Magic Motion Smart Adult Toy

Introduction

According to Wikipedia, “A sex toy is an object or device that is primarily used to facilitate human sexual pleasure, such as a dildo or vibrator. Many popular sex toys are designed to resemble human genitals, and may be vibrating or non-vibrating. The term sex toy can also include BDSM apparatus and sex furniture such as slings; however, it is not applied to items such as birth control, pornography, or condoms. Alternative expressions include adult toy and the dated euphemism marital aid, although "marital aid" has a broader sense and is applied to drugs and herbs marketed to supposedly enhance or prolong sex. Sex toys are most commonly sold at a sex shop, but they may also be sold in a pharmacy/chemist store, a pornographic DVD store, a head shop, or a department store. Today's sex toys are available in almost all countries for male and females.”

In this article, we are going to discuss the methods used by attackers as well as other security researchers to find security issues in MagicMotion adult toy(http://www.magicsmotion.com/p-flamingo.html). The chapter will give a quick introduction on the techniques used to exploit the toy. The vendor describes it as “Be controlled no matter where you are with our Magic Motion App.

Feel the adrenaline in your body thanks to our wearable vibrator Magic Flamingo. Magic Flamingo has been specially designed for couples who like to have sex in public and women who like to be teased when going out.

Now we released new partner Magic Elizebath Cap for Magic Flamingo, try it!.”

However, the same techniques can be used against other similar devices sold on Amazon by various other vendors. The chapter will talk about how the author mapped various components of the Adult Toy that would be susceptible to attacks and will describe some of the security vulnerabilities that were identified in this process.

We started our research by looking at various Smart fitness tracker manufacturers and the technologies used by these devices. MagicMotion Flamingo caught our attention due to its cheap price and functionality provided within it. Also, this device utilized Bluetooth Low Energy (BLE) which is pretty common amongst smart adult toys. When it comes to IoT devices how could we escape talking about IoT security without exploring a device that uses BLE.

Magic Motion Flamingo

Device Procurement

The first agenda on our list in this case was to procure the device itself. We found that the device was easily available on Amazon's website[1]. The next goal for us was to identify if the device firmware was available easily. However, after going through manufacturer website we realized that the device firmware was not easily downloadable from the website. Next, we decided to see how the firmware was upgraded by the manufacturer, it seems that mobile app VeryFit2 provided the capability of upgrading the device. We decided to decompile the application. For that we focused on using Android application version of the mobile application.

[1] https://www.amazon.com/Lintelek-Waterproof-Bluetooth-Pedometer-Smartphone/dp/B075PWY3TY

We were able to look at the source code and soon came across the call for firmware.json file in the source code in the file called UpdatePresenter.java

```
public UpdatePersenter()
{
}

private DeviceUpdateInfo getUpdateInfo()
{
    Object obj1;
    Object obj2;
    Object obj3;
    Object obj4;
    obj4 = HttpUtil.get("http://www.youduoyun.com/apps/firmwares/firmware.json", null);
    DebugLog.d("HttpUtil.PATH:http://www.youduoyun.com/apps/firmwares/firmware.json");
    obj1 = "";
    obj3 = null;
    obj2 = null;
    String s = URLDecoder.decode(((String) (obj4)), "UTF-8");
    obj1 = s;
_L2:
    Object obj = obj2;
    if(obj4 == null)
        break MISSING_BLOCK_LABEL_160;
    obj = obj3;
    int i;
```

Firmware Json file hardcoded

Finally, we downloaded the file and observed that the firmware files were available to be downloaded without any authentication as zip files. Here is the excerpt from firmware.json file.

```
{
    "firmwareInfo":
    [
        {
            "device_id": "0",
            "version": "0",
            "file": "http://www.youduoyun.com/apps/firmwares/aiju_fw0_4.zip",
            "info_ch": "1, 测试1;\n  2, 测试2;\n  3, 测试3.",
            "info_en": "1, test1;\n  2, test2;\n  3, test3."
        },
        {
            "device_id": "1",
            "version": "31",
            "file": "http://www.youduoyun.com/apps/firmwares/aiju_fw_ido100_id1_v31.zip",
            "info_ch": "1, 测试1;\n  2, 测试2;\n  3, 测试3.",
            "info_en": "1, test1;\n  2, test2;\n  3, test3."
        },
        {
            "device_id": "20",
            "version": "6",
            "file": "http://www.youduoyun.com/apps/firmwares/aiju_fw_id100_hr_v6.zip",
            "info_ch": "1, 测试1;\n  2, 测试2;\n  3, 测试3.",
            "info_en": "1, test1;\n  2, test2;\n  3, test3."
        },
        {
            "device_id": "21",
            "version": "1",
            "file": "http://www.youduoyun.com/apps/firmwares/aiju_fw_id100hr_id21_v35.zip",
            "info_ch": "1, 测试1;\n  2, 测试2;\n  3, 测试3.",
            "info_en": "1, test1;\n  2, test2;\n  3, test3."
        },
        {
            "device_id": "22",
            "version": "1",
            "file": "http://www.youduoyun.com/apps/firmwares/aiju_fw_id100hr_id22_v35.zip",
            "info_ch": "1, 测试1;\n  2, 测试2;\n  3, 测试3.",
            "info_en": "1, test1;\n  2, test2;\n  3, test3."
        },
```


