

Analysis-Report Chinese Police App “IJOP” 12.2018

Several Members of the Cure53 Team

Index

[Introduction](#)

[Scope of the Assessment](#)

[Classification of Findings](#)

[Questions & Answers](#)

[XJ1-01-001 WifiDetector class implementing War Driving database \(Assumed\)](#)

[XJ1-01-002 NewCollection activity tracks blood type and political views \(Proven\)](#)

[XJ1-01-003 App feature of scanning for book titles \(Unclear\)](#)

[XJ1-01-004 Local database of religious atmosphere & re-education \(Assumed\)](#)

[XJ1-01-005 App receives electricity consumption data from HQ \(Proven\)](#)

[XJ1-01-006 Reporting feature for problematic tools \(Proven\)](#)

[XJ1-01-007 Problem Survey Feed tracks the use of crypto tools & VPNs \(Evident\)](#)

[XJ1-01-008 PII collection via OverduePersonAddFeedActivity \(Assumed\)](#)

[XJ1-01-009 PII collection via SuspiciousPhone\[List\]Activity \(Assumed\)](#)

[XJ1-01-010 PII collection via PeerPersonnel Add/List activities \(Assumed\)](#)

[XJ1-01-011 PII collection via MapRadioPersonnelDetailActivity \(Assumed\)](#)

[XJ1-01-012 PII data via OutOfPersonDetailActivity \(Assumed\)](#)

[XJ1-01-013 PII data via InfluxPersonDetailActivity \(Assumed\)](#)

[XJ1-01-014 PII collection via MissPhoneTrailFeedbackActivity \(Assumed\)](#)

[XJ1-01-015 PII collection via FourAssociationFeedbackActivity \(Assumed\)](#)

[XJ1-01-016 PII collection via VehicleAddActivity \(Assumed\)](#)

[XJ1-01-017 PII collection via ToolsAddActivity \(Assumed\)](#)

[XJ1-01-018 PII collection via BankAddActivity \(Assumed\)](#)

[XJ1-01-019 Mention of the 26 sensitive countries for Uyghurs \(Unclear\)](#)

[XJ1-01-020 Usage of the terms “Picked” and “Radio” \(Unclear\)](#)

[XJ1-01-021 Usage of the terms “illegal”, “suspicious” and “problem” \(Unclear\)](#)

[XJ1-01-022 HQ communications for investigations and arrest \(Assumed\)](#)

[XJ1-01-023 Review of Warn activities \(Unclear\)](#)

[XJ1-01-024 Review of AntiRefluxActivity \(Assumed\)](#)[Conclusions](#)[Appendix](#)[Appendix 1: Working version of app APK and instructions on how to use it](#)[Appendix 2: Guide for checking relations between activities](#)

Introduction

“The Integrated Joint Operations Platform (IJOP) is a policing program based on big data analysis in Xinjiang, one of the most repressive regions in the world. The program aggregates data about people – often without their knowledge – and flags those it deems potentially threatening to officials.”

From Memo on IJOP provided by HRW

This report documents the findings of a Cure53 assessment targeting the Integrated Joint Operations Platform (IJOP) mobile app. This project was requested by Human Rights Watch (HRW) and funded by Open technology Fund (OTF). The main objective of the assessment was to find out whether the IJOP mobile application compound could violate human rights.

More specifically, Cure53 carried out a source code audit and a dedicated review of the IJOP mobile application in late November 2018. The project followed a so-called white box methodology as much as possible, particularly in a sense that the assessment included reviewing and targeted investigations of the decompiled source code of the application, which had been made available to Cure53 as an APK file by the HRW. In addition, HRW furnished other, previously acquired data and material to Cure53, so that a better coverage could be accomplished.

The assessment took a total of ten days, excluding a separate budget allocated to preparations, communications and write-up of this report. Note that the project was executed stealthily, as the Cure53 team aimed at avoiding detection. This meant refraining from any form of behavior that could be perceived as “noisy” and, thus, alert the maintainers of the application and/or server owners. As noted above, the assessment tackled both the application source code and the running application which has been repacked with English translations. The project also entailed a review of the APK-exposed activities.

Throughout this assignment, the Cure53 and HRW teams maintained frequent contact on a shared Slack channel. This allowed for fast communications and a good flow during

the test, additionally ascertaining that Cure53 does not pursue exploration avenues and directions useless from the HRW's point of view.

It should be emphasized that the European Convention on Human Rights (ECHR) served as a baseline for this project. To reiterate, Cure53 set out to determine - through the technical reviews and audits - what communication capabilities and functionality of the IJOP mobile application can be seen as standing in direct opposition to what the ECHR guarantees. Therefore, this was a search for any human rights violations that the IJOP's operational scope could enable and facilitate.

For the purpose of documenting the findings, Cure53 devised a classification system that could help discern the level of harm and certainty about each potential violation of human rights. This system is discussed in more detail below. At this stage, it can nevertheless be stated that Cure53 managed to prove four cases of clear human rights violations. All other items, in a vast array of as many as twenty-four total discoveries, should also be considered, yet only in the context of having all vital information at hand. In other words, twenty suspicious behaviors should be evaluated by HRW further as to whether the supplied evidence is sufficient or needs to be extended with more data.

In the following sections the report first elaborates on the scope and then sheds light on the link between severities of the findings and the employed classification system. The questions posed by HRW, along with detailed answers and research results, are discussed next. Further, Cure53 reiterates some of the key points in the *Conclusions* and, last but not least, supplies a selection of attachments that are relevant to this report. It is hoped that this will facilitate possible follow-up research or other actions that will be taken by HRW or any other involved or interested parties.

Scope of the Assessment

- **Chinese Police App**
 - In scope of the assessment was a mobile application for Android phones. The application (labelled as **IJOP**) is supposedly used in specific regions of China by Law Enforcement personnel to gather and manage data about specific groups of citizens and/or minorities.
 - In a nutshell, Cure53 got access to the APK of the app through HRW and was tasked with finding out what the app is capable of doing. Of particular importance was whether the activities and features of the app could be used in a way that violated human rights.
 - Cure53 received thorough briefing from HRW about the context of this assessment and communicated with the HRW team using a dedicated Slack channel.
 - Additional material that was collected by HRW prior to the assessment was also

shared with Cure53. It showed that HRW also disassembled the apps and started initial research into the identified features and activities.

Classification of Findings

During this assessment, Cure53 team has been using the following classification to specify the level of certainty regarding the documented findings. Given that this research had to happen on the basis of reverse-engineering and needed to be executed in a stealthy manner, it is necessary to classify the findings to make sure which level of reliability can be assumed.

- **Proven** - Source code and the analyzed activity clearly indicate a HR violation.
- **Evident** - Source code strongly suggests a HR violation.
- **Assumed** - Indications of a HR violation were found but a broader context remains unknown.
- **Unclear** - Initial suspicion was not confirmed. No HR Violation can be assumed.

Questions & Answers

HRW and Cure53 participated in frequent discussions throughout the project. They also shared and commented on the research results, largely doing so in real-time. In addition to the reverse-engineering of the app's features as means to finding HR Violation, Cure53 was also tasked with responding to specific questions sent in by the HRW team. To fully cover the scope of the assessment and all related exchanges, the questions asked by HRW are incorporated to this report. Cure53's research-based responses can be found in the ensuing sections.

XJ1-01-001 WifiDetector class implementing *War Driving* database (*Assumed*)

A question from the HRW team was as follows:

"Wifi Detectors - please look into what this is?"

The *Integration Joint Operations Platform* (IJOP) application employs a *WifiDetector* class which appears to collect data about wireless networks in range of the device. These are seemingly placed in a database. The collected data includes SSID, encryption method and GPS locations.

The Cure53 team assumes that this serves the purpose of creating a map of the existing wireless networks in the region, also known as *War Driving*¹. This could be potentially for

¹ <https://en.wikipedia.org/wiki/Wardriving>

targeted locating of weakly-secured wireless networks and joining them for the purpose of surveillance and infiltration. Furthermore, the evaluation of the metadata pertinent to the wireless networks in a certain area may signify information on the population density, connectivity and the produced data volume.

Affected File:

Collected Material/readable/code/com/fec/xjoneproject/xmpp/StatusManager.java

Affected Code:

```
private void sendLocationAndWifi(BDLocation paramBDLocation)
{
    Object localObject1 = WifiHelper.getScanResult();
    ScanResult localScanResult = (ScanResult)((Iterator)localObject2).next();
    localObject4 = new com/fec/report/dao/WifiDetector;
    ((WifiDetector)localObject4).<init>();
    localObject3 = localScanResult;
    localObject3 = localScanResult.SSID;
    localObject5 = localObject3;
    ((WifiDetector)localObject4).setName((String)localObject3);
    [...]
}
```

Affected File:

Collected Material/readable/simple_classes/com/fec/xjoneproject/util/WifiHelper.java

Affected Code:

```
private static WifiManager mWifiManager =
(WifiManager)IMSDroid.getInstance().getApplicationContext().getSystemService("wi
fi");
public static WifiInfo getConnectionInfo()
{
    return mWifiManager.getConnectionInfo();
}
public static List<ScanResult> getScanResult()
{
    return mWifiManager.getScanResults();
}
}
```

Affected Files:

Collected Material/readable/code/com/fec/report/dao/WifiDetector.java

Collected Material/readable/code/com/fec/report/dao/WifiDetectorDao.java

Affected Code:

```
localObject = "CREATE TABLE " + str + "\"WIFI_DETECTOR\" (\\"MAC_ADDRESS\" TEXT
PRIMARY KEY NOT NULL ,\"NAME\" TEXT,\"ENCRYPTION_METHOD\" TEXT,\"SIGNAL_LEVEL\"
```

```
INTEGER,\"CHANNEL_WIDTH\" INTEGER,\"CENTER_FREQUENCY\"  
INTEGER,\"CHANNEL_FREQUENCY\" INTEGER,\"NOTE\" TEXT,\"LATITUDE\"  
REAL,\"LONGITUDE\" REAL,\"LOCATION_DESCRIPTION\" TEXT);
```

XJ1-01-002 *NewCollection* activity tracks blood type and political views (*Proven*)

The HRW's question was as follows:

“Recording of Height and Blood Type: We want to know to what extent the authorities can justify this by saying this is all for counter-terrorism. So far, I see only a few mentions of terrorism”

Based on the observations made by the Cure53 team when browsing the activities, it can be assumed that the goal of this massive data collection is to have more reference data when it comes to mining and gathering data on individuals. With increasingly strong indicators, data that is not matching with information from the HQ might reveal more suspicious and problematic subsets of users/ actual people and groups.

