

# Rana Android Malware

 [blog.reversinglabs.com/blog/rana-android-malware](https://blog.reversinglabs.com/blog/rana-android-malware)



Threat Research | December 7, 2020



Blog Author

Karlo Zanki, Reverse Engineer at ReversingLabs. [Read More...](#)



## Introduction

---

On September 17th, the U.S. Department of the Treasury's Office of Foreign Assets Control imposed sanctions on Iranian cyber threat group APT39, also known as Chafer, Cadelspy, Remexi, and ITG07. On the same day, the FBI released a public threat analysis report describing several tools used by Rana Corp, a front company backed by the Iranian Ministry of Intelligence and Security (MOIS) which is behind the malicious cyber activities conducted by the APT39 group.

The tools described in the aforementioned report are implemented using different technologies including VisualBasic and Autoit scripts, Android applications and the more common PE executables. According to the report, the focus of Rana's cyber activities is tracking the movements of individuals whom the MOIS considers a threat. In today's world, the most valuable source of such information are smartphones. You carry a smartphone almost the entire time, and, besides being the main tool for everyday communication, smartphones also provide a large set of secondary functionalities, including visual and audio recording and location services. Because of all these capabilities, gaining control over someone's smartphone provides the malicious actor with a powerful espionage tool. For these reasons, we decided to take a better look at the information and IOCs provided in the referenced report to see if there is anything more to be found about this Android malware.

## Related samples

---

The report provided one sample hash of the malware APK file, and a YARA rule which can be used to search for additional samples. The first encountered problem was that the provided YARA rule wasn't triggered on the provided APK sample. It was triggered only on the .dex file contained inside the APK. This means that the APK sample can pass undetected if its contents are not extracted.

So, we had to take a different approach to find additional samples. Luckily, the Titanium Platform provides multiple ways to search for related samples based on metadata extracted from Android applications. The provided sample was signed with a self-signed certificate. Since searching for samples signed with the same certificate (based on the certificate thumbprint) didn't give any results, a more generic search query was required.

**0c23f62ba98ebfa2c062c485e...**  
[Preview Sample](#)

Size: 185.6 KB  
Type: Binary / Archive  
Format: Android:Generic  
Threat: ● Android.Trojan.Dingwe  
First seen (**cloud**): 2020-09-17 22:06 UTC  
Last seen (**local**): 2020-10-16 09:13 UTC  
User uploads: 1

---

**Summary**

- TitaniumCore
- ▼ Info
    - File
    - Hashes
    - Package
    - Validation
    - Statistics
  - ▼ Application (Android)
    - Capabilities
    - Package
    - Activities
    - Services
    - Receivers
    - Permissions
    - Features
  - ▼ **Certificates**
    - Signer information
  - ▼ Classification
    - Scanners
    - Tags

### Certificate Trust Chain ^

▼ **Company Name**

     SIGNER: Company Name

Serial Number	763FAA62	
Subject	Country Name	Country Code
	State Orprovince...State	
	Locality Name	Location
	Organization Na... Organisation	
	Organizational U... Organisational Unit	
	Common Name	Company Name
Issuer	Country Name	Country Code
	State Orprovince...State	
	Locality Name	Location
	Organization Na... Organisation	
	Organizational U... Organisational Unit	
	Common Name	Company Name
Expired	<b>No</b>	
Valid	2018-12-23 23:47 UTC to 2073-09-25 23:47 UTC	
Version	2	
Thumbprint	Algorithm	Sha256
	Value	531 F740 C519 Abd1 B960 Fe4 Ffe239 E3 Dc235 C9941 D0 D121126557 B3 Cd8543 B0 D0

## Certificate details

Looking at the certificate details reveals that malicious actors didn't even try to make the certificate look the least bit legitimate. They decided to leave all the default values for the fields describing the certificate receiver. These certificate fields look specific enough and can be used to detect related samples. We used a modified search query to find all Android APK files signed with a certificate issued to an organizational unit "Organisational Unit" in a company with the country "Country Code".

format:Binary/Archive/Android AND cert-subject-country:"Country Code" AND cert-subject-unit:"Organisational Unit"