Firmware.json file content

We were able to download the firmware file for our device which was called aiju_fw_id115hr_v4.zip. After downloading the zip file, we observed that it contained a hex and dat file. Hex file is called Intel Hex format and is a file format that conveys binary information in ASCII text form. Below is the image of that file.

```
:020000040001F9
:10800000C07A002045E501005FE501007DC0010068
:1080100000000000000000000000000000000060
:1080200000000000000000000000063E5010007
:10803000000000000000000065E5010067E50100A8
:1080400069E5010069E5010069E501003DF901000C
:108050005FF90100000000006109020037A701007C
:1080600069E5010069E5010069A2010069E5010017
:1080700069E5010069E5010069E5010069E50100C4
:1080800069E50100A9F3010069E5010069E5010066
:10809000BFF3010069E501003D06020069E501004A
:1080A00069E5010069E501000000000000000032
:1080B000000000000000000000000000000000C0
:1080C0000348854617F0F0FC00480047578501003B
:1080D000C07A0020401E00BF00BF00BF00BF00BF2D
:1080E00000BF00BF00BF00BF00BF00BF00BFF1D195
:1080F00070470000401E00BF00BF00BF00BF00BFB0
:1081000000BF00BF00BF00BF00BF00BF00BFF1D174
:10811000704700000056885F308884068FF2464B252
:10812000EFF30585002D01D1A646004725460646FA
:1081300021273FBAF0B400240025002600027F0B420
:10814000F92040B200470000401E00BF00BF00BF42
:1081500000BF00BF00BF00BF00BF00BF00BF00BF27
:1081600000BFF1D170470000401E00BF00BF00BF3C
:1081700000BF00BF00BF00BF00BF00BF00BF00BF07
```

Hex format firmware

The hex file is then converted to binary format by using Hex2Bin tool[2]. After converting it into binary format then we can apply the general aspect of reverse engineering to look at the assembly. However, in this case we will stay away from identifying security issues from assembly as it is much easier to exploit BLE devices directly using application security issues both in the device as well as in the application that support these devices. The next goal for us was to identify attack surfaces that would allow to identify security issues for this device. We mapped out the following components that an attacker would focus on

1. The device
2. BLE GATT services
3. Insecure mobile application
4. Insecure data storage

Lintelek Exploits

In the next section we are going to work towards various issues that were identified in the Lintelek fitness tracker.

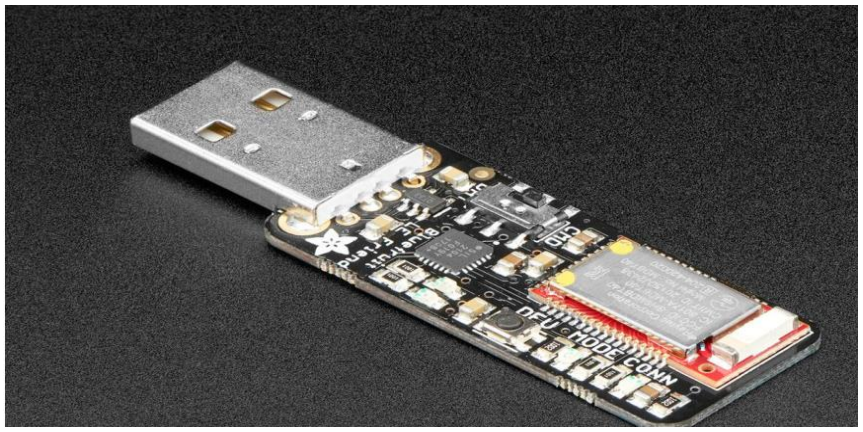
- ### Device

  As a part of security mapping earlier, we decided to focus on device and the GATT services exposed by the device.

