《智能家居安全——身份劫持》

——作者：挽秋(daizy)

1、概要

本文以如何劫持(窃取)智能家居时代设备的身份“安全凭证”为出发点，调研并分析了目前国内市场的主流产品和设备交互协议，及其所依赖身份凭证，通过介绍、分析和发现设备交互控制协议安全性，最终通过身份劫持，实现相关设备和产品的任意远程控制。

2、智能家居身份和劫持危害

先通过一张简图来了解一下应用、智能设备和云端三者交互时携带的身份标识，如图 2.1 所示：

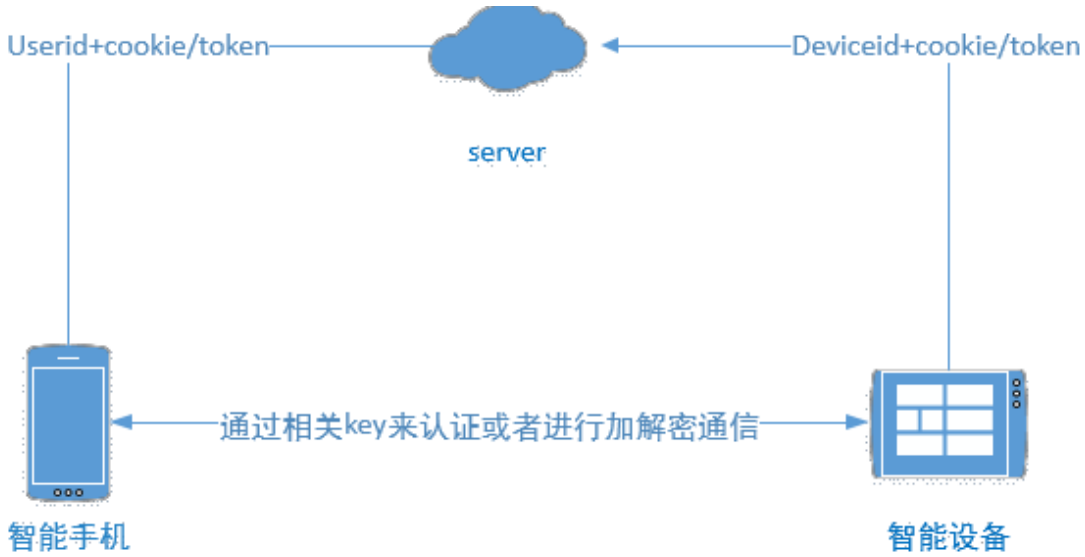


图 2.1：智能家居设备交互时携带的身份标识

从上图了解到，智能家居身份标识通常是以下几种情况：

1）账号 cookie 相关，如身份 Token；

2）用户 id：userid

3）设备 id：deviceid

4）认证或加密的 key

一旦用户或设备的身份被劫持，那么至少存在如下几方面危害：

1） 个人信息，聊天内容等隐私敏感信息泄露

2） 智能设备被任意控制

3） 财产损失

4） 随时被监控

以智能音箱和智能插座等设备为例，至少有两个环节设计 “身份”相关：

1） 账号同步

2） 设备交互操作

下面将分别介绍如何在这两个环节进行身份劫持。

3、账号同步

账号同步是指，在智能设备在初次激活使用(或更改绑定用户时)，用户将自己的身份信息同步给设备，并绑定设备。

一个简化的账号同步流程，如图 3.1 所示：

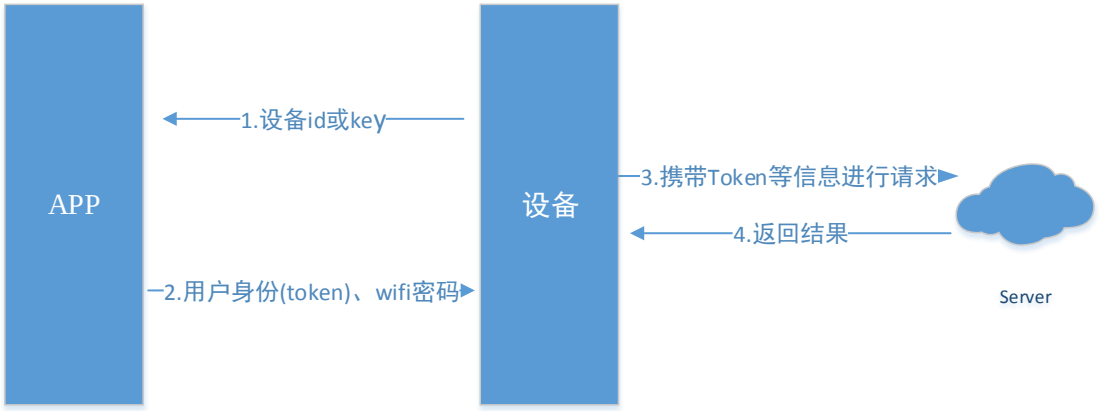


图 3.1：账号同步

账号同步通常会存在如下两类问题：

1） 设备是否合法：验证设备 id 还是设备 key？id 和 key 都很容易泄露伪造。

2） 账号 Token 如何安全传输：设备此时为入网，通过蓝牙、AP，还是其他何种方式传输账号信息。

账号同步时身份劫持

以厂商 A 音箱的配网和身份账号同步为例，其账号同步分为两种方式：

1）直接通过 UDP 广播 255.255.255.255:50000,发送 userid、token 和 wifi 的 ssid 和 wifi 密码。

2）将 userid、token、ssid 和 wifi 密码等转化成语音播放，音箱进行语音信息识别即可。

关于两种模式的选择：由本地 sharedPreferences 文件(tg_app_env.xml)中的 app_connect_mode 属性值决定，其账号同步代码如图 3.2 所示：

```
private void doConnectDevice(String str, String str2, String str3, String str4) {
