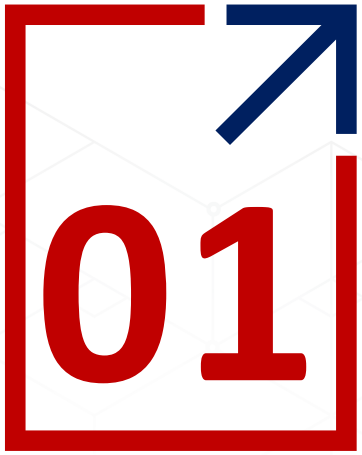
The Secret Codes Tell the Secrets

Zhang Qing@ByteDance and Bai Guangdong@UQ

CONTENTS



About Us

About Us

Zhang Qing

Senior Android security researcher from ByteDance

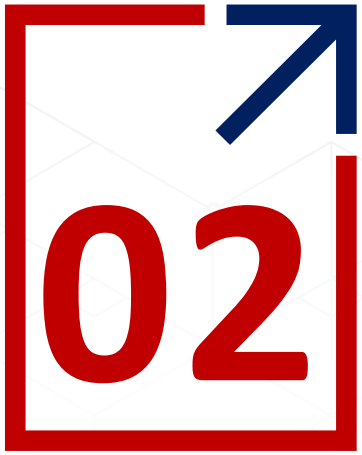
Research on Android security and payment security

Bai Guangdong

Senior Lecturer from

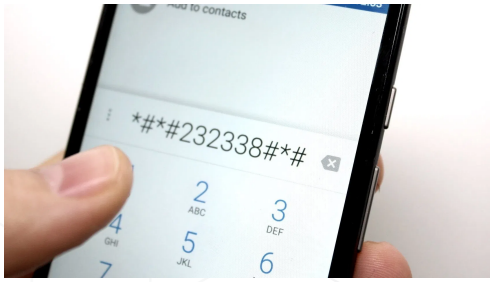
The University of Queensland, Australia

Research on mobile security and protocol analysis



Secret codes

Why focus on secret codes?



Android secret codes

- are well known by some geeks or engineers
- are used for test and for fun
- Secret codes' security implications have not received enough attention. On the other hand, these secret codes expose many attack surfaces which may be easy to access and have the system privilege.

In 2017, a security researcher pointed out that some Android devices have a backdoor in EngineerMode app for diagnostics mode which can lead to root exploit.

On Twitter, Qualcomm VP of Product Security Engineering Alex Gantman stated that the EngineerMode app was not authored by Qualcomm but others who had built it on top of a previous testing app.

Why focus on secret codes?

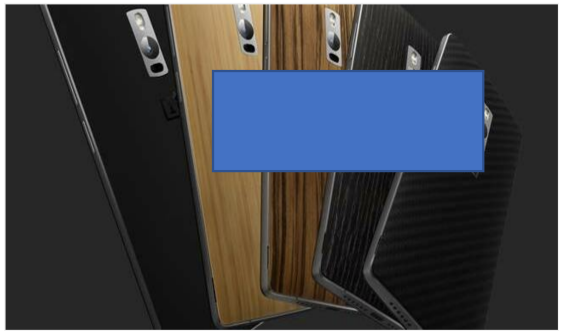
{* PERSONAL TECH *}

Heads up: [redacted]phones have a secret root backdoor and the password is 'angela'

Who left 'wipe the engineering toolkit' off the factory checklist?

TUE 14 NOV 2017 // 21:32 UTC 12 GOT TIPS?

Shaun Nichols in San Francisco BIO EMAIL TWITTER SHARE



Updated An apparent factory cockup has left [redacted] Android smartphones with an exposed diagnostics tool that can be potentially exploited to root the handsets.

QUICK-POLL

How has your budget for security proc changed post-COVID-19?

- Increased
- Decreased
- Stayed the same

Do you have a vulnerability managem solution in place?

- Yes
- No

If yes, where are the biggest gaps in y current solution?