Affected File:

Collected Material/readable/code/com/fec/report/dao/PersonInfoDao.java

Affected Code:

```
localObject = "CREATE TABLE " + str + "\"PERSON_INFO\" (\\"ID\" INTEGER PRIMARY  
KEY AUTOINCREMENT ,\\"SERVICE_ID\" TEXT,\\"BUILDING_ID\" INTEGER,\\"HOUSE_ID\"  
INTEGER,\\"NAME\" TEXT,\\"CARD\" TEXT,\\"ADDRESS\" TEXT,\\"PHOTO\"  
TEXT,\\"MODIFY_TYPE\" INTEGER,\\"PHONE\" TEXT,\\"CAR\" TEXT,\\"WORK\"  
TEXT,\\"EDUCATIONAL\" INTEGER,\\"RELIGIOUS_ATMOSPHERE\" INTEGER,\\"RELIGIOUS_NAME\"  
INTEGER,\\"RELIGIOUS_NAME_OTHER\" TEXT,\\"POLITICAL_STATUS\"  
INTEGER,\\"POLITICAL_STATUS_OTHER\" TEXT,\\"BIRTHDAY\" TEXT,\\"HEIGHT\"  
TEXT,\\"BLOOD\" INTEGER,\\"NATION\" TEXT,\\"RELATIONSHIP\"  
INTEGER,\\"RELATIONSHIPOTHER\" TEXT,\\"ADD_USER\" TEXT,\\"PERSON_TYPE\"  
TEXT,\\"PERSON_TYPE_OTHER\" TEXT,\\"CARD_TYPE\" INTEGER,\\"CARD_NUMBER\"  
TEXT,\\"DESTINATION_COUNTRY\" TEXT,\\"EXIT_TIME\" TEXT,\\"EXIT_REASON\"  
INTEGER,\\"EXIT_OTHER_REASON\" TEXT,\\"COLLECTION_THEME\" INTEGER,\\"CERTI_AGREE\"  
INTEGER,\\"TO_CENSUS\" TEXT,\\"IS_CHANGE_IDIN\" INTEGER,\\"NEW_NAME\"  
TEXT,\\"NEW_CENSUS\" TEXT,\\"NEW_ID_CARD\" TEXT,\\"NEW_NATION\" TEXT,\\"PASSPORT\"  
TEXT,\\"ASYLUM_EDUCATE_REASON\" TEXT,\\"ACTION\" TEXT,\\"DESCRIPTION\"  
TEXT,\\"CURRENT_ADDRESS\" TEXT,\\"SEND_PHOTO\" TEXT,\\"COUNT\" INTEGER);"
```

The following screenshot displays the *NewCollection* activity with the items of concern.

The screenshot shows an Android application window titled "Personnel Information". The interface is in Chinese and contains several input fields for personal information. The fields are: "原户籍地址" (Original household registration address), "迁入户籍地址" (Migrated household registration address), "height" (with a dropdown menu set to "未选择"), "blood type" (with a dropdown menu set to "未选择"), "degree" (with a dropdown menu set to "未选择"), "professional" (with the value "professional" entered), and "political look" (with a dropdown menu). Below these fields is a section titled "车辆信息采集" (Vehicle information collection) with a "车牌号" (License plate number) field.

Fig.: NewCollection activity renders input fields for blood type and political views.

XJ1-01-003 App feature of scanning for book titles (*Unclear*)

HRW asked Cure53 team about the following:

“Does the app has the functionality of scanning for book titles on Google? Is this code referring to a central database of banned titles?”

The HRW team discovered the file specified next during a review of the collected *IJOP* material. It was suspected that police officers might use the app to scan books and look up their ISBN via Google or possibly compare it with a database of banned titles.

Affected File:

Collected Material/readable/code/com/turui/bank/ocr/Intents\$SearchBookContents.java

Affected Code:

```
public static final String ACTION =  
"com.google.zxing.client.android.SEARCH_BOOK_CONTENTS";  
public static final String ISBN = "ISBN";
```

```
public static final String QUERY = "QUERY";
```

However, the Cure53 team did not discover this class being used anywhere in the reversed codebase.

XJ1-01-004 Local database of *religious atmosphere & re-education* (*Assumed*)

HRW raised the following questions to the testing team:

"With regard to RELIGIOUS_ATMOSPHERE and ASYLUM_EDUCATE_REASON.

It's unclear how this bit of code is used."

The Cure53 team reviewed the seemingly concerning code but found no traces of these terms being used in the activities. It is very likely that this information is a data object pushed by HQ and revealing how the collected data is grouped and categorized.

Affected File:

Collected Material/readable/code/com/fec/report/dao/PersonInfoDao.java

Affected Code:

```
localObject = "CREATE TABLE " + str + "\"PERSON_INFO\" (\\"ID\" INTEGER PRIMARY  
KEY AUTOINCREMENT ,\\"SERVICE_ID\" TEXT,\\"BUILDING_ID\" INTEGER,\\"HOUSE_ID\"  
INTEGER,\\"NAME\" TEXT,\\"CARD\" TEXT,\\"ADDRESS\" TEXT,\\"PHOTO\"  
TEXT,\\"MODIFY_TYPE\" INTEGER,\\"PHONE\" TEXT,\\"CAR\" TEXT,\\"WORK\"  
TEXT,\\"EDUCATIONAL\" INTEGER,\\"RELIGIOUS_ATMOSPHERE\" INTEGER,\\"RELIGIOUS_NAME\"  
INTEGER,\\"RELIGIOUS_NAME_OTHER\" TEXT,\\"POLITICAL_STATUS\"  
INTEGER,\\"POLITICAL_STATUS_OTHER\" TEXT,\\"BIRTHDAY\" TEXT,\\"HEIGHT\"  
TEXT,\\"BLOOD\" INTEGER,\\"NATION\" TEXT,\\"RELATIONSHIP\"  
INTEGER,\\"RELATIONSHIPOTHER\" TEXT,\\"ADD_USER\" TEXT,\\"PERSON_TYPE\"  
TEXT,\\"PERSON_TYPE_OTHER\" TEXT,\\"CARD_TYPE\" INTEGER,\\"CARD_NUMBER\"  
TEXT,\\"DESTINATION_COUNTRY\" TEXT,\\"EXIT_TIME\" TEXT,\\"EXIT_REASON\"  
INTEGER,\\"EXIT_OTHER_REASON\" TEXT,\\"COLLECTION_THEME\" INTEGER,\\"CERTI_AGREE\"  
INTEGER,\\"TO_CENSUS\" TEXT,\\"IS_CHANGE_IDIN\" INTEGER,\\"NEW_NAME\"  
TEXT,\\"NEW_CENSUS\" TEXT,\\"NEW_ID_CARD\" TEXT,\\"NEW_NATION\" TEXT,\\"PASSPORT\"  
TEXT,\\"ASYLUM_EDUCATE_REASON\" TEXT,\\"ACTION\" TEXT,\\"DESCRIPTION\"  
TEXT,\\"CURRENT_ADDRESS\" TEXT,\\"SEND_PHOTO\" TEXT,\\"COUNT\" INTEGER);"; 35  
paramDatabase.execSQL((String)localObject);
```


XJ1-01-005 App receives electricity consumption data from HQ (*Proven*)

HRW raised wanted to know about application's data on electricity consumption. Specifically, the questions were:

"Here's the authorities logging people's electricity use, how is it problematic? (it maybe problematic because the authorities are logging everyone's electricity use, and trying to see if there's a reason for "abnormal" level of electricity use, such as whether that person is a farmer, or has purchased new electric equipment. This is people's private matter, and illustrating how their privacy is being intruded."

The application employs a database which fetches various *utility* data about an individual. The Cure53 team assumes that the cause for filing such a report is a new task received from the HQ. In other words, a police officer can file a report to investigate the occurrence of unusual power consumption at a particular date and can mark reasons for it, preparing ground for further investigation by the public security agency. In case of a false positive, the officer can file in the actual electricity meter's value. A justification by law enforcement might be to monitor the use of electricity for cryptocurrency mining or growing cannabis indoors, as these types of activities lead to increased consumption.

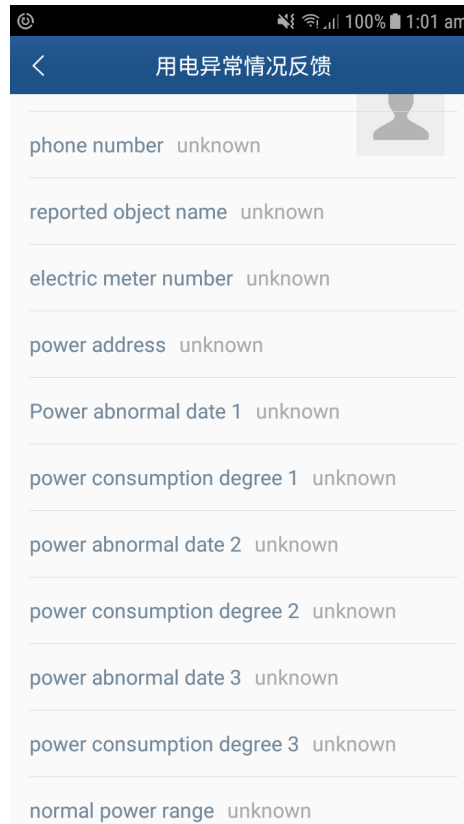


Fig.: Rendered Activity for CheckElcInfoActivity.

XJ1-01-006 Reporting feature for problematic tools (*Proven*)

HRW had doubts about the *tools*. Specifically,

“Tools they use and possess: An option under “electronic” appears to refer to problematic tools—such as tools that can be used to make explosives.”

The Cure53 team found that officers are capable of saving and submit a report to the HQ about explosive materials and tools. Furthermore, an officer can ask for an investigation of the matter to be carried on by the public security agency.

用电异常情况反馈

Click to select a region, county, township, village

detailed address

Is there a blasting tool or material * yes no

Exception reason *

Add new home appliances

renovation

farming

I found the use of non-home appliances such as cutting machines and electric welders unreasonable

Unable to explain, there is suspicious

other 其他异常原因

Do you need public security agencies to investigate * yes no

SUBMIT SAVE

Fig.: Rendered Activity for CheckElcInfoActivity.

XJ1-01-007 Problem Survey Feed tracks the use of crypto tools & VPNs (*Evident*)

HRW posed the following question to the Cure53 team:

“So there are a list of 9 “unlawful software” it seems that includes Whatsapp but here’s a much longer list of “network tools”—what’s the relationship between the list of 9 and this much longer list here?”