    XVb.d("connecting, userId: " + str + ", authCode: " + str2 + ", ssid: " + str3 + ", password: " + str4);
    int model = C2776Etb.getInstance().getModel();
    XVb.v("connect model: " + model);
    if (model != 2) {
        try {
            if (this.mNetConfig == null) {
                this.mNetConfig = C7407kmc.getInstance();
            }
            XVb.v("start wifi provision");
            this.mNetConfig.startProvision(str3, str4, str, str2);
        } catch (IOException e) {
            XVb.w("IOException, connect device failed !!!");
            connectDeviceFailed();
            e.printStackTrace();
        }
    }
    if (model != 1) {
        if (this.mSoundConfig == null) {
            this.mSoundConfig = C7601mmc.getInstance(this.activity.getApplicationContext());
        }
        XVb.v("start sound provision");
        this.mSoundConfig.startEncodeAndPlayAudio(str3, str4, str, str2);
    }
```

图 3.2：厂商 A 音箱的账号同步

厂商 A 的音箱将身份信息，通过固定“协议”的格式，在 UDP255.255.255.255:50000 端口进行身份信息发送，攻击者可以监听 UDP50000 端口，从而获取用户的 userid 和 token，窃取身份凭据；语音发送也是按照同一套固定的“协议”格式发送。协议格式通过破解后如图 3.3 所示：

