7e7b8706566788aa25a7d91843232ece85592b6bfe1eb4cd317a
25ea7f67638e7e7b8706566788aa25a7d91843232ece85592b6bfe1eb4cd317a
1e4d34fe08da837f16d1c04c313c3b2e46a09bf55376d4573ae1ebff1fae53f4
5d25465ec4d51c6b61947990fb148d0b1ee8a344069d5ac956ef4ea6a61af879
707e37e085014d2a8e6b0596322ac8f13664666c0d44f7963d2174dc4dc37ef6
2151c1977b4555a1761c12f151969f8e853e26c396fa1a7b74ccbaf3a48f4525
70034b33f59c6698403293cdc28676c7daa8c49031089efa6eefce41e22dccb3
49757cf85657757704656c079785c072bbc233cab942418d99d1f63d43f28359
05feed9762bc46b47a7dc5c469add9f163c16df4ddaafe81983a628da5714461
6323816c6476cb3d9c28461c5c45c858087f1eefed96e3f83843f5bcc299219e
b3de3f9309b2f320738772353eb724a0782a1fc2c912483c036c303389307e2e
a29b07a6fe5d7ce3147dd7ef1d7d18df16e347f37282c43139d53cce25ae7037
b66624ab8591c2b10730b7138cbf44703abec62bfc7774d626191468869bf21c
f4e0f145830ec7a9dace5a4b7d5af5f1e93662edcad40c08d57dc825d316174d
fe696f8fb3f927bfbc9dbdcb067f87f3ada1afa8a76385f16e5b3dd70adf5ca2
2011b9aa61d280ca9397398434af94ec26ddb6ab51f5db269f1799b46cf65a76
ef40f7ddff404d1193e025081780e32f88883fa4dd496f4189084d772a435cb2
4d37f80da97845129debf3244e1f731d2c93a02519f9fdaa059f5f124cf7c26f
fb94a5e30de7afd1d9072ccedd90a249374f687f16170e1986d6fd43c143fb3a
f31bee70fd10f6846890f42947de40061bacb24fb51f43ef6c75325ec9b95de8
6f37b758a7a015c2abdab7941b416deb508f2ab9143a64f9a8188ed0d0db3d14
2011b9aa61d280ca9397398434af94ec26ddb6ab51f5db269f1799b46cf65a76
a9bc09a17d55fc790568ac864e3885434a43c33834551e027adb1896a463aafc
4a740227eeb82c20286d9c112ef95f0c1380d0e90ffb39fc75c8456db4f60756
820ca1903a30516263d630c7c08f2b95f7b65dffceb21129c51c9e21cf9551c6
032ccd6ae0a6e49ac93b7bd10c7d249f853fff3f5771a1fe3797f733f09db5a0
82645e88736e11321774db7a7b28bd62d4ab133f859ecd35a4b2fa1d471412b7
98ccf3a463b81a47fdf4275e228a8f2266e613e08baae8bdcd098e49851ed49a
dee482e5f461a8e531a6a7ea4728535aafdc4941a8939bc3c55f6cb28c46ad3d
94aa827a514d7aa70c404ec326edaaad4b2b738ffaea5a66c0c9f246738df579
111ab6aa14ef1f8359c59b43778b76c7be5ca72dc1372a3603cd5814bfb2850d
0ca12b78644f7e4141083dbb850acbacbebfd3cfa17a4849db844e3f7ef1bee5
0852f2c5741997d8899a34bb95c349d7a9fb7277cd0910656c3ce37a6f11cb88
ee7a9a7589cbbcac8b6bf1a3d9c5d1c1ada98e68ac2f43ff93f768661b7e4a85
c54837d0b856205bd4ae01887aae9178f55f16e0e1a1e1ff59bd18dbc8a3dd82
ae1b32aac4d8a35e2c62e334b794373c7457ebfaaab5e5e8e46f3928af07cde4
2981e1a1b3c395cee6e4b9e6c46d062cf6130546b04401d724750e4c8382c863
9097b372f7f844c430aa8c1b217a50754b28434172d5af5d992bfcbce9dfeb4f
e7a542312ec718300ed9f229aaa60e5e2ec11aaa99387b76ed2e377bfad8b86e
7334209ace81d67babbbb37f5a0d2af24160f637a8559483e14685927df6b7fa
45bfa1327c2c0118c152c7192ada429c6d4ae03b8164ebe36ab5ba9a84f5d7aa
5cbc07895d099ce39a3142025c557b7fac41d79914535ab7ffc2094809f12a4b
db350bb43179f2a43a1330d82f3afeb900db5ff5094c2364d0767a3e6b97c854
90eba6416f5e1b35c9bf41b4a25ac880c491dd2f10d993d8a65091f1adf68ee8
29431dc086499c7ee64236a365615be5e5c861452f047ffac5656120ece59266
2230edace3f42a5750f738f28814759b670922f16aa778e97d039d10fe9bab02
7aa99ebc49a130f07304ed25655862a04cc20cb59d129e1416a7dfa04f7d3e51
496841be8fb9d0042180a2bccf205e1e0bd0b41c537798265da7ad8f85cc35a2
bf2534b2f059547967bb453d67909921a41c10cdd19c1ec346a193060b094e2e