The IJOP app employs a *Problem Survey Feed*, which the team believes to act as a reporting tool for police officers who are interviewing subjects at road checkpoints. The *Feed* can be populated with various types of information about an individual, including their usage of crypto messenger and VPN applications.

It appears to be a general reporting tool that could be used in all kinds of situations that require filing a report about an individual like, for instance during an interrogation. The relationship between the list of nine items and the longer list in question appears to also include VPN services and various other crypto messengers. It could be assumed that the more extensive list includes less popular software. It may have the purpose of generally painting a picture of whether an individual knows and uses certain type of software. When this report is sent to the HQ, this might be the basis to raise a red flag on an individual's case. However, the code does not reveal what happens with the report next.

Affected File:

*Collected Material/readable/code/com/fec/xjoneproject/ui/task/problem_survey/
ProblemSurveyAddFeedFragment.java*

Affected Code:

```
localTextView = this.mNetworkToolCountTv;
```

Affected File:

Documentation/arrays file.docx

Affected Code:

```
<string-array name="network_tool">  
    <item>line</item>  
    <item>voxer</item>  
    <item>SKYPE</item>  
    <item>DiDi</item>  
    <item>whatsapp</item>  
    <item>ChatOn</item>  
    <item>OpenVPN</item>  
    <item>vpn dialogs</item>  
    <item>easyVPN</item>  
    <item>VPN Shield</item>  
    <item>GreenVPN</item>  
    <item>Astrill VPN</item>  
    <item>VPN for Phone</item>  
    <item>Global VPN</item>
```

[...]

XJ1-01-008 PII collection via *OverduePersonAddFeedActivity* (Assumed)

Another doubt shared by HRW was:

“Regarding OverduePersonAddFeedActivity, ‘Overdue person’ refers to people who have stayed abroad. Authorities are punishing people simply for having been abroad—we documented that, but it’d be good to flag that in this piece of research.”

Summary answer:

Reviewing the activity and data processing from the decompiled source code provides evidence of tracking people who have stayed abroad, along with geolocation, police officer’s ID and other metadata. No data was found on actual punishment as that is likely handled outside of the application. The fields used by the app for this purpose are:

actionSuspiciousInfo, activityDetail, addUser, censusRegisterDetail, enter, fkOp, idCard, isActionSuspicious, isLeaveCountry, isSuspicious, kinship, latitude, leaveFor, leaveReason, locationDescription, longitude, name, note, otherReason, otherRelation, phone, relation, reson, sex, suspiciousActivity, suspiciousInfo, type, userOrganizationId, userType

Detailed answer with technical details:

The targeted APK was modified so all activities could be invoked. Exploration of the relevant activity is presented next.

ADB Command:

```
adb shell am start -n  
"com.hbfec.xjoneproject/com.fec.xjoneproject.ui.task.overdue_person.OverduePersonAddFeedActivity"
```

This resulted in the PII collection form being rendered and this can be observed next.

Fig.: PII collection form.

This activity, which includes the police officer's ID as a *key_warn_id* intent extra, can be consulted next.

File:

com/fec/xjoneproject/ui/task/CheckDetailFragment.java

Code:

```
Intent localIntent22 = new Intent(getActivity(),
OverduePersonAddFeedActivity.class);
    localIntent22.putExtra("key_warn_id",
this.mMissionCommand.getCheckedPersonnelId());
    startActivity(localIntent22);
```

By observing the activity itself, it can be seen that the actual data is processed via the *OverduePersonAddFeedFragment*.

File:

com/fec/xjoneproject/ui/task/overdue_person/OverduePersonAddFeedActivity.java

Code:

```
return
OverduePersonAddFeedFragment.newInstance(getIntent().getStringExtra("key_id"));
```

The processed data fields include more options than the items displayed in the UI of the app and were gathered with a specific command supplied below. This provides evidence of information being captured by the app beyond the scope of what is displayed on the

screen. Some examples pertain to geolocation or the reporting police officer's identification.

Command:

```
cat $(find . -name OverduePersonAddFeedFragment.java) | grep localObject |  
grep put|sed 's|^ *||'|sort -u
```

Output:

```
localObject.put("actionSuspiciousInfo",  
this.mSuspiciousInfoMaterialEditText.getText().toString());  
localObject.put("activityDetail",  
this.mRecentEventsInfoMaterialEditText.getText().toString());  
localObject.put("addUser", str1);  
localObject.put("censusRegisterDetail",  
this.mAddHouseMaterialEditText.getText().toString());  
localObject.put("enter",  
this.mCountryEntryMaterialEditText.getText().toString());  
localObject.put("fkOp", this.mWarnId);  
localObject.put("idCard",  
this.mAddIdentityMaterialEditText.getText().toString());  
localObject.put("isActionSuspicious", 0);  
localObject.put("isLeaveCountry", 0);  
localObject.put("isSuspicious", 0);  
localObject.put("kinship", this.mFamilySpinner.getSelectedItemPosition() +  
"" );  
localObject.put("latitude", getLatitude());  
localObject.put("leaveFor",  
this.mCountryDestinationMaterialEditText.getText().toString());  
localObject.put("leaveReason",  
this.mExitReasonSpinner.getSelectedItemPosition());  
localObject.put("locationDescription", getLocationDescription());  
localObject.put("longitude", getLongitude());  
localObject.put("name", localPeerItem.getName());  
localObject.put("note", localPeerItem.getNote());  
localObject.put("otherReason",  
this.mExitReasonMaterialEditText.getText().toString());  
localObject.put("otherRelation",  
this.mOtherFamilyMaterialEditText.getText().toString());  
localObject.put("phone", localPeerItem.getPhone());  
localObject.put("relation", localPeerItem.getRelation() + "");  
localObject.put("reason", this.mReasonMaterialEditText.getText().toString());  
localObject.put("sex", sexInfo());  
localObject.put("suspiciousActivity",  
this.mSuspiciousEventsMaterialEditText.getText().toString());  
localObject.put("suspiciousInfo", 0);  
localObject.put("type", "2");  
localObject.put("userOrganizationId", str2);  
localObject.put("userType", str3);
```

XJ1-01-009 PII collection via *SuspiciousPhone[List]Activity* (Assumed)

HRW shared the following idea with Cure53:

“Phone activity would be of interest too. My guess is it has to do with people suddenly stopping their phone use rather than, bone fide suspicious phone calls to ISIS, for example...”

Summary answer:

Reviewing the activity and data processing from the related decompiled source code provides evidence of the phone activity being tracked. However, the purpose of this seems unclear. Perhaps more interestingly, the tracking also extends to cars' radios. This is performed on the same file and compounded together with the tracking of the phone activity. Again, this includes information such as geolocation, police officer's ID, driver and vehicle details, mentions of contraband and other metadata.

The following list summarizes the fields used by the app for phone-tracking purposes:

idCard, phone, susDescription, susOther, addUser, userOrganizationId

The following list includes fields related to car radio tracking:

addUser, dfyy, driverConsDetail, driverConsistency, driverId, driverPhone, fkMv, idCard, isConsistency, isDriverConsis, isForbiddenObj, isNeedSecondcheck, isPhoneSus, latitude, locationDescription, longitude, monitPeopleRelation, noNeedCheckedReason, phone, problemCarType, susDescription, susOther, userIdCard, userOrgName, userOrganizationId, userType, username, vehicleOrgName, vehicleProperty

Source code related to car-radios can be seen below with mentions of contraband highlighted.

```
setData(this.mEntity, this.mLocalRespondent.getDriver_id_number(),
this.mLocalRespondent.getDriver_phone(),
this.mLocalRespondent.getPerson_to_card(),
this.mLocalRespondent.getWhether_contraband(),
this.mLocalRespondent.getContraband_photo(),
this.mLocalRespondent.getReason_for_differ(),
this.mLocalRespondent.getOther_reason(),
this.mLocalRespondent.getCheck_relation_to_driver(),
this.mLocalRespondent.getVehicleProperty(),
this.mLocalRespondent.getVehicleOrgName(),
this.mLocalRespondent.getIsNeedSecondcheck(),
this.mLocalRespondent.getCheckedReason(),
this.mLocalRespondent.getNoNeedCheckedReason());
```


Detailed answer with technical details:

The targeted APK was modified so that all activities could be invoked. In terms of exploration, the basic information gathered by the application was first trivially inferred by invoking the pertinent [...]Add[...] activity, intended for data collection by the police officer.

ADB Command:

```
adb shell am start -n  
"com.hbfec.xjoneproject/com.fec.xjoneproject.ui.task.radio_car.SuspiciousPhoneActivity"
```

This resulted in the following PII collection form being rendered.



Fig.: SuspiciousPhoneActivity data collection form.

Secondly, the *SuspiciousPhoneListActivity* was invoked, yet this resulted in a blank screen as the user is not logged into the application and the activity is not meant to be invoked in this fashion (i.e. in a normal, logged-in app flow).

ADB Command:

```
adb shell am start -n  
"com.hbfec.xjoneproject/com.fec.xjoneproject.ui.task.radio_car.SuspiciousPhoneListActivity"
```

However, upon closer inspection of the decompiled source code, it was determined that the *List* activity is called from the Java file furnished next.

File:

com/fec/xjoneproject/ui/task/radio_car/CheckRadioCarInfoFragment.java

Code:

```
Intent localIntent1 = new Intent(getActivity(),
SuspiciousPhoneListActivity.class)
```

The information sent from this Java class to the *List* activity was gathered with the following command. This reveals substantially more information being collected, including the user-information of the police officer gathering the data, as well as geolocation.