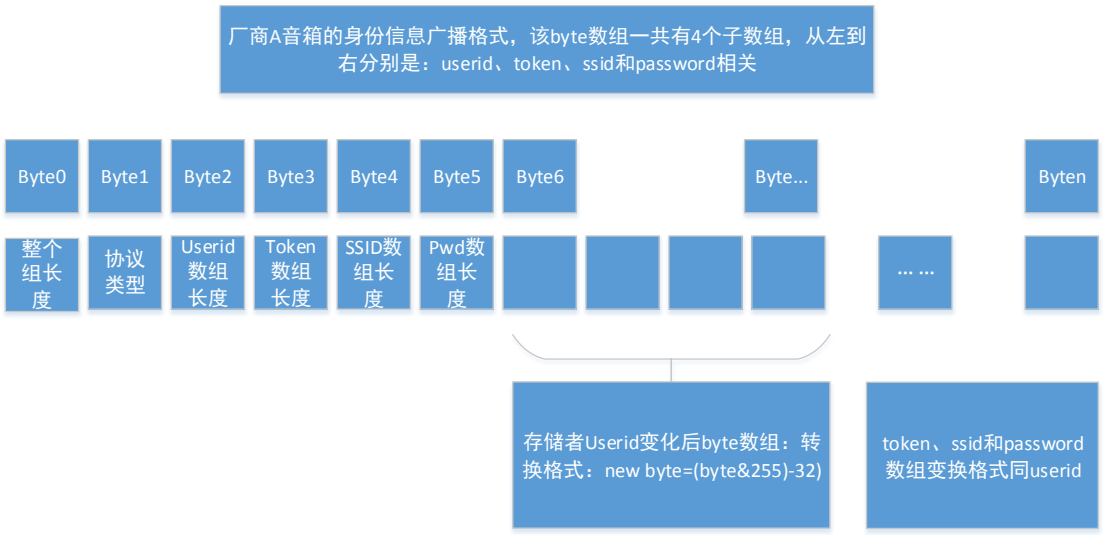


图 3.3：厂商 A 音箱的账号信息同步格式

4、设备交互

设备交互是指应用、设备和云端的三者交互访问；交互操作大体分为两种方式：

1） 只支持广域网：厂商 A 为代表；

2） 支持广域网和局域网：厂商 B 和 C 为代表。

广域网应用与设备交互、设备与设备的交互方式如图 4.1 和 4.2 所示：

图 4.1：应用和设备广域网交互

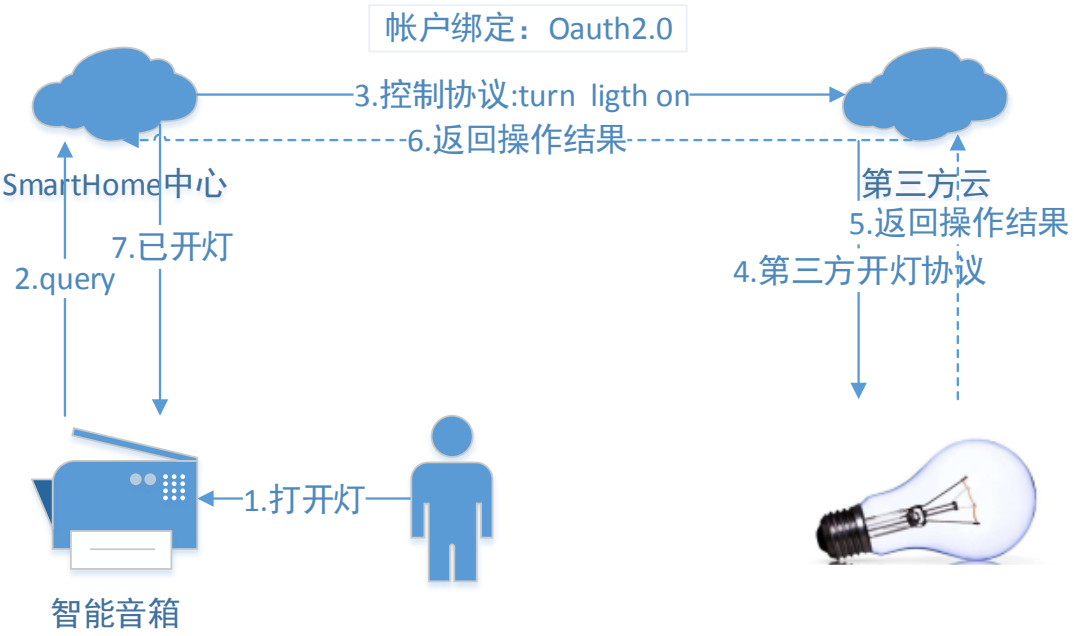


图 4.2：设备和设备广域网交互

厂商 A 的智能家居接入方式：已开灯为例

第一步：**厂商 A 的音箱→音箱 server**

url：https://***.com/***

Payload: { Uderid, Deviceid, Accesstoken, 打开灯的语音}

第二步：**厂商 A 的音箱 sever→第三方 server**

用户需要在第三方产品 server 注册并通过 Oauth 授权给厂商 A 的 Server，消息格式如下：

```
 {
"header":{
  “namespace”:” ***Genie.Iot.Device.Control",
   "name":"TurnOn",
   "messageId":"1bd5d003-31b9-476f-ad03-71d471922820",
   "payLoadVersion":1
 },
 "payload":{
    "accessToken":"access token",
    "deviceId":"34234",
    "deviceType":"XXX",
    "attribute":"powerstate",
    "value":"on",
    "extensions":{
      "extension1":"",
      "extension2":""
     }
    }
}
```

第三步：**第三方 server→设备**

Payload：{command: turn-on, currentValue:0 }

厂商 A 音箱的身份劫持

厂商 A 的音箱每次交互时，都会携带: token、userid、deviceid、action 来进行，并且 server 会依据 userid 来进行身份判断。

1）有了 userid 就可以身份劫持——远程设备任意操作；

2）userId 是顺序的，可遍历的 9 位数字：比如一个 userid 是 50****123，另一个 userid 则是 50****397 这几位数字;

3） userid 还有其他多种方式获得：配网时窃取、APP 端上获取；

厂商 A 音箱被劫持后，可以用户查看聊天记录，自定义问答，设置闹钟、话费充值、智能家居控制等等，此外音箱 “被分享”之后，宿主不能主动取消分享，只能等“攻击者”取消分享，身份劫持危害如图 4.3 所示，中间的攻击者可以任意查看用户的聊天记录：

图 4.3：厂商 A 音箱的身份劫持

如何发现这类身份劫持？

应用或设备通过携带 4 元组信息：userid、deviceid、token 和 action，向云端进行请求时，如图 4.4 所示，如果云端对 4 元组信息校验出现不一致的情况下，就会导致身份劫持：

1） 把 userid、deviceid、token 三者信息中的一种直接当成用户身份，而不是进行严格的身份一致性判断：判断 userid 和 token 是否一致，用户身份和设备列表是否是绑定关系。