- Lack of vulnerability context
- Too many false positive alerts
- Lack of risk-based prioritization
- Lack of remediation guidance
- Poor reporting capabilities
- N/A - No current solution

Elliot Alderson
@fs0c131y

With telephony secret code you can access to manual tests like GPS test, root status test as stated in this article xda-developers.com/oneplus-hardwa… pointed by @AleGrechi . But can do better...

```
<receiver android:name="com.android.engineeringmode.
    <intent-filter>
        <action android:name="com.oem.engineermode.E
        <action android:name="com.[redacted]engineermode.
    </intent-filter>
    <intent-filter>
        <action android:name="android.provider.Telep

        <data
            android:host="818"
            android:scheme="android_secret_code" />
        <data
            android:host="838"
            android:scheme="android_secret_code" />
        <data
            android:host="7668"
            android:scheme="android_secret_code" />
        <data
            android:host="3439"
            android:scheme="android_secret_code" />
        <data
            android:host="3392"
            android:scheme="android_secret_code" />
    </intent-filter>
</receiver>
```

```
g: "onReceive intent.getAction() = " + intent.getActio
 null) {
).equals("android.provider.Telephony.SECRET_CODE")) {
ent.getData() != null ? intent.getData().getHost() : n
(host)) {
ies.set("oem.cust.flag", "1");
t(context,  resId: 2131297667,  duration: 3000).show();
.equals(host)) {
ies.set("oem.cust.flag", "0");
t(context,  resId: 2131297668,  duration: 3000).show();

= new Intent();
addFlags(268435456);
als(host)) {
ion.setAction("com.android.engineeringmode.NetworkSear
ion.putExtra( name: "Step_Num",  value: "Third");
38".equals(host)) {
ion.setAction("com.android.engineeringmode.NetworkSear
ion.putExtra( name: "Step_Num",  value: "Fourth");
668".equals(host)) {
ion.setAction("com.android.engineeringmode.manualtest.

Activity(intentAction);
```