Local (2) Cloud - Shareable (3) Private (6) Export

<input type="checkbox"/>	First Seen	Threat	Name	Format	Files	Size
<input type="checkbox"/>	4 years ago	Android.Trojan.Dingwe	c2694dae46fd2846368731d92e810f32c2c9a2f9	Binary/Archive/A...	1	121.7 KB
<input type="checkbox"/>	4 years ago	Android.Trojan.Boogr	c552f74bf23211428b7fab141a72db9073a98729	Binary/Archive/A...	1	121.6 KB
<input type="checkbox"/>	4 years ago	Android.Trojan.Dingwe	28fa9354be6ce503ee7c1f7615a26cdd99d7b801	Binary/Archive/A...	1	181.5 KB

1 3 results 100 ^

## Search based on certificate data

This resulted in three new samples. Analyzing these samples with the Titanium Platform showed that all of them had the same “**com.android.providers.optimizer**” package name as the sample provided in the report. We can assume that these are samples of the same malware. Comparing the time when the samples were first seen in the ReversingLabs cloud with the time of validation of the certificates used for signing the APK binaries can give us a clue about the evolution of the malware.

Sample SHA1	Cert valid from	First seen
28fa9354be6ce503ee7c1f7615a26cdd99d7b801	2016-01-04 13:14	2016-06-21 16:39
c2694dae46fd2846368731d92e810f32c2c9a2f9	2016-06-01 00:11	2016-11-19 05:59
c552f74bf23211428b7fab141a72db9073a98729	2016-06-26 11:09	2016-10-15 20:51
0c23f62ba98ebfa2c062c485e5704f193909e421	2018-12-23 23:47	2020-09-17 22:06

## Time relations between malware samples

## Reversing the sample

The found samples are obviously older versions of malware, which were analyzed and compared to the version described in the report. A brief static analysis shows that the older samples require a more extensive list of permissions, and that they don't include the libOptimizer.so file which is, as described in the report, responsible for the generation of the AES key. Older versions also don't include the tmp.tmp resource referenced by the mentioned libOptimizer.so file.

## Permissions

```
android.permission.ACCESS_COARSE_LOCATION
android.permission.ACCESS_FINE_LOCATION
android.permission.ACCESS_NETWORK_STATE
android.permission.CAMERA
android.permission.INTERNET
android.permission.READ_CALENDAR
android.permission.READ_CONTACTS
android.permission.READ_EXTERNAL_STORAGE
android.permission.READ_PHONE_STATE
android.permission.READ_SMS
android.permission.RECEIVE_BOOT_COMPLETED
android.permission.RECORD_AUDIO
```

## Permissions

```
android.permission.ACCESS_COARSE_LOCATION
android.permission.ACCESS_FINE_LOCATION
android.permission.ACCESS_NETWORK_STATE
android.permission.ACCESS_WIFI_STATE
android.permission.BLUETOOTH
android.permission.BLUETOOTH_ADMIN
android.permission.CALL_PHONE
android.permission.CAMERA
android.permission.CHANGE_NETWORK_STATE
android.permission.CHANGE_WIFI_STATE
android.permission.GET_TASKS
android.permission.INTERNET
android.permission.MODIFY_AUDIO_SETTINGS
android.permission.PROCESS_OUTGOING_CALLS
android.permission.READ_CALENDAR
android.permission.READ_CONTACTS
android.permission.READ_EXTERNAL_STORAGE
android.permission.READ_LOGS
android.permission.READ_PHONE_STATE
android.permission.READ_SMS
android.permission.RECEIVE_BOOT_COMPLETED
android.permission.RECEIVE_SMS
android.permission.RECORD_AUDIO
android.permission.SEND_SMS
android.permission.WAKE_LOCK
android.permission.WRITE_CALENDAR
android.permission.WRITE_CONTACTS
android.permission.WRITE_EXTERNAL_STORAGE
android.permission.WRITE_SETTINGS
android.permission.WRITE_SMS
com.android.browser.permission.READ_HISTORY_BOOKMARKS
```