Command:

```
cat $(find . -name CheckRadioCarInfoFragment.java) | grep localJSONObject.put |
sed 's|^ *||'|sort -u
```

Output:

```
localJSONObject.put("addUser", ConnectionUtils.getLoginName());
localJSONObject.put("dfyy",
this.uncheckReasonOneProjectSimpleSpinnerFieldView.getSelection());
localJSONObject.put("driverConsDetail",
this.differReasonOneProjectSimpleSpinnerFieldView.getOtherEditText());
localJSONObject.put("driverConsistency", 1);
localJSONObject.put("driverId",
this.dividerCardOneProjectSimpleEditTextFieldView.getText());
localJSONObject.put("driverPhone",
this.dividerPhoneOneProjectSimpleEditTextFieldView.getText());
localJSONObject.put("fkMv", this.mId);
localJSONObject.put("idCard", localPeerItem.getIdCard());
localJSONObject.put("isConsistency", i);
localJSONObject.put("isDriverConsis",
this.resultOneProjectSimpleRadioGroupWithTwoButtonFieldView.getValue());
localJSONObject.put("isForbiddenObj",
this.contrabandOneProjectSimpleRadioGroupWithTwoButtonFieldView.getValue());
localJSONObject.put("isNeedSecondcheck",
this.checkAgainOneProjectSimpleRadioGroupWithTwoButtonFieldView.getValue());
localJSONObject.put("isPhoneSus", 2);
localJSONObject.put("latitude", getLatitude());
localJSONObject.put("locationDescription", getLocationDescription());
localJSONObject.put("longitude", getLongitude());
localJSONObject.put("monitPeopleRelation",
this.relationOneProjectSimpleSpinnerFieldView.getSelection());
localJSONObject.put("noNeedCheckedReason",
this.uncheckReasonOneProjectSimpleSpinnerFieldView.getOtherText());
localJSONObject.put("phone", localPeerItem.getPhone());
localJSONObject.put("problemCarType", this.mEntity.getProblemCarType());
localJSONObject.put("susDescription", localPeerItem.getReason());
localJSONObject.put("susOther", localPeerItem.getOtherReason());
localJSONObject.put("userIdCard", "");
```

```
localJSONObject.put("userOrgName", str3);
localJSONObject.put("userOrganizationId", str);
localJSONObject.put("userType", 3);
localJSONObject.put("username", str1);
localJSONObject.put("vehicleOrgName",
this.vehicleUnitOneProjectSimpleEditTextView.getText());
localJSONObject.put("vehicleProperty",
String.valueOf(this.vehiclePropertyOneProjectSimpleSpinnerFieldView.getSelection
()));
```

XJ1-01-010 PII collection via *PeerPersonnel Add/List* activities (*Assumed*)

HRW raised the following question with the Cure53 team:

“Regarding PeerPersonnelAddActivity and PeerPersonnelListActivity, I believe this refers to who a person is travelling with. It would be interesting. Not sure what “personnel” refers to—it could be referred”

Summary answer:

Reviewing the activity and data processing from the decompiled source code provides evidence of tracking people who are travelling, along with destination, travel time, geolocation, police officer’s ID and other metadata. The list below summarizes the fields used by the app for this purpose:

action, addUser, arrest, fkMp, idCard, ifHavePeer, info, isDubious, isMatch, isWarnObj, latitude, leave, leaveLocal, locationDescription, longitude, matchPerRes, name, phone, returnTime, toInlandDesc, toInlandRes, userIdCard, userOrgName, userOrganizationId, userType, username

Radio personnel appears to refer to police officers in charge of tracking accompanying persons. These are looked up by Mission ID in a database.

File:

com/fec/xjoneproject/ui/task/radio_personnel/CheckRadioPersonnelInfoFragment.java

Code:

```
private RadioPersonnelRespondent getLocalRespondentByCheckMissionID(String
paramString)
{
    return
(RadioPersonnelRespondent)OneProjectDao.getInstance().getDaoSession().getRadioPe
rsonnelRespondentDao().queryBuilder().where(RadioPersonnelRespondentDao.Properti
es.Task_id.like(paramString), new WhereCondition[0]).build().unique();
}
```

Detailed answer with technical details:

First, some of the information captured by the application was trivially confirmed by invoking the *PeerPersonnelAddActivity*, designed for data capture by police officers.

ADB Command:

```
adb shell am start -n
"com.hbfec.xjoneproject/com.fec.xjoneproject.ui.task.radio_personnel.PeerPersonnelAddActivity"
```

This action resulted in the following PII collection form being rendered.

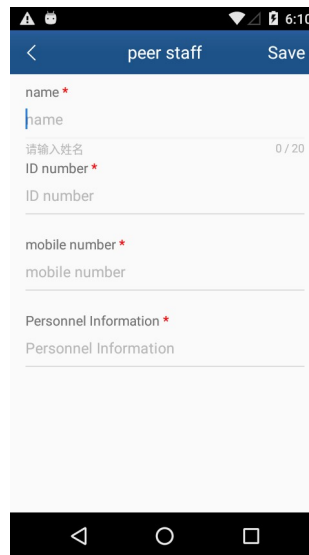
A screenshot of a mobile application interface showing a form titled "peer staff". The form has a blue header with a back arrow, the title "peer staff", and a "Save" button. The form contains several input fields: "name" (with a red asterisk), "ID number" (with a red asterisk), and "mobile number" (with a red asterisk). There is also a section titled "Personnel Information" with a red asterisk. The form is displayed on a white background with a black navigation bar at the bottom.

Fig.: PII collection form for *PeerPersonnelAddActivity*.

Secondly, the *PeerPersonnelListActivity* was invoked but this resulted in a blank screen when called directly.

ADB Command:

```
adb shell am start -n
"com.hbfec.xjoneproject/com.fec.xjoneproject.ui.task.radio_personnel.PeerPersonnelListActivity"
```

Upon closer inspection of the decompiled source code, it was found that the reason behind the screen being empty is because the *list* activity simply renders the data provided via intent extras.

File:

com/fec/xjoneproject/ui/task/radio_personnel/PeerPersonnelListActivity.java

Code:

```
public Fragment createFragment()
{
    return PeerPersonnelListFragment.newInstance((ArrayList) getIntent()
        .getSerializableExtra("item_list"));
}

public void finish()
{
    ArrayList localArrayList = (ArrayList)
((PeerPersonnelListFragment) getSupportFragmentManager().getFragments()
    .get(0)).getItemList();
    Intent localIntent = new Intent();
    localIntent.putExtra("item_list", localArrayList);
    setResult(-1, localIntent);
    super.finish();
}
```

The *item_list* was then found to be supplied to this activity from the Java file specified next.

File:

com/fec/xjoneproject/ui/task/radio_personnel/CheckRadioPersonnelInfoFragment.java

The actual fields passed in the list were gathered with the command presented next and this reveals the metadata gathered, in addition to geolocation information and other details. The mentioned data goes beyond what is displayed on the screen.

Command:

```
cat $(find . -name CheckRadioPersonnelInfoFragment.java) | grep
localJSONObject.put|sed 's|^ *||'|sort -u
```

Output:

```
localJSONObject.put("action", this.mEntity.getAction());
localJSONObject.put("addUser", ConnectionUtils.getLoginName());
localJSONObject.put("arrest", 1);
localJSONObject.put("fkMp", this.mId);
localJSONObject.put("idCard", localPeerItem.getCard());
localJSONObject.put("ifHavePeer", 1);
localJSONObject.put("info", localPeerItem.getInfo());
localJSONObject.put("isDubious",
this.dubiousOneProjectSimpleRadioGroupWithTwoButtonFieldView.getValue());
```

```
localJSONObject.put("isMatch",
this.sameOneProjectSimpleRadioGroupWithTwoButtonFieldView.getValue());
localJSONObject.put("isWarnObj",
this.warningOneProjectSimpleRadioGroupWithTwoButtonFieldView.getValue());
localJSONObject.put("leave",
this.leaveOneProjectSimpleRadioGroupWithTwoButtonFieldView.getValue());
localJSONObject.put("leaveLocal",
this.localOneProjectSimpleRadioGroupWithTwoButtonFieldView.getValue());
localJSONObject.put("locationDescription", getLocationDescription());
localJSONObject.put("latitude", getLatitude());
localJSONObject.put("longitude", getLongitude());
localJSONObject.put("name", localPeerItem.getName());
localJSONObject.put("phone", localPeerItem.getMobile());
localJSONObject.put("returnTime",
this.activityOneProjectSimpleDateSelectFieldView.getText());
localJSONObject.put("toInlandDesc",
this.reasonOneProjectSimpleSpinnerFieldView.getOtherText());
localJSONObject.put("toInlandRes",
this.reasonOneProjectSimpleSpinnerFieldView.getValue());
localJSONObject.put("userIdCard", "");
localJSONObject.put("userOrgName", str3);
localJSONObject.put("userOrganizationId", str2);
localJSONObject.put("userType", new
GetPrivilegeService(IMSdroid.getContext()).getString("privilege", ""));
localJSONObject.put("username", str1);
```

XJ1-01-011 PII collection via *MapRadioPersonnelDetailActivity* (Assumed)

HRW posed another question to the testing team:

"The map (or map radio) activity could be interesting—let's investigate. Again, if it's referring to police's location, not so; it'd be interesting if it's about the person being investigated."

Summary answer:

Reviewing the activity and data processing from the decompiled source code provides evidence of information collection such as home address, ID card, ID photo, IMEI, IMSI, geolocation, police officer's ID and other metadata. It seems to refer to the person being investigated. This can be inferred from the fields the application employs for this purpose:

action, addTime, addUser, dataId, deviceCode, disposition, endTime, gatherPhoto, gatherTime, homeAddress, id, idCard, idCardPhoto, imei, imsi, label, latitude, location, longitude, mac, matchType, name, note, peopleType, similarity, startTime, stationId, warnType

Detailed answer with technical details:

This activity was investigated by running the command detailed next.

ADB Command:

```
adb shell am start -n
"com.hbfec.xjoneproject/com.fec.xjoneproject.ui.task.radio_personnel.MapRadioPer
sonnelDetailActivity"
```

Running the command resulted in the following PII collection form.

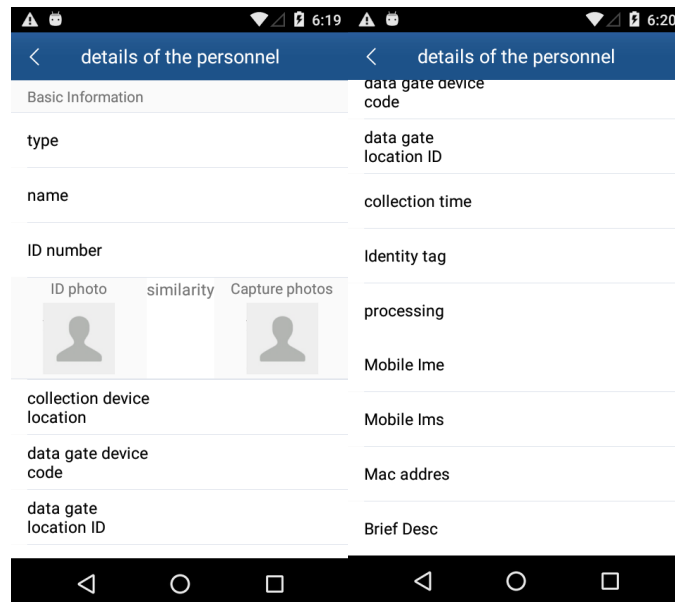


Fig.: PII collection form for MapRadioPersonnelDetailActivity.