2df9e274ce0e71964aca4183cec01fb63566a907981a9e7384c0d73f86578fe4
f12db45c32bda3108adb8ae7363c342fdd5f10342945b115d830701f95c54fa9
bd6efb16527b025a5fd256bb357a91b4ff92aff599105252e50b87f1335db9e1
4b3416fb6d1ed1f762772b4dd4f4f652e63ba41f7809b25c5fa0ee9010f7dae7
e23900b00ffd67cd8dfa3283d9ced691566df6d63d1d46c95b22569b49011f09
0753f8a7ae38fdb830484d0d737f975884499b9335e70b7d22b7d4ab149c01b5
4c2efe2f1253b94f16a1cab032f36c7883e4f6c8d9fc17d0ee553b5afb16330c
201a9c5fe6a8ae0d1c4312d07ef2066e5991b1462b68f102154bb9cb25bf59f9
4d4b17ddbcf4ce397f76cf0a2e230c9d513b23065f746a5ee2de74f447be39b9
ebba2aa065059f1f841a86100905310d11e1b8d7a0f8e89bc1227b19ab69e9af
2e835c7496fb4fc1c53665ef89fffdcbcc8dc49bea0baecc5b8496006ea601bb
2e373e199d2b6dea0241c672bbcbccedac86cba2ed2fdefc84a5d8187acb896f
ffa97eb4875129646376bc88e9ff99ffeff2c6bba3a06f6727d5f343fc7f6b51
0efd49bfbdc8655e5db47d45b6ce4c2c64d6152665f45ef7ac57f04459369487
2cab9946741fc4cddefcec2104d4fe6d76390868f60f3207e9cb0e210bbe8db0
de4ff8901766e8fc89e8443f8732394618bf925ce29b6a8aafe1d60f496e7f0e
1a8655886ea6be9ae0a71e845b5a334b476494b3aad7bfe6510218059eba5788
0b269bdd4c2d11ce0cd050bddf8f6ff618126c2b531e8ad3ab36ecc1a88d8162
0a812976b9412ed28aee3ac3de57873fafe1ddfa0e6b9026078017b810d1b24e
2b6288bbd81bb9d666c3a0372f26ede47c8c9ff11c604307982d51654fb9e850
1ec2e4d02a89277afc0ee35d2d72009a5dbe96f88e1bc70bbfb3a9224478b7d5
0af925cd9d9a417f47882391555fa207398bfb87c3c6edc65f2ea42843cbdc3d
1d24d8268c2f8e82b65d58429c166367eee9683c38a1408910536d8084f4ad46
2db8a9c401911c7317e8a89c35d979d0e8e9ba718ae13a0a0cfedd957654ec10
276af9add9b6d8d96a950525b647d1eed247b3f63101bd942bd9816d0f8f9a6b
9661f70ead79a1ff35282bf2d061acb2733900eea87e2233ac7b8f8d3a80ad75
eceb01a902c2b7ccd580dd639ae5a1dc366e73cc8a27230557bb2237bd20e452
7f4e7618af45a61003c74b373095b206a885b26079830f7ee0dada28f8429623
28006dc505920fc3589933fda216052abc09d4a007bc76ba2542f7876ebd299b
24bce152c7f884a923b29a4130931c63cd3f9c0ab08a28a79c7995356a146131
c6a78c2c4d5078a1a769bdfb071311eb3bb01750e8bf101026102 8a1db68671b
373938e958030f1764b4db71df953df5c460a30e895583b7901da5c6954b0739
dc827f7a1e5ee4600697d7d3efdeb8401b7a9af3d704d0462e7d3e0804a9069d
32e98f39bcde86885c527ddcf68fad67d0a7e6c23877672ebfd4c2a6a3f545e5
90abfe3e4f21b5a16cd1ff3c485f079f73f5e7bbaca816917204858bb08007fc
e69d6c2d3e9c4beebee7f3a4a3892e5fdc601beda7c3ec735f0dfba2b29418a7
36f43755e5e5988d112f28fbc1dcd9bdee4a31fb7004b52db26dacdbfe7cb72f
929dc09a8bd8491b77f050a2736d39c30597ec7090d8f081eeb6179b6f8ab033
6c627a4be54b6377af9f73ab0923aeebcccbb57ec94e995a2171deb69d61af9d
d7a71f83d576fdf75e7978539bac04ad8b6605207b29379b89c24c0d0f31da61
d9757441e40d05a863d8dcfedab684d6644061231341c4106a3721436bc034ea
510f83af3c41f9892040a8a80b4f3a4736eebee2ec4a7d4bfee63dbe44d7ecff
1fbecec5da37b9a6e6dd63add4988fe7e2c4249aada883f58bcb794020455b77
422c767682bee719d85298554af5c59cf7e48cf57daaf1c5bdd87c5d1aab40cc
239ed753232d3cc0e75323d16d359150937934d30da022628e575997c8dd60a2