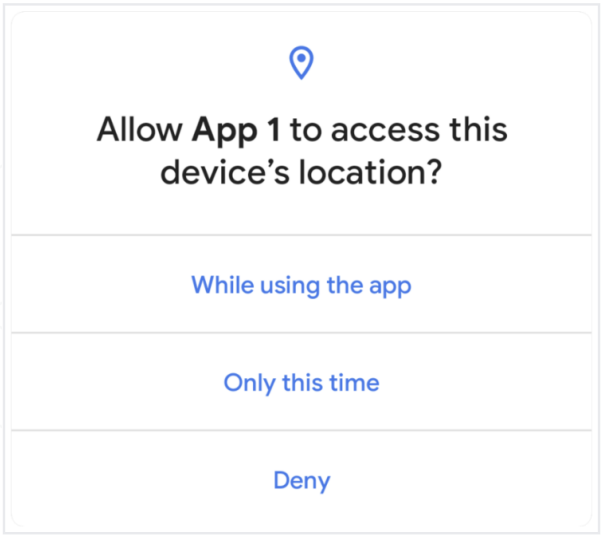
51 1:14 AM - Nov 14, 2017

See Elliot Alderson's other Tweets

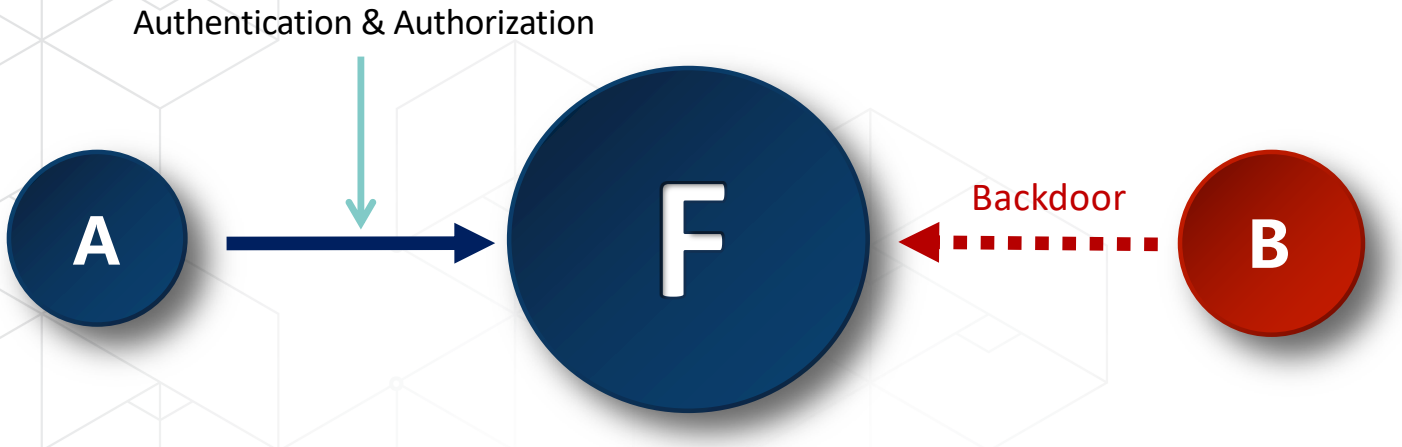
Android Permissions System

As is well known, Google has tightened Android's permissions in recent years

- Gives users interfaces to specify fine-grained permissions for location, microphone, and camera in Android 10/11
- Granting at run time



Android Access Control Mechanism

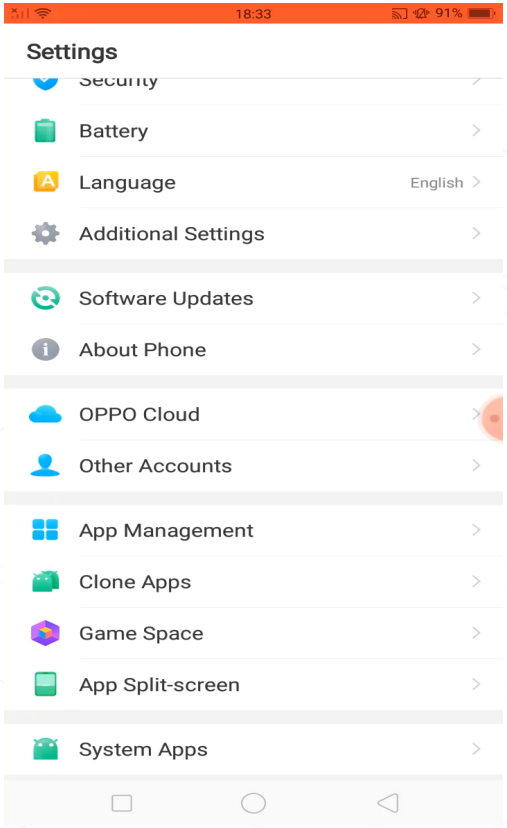


When we design an access control system, permission policies must be enforced consistently and globally.

If any interface can access the system resource with different permission requirement or even without permission, the permission system is breached.

To some extent, this kind of entrance is backdoor.

Android Permissions System



Secret codes

```
static boolean handleSecretCode(Context context, String input) {
    // Secret codes are in the form *#*#<code>#*#*
    int len = input.length();
    if (len > 8 && input.startsWith("*#*#") && input.endsWith("#*#*")) {
        Intent intent = new Intent(TelephonyIntents.SECRET_CODE_ACTION,
                Uri.parse("android_secret_code://" + input.substring(4, len - 4)));
        context.sendBroadcast(intent);
        return true;
    }

    return false;
}
```

Secret Codes refers to codes which can access hidden features or secret menu (such as display information, testing hardware, and software, etc).

These codes provide a fast way for manufacturers to verify that their smartphone and tablets are working as intended.

For example, Instead of “go to settings app > about phone > check IMEI number of your Android Phone”, you can simply dial *#06# to know IMEI number. If such a secret code is executed, the system Dialer app will trigger the above code

Secret codes

How to create your own secret code?