2） 用户身份和 action 判断，存在逻辑漏洞，导致攻击者可以进行操作提权，比如子设备提权可以操作“属主”身份的一些权限，OTA 更新等等。

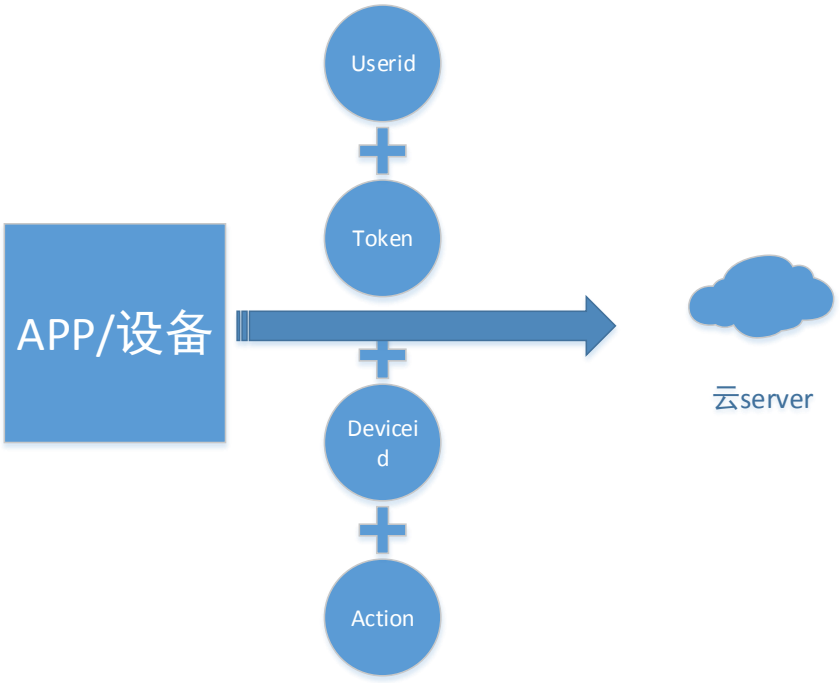


图 4.4: 4 元组访问请求

局域网中应用与设备交互、设备与设备的交互方式如图 4.5 和 4.6 所示：

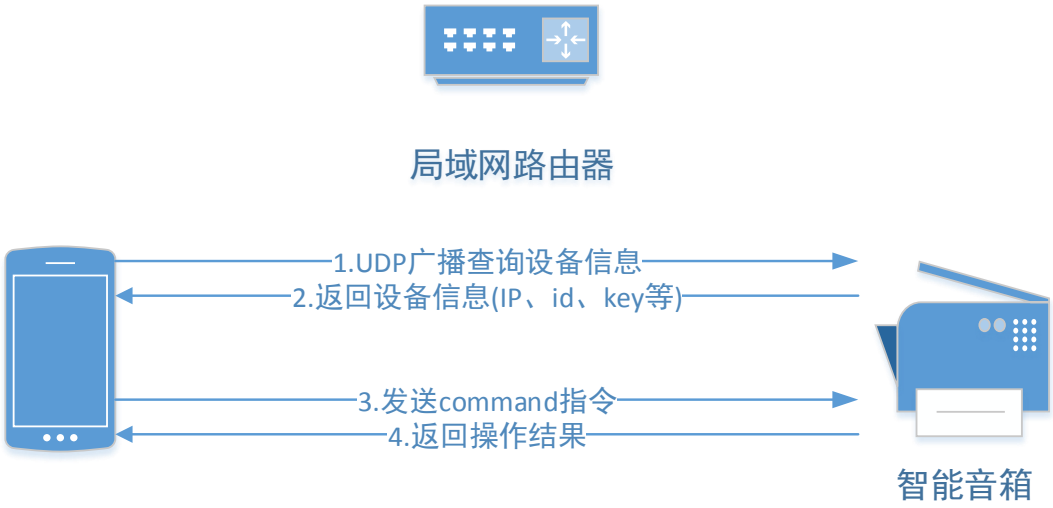


图 4.5：应用和设备局域网交互

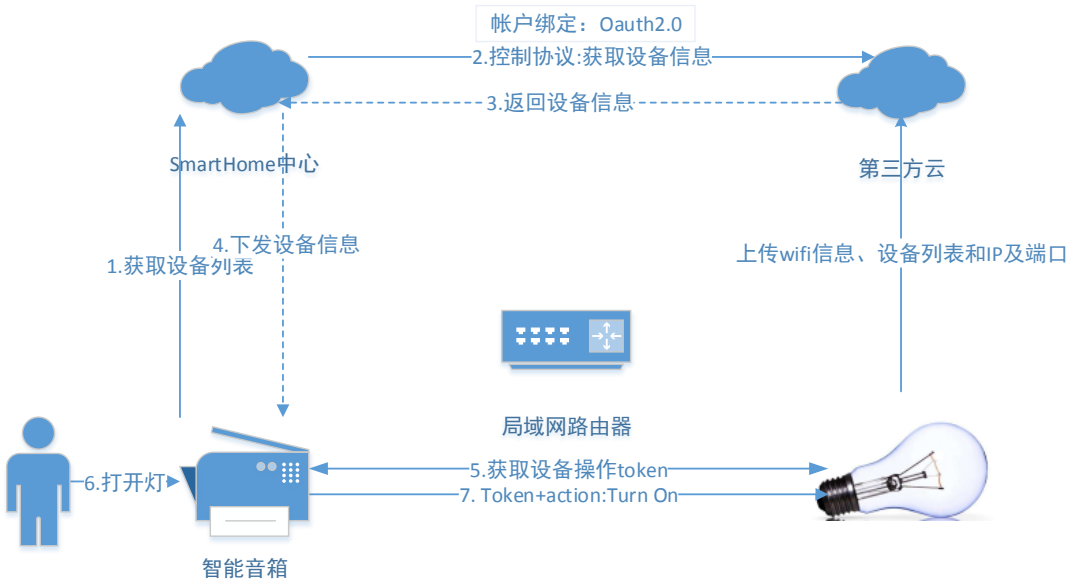


图 4.6：设备和设备局域网交互

厂商 B 的设备局域网身份劫持

在同一局域网下，厂商 B 设备通过专有的加密 UDP 网络协议——miio 协议，进行通信控制。

1）通过广播发送一个握手协议包，如果设备支持 miio 协议，那么设备就会回复自身信息：token、ip 和 ID。

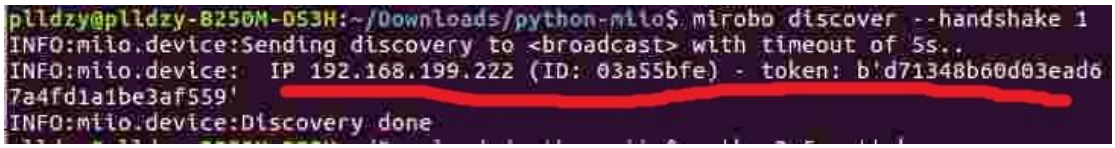
2）向指定设备发送一串 hello bytes 获得设备信息结构体”header”

3）凭借 token、ID 等信息构造信息结构体”header”，跟随控制消息发送给设备，实现设备控制。

厂商 B 的设备局域网身份劫持交互如图 4.7 所示：

图 4.7：厂商 B 的设备局域网身份劫持交互

第一步：安装 python-miio 库，然后执行：mirobo discover --handshake 1，获取设备 IP、ID 和 Token 信息。



第二步：发送 hello bytes 消息给设备 54321 端口，获取设备消息结构体 Header：

```
plldzy@plldzy-B250M-DS3H:~/Downloads/python-miio/miio$ python3.5 gettoken.py
Container:
    data = Container:
        offset2 = 32