e9f3f6e286f5d06addb82a2fc4b3bcdf1142570183c5cac8e8156b2f1c26b74f
0d99b59ee6427f62596dbd7d016cc9ad5b365da152806703dbc5a5225164bbd5
d1515a888defff96f724d49fe05bace85066f6eeafd81cd0d9c4c27fdebc9cbb
7af070db3f5a3a08eeb5439039c1eee30f10c637b1c0d88e723104d422048863
d060123c21869b765b22b712a8ca47266a33464095411e2b7bdf7e327d23ed07
1b8d3e69fc214cb7a08bef3c00124717f4b4d7fd6be65f2829e9fd337fc7c03c
7f000893320d77e012686e20e1212e297408d5684335f7f24e40889401e24dff
0b6056e7ce278fb31bf644ef41e9532009e5dfbc33849b29f59c77ec993a8f46
cf065e50a5bef24099599af6a60a78c1607a04b21d3573a25ab26bf044a119d6
ca70aa2f89bee0c22ebc18bd5569e542f09d3c4a060b094ec6abeeeb4768a143
c0e22e80ea020ca8f71f58a8b53855293abdf8d4e0b34a69068004abaac60f42
0237b186086fa4d13e8c854dcf2d0f8a19fcbe62a58a415e9a5a933f1154e7d8
0c06e129902925c7ebd70e93d4d09707add781d8bd89cd557cda023045f3853e
2eb447785e5b35c42d842706d593a907d0bdbc50ad9d0327c3591ac4ef17ce6e
b783a2a69591cc1509acd0d3b33bdf69c87908669741f03a06f7d152cbe2923e
a917c1cc198cf36c0f2f6c24652e5c2e94e28d963b128d54f00144d216b2d118
a1260fd3e9221d1bc5b9ece6e7a5a98669c79e124453f2ac58625085759ed3bb
fd5a7e54cfdd3b3f32b44d8fdd845e62d6b86c0ddb550c544d659588d06ceaee
37f652e2060066a1c2c317195573a334416f5a9b9933cfb1ece55bea8048d80f
d4616f9706403a0d5a2f9a8726230a4693e4c95c58df5c753ccc684f1d3542e2
5d25465ec4d51c6b61947990fb148d0b1ee8a344069d5ac956ef4ea6a61af879
3cc9d9a12f3b884582e5c4daf7d83c4a510172a836de90b87439388e3cde3682
bfb39f486372a509f307cde3361795a2f9f759cbeb4cac07562dcbaebc070364
fe51590db6f835a3a210eba178d78d5eeafe8a47bf4ca44b3a6b3dfb599f1702
dfa984f8d6bfc4ae3920954ec8b768e3d5a9cc4349966a9d16f8bef658f83fcd
9b86a50b36aea5cc4cb60573a3660cf799a9ec1f69a3d4572d3dc277361a0ad2
37b04dcdcfdcaa885df0f392524db7ae7b73806ad8a8e76fbc6a2df4db064e71
4a84452752cf8e493ae820871096044edd9f6453366842927148e7d8e218dc87
dbae68e4cab678f2678da7c48d579868e35100f3596bf3fa792ee000c952c0ed
4e8c10a7fa51a3ab089b284e86a7daaca779ed82ba1750607fc3bfa91681f9b1
e79bbb45421320be05211a94ed507430cc9f6cf80d607d61a317af255733fcf2
7429a6b6e8518a1ec1d1c37a8786359885f2fd4abde560adaef331ca9deaeefd
80b5cc9feb10fac41ee2958ab0f751bf807126e34dcb5435d2869ef1cf7abc41
a606716355035d4a1ea0b15f3bee30aad41a2c32df28c2d468eafd18361d60d6
a4a2e47161bbf5f6c1d5b1b3fba26a19dbfcdcf4eb575b56bde05c674089ae95
9f177a6fb4ea5af876ef8a0bf954e37544917d9aaba04680a29303f24ca5c72c
6f474f2af52961e9d7bbd467d98fb7886579932e2fe9567c28c8be3ab845dc5d
9b383ebc1c592d5556fec9d513223d4f99a5061591671db560faf742dd68493f
636844ce36f41641d854a1b239df91da3103873d3dfec0c25087582eec064e4d
34ad7b845707674e5f4f52e7bc60148a0971ec2f375d80ec3dc48387848973ba
4659dadbf5b07c8c3c36ae941f71b631737631bc3fded2fe2af250ceba98959a
5b7c970fee7ebe08d50665f278d47d0e34c04acc19a91838de6a3fc63a8e5630
4cf164497c275ae0f86c28d7847b10f5bd302ba12b995646c32cb53d03b7e6b5
ae086350239380f56470c19d6a200f7d251c7422c7bc5ce74730ee8bab8e6283
4659dadbf5b07c8c3c36ae941f71b631737631bc3fded2fe2af250ceba98959a
6dae368eecbcc10266bba32776c40d9ffa5b50d7f6199a9b6c31d40dfe7877d1