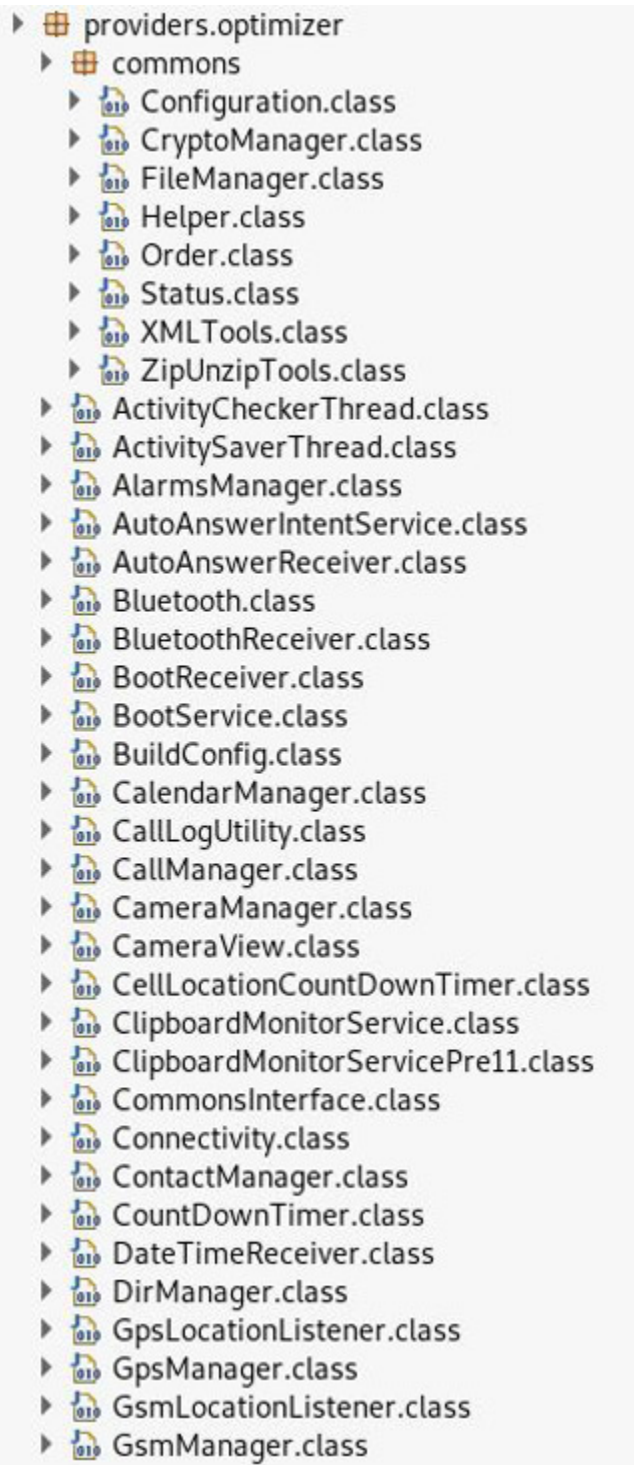
### Permissions requested by the newer and the older version

For a better understanding of the malware's capabilities, the Dalvik bytecode inside classes.dex file was converted to a corresponding .jar file with the open-source tool Dex2jar. The contents of the .jar file were then decompiled to original Java source code with the [Java Decompiler](#). As mentioned in the report where the same procedure was performed, the decompiled output included around 200 classes and almost 600 methods, all with obfuscated names. This obfuscation is performed at the moment when APK files are generated, and is most often done using tools like ProGuard.

When analyzing such a big number of generically named methods (method a, b, c...) some interesting functionality can often remain undetected. Finding an older malware sample that didn't use obfuscation would make the analysis much easier. For some lucky reason, sample 28fa9354be6ce503ee7c1f7615a26cdd99d7b801 was not obfuscated. This made the

analysis much simpler. The Java source code contained all the original class and method

names.