[2] http://hex2bin.sourceforge.net/

No Encryption (CVE-2020-12730)

The very first thing that we wanted to find was whether there was any encryption between the mobile application and the device. There are different ways to sniff communication between a BLE client and server. We could go the way of Hardware peripherals like Ubertooth[3] or use cheaper alternative such as Adafruit BLE-Sniffer[4]. The author prefers the Adafruit BLE-Sniffer as it is cheap and plus It works with windows. The details of how to use the BLE-Sniffer using Windows are a provided by AdaFruit guys[5].



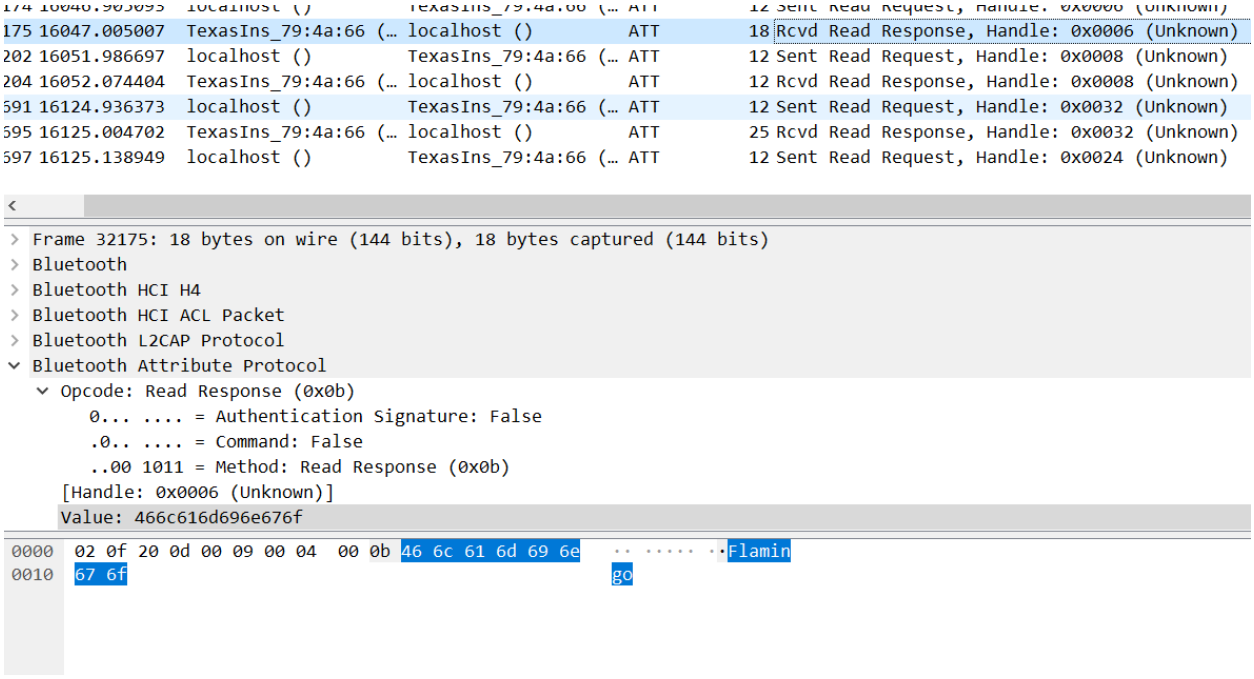
Ada Fruit BLE sniffer

However, there is even an easier option if we own an Android device. All we need to do is enable logging of HCI logs following the steps below and we can get the Bluetooth_hci.log file written to sdcard on the device.

1. On the Android device go to Settings.
2. Select Developer options.
3. Click to enable Bluetooth HCI snoop logging.
4. Return to the Settings screen and select Developer options.
5. In the Developer options screen select Enable Bluetooth HCI snoop log.
6. The log file is now enabled.

The file is located as /sdcard/btsnoop_hci.log on author's Android Nexus device and the location might vary as per the device. Once we get access to that log file, we can view the log files using Wireshark. Below is the Wireshark display of Lintelek's BLE communication. We can clearly see that the application does not use any sort of encryption on the communication which would allow any attacker to use external sniffers as mentioned above to sniff all the communication between the app and the device easily. Obviously an attacker would need to sniff the 2.4 Ghz band for that.