VII. Appendices

53e9bca505652ef23477e105e6985102a45d9a14e5316d140752df6f3ef43d2d
8fcd303e22b84d7d61768d4efa5308577a09cc45697f7f54be4e528bbb39435b
e79bbb45421320be05211a94ed507430cc9f6cf80d607d61a317af255733fcf2
eff3e37d0406c818e3430068d90e7ed2f594faa6bb146ab0a1c00a2f4a4809a5
1d0999ba3217cbdb0cc85403ef75587f747556a97dee7c2616e28866db932a0d
e40a46e95ef792cf20d5c14a9ad0b3a95c6252f96654f392b4bc6180565b7b11
ff2eb800ff16745fc13c216ff6d5cc2de99466244393f67ab6ea6f8189ae01dd
fee0081df5ca6a21953f3a633f2f64b7c0701977623d3a4ec36fff282ffe73b9
9f177a6fb4ea5af876ef8a0bf954e37544917d9aaba04680a29303f24ca5c72c
16fe4de2235850a7d947e4517a667a9bfcca3aee17b5022b02c68cc584aa6548
f51336e862b891f78f2682505c3d38ea7de5b0673d6ef7a3b0907c0996887c22
c9209951f7866849c9b1e5375bfb511b368394e52f6a276e86fdd542a79c2cd5
2223a93521b261715767f00f0d1ae4e692bd593202be40f3508cb4fd5e21712b
39cbad3b2aac6298537a85f0463453d54ab2660c913f4f35ba98fffeb0b15655
ae9a4e244a9b3c77d489dee8aeaf35a7c3ba31b210e76d81ef2e91790f052c85
32f24601153be0885f11d62e0a8a2f0280a2034fc981d8184180c5d3b1b9e8cf
ed01ebfbc9eb5bbea545af4d01bf5f1071661840480439c6e5babe8e080e41aa
1be0b96d502c268cb40da97a16952d89674a9329cb60bac81a96e01cf7356830
3e6de9e2baacf930949647c399818e7a2caea2626df6a468407854aaa515eed9
d8a9879a99ac7b12e63e6bcae7f965fbf1b63d892a8649ab1d6b08ce711f7127
32f24601153be0885f11d62e0a8a2f0280a2034fc981d8184180c5d3b1b9e8cf
ed01ebfbc9eb5bbea545af4d01bf5f1071661840480439c6e5babe8e080e41aa
aff73144a359020abbb4bde3f80858d822b840dd7171ba7946b77ba9b3487831
deefab8ee3d082119cc69c5dbdaf5faddeae36fbbd2345b1dc0463d07b65f13b
4e1c5141652acf8ea66b7d6dbb3fcdd96353e7d27c9e5698792c199aaf3f05c4
216d262e614e0bacff4e23077492ab9711b68b7ba2fbc17609ee1052093f59fc
95c2186be69601ae37f8269cb487f8f19d495b9f811908f90ec97bae9333db20
dcccd8859e532cc54f66f54e88fbe6eb52b3d5175233da65233bfddf49c165b4
e0b1ed0f1fb8648ccdbb8a844fef5cf9b3b9eb46902289122c508bbf7d2e8d6e
63d49254ee2d07ce08bd981743c17f3d5a3242478cea883332e0cc1ae43c0fe6
6cec00f9d3b7a34c899b1b0cdb69eb5356fa33b80144a10499b7ec905b12e903
7a57d3b9da733bf66894341e70ba5a0059f1046576d9f8ae07b7a48945bdda66
aff73144a359020abbb4bde3f80858d822b840dd7171ba7946b77ba9b3487831
302e75fe7e40e1637512e1c439d6fb3913945007428ed5d1a9bd198f08f38292
838286ef99986dbb65cf0b939e6c70a7fb7a47f79198b75c3c45a54a3c8666b6
16db0063e4aa666d94752414549fa09fb33142481d894b01a0fae45b339a09fb
49a63ae5e65bf75777d49d37eb1d23fd3f2f584ae57758e3016a312d9716fa9f