So that whenever *#*#111222#*#* is submitted, your receiver will be notified.

```
<receiver android:name=".DiagnoserReceiver" android:enabled="true" android:exported="true">
    <intent-filter>
        <action android:name="android.provider.Telephony.SECRET_CODE"/>
        <data android:scheme="android_secret_code" android:host="111222"/>
    </intent-filter>
</receiver>

public class DiagnoserReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        if ("android.provider.Telephony.SECRET_CODE".equals(intent.getAction())) {
            Log.e("proyx", intent.getDataString() + "   I   am   here");
        }
        Log.e("proyx", intent.getDataString() + "   I   am   here2");
    }
}
```

Secret codes

Trigger the function

```
public static void useSecretCodeActivity(Context context){
    String secretCode = "111222";
    Intent intent = new Intent(Intent.ACTION_DIAL);
    intent.setData(Uri.parse("tel:*#*#" + secretCode + "#*#*"));
    context.startActivity(intent);
}

public static void useSecretCodeBroadcast(Context context){
    String secretCode = "111222";
    String action = "android.provider.Telephony.SECRET_CODE";
    Uri uri = Uri.parse("android_secret_code://" + secretCode);
    Intent intent = new Intent(action, uri);
    context.sendBroadcast(intent);
}
```

Why would secret codes breach Android’s permission system?

Most of these secret codes come from the factory-installed apps such as EngineerMode or wt_secret_code_manager, and few may be embedded in the basic apps such as contacts, calendar and so on.

Engineermode app and other apps which have the secret codes mostly are built-in and system-signed, and they possess many special and externally accessible privileges (for convenience of developers)

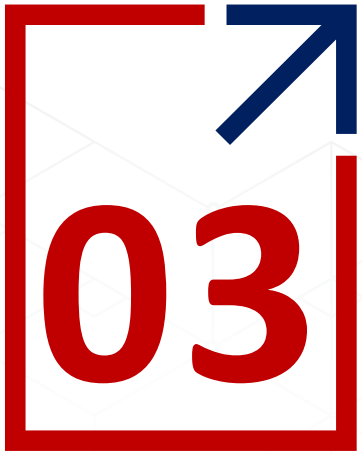
Therefore, there are broad attack surfaces and severe security impact to the users.

```
λ grep -ri "android.provider.Telephony.SECRET_CODE" .
Binary file ./framework/boot-framework.oat matches
./framework/CalendarProvider.apk1a/AndroidManifest.xml:
./framework/Contacts.odex/com/android/contacts/Engineer/d.smali:    cons
./framework/[redacted]EngineerMode.apk1a/AndroidManifest.xml:
./framework/[redacted]EngineerMode.odex/com/oppo/engineermode/CustomizedResetF
./framework/[redacted]PowerMonitor.apk1a/AndroidManifest.xml:
./framework/Settings.apk1a/AndroidManifest.xml:                <action a
./framework/Telecom.odex/com/android/server/telecom/DialerCodeReceiver.s
./framework/Telecom.odex/com/android/server/telecom/TelecomSystem.smali
```

Why would secret codes breach Android’s permission system?

Code	Function
##7780#*#*	Resetting your phone to factory state - Only deletes application data and applications
*2767*3855#	It’s a complete wiping of your mobile also it reinstalls the phones firmware
##273283*255*663282*#*#*	For a quick backup to all your media files
*#06#	Display the IMEI (International Mobile Equipment Identity)
*#9900#	System dump mode
##67#	Erase call diversion
*43#[dial]	Turn on call waiting
#43#[dial]	Turn off call waiting
*#91909#	Fingerprint test
*#301279#	HSDPA/HSUPA settings
**04*[old Pin]*[new Pin]*[new Pin]#	Change Pin (do not enter [and])
**05*[PUK]*[new Pin]*[new Pin]#	Unlock Pin (do not enter [and])

Info codes
Backup codes
Testing codes
Configuration codes
Developer codes
USSD codes



Vulnerabilities

Vulnerabilities caused by secret codes

We found that system reset bypass vulnerabilities (formatting and resetting Android devices) almost influences all OEMs' devices