[3] https://github.com/greatscottgadgets/ubertooth
[4] https://www.adafruit.com/product/2269
[5] https://learn.adafruit.com/introducing-the-adafruit-bluefruit-le-sniffer/nordic-nrfsniffer



Wireshark for BLE

GATT Services

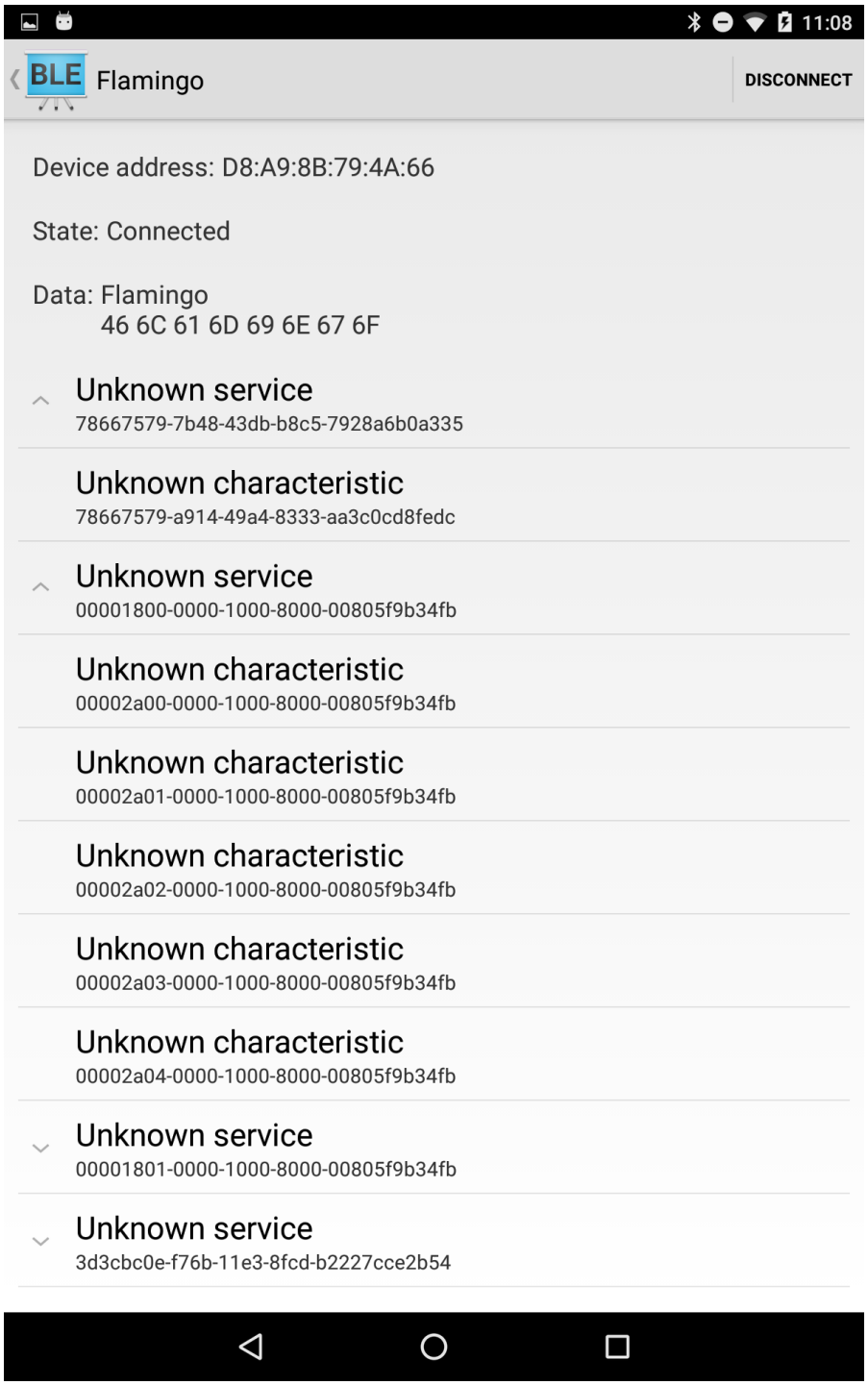
As a part of security mapping earlier, we decided to focus on device GATT services exposed by the device

No Auth/Authz reading from device descriptors (CVE-2020-12729)

Now that we know that there is no encryption on communication between application and the device, we wanted to see if the device's GATT services required any kind of authentication or authorization before accessing them. We can use Gattacker[6] tool. We would need to buy BLE USB dongles which are usually for 4 or 5$ max. The most popular ones are CSR 8510 USB dongles and available on Amazon[7]. The details of how to use the tool are given on the GitHub page as well. However, we can use even a better alternative if we have an Android device. We can use Google Android Bluetooth application for scanning devices and retrieving values from the characteristics as well as descriptors. Below is an image where we can see all the services and characteristics of the device. Also, we can see that characteristic that stores the name of the device.