### Unobfuscated class files

Comparing source code of the obfuscated and unobfuscated samples shows that the codebase is very similar. Most of the functionalities are identical, and can be easily understood when looking at unobfuscated code. Probably the best example are the obfuscated SMS command codes.

```

public static String b = "-saau";
public static String c = "-gurls";
public static String d = "-surls";
public static String e = "-gspns";
public static String f = "-sspns";
public static String g = "-gdvcinf";
public static String h = "-satawr";
public static String i = "-gatawrpns";
public static String j = "-satawrpns";
public static String k = "-smtrtxcb";
public static String l = "-gcntcs";
public static String m = "-gclls";
public static String n = "-gsmss";
public static String o = "-gclrevs";
public static String p = "-gbwrhty";
public static String q = "-gildaps";
public static String r = "-sblth";
public static String s = "-sprfle";
public static String t = "-swifi";
public static String u = "-updfle";
public static String v = "-gdirtry";
public static String w = "-gstgsze";
public static String x = "-ggsmlcn";
public static String y = "-sgsmlcn";

static {
    GetSmsPhoneNumbers = "-gspns";
    SetSmsPhoneNumbers = "-sspns";
    GetDeviceInformation = "-gdvcinf";
    SetAutoAnswer = "-satawr";
    GetAutoAnswerPhoneNumbers = "-gatawrpns";
    SetAutoAnswerPhoneNumbers = "-satawrpns";
    SetMonitorTextClipboard = "-smtrtxcb";
    GetContacts = "-gcntcs";
    GetCalls = "-gclls";
    GetSMSs = "-gsmss";
    GetCalendarEvents = "-gclrevs";
    GetBrowserHistory = "-gbwrhty";
    GetInstalledApplications = "-gildaps";
    SetBluetooth = "-sblth";
    SetProfile = "-sprfle";
    SetWiFi = "-swifi";
    UploadFile = "-updfle";
    GetDir = "-gdirtry";
    GetStorageSize = "-gstgsze";
    GetGSMLoction = "-ggsmlcn";
    SetGSMLoction = "-sgsmlcn";
    SetGSMTracking = "-sgsmtkg";
    SetGPSTracking = "-sgpstkg";
    UploadCameraTakenPhotos = "-gcmapcr";
    GetCurrentOrderFile = "-gctodfle";
    TakeCameraPhoto = "-tkephto";
    AudioRecord = "-adorcd";
    WifiAddWorkRemove = "-wifiawr";
}

```

## Comparison of obfuscated and unobfuscated commands

Besides the commands visible in the list, there are a few more quite interesting commands that support the hypothesis that this malware is used for government surveillance purposes, and these were not explicitly mentioned in the report. The first one is the ability to take camera photos at login success or failure. This can obviously be used to visually identify any person who tries to use the mobile device. Another typical surveillance feature is highly configurable audio recording. Recording can be scheduled periodically, or at specific moments, with a configurable recording duration. The malware also enables scheduling a device boot at some specific moment, ensuring malware activation even when someone turns off the phone.

Besides these commands which are common for Android malware, there are a few not so common ones. The first one is the ability to add a custom WiFi access point and to force the device to connect to it. This feature was probably introduced to avoid possible detection due

to unusual data traffic usage on the target's mobile account. The second one is the ability to automatically answer calls from specific phone numbers. Such a feature can also be used for audio tapping, and is probably used when tapping needs to be started urgently and couldn't be previously scheduled.

The malware supports receiving commands sent by SMS. In that case, the malware intercepts the received SMS and, if it starts with a predefined command header, the malware aborts further propagation of the SMS\_RECEIVED Intent. This prevents the received SMS from ending up in the default SMS application. The predefined command header is usually set to "**opt -cmd**", but is configurable and can be set to different values.

```
public class SmsListener extends BroadcastReceiver {
    public void onReceive(Context paramContext, Intent paramInt) {
        try {
            if (paramInt.getAction().equals("android.provider.Telephony.SMS_RECEIVED")) {
                Bundle bundle = paramInt.getExtras();
                if (bundle != null) {
                    Object[] arrayOfObject = (Object[])bundle.get("pdus");
                    SmsMessage[] arrayOfSmsMessage = new SmsMessage[arrayOfObject.length];
                    byte b = 0;
                    while (true) {
                        if (b < arrayOfSmsMessage.length) {
                            arrayOfSmsMessage[b] = SmsMessage.createFromPdu((byte[])arrayOfObject[b]);
                            String str = arrayOfSmsMessage[b].getMessageBody();
                            if (str.indexOf(Configuration.Consts.SMS_CMD_HEADER) == 0) {
                                abortBroadcast();
                                Order.updateOrderBySmsLocked(str);
                            }
                            b++;
                            continue;
                        }
                    }
                    return;
                }
            }
        } catch (Exception exception) {}
    }
}
```