Reviewing the decompiled application source code of *MapRadioPersonnelDetail-Activity.java* leads to *MapRadioPersonnelDetailFragment*, which loads the person's details over the API.

File:

```
com/fec/xjoneproject/ui/task/radio_personnel/MapRadioPersonnelDetailFragment.java
```

Code:

```
AttendanceService.getApi().getRadioPersonDetail(paramString).enqueue(new
OneProjectCallback(this)
{
    public void onResponse(MapRadioPersonnelResponse
paramAnonymousMapRadioPersonnelResponse)
[...]
```

MapRadioPersonnelResponse.java then retrieves the response from the API and saves it in a model called *MatchPerWarnEntity*.

File:

com/fec/xjoneproject/ui/map/MapRadioPersonnelResponse.java

Code:

```
public List<MatchPerWarnEntity> getRes()  
{  
    return this.res;  
}  
  
public void setRes(List<MatchPerWarnEntity> paramList)  
{  
    this.res = paramList;  
}
```

The *MatchPerWarnEntity* data model defines the fields listed next. These are the ones tracked by this activity and the relevant underlying logic. One can consult how they are defined next.

File:

com/fec/xjoneproject/ui/task/bean/MatchPerWarnEntity.java

Field data was extracted from the above file and this is shown next.

Command:

```
cat $(find . | grep -i MatchPerWarnEntity | grep -i java ) | grep -i private|cut  
-f5 -d" "|cut -f1 -d';' | sort -u | tr "\n" ", "|sed 's|,|, |g'
```

Output:

action, addTime, addUser, dataId, deviceCode, disposition, endTime, gatherPhoto, gatherTime, homeAddress, id, idCard, idCardPhoto, imei, imsi, label, latitude, location, longitude, mac, matchType, name, note, peopleType, similarity, startTime, stationId, warnType

XJ1-01-012 PII data via *OutOfPersonDetailActivity* (Assumed)

Another item of interest for the HRW team was the following:

“OutOfPersonDetailActivity: This refers to people who have left Xinjiang—i.e. people of interests to police.”

Summary answer:

Reviewing the activity and data processing from the decompiled source code provides evidence of tracking people who have left Xinjiang. The following list summarizes the fields used by the app for this purpose:

addTime, addUser, addUserName, address, areaCode, description, feedbackTime, fkCm, id, idCard, isChecked, mobilePhone, name, note, policeCheck, pushTime, relationType, sendTime, sex, track, visitorAddress, visitorIdCard, visitorMobilePhone, visitorName, visitorPic

Detailed answer with technical details:

This activity was investigated running the ADB command provided below.

ADB Command:

```
adb shell am start -n  
"com.hbfec.xjoneproject/com.fec.xjoneproject.ui.task.personnel_outflow.activity.  
OutOfPersonDetailActivity"
```

This resulted in the following PII collection form being rendered:



Fig.: PII data via *OutOfPersonDetailActivity*.

Upon closer inspection of the decompiled source code, it appeared that data included next was being gathered by this activity.

File:

com/fec/xjoneproject/ui/task/personnel_outflow/activity/OutOfPersonDetailActivity.java

Code:

```
this.tvWarningDescription.setText(paramPlaceOfDischargeEntity.getDescription());
    this.mWarnNameText.setText(paramPlaceOfDischargeEntity.getName());
    this.mWarnAddressText.setText(paramPlaceOfDischargeEntity.getAddress());
    this.mWarnPhoneText.setText(paramPlaceOfDischargeEntity.getMobilePhone());
    this.mWartIdCard.setText(paramPlaceOfDischargeEntity.getIdCard());
    this.tvOutPersonTrack.setText(paramPlaceOfDischargeEntity.getTrack());
    localInteger = paramPlaceOfDischargeEntity.getSex();
    if (localInteger == null) {}
}
switch (localInteger.intValue())
{
default:
    return;
case 0:
    this.mWarnSexText.setText("女");
    return;
}
this.mWarnSexText.setText("男");
```

The list of fields was extracted from the above source code. A command used for this purpose is furnished next.

Command:

```
cat $(find . -name OutOfPersonDetailActivity.java) | grep -i 'setText' | cut -f2 -d'.' | sort -u | sed 's| *||g' | tr "\n" "," | sed 's|,|, |g'
```

Output:

```
mTitleText, mWarnAddressText, mWarnNameText, mWarnPhoneText, mWarnSexText,
mWartIdCard, tvOutPersonTrack, tvWarningDescription
```

This data is retrieved by the app from the API as follows:

File:

com/fec/xjoneproject/ui/task/personnel_outflow/activity/OutOfPersonDetailActivity.java

Code:

```
private void getDataFromNet(String paramString)
{
```

```
try
{
    AttendanceService.getApi().getOutOfPersonDetail(paramString).enqueue(new
Callback()
    {
        [...]
        public void onResponse(Call<OutOfPersonResponse>[...])
    }
}
```

The above points to *OutOfPersonResponse*, which is defined on the following file and expects a *PlaceOfDischargeEntity* data model.

File:

com/fec/xjoneproject/ui/task/personnel_outflow/bean/OutOfPersonResponse.java

Code:

```
public List<PlaceOfDischargeEntity> getRes()
{
    return this.res;
}

public void setRes(List<PlaceOfDischargeEntity> paramList)
{
    this.res = paramList;
}
```

The associated fields were then extracted from this Java file in a manner presented next.

Command:

```
cat $(find . -name PlaceOfDischargeEntity.java) | grep -i private|cut -f5 -d" "|
cut -f1 -d';' | sort -u | tr "\n" ", "|sed 's|,|, |g'
```

Output:

```
addTime, addUser, addUserName, address, areaCode, description, feedbackTime,
fkCm, id, idCard, isChecked, mobilePhone, name, note, policeCheck, pushTime,
relationType, sendTime, sex, track, visitorAddress, visitorIdCard,
visitorMobilePhone, visitorName, visitorPic
```

XJ1-01-013 PII data via *InfluxPersonDetailActivity* (Assumed)

HRW similarly questioned another activity, specifically:

“Same as above. Though “inflow” and “influx” are referring to different concepts both are about monitoring people for their movements.”

Summary answer:

Reviewing the activity and data processing from the decompiled source code provides evidence of monitoring people's movements. The following list summarizes the fields used by the app for this purpose:

address, description, feedbackTime, fkCm, id, idCard, isChecked, isLogicDelete, mobilePhone, name, note, policeCheck, pushTime, relationAddress, relationIdCard, relationName, relationPhone, relationSex, sendTime, sex, track, userOrgName, userOrganizationId

Detailed answer with technical details:

This activity was investigated by running the ADB command included next.

ADB Command:

```
adb shell am start -n  
"com.hbfec.xjoneproject/com.fec.xjoneproject.ui.task.personnel_inflow.activity.I  
nfluxPersonDetailActivity"
```

This resulted in the following PII collection form being rendered:



Fig.: PII data processed via InfluxPersonDetailActivity.

Upon closer inspection of the decompiled source code, it was found that the data is retrieved from the API and expects an *InfluxPersonReponse* object.

File:

com/fec/xjoneproject/ui/task/personnel_inflow/activity/InfluxPersonDetailActivity.java

Code:

```
private void getDataFromNet(String paramString)
{
    try
    {
        AttendanceService.getApi().getInfluxPersonDetail(paramString).enqueue(new
        Callback()
        {
            [...]
            public void onResponse(Call<InfluxPersonReponse>
```

In turn, investigating *InfluxPersonReponse.java* reveals usage of a *PlaceOfInfluxEntity* data model.

File:

com/fec/xjoneproject/ui/task/personnel_inflow/bean/InfluxPersonReponse.java

Code:

```
public List<PlaceOfInfluxEntity> getRes()
{
    return this.res;
}

public void setRes(List<PlaceOfInfluxEntity> paramList)
{
    this.res = paramList;
}
```

This leads to *PlaceOfInfluxEntity.java*, from where the tracking fields were extracted with the command shown next.

Command:

```
cat $(find . -name PlaceOfInfluxEntity.java) | grep -i private|cut -f5 -d" "|cut -f1 -d';' | sort -u | tr "\n" ", "|sed 's|,|, |g'
```

Output:

```
address, description, feedbackTime, fkCm, id, idCard, isChecked, isLogicDelete,
mobilePhone, name, note, policeCheck, pushTime, relationAddress, relationIdCard,
relationName, relationPhone, relationSex, sendTime, sex, track, userOrgName,
userOrganizationId
```

XJ1-01-014 PII collection via *MissPhoneTrailFeedbackActivity* (Assumed)

HRW was also curious about another activity, namely:

"MissPhoneTrailFeedbackActivity: This refers to when someone's gone off grid — suddenly stopped using their phone."

Summary answer:

Reviewing the activity and data processing from the decompiled source code provides evidence of tracking this information. The following list summarizes the fields used by the app for this purpose:

addUser, addUserName, expression, expressionDesc, fkMptv, id, latitude, locationDescription, longitude, missTrailReason, note, otherReason, police, policeCheck, policeReason, relationship, telNumber, userOrgName, userOrganizationId

Detailed answer with technical details:

This activity was investigated with the command included in the following.

ADB Command:

```
adb shell am start -n
"com.hbfec.xjoneproject/com.fec.xjoneproject.ui.task.miss_phone_trail.activity.MissPhoneTrailFeedbackActivity"
```

This resulted in the following PII collection form being rendered:

The screenshot shows a mobile application interface with three instances of a form titled "没有手机号码轨迹" (No mobile phone number trail). The form contains the following fields and options:

- 新增走访对象** (Add new visit object)
- * 号码开户人与走访对象关系:** 未选择 (Dropdown menu)
- * name:** 点击图标扫描身份证自动获取 (Text input)
- * 未产生轨迹原因:** 未选择 (Dropdown menu)
- 走访对象** (Visit object)
- * ID number:** 点击图标扫描身份 (Text input)
- 现用手机号:** 现用手机号 (Text input)
- 姓名:** 姓名 (Text input)
- 手机号:** 手机号 (Text input)
- * 现实表现:** 有异常 无异常 (Radio buttons)
- 身份证号:** 身份证号 (Text input)
- * 地址:** 点击图标扫描身份证自动获取 (Text input)
- 备注说明:** 备注说明 (Text input)
- 手机号:** 手机号 (Text input)
- * 号码开户人与走访对象关系:** 未选择 (Dropdown menu)
- * 是否需要公安机关深入调查:** 是 否 (Radio buttons)
- 地址:** 地址 (Text input)
- * 未产生轨迹原因:** 未选择 (Dropdown menu)
- 新增走访对象** (Button)
- SUBMIT** (Button)
- SAVE** (Button)

Fig.: Data collection form rendered by *MissPhoneTrailFeedbackActivity*.