        value = b'' (total 0)
        data = b'' (total 0)
        length = 0
        offset1 = 32
    header = Container:
        offset2 = 16
        value = Container:
            length = 32
            unknown = 0
            device_id = b'\x03\xa5[\xfe' (total 4)
            ts = 1970-01-01 00:38:51
        data = b'!1\x00 \x00\x80\x00\x00\x03\xa5[\xfe\x00\x00\t\x1b' (total 16)
        length = 16
        offset1 = 0
    checksum = b'\xd7\x13H\xb6\r\x03\xea\xd6z0\xd1\xa1\xbe:\xf5Y' (total 16)
b'd71348b60d03ead67a4fd1a1be3af559'
```

第三步：伪造控制消息结构体 Header、消息指令 cmd 和 checksum(Token)，给设备发送；

```
typedef struct{
    Header,
    cmd,
    checksum
}Msg
```

控制消息结构体如图 4.8 所示：

图 4.8：厂商 B 的设备控制消息结构体

已打开智能插座为例：cmd={'id':1,'method':'set_power','params':['on']}

厂商 C 的局域网交互控制

厂商 C 为了实现智能家居生态，主推一套实现产品智能化互联互通的协议——“***Link”，目前所有的产品都可以与 APP，以及音箱进行交互控制，是一套“带认证的密钥协商+对称密钥加密”的设备操作和交互控制协议。