d246669cf1e25860f8601e456edd7156aa7304026ff4eadac18a2a82a18fabbf
315c06bd8c75f99722fd014b4fb4bd8934049cde09afead9b46bddf4cdd63171
480b0eb4636d6a78b62e7b52b773ec0a4e92fe4a748f9f9e8bd463a3b8dd0d83
f895757608b7725674628d731ec9fe90fd310eb65e7041bc6617ba0b831649b4
16db0063e4aa666d94752414549fa09fb33142481d894b01a0fae45b339a09fb
838286ef99986dbb65cf0b939e6c70a7fb7a47f79198b75c3c45a54a3c8666b6
eebc86e67a4a88f8cd5022adaa15b33a21ee609947dfcff75345f63d577bcd98
4659dadbf5b07c8c3c36ae941f71b631737631bc3fded2fe2af250ceba98959a
5b7c970fee7ebe08d50665f278d47d0e34c04acc19a91838de6a3fc63a8e5630
ae086350239380f56470c19d6a200f7d251c7422c7bc5ce74730ee8bab8e6283
4d4b17ddbcf4ce397f76cf0a2e230c9d513b23065f746a5ee2de74f447be39b9
45e68dce0f75353c448865b9abafbef5d4ed6492cd7058f65bf6aac182a9176a
05a567fe3f7c22a0ef78cc39dcf2d9ff283580c82bdbe880af9549e7014becfc
ae65288f5c96b4656402853b14acd1d060b2a6303d833df5b1f10cc7a34b0025
7cf5d86cc75cd8f0e22e35213a9c051b740bd4667d9879a446f06277782bffd1
26a2fa7b45a455c311fd57875d8231c853ea4399be7b9344f2136030b2edc4aa
ec254c40abff00b104a949f07b7b64235fc395ecb9311eb4020c1c4da0e6b5c4
c6930e298bba86c01d0fe2c8262c46b4fce97c6c5037a193904cfc634246fbec
16c3a7f143e831dd0481d2d57aae885090e22ec55cc8282009f641755d423fcd
7429a6b6e8518a1ec1d1c37a8786359885f2fd4abde560adaef331ca9deaeefd
e0cd4eb8108dab716f3c2e94e6c0079051bfe9c7c2ed4fcbfdd16b4dd1c18d4d
163571bd56001963c4dcb0650bb17fa23ba23a5237c21f2401f4e894dfe4f50d
efd470cfa90b918e5d558e5c8c3821343af06eedfd484dfeb20c4605f9bdc30e
f3ca8f15ca582dd486bd78fd57c2f4d7b958163542561606bebd250c827022de
c6930e298bba86c01d0fe2c8262c46b4fce97c6c5037a193904cfc634246fbec
6f76a8e16908ba2d576cf0e8cdb70114dcb70e0f7223be10aab3a728dc65c41c
851032eb03bc8ee05c381f7614a0cbf13b9a13293dfe5e4d4b7cd230970105e3
9a776b895e93926e2a758c09e341accb9333edc1243d216a5e53f47c6043c852
8dcca8c720fdb9833455427cd9b2146e2e9581e3bc595e8d97e562854133542b
70b494b0a8fdf054926829dcb3235fc7bd0346b6a19faf2a57891c71043b3b38
059aab1a6ac0764ff8024c8be37981d0506337909664c7b3862fc056d8c405b0
e08fc761cc22953de7fcc1684b7424755fa52f361dd5c6605b1469a80cb858bb
9bf8e8ac82b8f7c3707eb12e77f94cd0e06a972658610d136993235cbfa53641
357b5b8ba2dd4fb3196ba5ad45b7162d8115186bac3eb33b87f2942491656f8b
efd470cfa90b918e5d558e5c8c3821343af06eedfd484dfeb20c4605f9bdc30e
ccafbcff1596e3dfd28dcb97a5ba85e6845e69464742edfe136fe09bbec86ba1
f9686467a99cdb3928ccf40042d3e18451a9db97ef60f098656725a9fc3d9025
44884565800eebf41185861133710b4a42a99d80b6a74436bf788c0e210b9f50