Upon closer inspection of the decompiled source code of the application, it was found that *MissPhoneTrailFeedbackActivity* makes use of the *MissPhoneTrailFeedbackViewModel*.

File:

*com/fec/xjoneproject/ui/task/miss_phone_trail/activity/
MissPhoneTrailFeedbackActivity.java*

Code:

```
this.mBinding.setViewModel(MissPhoneTrailFeedbackViewModel)getViewModel();  
protected void onCreate(Bundle paramBundle)  
{  
    super.onCreate(paramBundle);  
    this.mBinding =  
    ((ActivityMissPhoneTrailFeedbackBinding)DataBindingUtil.setContentView(this,  
2131427398));  
    setViewModel(new MissPhoneTrailFeedbackViewModel(this,  
getIntent().getStringExtra("key_warn_id"), 28));
```

The sequence submits the information to the API in a fashion illustrated next and making use of a *MissPhoneTrailResEntity*.

File:

*com/fec/xjoneproject/ui/task/miss_phone_trail/viewModel/
MissPhoneTrailFeedbackViewModel.java*

Code:

```
public void submit()  
{  
    MissPhoneTrailFeedbackActivity localMissPhoneTrailFeedbackActivity =  
(MissPhoneTrailFeedbackActivity) getActivity();  
    String str = localMissPhoneTrailFeedbackActivity.check();  
    if (TextUtils.isEmpty(str))  
    {  
        getActivity().mWaitingDialog.show("正在上传.....");  
        Gson localGson = new Gson();  
        HashMap localHashMap = new HashMap();  
        ((MissPhoneTrailResEntity) this.resEntity.get()).setAddUser();  
  
        (MissPhoneTrailResEntity) this.resEntity.get().setLongitude(getActivity().getLo  
ngitude());  
        [...]
```

From the *MissPhoneTrailResEntity.java* file, a more readable list of fields was collected.

Command:

```
cat $(find . -name MissPhoneTrailResEntity.java) | grep -i private|cut -f5 -d"  
"|cut -f1 -d';' | sort -u | tr "\n" ", "|sed 's|,|, |g'
```

Output:

```
addUser, addUserName, expression, expressionDesc, fkMptv, id, latitude,  
locationDescription, longitude, missTrailReason, note, otherReason, police,  
policeCheck, policeReason, relationship, telNumber, userOrgName,  
userOrganizationId
```

XJ1-01-015 PII collection via *FourAssociationFeedbackActivity* (Assumed)

HRW also wanted to know whether PII could be collected with another activity. More specifically:

"FourAssociationFeedbackActivity: Four associations refer to people's relationships to four categories of people the authorities are concerned about—people who have been abroad, I believe."

Summary answer:

Reviewing the activity and data processing from the decompiled source code provides evidence of tracking the following information about people's relationships to the noted four categories of people:

address, connectCount, connectDuration, description, idCard, name, peerCount, peerFlightPlace, peerTime, personType, personTypeName, phone, phoneOrAccount, photo, pushTime, relevanceType

Detailed answer with technical details:

This activity was investigated by running the following ADB command.

ADB Command:

```
adb shell am start -n  
"com.hbfec.xjoneproject/com.fec.xjoneproject.ui.task.four_association.activity.FourAssociationFeedbackActivity"
```

This resulted in the following PII collection form being rendered:

Fig.: Data collection form rendered by *FourAssociationFeedbackActivity*.

At the source code level, *FourAssociationFeedbackActivity* makes use of the *FourAssociationFeedbackViewModel*.

File:

com/fec/xjoneproject/ui/task/four_association/activity/
FourAssociationFeedbackActivity.java

Code:

```
setViewModel(new FourAssociationFeedbackViewModel(this,
getIntent().getStringExtra("key_warn_id")));
```

FourAssociationFeedbackViewModel.java then leads to *FourRelevanceResponse*. In turn, this makes use of *FourRelevanceEntity*, from which data fields were extracted and can be found below.

Command:

```
cat $(find . -name FourRelevanceEntity.java) | grep -i private|cut -f5 -d" "|cut -f1 -d';' | sort -u | tr "\n" " "|sed 's|,|, |g'
```

Output:

```
address, connectCount, connectDuration, description, idCard, name, peerCount,
peerFlightPlace, peerTime, personType, personTypeName, phone, phoneOrAccount,
photo, pushTime, relevanceType
```

XJ1-01-016 PII collection via *VehicleAddActivity* (Assumed)

Another question from the HRW team concerned:

“VehicleListActivity: Vehicle—the intersection between people, things and vehicles is a main way for mass surveillance.”

Summary answer:

Reviewing the activity and data processing from the decompiled source code provides evidence of tracking the following data fields for this activity:

account, addUser, address, bank, dataType, fkMiw, fkSDic, idCard, latitude, locationDescription, longitude, name, note, passport, phone, plateNumber, policeCheck, policeReason, relation, response, userIdCard, userOrgName, userOrganizationId, userType, username, vehicleColor, vehicleType, work, workUnit

Detailed answer with technical details:

First, an initially high-level view of the data gathered was trivially obtained via *VehicleAddActivity*.

ADB Command:

```
adb shell am start -n  
"com.hbfec.xjoneproject/com.fec.xjoneproject.ui.task.manual_new_task.VehicleAddA  
ctivity"
```

This resulted in the following PII collection form being rendered:



Fig.: Data collection form rendered by *VehicleAddActivity*.

The relevant *List* activity was then investigated by running a specific command supplied below.

ADB Command:

```
adb shell am start -n  
"com.hbfec.xjoneproject/com.fec.xjoneproject.ui.task.manual_new_task.VehicleList  
Activity"
```

Unfortunately, this crashes the app as the user is not logged in. In that sense, the application is not meant to be invoked this way. However, upon closer inspection of the decompiled source code, it was found that this activity is invoked from the classes presented next.

File:

com/fec/xjoneproject/ui/task/manual_new_task/NewTaskAddFeedFragment.java

Code:

```
Intent localIntent = new Intent(NewTaskAddFeedFragment.this.getActivity(),  
VehicleListActivity.class);
```

File:

com/fec/xjoneproject/ui/task/problem_survey/ProblemSurveyAddFeedFragment.java

Code:

```
Intent localIntent = new Intent(ProblemSurveyAddFeedFragment.this.getActivity(),  
VehicleListActivity.class);
```

As information passed from these activities is largely the same, it was gathered together with the following command. The process reveals the information from the screen, as well as all relevant metadata gathered by the app in the background (i.e. police officer's details, geolocation, etc.).

Command:

```
cat $(find . -name NewTaskAddFeedFragment.java) $(find . -name  
ProblemSurveyAddFeedFragment.java) | grep localJSONObject.put|sed 's|^ *||'  
sort -u
```

Output:

```
localJSONObject.put("account", localPeerItem.getAccount());  
localJSONObject.put("addUser", ConnectionUtils.getLoginName());  
localJSONObject.put("address",  
this.mOwnerLivingAddressMaterialEditText.getText().toString());  
localJSONObject.put("bank", String.valueOf(localPeerItem.getBank()));  
localJSONObject.put("dataType", "1");  
localJSONObject.put("fkIi", this.mId);  
localJSONObject.put("fkMiW", this.mId);  
localJSONObject.put("fkSDic", String.valueOf(localPeerItem.getName()));  
localJSONObject.put("idCard", localPeerItem.getIdCard());  
localJSONObject.put("latitude",  
Double.valueOf(this.mLocalRespondent.getLatitude()));  
localJSONObject.put("latitude", getLatitude());  
localJSONObject.put("locationDescription", getLocationDescription());  
localJSONObject.put("longitude",  
Double.valueOf(this.mLocalRespondent.getLongitude()));  
localJSONObject.put("longitude", getLongitude());  
localJSONObject.put("name", localPeerItem.getName());  
localJSONObject.put("note", localPeerItem.getNote());  
localJSONObject.put("passport",  
this.mOwnerPassportMaterialEditText.getText().toString());  
localJSONObject.put("phone", localPeerItem.getPhone());  
localJSONObject.put("plateNumber", localPeerItem.getPlateNumber());  
localJSONObject.put("policeCheck", "0");
```

```
localJSONObject.put("policeReason",
this.mPoliceReasonMaterialEditText.getText().toString());
localJSONObject.put("relation", localPeerItem.getRelation() + "");
localJSONObject.put("response",
this.mProblemFeedbackMaterialEditText.getText().toString());
localJSONObject.put("userIdCard", "");
localJSONObject.put("userOrgName", str3);
localJSONObject.put("userOrganizationId", str);
localJSONObject.put("userType", str4);
localJSONObject.put("username", str1);
localJSONObject.put("vehicleColor", localPeerItem.getVehicleColor() + "");
localJSONObject.put("vehicleType", localPeerItem.getVehicleType() + "");
localJSONObject.put("work",
this.mOwnerProfessionMaterialEditText.getText().toString());
localJSONObject.put("workUnit",
this.mOwnerWorkUnitMaterialEditText.getText().toString());
```

A more readable list of fields was extracted and can be found below.

Command:

```
cat $(find . -name NewTaskAddFeedFragment.java) | grep localJSONObject | grep
put|sed 's|^ *|'|sort -u | cut -f2 -d'"' | sort -u | tr "\n" ", " | sed 's|,|, |
g'
```

Output:

account, addUser, address, bank, dataType, fkMiw, fkSDic, idCard, latitude, locationDescription, longitude, name, note, passport, phone, plateNumber, policeCheck, policeReason, relation, response, userIdCard, userOrgName, userOrganizationId, userType, username, vehicleColor, vehicleType, work, workUnit

XJ1-01-017 PII collection via *ToolsAddActivity* (Assumed)

HRW raised the following doubt with the Cure53 team:

"ToolsListActivity: Tools can be interesting—I know they look for people with certain tools—like to sharpen stuff."

Summary answer:

This activity simply seems to record information about the applications installed on the phone.

Detailed answer with technical details:

This activity was investigated by running the following ADB command.

ADB Command:

```
adb shell am start -n  
"com.hbfec.xjoneproject/com.fec.xjoneproject.ui.task.problem_survey.ToolsListAct  
ivity"
```

Although this crashes the app because the user is not logged in into the system, it is possible to determine the information gathered via the relevant *Add* activity.