[6] https://github.com/securing/gattacker

[7] https://www.amazon.com/Bluetooth-Dongle-Adapter-Raspberry-Windows/dp/B073H4GQ9Q



BLE app on device

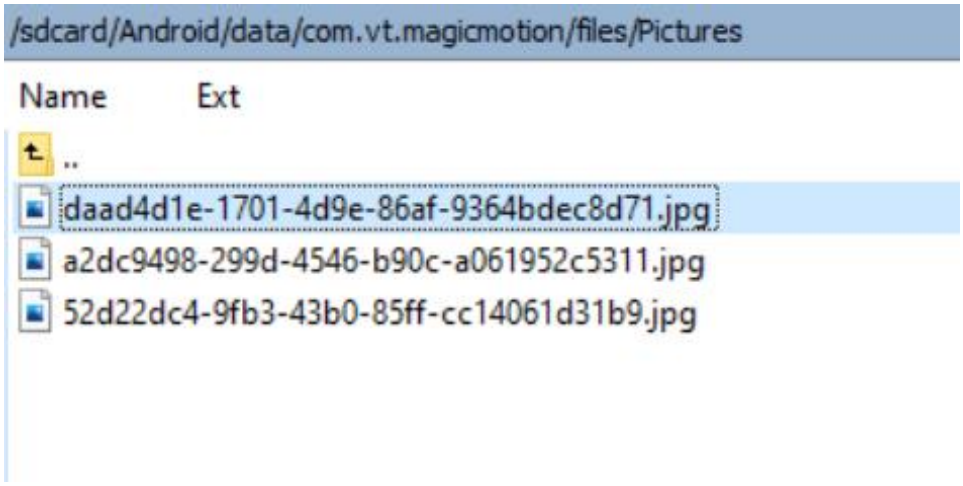
This indicates very clearly that the device has no access controls when connecting to the device and any application that can talk the BLE protocol can easily retrieve all the values from the device.

Insecure data storage (CVE-2020-12731)

As a part of security mapping earlier, we decided to focus on mobile device storage.

Pictures

It seems that the mobile application “MagicMotion” would log all the images to the device sdcard in a folder called “/sdcard/Android/data/com.vt.magicmotion/files/Pictures”. We can clearly see that the files are stored on the sdcard as in the image below.



Pictures in original application folder on sdcard

Below is the snippet of code that allows an attacker to create a malicious application that can copy every file from the sdcard folder that belongs to the VeryFit application to a sperate folder on the scdcard. However, an attacker can actually transfer those files to his/her server as well.

```
copyDirectory(new
File("/sdcard/Android/data/com.vt.magicmotion/files/Pictures"),new
File("/sdcard/Magicmotion_images_CP"));

 public void copyDirectory(File sourceLocation , File targetLocation)
         throws IOException {

       if (sourceLocation.isDirectory()) {
          if (!targetLocation.exists() && !targetLocation.mkdirs()) {
             throw new IOException("Cannot create dir " +
targetLocation.getAbsolutePath());
          }

          String[] children = sourceLocation.list();
          for (int i=0; i<children.length; i++) {
             copyDirectory(new File(sourceLocation, children[i]),
                   new File(targetLocation, children[i]));
```