APPENDIX C: SMB BRUTEFORCE PASSWORD LIST

This password list has been used on numerous occasions by Lazarus to perform SMB bruteforce attacks.

```
!@#$
!@#$%
!@#$%^
!@#$%^&
!@#$%^&*
!@#$%^&*()
"KGS!@#$%"
0000
00000
000000
00000000
1111
11111
111111
11111111
11122212
1212
121212
123123
123321
1234
12345
123456
1234567
12345678
123456789
123456^%$#@!
1234qwer
123abc
123asd
123qwe
1313
1q2w3e
1q2w3e4r
1qaz2wsx
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
4321
54321
654321
6969
666666
7777
8888
88888
888888
8888888
88888888
Admin
abc123
abc@123
abcd
admin
admin123
admin!23
admin!@#
administrator
administrador
asdf
asdfg
asdfgh
asdf123
asdf!23
baseball
backup
blank
cisco
compaq
control
computer
cookie123
database
dbpassword
db1234
default
dell
enable
fish
foobar
gateway
guest
golf
harley
home
iloveyou
internet
letmein
Login
login
love
manager
oracle
owner
pass
passwd
password
p@ssword
password1
password!
passw0rd
Password1
pa55w0rd
pw123
q1w2e3
q1w2e3r4
q1w2e3r4t5
q1w2e3r4t5y6
qazwsx
qazwsxedc
qwer
qwert
qwerty
!QAZxsw2
root
secret
server
sqlexec
shadow
super
sybase
temp
temp123
test
test!
test1
test123
test!23
winxp
win2000
win2003
Welcome1
Welcome123
xxxx
yxcv
zxcv
Administrator
Admin
```