ADB Command:

```
adb shell am start -n  
"com.hbfec.xjoneproject/com.fec.xjoneproject.ui.task.problem_survey.ToolsAddActi  
vity"
```

This resulted in the following PII collection form being rendered:

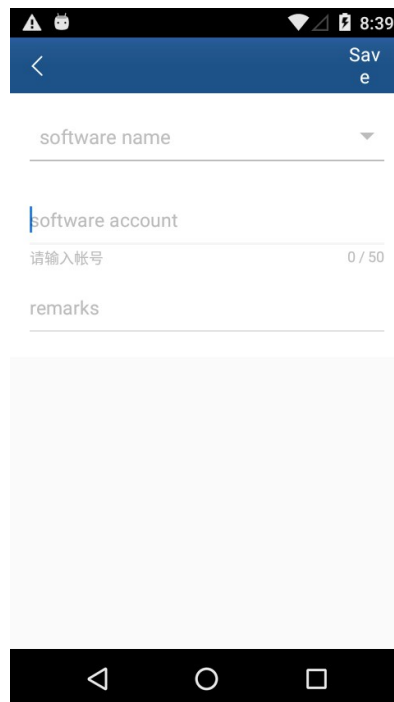


Fig.: Data collection form rendered by ToolsAddActivity.

It is shown that *ToolsAddActivity* makes use of *ToolsListFragment* and *ToolsAddFragment* in a manner included below.

File:

com/fec/xjoneproject/ui/task/manual_new_task/ToolsAddActivity.java

Code:

```
public Fragment createFragment()
{
    ToolsListFragment.PeerItem localPeerItem =
    (ToolsListFragment.PeerItem) getIntent().getSerializableExtra("peer_bean");
    Integer localInteger = Integer.valueOf(getIntent().getIntExtra("state", 9));
    int i = getIntent().getIntExtra("bean_position", -1);
    return ToolsAddFragment.newInstance(localPeerItem, localInteger.intValue(),
    i);
}
```

Looking at these fragments further reveals that this activity is simply tracking installed software.

File:

com/fec/xjoneproject/ui/task/manual_new_task/ToolsListFragment.java

Code:

```
ToolsListFragment.PeerItem localPeerItem =
    (ToolsListFragment.PeerItem)ToolsListFragment.this.mItemList.get(paramInt);
    localView.setTag(localPeerItem);

localTextView1.setText(ToolsListFragment.this.getSoftwareName(localPeerItem.getName()));
```

XJ1-01-018 PII collection via *BankAddActivity* (Assumed)

Another concern of raised by HRW was:

"BankAddActivity: We know the authorities monitor how much money Uyghurs withdraw or keep in banks"

Summary answer:

Reviewing the activity and data processing from the decompiled source code provides evidence that the bank's name is selected from a *dropdown*, the bank account is entered in a text field, and a police officer can enter free-form notes in a text-area field. An inspection of the source code confirmed this finding.

Detailed answer with technical details:

This activity was investigated by running the following ADB command.

ADB Command:

```
adb shell am start -n  
"com.hbfec.xjoneproject/com.fec.xjoneproject.ui.task.manual_new_task.BankAddActi  
vity"
```

This resulted in the following PII collection form being rendered:

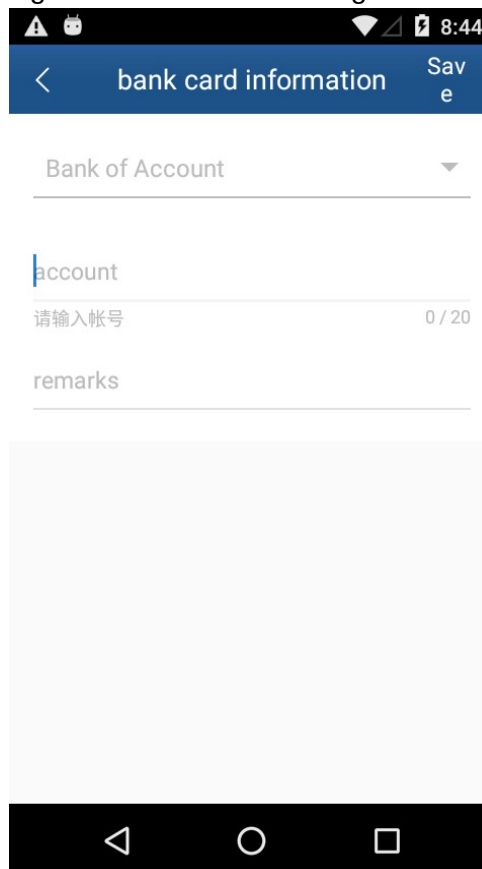


Fig.: Data collection form rendered by BankAddActivity.

At the source code level, this activity makes use of two fragments specified next.

File:

`com/fec/xjoneproject/ui/task/manual_new_task/BankAddActivity.java`

Code:

```
BankListFragment.PeerItem localPeerItem =  
(BankListFragment.PeerItem) getIntent().getSerializableExtra("peer_bean");  
int i = getIntent().getIntExtra("peer_state", 9);
```



```
return BankAddFragment.newInstance(localPeerItem,  
getIntent().getIntExtra("bean_position", -1), i);
```

From here, it can be confirmed that the account number and free-form notes are taken.

File:

com/fec/xjoneproject/ui/task/manual_new_task/BankAddFragment.java

Code:

```
this.mItem.setBank(Integer.valueOf(this.mBankSpinner.getSelectedItemPosition()))  
;  
this.mItem.setAccount(this.mAccountText.getText().toString());  
this.mItem.setNote(this.mNoteText.getText().toString());  
Intent localIntent = new Intent();  
localIntent.putExtra("peer_bean", this.mItem);  
localIntent.putExtra("bean_position", this.mPosition);
```

XJ1-01-019 Mention of the 26 sensitive countries for Uyghurs (Unclear)

HRW asked Cure53 the following question:

“The authorities consider any Uyghur’s links to 26 “sensitive countries” grounds for detention—these countries are Turkey, Indonesia, etc.—and yet I can’t find any of these country names in the app...or maybe I’m not looking in the right place?”

The Cure53 team found no arrays or strings that indicate a list of twenty-six sensitive countries. However, it might be that this information is nevertheless present in the form of codes or IDs that are used internally by the Ministry of Security.

XJ1-01-020 Usage of the terms “Picked” and “Radio” (Unclear)

HRW shared the following doubts:

“Then there’s 比中—it means “picked” (by the IJOP system, possibly)—there’s picked car, picked people. Can you see what this means? It’s a key concept but it seems to be referred to differently in the codes and the only consistency is the Chinese term. However, “radio” seems to refer to 比中 at times.”

The *CheckSearchActivity* has two mentions of 比中, however it remains unclear how this activity is used in a broader context. For further investigation, it is recommended to review all mentions of “picked” and see which activities employ the classes related to it. By reviewing these activities, certain patterns in usage might provide more context about the use.

XJ1-01-021 Usage of the terms “illegal”, “suspicious” and “problem” (Unclear)

Related to the above, HRW raised the following issue:

“Each category has “illegal” to it—illegalperson, illegalcar, illegalthing. And then there’s also the term “suspicious.” What’s the difference between “illegal” and “suspicious”? There’s also the term “problem.” What’s the difference between these terms?”

Based on the various activities, the Cure53 team browsed during this test, it is assumed that *illegal* refers to an object like a car or a tool that is not authorized to be driven or be in possession of a certain person. *Suspicious* appears to be related to events pushed by the HQ to be investigated by a police officer. For instance, the activity example provided next updates an officer on photos, date and location of a suspicious car that was recorded.

ADB Command:

```
adb shell am start -n
"com.hbfec.xjoneproject/com.fec.xjoneproject.ui.report.ui.activity.SusCarDetailA
ctivity"
```

The term *problem* appears to be used in the context of persons that have raised several red flags with the police, likely because of unauthorized possession of dangerous objects, government-opposing political views or being abroad for too long.

Affected File:

Collected

Material/readable/code/com/fec/xjoneproject/dao/OneProjectDao\$OpenHelper.java

Affected Code:

```
private void updateDatabaseTo40(Database paramDatabase)
{
    safeAlterDB(paramDatabase, "ALTER TABLE RADIO_CAR_RESPONDENT ADD
CHECKED_REASON INTEGER");
    dropTable(paramDatabase, "CHECK_MISSION");
    dropTable(paramDatabase, "ELECTRON_CHECK_FEEDBACK");
    dropTable(paramDatabase, "MISSING_PERSON_CONTACT");
    dropTable(paramDatabase, "MISSING_PERSON_INFO");
    dropTable(paramDatabase, "MISSING_PERSON_VISITOR");
    dropTable(paramDatabase, "MISS_RESPONDENT");
    dropTable(paramDatabase, "MISS_RESULT");
    dropTable(paramDatabase, "MISS_VISITOR");
    dropTable(paramDatabase, "PERSON_CAR_MATCH");
    dropTable(paramDatabase, "POPULATION_CHECK_INFO");
    dropTable(paramDatabase, "POPULATION_INFO");
    dropTable(paramDatabase, "POPULATION_SOCIAL");
```

```
dropTable(paramDatabase, "PROBLEM_CAR");
dropTable(paramDatabase, "PROBLEM_FAMILY");
dropTable(paramDatabase, "PROBLEM_HOUSE");
dropTable(paramDatabase, "PROBLEM_INFO");
dropTable(paramDatabase, "PROBLEM_SOCIAL");
}
```

XJ1-01-022 HQ communications for investigations and arrest (*Assumed*)

HRW was also wondering about the following:

“How does the officer communicate with the HQ. I see there are terms like “immediate arrest” or “detain for investigation”—are these decisions officers make or HQ makes.”

Most likely communication and some form of chat exchange takes place via the XMPP protocol. This can be assumed from the functions shown next.

Affected File:

com/fec/xjoneproject/ui/LoginFragment.java

Affected Code:

```
private void initAccount()
{
    String str1 = this.mXmppConfig.getString("uum_username", "");
    String str2 = this.mXmppConfig.getString("uum_password", "");
    if (this.loginFlag == 1)
    {
        ...
private void initXmppConfig()
{
    String str = this.mXmppConfig.getString("xmpp_host", "61.182.226.81");
    int i = this.mXmppConfig.getInt("xmpp_prot", 5222);
    ConnectionConfiguration localConnectionConfiguration = new
        ConnectionConfiguration(str, i, "");
    localConnectionConfiguration.setSecurityMode(
        ConnectionConfiguration.SecurityMode.enabled);
    localConnectionConfiguration.setSASLAuthenticationEnabled(true);
    localConnectionConfiguration.setReconnectionAllowed(true);
    localConnectionConfiguration.setSendPresence(true);
    ConnectionUtils.setHostPord(str, i);
    ConnectionUtils.setConnectionConfig(localConnectionConfiguration);
    Log.d("LoginFragment", "init XMPP host:" + str + " port:" + i);
}
```

While it is unclear how exactly these variables are used in the context of the app, there are indicators that information pushed by the HQ is setting the values for immediate

investigations and and immediate arrests. However, it seems police officers might be able to edit this information as well.