## Receiving commands using SMS messages

There is one interesting functionality that isn't present in the older versions, but is in the latest, and hasn't been explicitly mentioned in the FBI report. The latest version uses accessibility services to get contents of instant messaging applications.

```
|| paramAccessibilityEvent.getSource().getPackageName().toString().equals("org.ir.talaeii")
```

## Code related to monitoring IM applications



Looking at the monitored IM applications additionally proves that this malware is probably used for the surveillance of Iranian citizens. One of the monitored IM applications is a package named “**org.ir.talaeii**”, which is described as “an unofficial Telegram client developed in Iran”. The full list of monitored IM applications can be found in the following table:

com.instagram.android
com.skype.raider
org.telegram.messenger
org.ir.talaeii
com.viber.voip
com.whatsapp
com.imo.android.imoim

**List of monitored IM applications**

Generally, most of the settings are configurable. The initial configuration is provided in the cgn.cn resource. Resource contents are encrypted with the AES algorithm. Unlike the latest version which uses the libOptimizer.so file to generate the AES key, older versions use a hardcoded key provided inside the binary. It is constructed in the same way as described in the report. The 96-byte base64 string “**U3wzfSpoSkBrlys7...**” is decoded to a 72-byte key “S|3}\*hJ@k#+;!b-F...”, and only the first 16-bytes of the decoded 72-byte key are then used for AES decryption.

When analyzing Android binaries, it is always good to take a detailed look at the resources contained in the “res/raw/” folder. This is usually the place for application-specific binaries, and malware authors sometimes use it for the same purpose. Unlike the newer version, the older ones have only three raw resources and don’t contain the tmp.tmp resource. The att.cn resource usually contains the plaintext “Test” string. The previously mentioned cgn.cn file contains the AES encrypted configuration that includes configuration settings for malware functionalities, C2 domains, and another AES encryption key that is used to decrypt the last raw resource odr.od. This resource contains preconfigured commands that are to be

scheduled for execution by the malware.

28fa9354be6ce503ee7c1f7615a26cdd99d... / ... / raw /

<input type="checkbox"/>	Threat	File Name	Format	Files	Size	
<input type="checkbox"/>	● --	att.cn	Text/None	1	4 Bytes	≡
<input type="checkbox"/>	● --	cng.cn	Binary/None	1	2.5 KB	≡
<input type="checkbox"/>	● --	odr.od	Binary/None	1	608 Bytes	≡

1 - 3 of 3 items

### Resources contained in the “res/raw/” folder

## Threat actors

Each malware sample contained a list of C2 domains embedded in their configuration file. Historical domain registration data was searched for each of the extracted domains.

Sample SHA1	C2 domains	Domain valid from	Cert valid from
28fa9354be6ce503ee7c1f7615a26cdd99d7b801	wherisdomaintv.com whoisdomainpc.com	2014-12-25 2014-12-25	2016-01-04 13:14
c2694dae46fd2846368731d92e810f32c2c9a2f9	fullplayersoftware.com softwareplayertop.com	2016-06-01 2016-06-01	2016-06-01 00:11
c552f74bf23211428b7fab141a72db9073a98729	whoisdomainpc.com	2014-12-25	2016-06-26 11:09
0c23f62ba98ebfa2c062c485e5704f193909e421	saveingone.com	2018-12-24	2018-12-23 23:47

### Resources contained in the “res/raw/” folder


Comparing the domain registration dates with the certificate validation dates extracted from the malicious samples shows that, in some cases, the domains were registered at the time when the samples were signed (created). In other cases, the domain was registered about a year before the samples were signed. This suggests that the earlier versions of this malware reused the domains, and that it was active since the end of 2014. This further suggests that older, and currently undetected, versions of this malware probably exist as well.

Looking at the domain registrant data for the extracted domains reveals more connections to Iran. [DomainBigData](#) service was used to search WHOIS records for the “whoisdomainpc.com” domain. The records show that this domain was registered by an

Iranian resident named [redacted.1] using the email address [redacted.1]domain@chmail.ir.