再介绍和认识“带认证的密钥协商”之前，我们先介绍一下 ECDH 密钥协商及其存在的安全问题。

有两个用户 Bob 和 Alice，使用 ECDH 密钥协商，交互过程如图 4.9 所示：

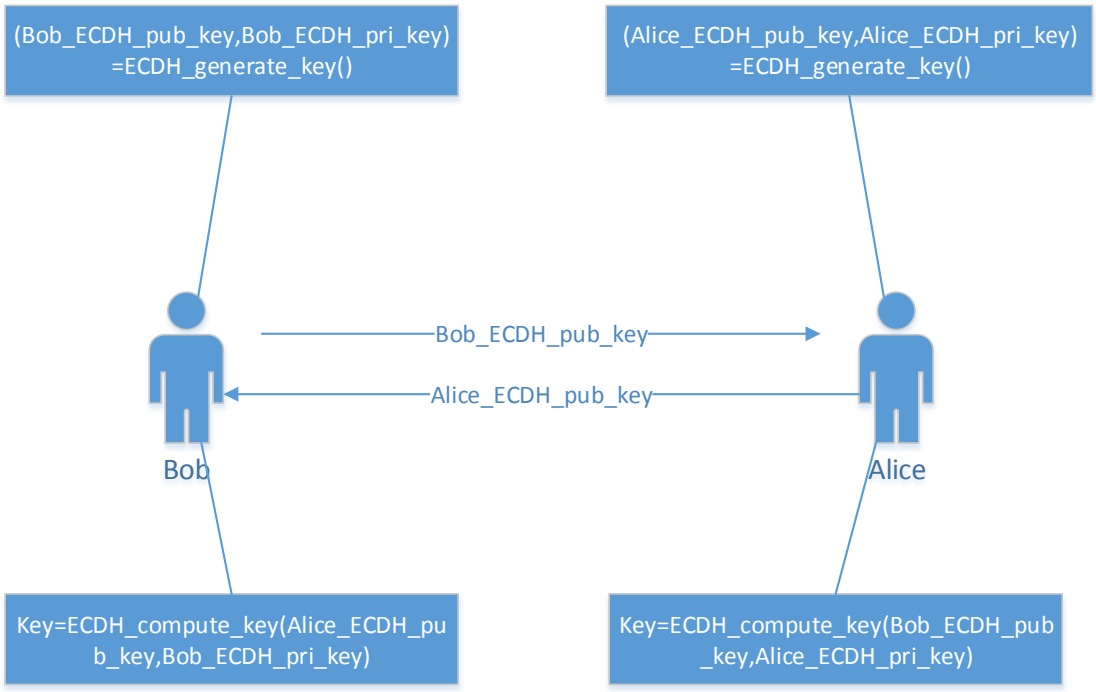


图 4.9：ECDH 密钥协商

但是 ECDH 密钥协商是无法防御中间人攻击的，假设在 Bob 和 Alice 存在一个攻击者——Attack，对 Bob 和 Alice 进行中间人攻击，ECDH 协商流程如图 4.10 所示：

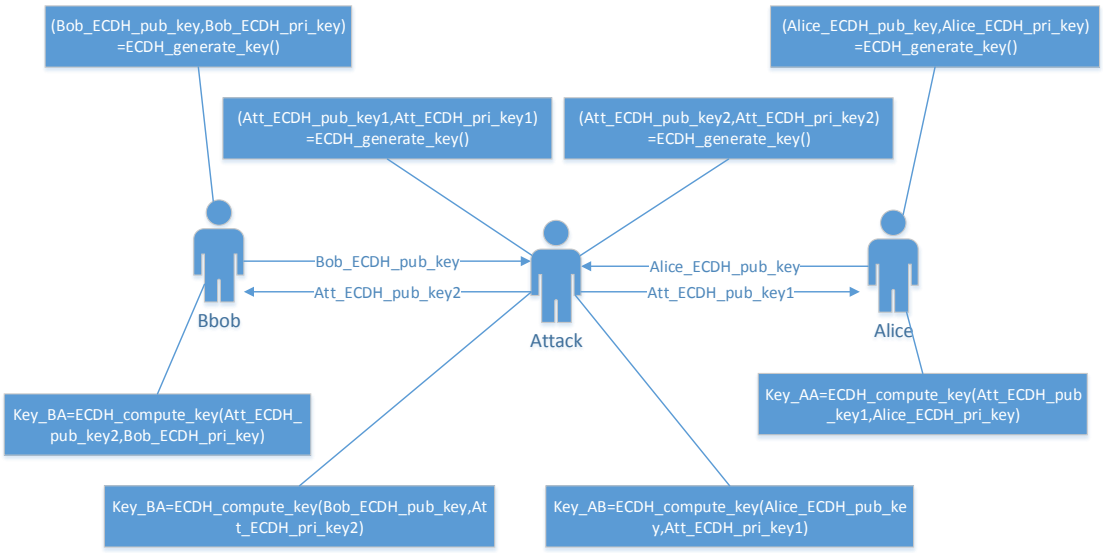


图 4.10：ECDH 密钥协商之中间人攻击

为了防御中间人攻击，需要在 ECDH 密钥协商过程中加入“一套身份认证机制”——EccSignKey 和 EccVerifyKey，EccVerifyKey 提前存储在需要协商密钥的用户设备上，整个“待认证的 ECDH 密钥协商”交互过程如图 4.11 所示：