Affected File:

Collected Material/readable/code/com/fec/xjoneproject/ui/task/radio_personnel/CheckRadioPersonnelInfoFragment.java

Affected Code:

```
private String checkInput()
{
    String str = CheckUtils.check(this.mScrollView);
    Object localObject = this.mImmediateArrestLinearLayout;
    int i = ((LinearLayout)localObject).getVisibility();
    if (i == 0)
    {
        localObject = this.mArrestCheckBox;
        boolean bool = ((CheckBox)localObject).isChecked();
        if (!bool)
        {
            bool = TextUtils.isEmpty(str);
            if (bool) {
                str = "请选择反馈已抓捕";
            }
        }
    }
    return str;
}

private void getDataFromNet(String paramString)
{
    try
    {
        localObject = AttendanceService.getApi();
        Call localCall =
((AttendanceApi)localObject).getRadioPersonDetail(paramString);
        localObject = new
com/fec/xjoneproject/ui/task/radio_personnel/CheckRadioPersonnelInfoFragment$3;
        ((CheckRadioPersonnelInfoFragment.3)localObject).<init>(this, this);
        localCall.enqueue((Callback)localObject);
        return;
    }
    catch (RetrofitUrlNullException localRetrofitUrlNullException)
    {
        for (;;)
        {
            Object localObject = IMSDroid.getContext();
```

```
        FecUtil.showUrlIsNullToast((Context)localObject);  
    }  
}  
}
```

XJ1-01-023 Review of *Warn* activities (*Unclear*)

HRW asked the following question:

“What are “warn activities”? I presume it's when officers file a warning (a discrepancy, a lost of track) to the HQ as well as when HQ tells officers about such a warning?”

Judging from the variable names, Cure53 assumes that *Warn* activities are something pushed by the HQ to the officer. There are multiple mentions of setting a *“new mission”*, as well as an *“update a task”* and *“retrieving”* matters.

Affected File:

com/fec/xjoneproject/ui/task/face_warning/FaceWarningListActivity.java

Affected Code:

```
public void getDataFromNet(String paramString)  
{  
    Object localObject = this.mWaitingDialog;  
    String str = “正在获取……”;
```

Furthermore, most of the *warning* classes inherit their capabilities from the *SingleFragmentActivity* class, which seems to establish a connection with HQ. The inheritance relationship is implemented by writing, for instance, that *FaceWarningListActivity* extends *SingleFragmentActivity* in the source code.

Affected File:

com/fec/xjoneproject/ui/task/face_warning/FaceWarningListActivity.java

Affected Code:

```
public class FaceWarningListActivity  
extends SingleFragmentActivity  
  
    Object localObject = ConnectionUtils.isConnect();  
    boolean bool1 = ((Boolean)localObject).booleanValue();  
    if (bool1)  
    {  
        localObject = ConnectionUtils.getConnection();  
        bool1 = ((XMPPConnection)localObject).isAuthenticated();  
        if (bool1) {}
```

```
}
else
{
    localObject = LogUtil.getLogger(getClass());
    String str = "XMPP 未连接, 发送重连广播";
    ((Logger)localObject).error(str);
    Intent localIntent = new android/content/Intent();
    localIntent.<init>("network_activated");
    localObject = ConnectionUtils.isConnect();
    bool1 = ((Boolean)localObject).booleanValue();
    if (bool1)
    {
        localObject = ConnectionUtils.getConnection();
        bool1 = ((XMPPConnection)localObject).isAuthenticated();
        if (bool1) {}
    }
    else
    {
        localObject = "connect_xmpp";
        boolean bool2 = true;
        localIntent.putExtra((String)localObject, bool2);
    }
}
```

This is used by developers to avoid double work for what is basically the same functionality. As a general rule of thumb, from the experience of reviewing most activities of *IJOP*, activated input fields, as well as *save* and *submit* buttons are strong indicators of an activity that is initiated by a user. If all the fields are static and not editable, this means an item likely being sent from the HQ. Furthermore, the *FaceWarningActivity* contains an ID check which a police officer can confirm, thus showing that *Warning* activities are intended to send some information back.



Fig.: Confirming an ID Check in FaceWarning Activity.

XJ1-01-024 Review of *AntiRefluxActivity* (Assumed)

HRW also asked for attention to be given to:

“AntiRefluxActivity: This, again, refers to movement of people—from abroad—definitely interesting.”

Summary answer:

An inspection of *AntiRefluxActivity* led to *AntiRefluxFragment*, which revealed that this functionality requires “*Passport Manager*” privileges from the police officer. While the screen is empty when one is not logged in, inspecting the source code suggests that it is

used for XMPP communications with the server, whereby notification updates are shown to the user.

Detailed answer with technical details:

This activity was investigated by running the following ADB command.

ADB Command:

```
adb shell am start -n  
"com.hbfec.xjoneproject/com.fec.xjoneproject.ui.task.anti_reflux.AntiRefluxActiv  
ity"
```

This resulted in the following PII collection form being rendered:



Fig.: The screen is empty due to not being logged into the database.

At the source code level, it can be seen that *AntiRefluxFragment* is referenced.

File:

com/fec/xjoneproject/ui/task/anti_reflux/AntiRefluxActivity.java

Code:

```
AntiRefluxFragment localAntiRefluxFragment = new AntiRefluxFragment();
```



```
getSupportFragmentManager().beginTransaction().add(2131296837,  
localAntiRefluxFragment).commit();
```

Further inspection of *AntiRefluxFragment.java* reveals that this requires “*Passport Manager*” privileges and updates notification information via XMPP communications.

File:

com/fec/xjoneproject/ui/task/anti_reflux/AntiRefluxFragment.java

Code:

```
public View onCreateView(LayoutInflater paramLayoutInflater, ViewGroup  
paramViewGroup, Bundle paramBundle)  
{  
    View localView = paramLayoutInflater.inflate(2131427645, paramViewGroup,  
false);  
    this.privilege = new  
GetPrivilegeService(IMSDDroid.getContext()).getString("privilege", null);  
    ArrayList localArrayList = new ArrayList();  
    if  
(PrivilegeUtil.isHavePrivilege(PrivilegeUtil.PRIVILEGE_RIGHT_PASSPORT_MANAGER))  
{  
        localArrayList.add(new MyTaskIconRecyclerViewAdapter.TaskIcon(2131231250,  
2131690341, 26));  
    }  
    [...]  
    public void update(Observable paramObservable, Object paramObject)  
    {  
        if (((paramObject instanceof XmppObservable.UpdateData)) &&  
((XmppObservable.UpdateData)paramObject).getType() ==  
XmppObservable.UpdateType.refreshTaskNumber) &&  
("SmartUc_notify".equals(((XmppObservable.UpdateData)paramObject).getBundle().ge  
tString("key_type", null))))  
        {  
            new  
XMPPConfiguration(IMSDDroid.getContext()).putBoolean(ConnectionUtils.getLoginName  
() + "notify_not_get_list", true, true);  
            this.mAdapter.notifyDataSetChanged();  
        }  
    }  
}
```

Conclusions

This technical analysis and review of the Integrated Joint Operations Platform (IJOP) mobile application has demonstrated that the concerns expressed by Human Rights Watch are valid. Funded by The Open Technology Fund and carried out by Cure53 in close collaboration with the Human Rights Watch team, this November 2018 project sheds light on twenty-four items from a perspective of potential violations of human rights.

In a nutshell, judging by the research outputs and results of an in-depth analysis, the Cure53 team finds it evident and undeniable that the application is capable of collecting and managing vast amounts of very specific data. It is certain that the gathered material can become a basis for further action concerning a specific group (or groups) of citizens. According to the European Convention on Human Rights, which stands among other examples of agendas and corresponds to related court rulings, the above practice can be considered a human rights violation.

The main aspects that should be highlighted among the many findings with differently-evaluated severities pertain to the possible *Wardriving* capability of the app (see [XJ1-01-001](#)). Beyond mentioning re-education reasons, the application has a tracking of power over energy consumption, the recording of political views and the religious atmosphere (see [XJ1-01-004](#)). The presence of those terms gives away information about the scope and type of data the Ministry of Security collects and shares with its police officers.

In a broader sense, the application's functionality makes an impression that leads Cure53 to believe that it can indeed violate human rights. Especially the items noted with a "Proven" status serve as solid evidence of this fact. At the same time, it should be noted that Cure53 operated as a purely technically-driven team and an unbiased investigating entity. Therefore, it is not a party in any way involved in making final judgements as to whether human right violations take place from legal, social or political standpoints. The Cure53 team works from a premise of technical evidence, which is based on reverse-engineering operations

The Appendix incorporated to this document is intended as means to enable HRW to investigate the app more on their own, which Cure53 believes to be of a critical importance given the above borders of expertise. Among other deliverables, this Appendix contains a guide to browsing through all activities that were previously hidden, as well as can help understand the relationship between them. Based on the technical evidence and material provided by Cure53, it should be possible for HRW to rely on the deliverables and build a solid case if the results indeed allow it.

Cure53 would like to thank Maya Wang, Greg Walton and Seamus Tuohy from the Human Rights Watch team for their excellent project coordination, support and assistance, both before and during this assignment.

Appendix

Appendix 1: Working version of app APK and instructions on how to use it

HRW asked the following question:

“One way to investigate the app would be for Cure53 to construct a working version of the app so I can explore the functions myself—let me know if it's possible?”

The Cure53 team reverse-engineered the provided *IJOP* app and repackaged it with all activities being exported. Furthermore, a step-by-step guide and commands to browse these activities was written and shared with the HRW team.

Appendix 2: Guide for checking relations between activities

HRW requested the following:

“Is it possible to sketch out the workflow of particularly important Activities though? I find it hard to understand without knowing how the screens relate to each other.”

The Cure53 team provided a guide to back-check the relations between activities, which can be further elaborated to construct a flow diagram detailing which activity is called by which other activity. This might help understanding the workflow of a police officer using this app better.

The following files were shared with the HRW team:

- *IJOP_App_exported_in_chinese.apk*
- *IJOP_App_exported_in_english.apk*
- *Activity_adb_commands.txt*
- *HowTo_invoke_Activities_and_fetch_Screenshots.txt*
- *HowTo_Flow_Diagram_Guide.txt*