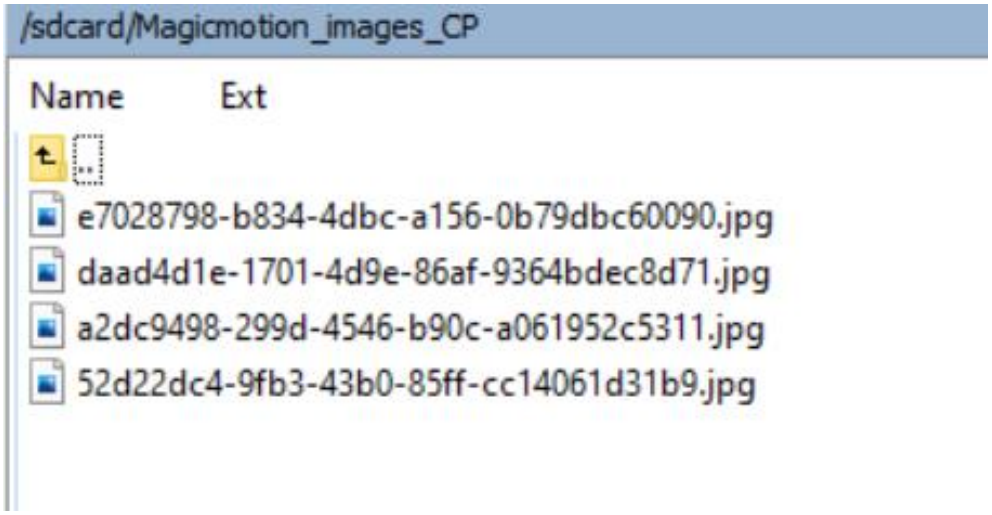
```
        }
    } else {

        // make sure the directory we plan to store the recording in exists
        File directory = targetLocation.getParentFile();
        if (directory != null && !directory.exists() && !directory.mkdirs()) {
            throw new IOException("Cannot create dir " + directory.getAbsolutePath());
        }

        InputStream in = new FileInputStream(sourceLocation);
        OutputStream out = new FileOutputStream(targetLocation);

        // Copy the bits from instream to outstream
        byte[] buf = new byte[1024];
        int len;
        while ((len = in.read(buf)) > 0) {
            out.write(buf, 0, len);
        }
        in.close();
        out.close();
    }
}
```

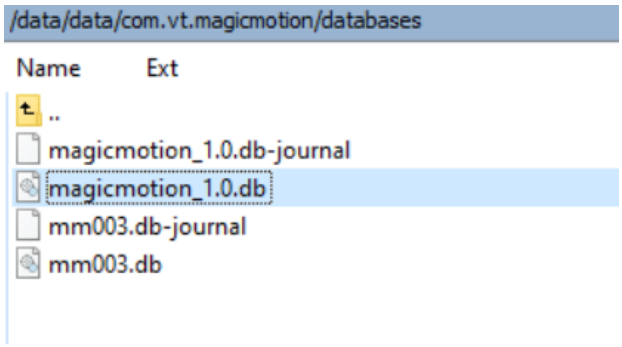
We can see in the image below that the code above executed by a malicious app can transfer the images to a separate folder on the same device. Then an attacker can transfer it to their server using the same malicious application.



Pictures in stolen folder on sdcard

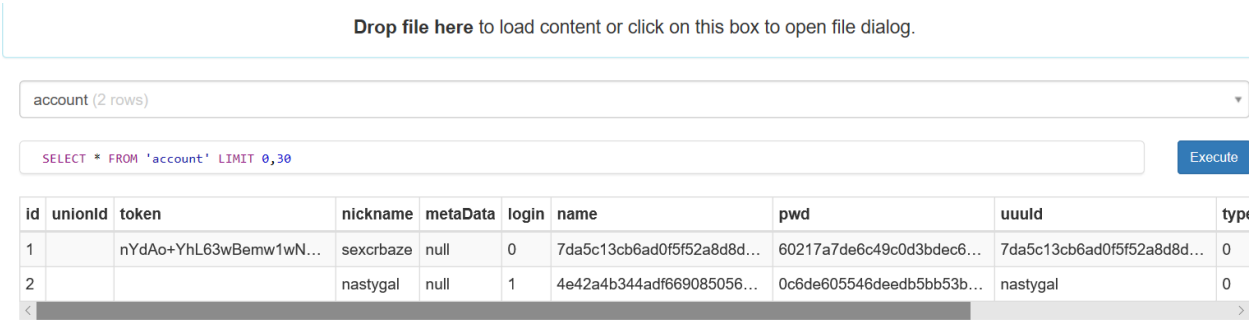
User Information

It seems that the mobile application “MagicMotion” would log all the user information to the device in clear text in a folder called “/data/data/com.vt.magicmotion/databases”. Although the data is stored in an application folder, however rooted device can allow malicious application with root rights to look at that data.



Logcat.txt

Below is the data that can be seen by an attacker in clear text.



id	unionId	token	nickname	metaData	login	name	pwd	uuuId	typ
1		nYdAo+YhL63wBemw1wN…	sexcrbaze	null	0	7da5c13cb6ad0f5f52a8d8d…	60217a7de6c49c0d3bdec6…	7da5c13cb6ad0f5f52a8d8d…	0
2			nastygal	null	1	4e42a4b344adf669085056…	0c6de605546deedb5bb53b…	nastygal	0