Recorded : 2014-12-26

### Historic Registrant

Name	[redacted]	is associated with 8 domains
Email	[redacted]domain@chmail.ir	is associated with 7 domains
Address	tehran	
City	tehran	
State	tehran	
Country	 Iran (Islamic Republic Of)	
Phone	+98.21 [redacted] 281	
Private	no	

### Historic Whois Record

Domain Name: whoisdomainpc.com  
Registry Domain ID: 1892293605\_DOMAIN\_COM-VRSN  
Registrar WHOIS Server: whois.realtimeregister.com  
Registrar URL: http://www.realtimeregister.com  
Updated Date: 2014-12-25T08:29:48Z  
Created Date: 2014-12-25T07:18:10Z  
Registrar Registration Expiration Date: 2019-12-25T07:18:10Z  
Registrar: REALTIME REGISTER B.V.  
Registrar IANA ID: 839  
Registrar Abuse Contact Email: abuse[at]realtimeregister.com  
Registrar Abuse Contact Phone: +31.384530759  
Reseller: Hostcontrol  
Domain Status: ok  
Domain Status: renewPeriod  
Registry Registrant ID:  
Registrant Name: [redacted]  
Registrant Organization:  
Registrant Street: tehran  
Registrant City: tehran  
Registrant State/Province: tehran  
Registrant Postal Code: 1589947811  
Registrant Country: IR  
Registrant Phone: +98.21 [redacted] 281

## WHOIS data for whoisdomainpc.com

The same person is associated with 7 more domains, including some of the already found ones. Domains “**facedomainpc.com**” and “**facedomaintv.com**” are interesting since they follow the same naming convention ending in \*.pc.com and \*.tv.com, and were also registered on the same date as the domains used in the found malware samples. It is quite possible that all of these domains were used, or were meant to be used, in some kind of a malicious activity.

Domain Name	Creation Date	Registrar
<a href="#">facedomainpc.com</a>	2014-12-25	joker.com
<a href="#">facedomaintv.com</a>	2014-12-25	joker.com
<a href="#">wherisdomaintv.com</a>	2014-12-25	joker.com
<a href="#">whoisdomainpc.com</a>	2014-12-25	joker.com
<a href="#">ccloudflare.com</a>	2018-02-14	tu cowsdomains.com
<a href="#">sadostad.com</a>	2015-10-22	realtimeregister.com
<a href="#">irchemistry.net</a>	2016-04-23	realtimeregister.com
<a href="#">irchemistry.com</a>	2016-04-23	realtimeregister.com

### List of domain names registered by [redacted.1]

The list of domains registered by the same email account reveals two more domains registered by the same person. These newly discovered domains were registered 3 days before the already mentioned ones.

Domain Name	Creation Date	Registrar
<a href="#">facedomainpc.com</a>	2014-12-25	joker.com
<a href="#">facedomaintv.com</a>	2014-12-25	joker.com
<a href="#">whoisdomainpc.com</a>	2014-12-25	joker.com
<a href="#">ccloudflare.com</a>	2018-02-14	tu cowsdomains.com
<a href="#">wherisdomaintv.com</a>	2014-12-25	joker.com
<a href="#">lifedomainwar.com</a>	2014-12-22	realtimeregister.com
<a href="#">elfdomainone.com</a>	2014-12-22	realtimeregister.com

### List of domain names registered by [redacted.1]domain@chmail.ir

Three more domains were related to the Iranian national, but a different email address was used for their registration. Domains “**sadostad.com**”, “**irchemistry.net**” and “**irchemistry.com**” were registered using the [redacted.1]@gmail.com account.

The RiskIQ service was used to search for more email accounts related to the Iranian national. A very similar account was detected: [redacted.1]@ymail.com. It was also used to register domain names similar to previous ones (sadostad.ir - sadostad.com). Two more, probably related, accounts were also found, and they were used more recently to register two more domains. Even though no concrete proof can be given to reliably declare these domains malicious, the aforementioned relations should be enough to keep you alert and

treat them as suspicious.

Focus	Email	Registered	Expires
chembook.ir	[redacted]@ymail.com	2017-08-27	2019-11-17
100ostad.ir	[redacted]@ymail.com	2017-08-27	2020-11-07
sadostad.ir	[redacted]@ymail.com	2017-08-27	2020-11-07
[redacted]online.ir	[redacted]@ymail.com	2016-06-15	2017-06-12

Focus	Email	Registered	Expires
ctci.ir	[redacted]70@gmail.com	2020-02-17	2023-02-07

Focus	Email	Registered	Expires
ktci.ir	[redacted]2012@outlook.com	2019-06-25	2021-02-06



### List of related domain names and email addresses

Using one of the Iranian domain name registration services, [selva.ir](http://selva.ir), both the registrant data for the 100ostad.ir domain and address information of the registrant were found.