图 4.11：待认证的 ECDH 密钥协商

设备和厂商 C 的应用(或音箱)基于***Link 协议来进行交互，第三方设备制造商首先在云端通过 ECC 算法一对生成公私钥：Ecc-sPrivateKey/Ecc-sPubkey，其中公钥 Ecc-sPubkey 内置在设备端，用于加密发送随机数到云端，进行设备的身份认证，设备认证合法后，云端下发后续通信加密的 key：accessKey 到设备上，然后应用使用 ECDH 密钥协商算法协商出的密钥，通过 AES-CBC 模式加密传输 accessKey；此外设备和应用进行局域网通信时，都是通过 localkey 进行加解密来进行的，其中 localkey 就是 accessKey。设备和厂商 C 的应用局域网交互流程如图 4.12 所示：

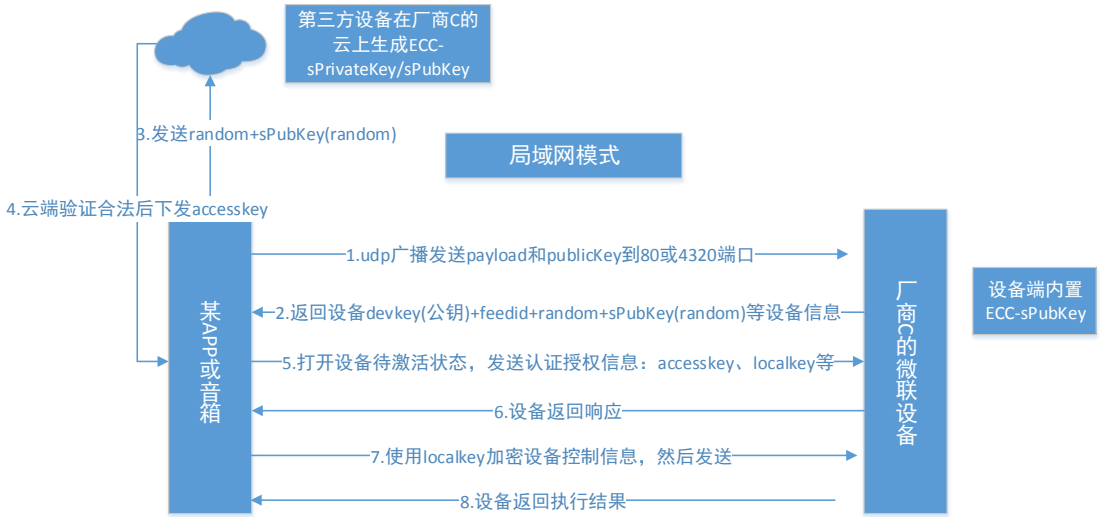


图 4.12：设备和厂商 C 的应用局域网通信交互

厂商 C 的设备局域网身份劫持

厂商 C 的***Link 协议的交互控制的消息结构体如下所示：

已打开智能插座为例：

Packet_t=协议包头，opt=null，Payload=LocalKey 密钥加密

```
Time[时间戳]        //4 字节 int 类型时间戳，小端在前
{
    “cmd”:5,
    "data":{
    "streams":[{"current_value":"0","stream_id":"power"}],
    "snapshot":[{"current_value":"1","stream_id":"power"}]
}
```

设备交互方式总结和比较

属性\公司	厂商 A	厂商 B	厂商 C
交互方式	只允许云端交互	允许云端和局域网	允许云端和局域网
是否可劫持	音箱 server 和第三方设备进行控制协议交互； 身份凭证是 userid，可劫持	生态链企业，云端统一走厂商 B 的生态链云；基于 miio 协议 局域网交互，身份凭证 token 可劫持	第三方企业使用其 link 协议，云端使用厂商 C 的云作为 server；局域网交互依赖 localkey，目前安全。但是设备身份依赖于 ECC-sPubKey(多个设备一个 key)，该 key 失窃后，设备可以被伪造。
产品安全性负责	厂商 A 只负责自己音箱自生的安全性，第三方产品的安全性自行负责。	厂商 B 负责	第三方自己负责，但是***link 协议统一交互控制、OTA 更新等，安全性极大的有保障
账号	第三方 Oauth 登录授权	统一厂商 B 的帐户	厂商 C 的帐户
APP 控制	第三方有独立 APP	厂商 B 的 APP	厂商 C 的 APP(H5 小程序)

5、通过应用实现身份劫持

通过应用实现身份劫持，常用的方法有如下两种：

1）通过 webview JS 交互接口远程命令执行或泄露身份账号

应用 APP 通过为 webview @JavascriptInterface 关键字，自定义添加身份获取的函数，并

且没对加载 url 做好限制，导致身份信息远程泄露或者远程命令执行。

2）Webview file 域远程信息泄露

应用开启 WebSettings.setAllowUniversalAccessFromFileURLs(true)，并且 webview 对加载的 url 没有任何限制，则应用 APP 下所有私有目录信息都会被窃取。