VII. Appendices

REFERENCES

1. Operation Blockbuster, Novetta, 2016
2. HiddenCobra, US-CERT
3. FBI boss: Sony hack was DEFINITELY North Korea, haters gonna hate, The Register, 2015
4. APT37 (Reaper) - The Overlooked North Korean Actor, FireEye, 2018
5. APT38 Un-usual suspects, FireEye, 2018
6. Big Game Hunting with Ryuk: Another Lucrative Targeted Ransomware, CrowdStrike, 2019
7. Targeted TrickBot activity drops ‘PowerBrace’ backdoor, NttSecurity, 2019
8. OSINT Reporting Regarding DPRK and TA505 Overlap, NorfolkInfosec, 2019
9. DPRK cyberattack timeline, Intezer
10. Sanctions against North-Korea, Wikipedia
11. From stealing confidential data to revenue-generating attacks, AhnLab, 2018
12. Full Discloser of Andariel, A Subgroup of Lazarus Threat Group, AhnLab, 2018
13. Lazarus & Watering-hole attacks, BAE Systems Threat Research Blog, 2017
14. NATION-STATE MONEYMULE’S HUNTING SEASON - APT ATTACKS TARGETING FINANCIAL INSTITUTIONS, BlackHat EU, 2017
15. Alert (TA18-275A) HIDDEN COBRA – FASTCash Campaign, US-CERT, 2018
16. FASTCash: How the Lazarus Group is Emptying Millions from ATMs, Symantec, 2018
17. North Korea Bitten by Bitcoin Bug - Financially motivated campaigns reveal new dimension of the Lazarus Group, Proofpoint, 2017
18. Lazarus Under The Hood, PDF, Securelist, 2017
19. Group-IB: 14 cyber attacks on crypto exchanges resulted in a loss of $882 million, Group-IB, 2018
20. Operation AppleJeus: Lazarus hits cryptocurrency exchange with fake installer and macOS malware, Securelist, 2018
21. North Korea Bitten by Bitcoin Bug: Financially motivated campaigns reveal new dimension of the Lazarus Group, Proofpoint, 2017
22. Operation AppleJeus: Lazarus hits cryptocurrency exchange with fake installer and macOS malware, Securelist, 2018
23. Mac Backdoor Linked to Lazarus Targets Korean Users, TrendMicro, 2019
24. Android Malware Appears Linked to Lazarus Cybercrime Group, McAfee, 2017

25. Lazarus Arisen - ARCHITECTURE / TOOLS / ATTRIBUTION, Group-IB, 2017
26. Disclosure of Chilean Redbanc Intrusion Leads to Lazarus Ties, Flashpoint, 2019
27. Lazarus Arisen - ARCHITECTURE / TOOLS / ATTRIBUTION, Group-IB, 2017
28. Lazarus' Flase Flag Malware, BAE Systems Threat Research Blog, 2017
29. Alert (TA17-318A) - HIDDEN COBRA – North Korean Remote Administration Tool: FALLCHILL, US- CERT, 2017
30. Android Malware Appears Linked to Lazarus Cybercrime Group, McAfee, 2017
31. Dissecting Operation Troy:Cyberespionage in South Korea, McAfee,2013
32. Lazarus Under The Hood, PDF, Securelist, 2017
33. Lazarus Arisen - ARCHITECTURE / TOOLS / ATTRIBUTION, Group-IB, 2017
34. Lazarus Group, MITRE ATT&CK
35. Lazarus Group, MITRE ATT&CK
36. Lazarus Continues Heists, Mounts Attacks on Financial Organizations in Latin America, TrendMicro, 2018
37. Malpedia
38. The devil's in the Rich header, Securelist, 2018
39. APT37 (Reaper) - The Overlooked North Korean Actor, FireEye, 2018
40. Alert (TA17-318A) - HIDDEN COBRA – North Korean Remote Administration Tool: FALLCHILL, US- CERT, 2017

CONTACT

LEXFO
OFFENSIVE SECURITY

5 RUE DROUOT 75009 PARIS | + 33 (0) 1 40 17 91 28 | CONTACT@LEXFO.FR

WWW.LEXFO.FR

@LEXFOSECURITE

/COMPANY/LEXFO/

PRIS@LEXFO.FR

White Paper

The Lazarus Constellation

A study on North Korean malware

06/01/2020

LEXFO

avisa partners