100ostad.ir	:Domain name
[redacted]	:Copyright Name
[redacted]59@ymail.com	:Franchise Email
442 [redacted] 2198+	: Phone number
1399/8/17	: Expiration date
1399/8/17	: last update
[redacted] Tehran, Tehran, IR	: Address

### [redacted.1]'s potential addresses

The second interesting person related to the domains “fullplayersoftware.com” and “softwareplayertop.com” is [redacted.2] . The domains were registered using the [redacted.2]@gmail.com email account.

👤 Registrant		🕒 Recorded : 2016-06-02	
Name	[redacted]	👤 Historic Registrant	
Email	[redacted]@gmail.com	Name	[redacted]
Address	Tehran [redacted]	Email	[redacted]@gmail.com
City	Tehran	Address	noadress noneadress
Country	 Iran (Islamic Republic Of)	City	noname
Phone	+98. [redacted] 776	State	jivhk
Fax	+98. [redacted] 776	Country	 Iran (islamic Republic Of)
Private	no	Phone	+98.21 [redacted] 744
		Fax	+98.21 [redacted] 744

### [redacted.2]’s registration data

## Conclusion

---

It’s important to remember that there are many reasons that cause threat groups to turn their focus to specific targets. Whether it’s political dissidents, opposition in countries under authoritarian regimes, or corporations the threat actors goal is to make gains monetarily or politically. When targeting individuals, threat actors often want to monitor their communication and movement. Mobile phones are most suitable for such goals because of the computing power contained in your pocket, and the fact that most people carry them all the time. Since the Android platform maintains the biggest part of the global smartphone market share, it follows that it is also the primary target of mobile malware.

Development environments used in the creation of Android applications are often preconfigured to perform obfuscation by default making malicious apps an easy target. Since malware actors very often use the same tools as legitimate developers, malware analysts frequently have to find ways to overcome the obfuscation barriers in order to study how a particular malware works. Finding an older, unobfuscated, version of malware can be an excellent workaround for this problem. Older versions are also very interesting because malware authors often primarily focus on implementing the necessary functionalities before trying to hide clues that can later be used to associate them to the malware.

Fortunately, older malware samples can be a real treasure in analysis. Therefore it is always good to search for related samples first - finding an older one can be worth the effort, and can make research much simpler. What we can take away from this analysis is the importance of maintaining control over your device to reduce the risk of infection. On an individual level this includes knowing which apps have access to microphones and sensitive

information. If you are part of a government agency, or even a private corporation, it means having a solid BYOD policy, that includes application control, continually auditing the system setting, and malware scanning.

Keep in mind that any person-related data in this blog post should be taken with caution since there is always a possibility that such information is forged or stolen.

## IOC list

---

The following links contain the data extracted from the discovered samples related to the Rana Android malware.

[https://blog.reversinglabs.com/hubfs/Blog/rana\\_android\\_malware/IOC\\_SHA1\\_list.txt](https://blog.reversinglabs.com/hubfs/Blog/rana_android_malware/IOC_SHA1_list.txt)

[https://blog.reversinglabs.com/hubfs/Blog/rana\\_android\\_malware/IOC\\_C2\\_list.txt](https://blog.reversinglabs.com/hubfs/Blog/rana_android_malware/IOC_C2_list.txt)

[https://blog.reversinglabs.com/hubfs/Blog/rana\\_android\\_malware/IOC\\_suspicious\\_domains.txt](https://blog.reversinglabs.com/hubfs/Blog/rana_android_malware/IOC_suspicious_domains.txt)

## YARA rule

---

```
rule Rana_Android_resources {
  strings:
    $res1 = "res/raw/cng.cn" fullword wide ascii
    $res2 = "res/raw/att.cn" fullword wide ascii
    $res3 = "res/raw/odr.od" fullword wide ascii
  condition:
    any of them /* any string in the rule */
}
```

## MORE BLOG ARTICLES

---