通过 webview JS 交互接口远程命令执行或泄露身份账号

应用扫一扫时(CaptureActivity),当 CaptureActivity 扫描到是“合法”url 时，会调用 com.***.WebViewActivity 进行 url 加载，但是 url 判断逻辑存在漏洞，导致攻击者可以调用 WebViewActivity 定义的交互接口，远程获取用户账号等敏感身份信息，漏洞执行效果如图 5.1 所示。

漏洞案列简化：

```
if(loadurl.contains(“***”)){
        //合法
} else{
        //不合法
}
```

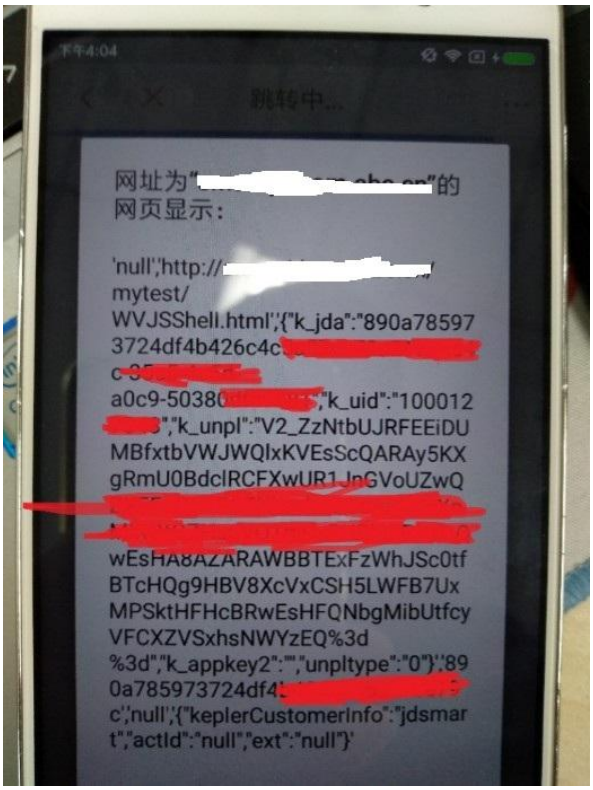


图 5.1：通过 webview-JS 交互接口获取厂商 C 控制应用的身份

Webview file 域远程信息泄露

厂商 A 的音箱控制 APP 中 WVWebViewActivity 对外导出，并接收如下远程 uri scheme：**assistant://hsend***Poc5_web_view?direct_address=url**。

```
<activity android:name="com.i                .tg.activity.WVWebViewActivity" android:screenOrientation="portrait" android:windowSoft
    <intent-filter>
        <action android:name="android.intent.action.VIEW"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <category android:name="android.intent.category.BROWSABLE"/>
        <data android:scheme="assistant" android:host="h5_web_view"/>
    </intent-filter>
```

WVWebViewActivity 接受外部的 url 会传入 Fragment 中的 webview 中进行加载，并且 WVWebViewActivity 中对 webview 进行了设置，开启了 JS 和 file 文件访问能力，并设置了

WebSettings.setAllowUniversalAccessFromFileURLs(true)。

攻击者可以将 assistant 伪协议中的 url 先通过 url 加载任意 html，然后下载恶意 html 文件到本地，然后 webview 跳转加载本地的恶意 html 文件，窃取用户私有目录内的身份信息。

assistant://hsend*Poc5_web_view?direct_address=http://www.test.com**

assistant://hsend*Poc5_web_view?direct_address=file:///*/***.html**

6、智能家居身份劫持漏洞总结

1：配网泄露

2：设备交互控制时，劫持

1）app/设备->server：厂商 A 为代表，userid 为身份凭证，可劫持；

2）局域网控制：

A、厂商 B 的局域网控制基于 miio 协议：token 泄露，可劫持；

B、厂商 C 的局域网控制：带认证的密钥协商+对称密钥加密(localkey),协议安全，但是设备身份依赖于 ECC-sPubKey(多个设备一个 key)，设备可被伪造；

3：app 应用存在身份穿越漏洞

A、Webview JS 交互接口远程命令执行或远程信息泄露

B、Webview File 域远程信息克隆

7、参考文章

1.https://github.com/WeMobileDev/article/blob/master/%E5%9F%BA%E4%BA%8ETLS1.3%E7%9A%84%E5%BE%AE%E4%BF%A1%E5%AE%89%E5%85%A8%E9%80%9A%E4%BF%A1%E5%8D%8F%E8%AE%AEmmtls%E4%BB%8B%E7%BB%8D.md

2. https://github.com/rytilahti/python-miio

3.https://paper.seebug.org/616/