- the problem is gradually fixed by manufacturers as their Android OSes are upgraded

We found some vulnerabilities, e.g., using engineering mode to change the language violates the permission policies, and turning on cameras may leak users' privacy and information of users' surroundings

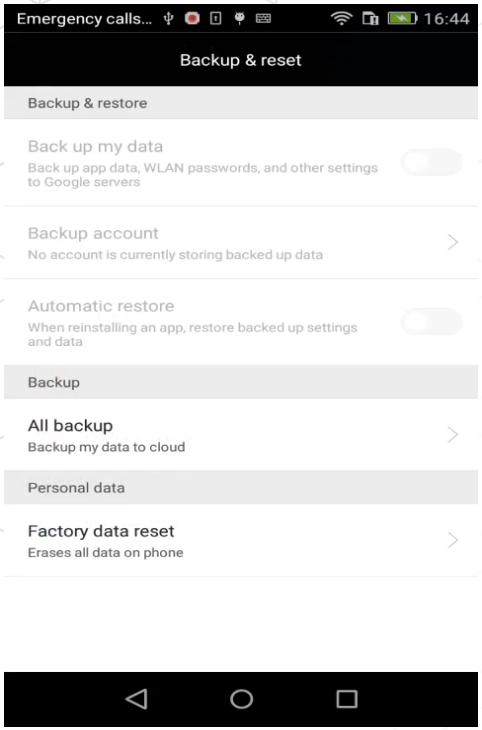
In several phones, we also found that lock-screen PINs are leaked through logcat

What's more, in some devices using the Engineering mode can reboot to Qualcomm's Kernel FFBM mode.

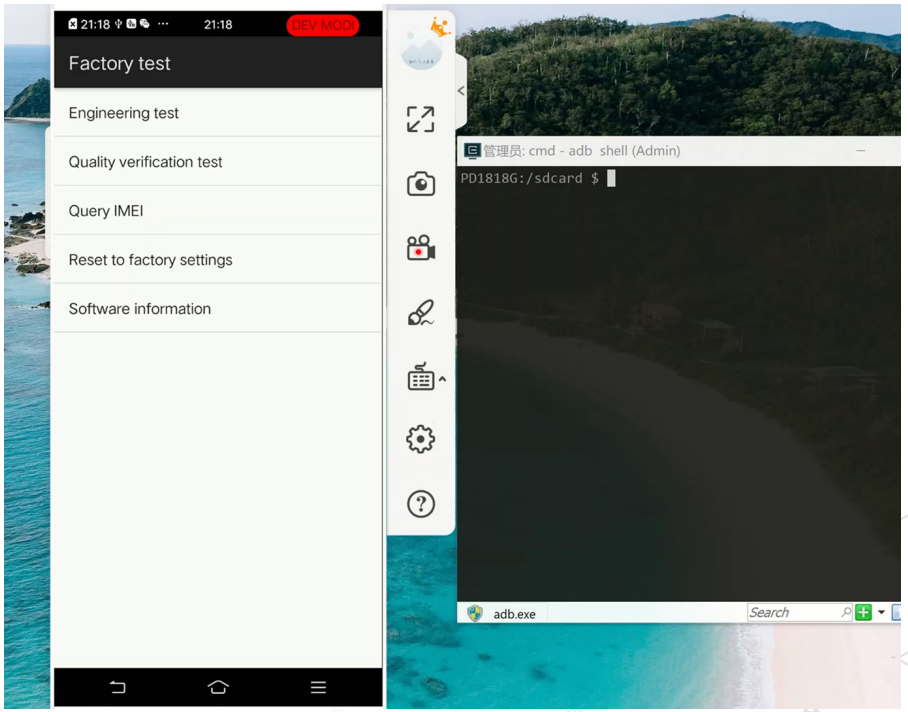
The last but not the least, using the Engineering mode to reset and disable fingerprint lock is commonly found.

Vulnerabilities because of secret codes

System reset bypass (CVE-2017-8152 and another to be assigned soon)



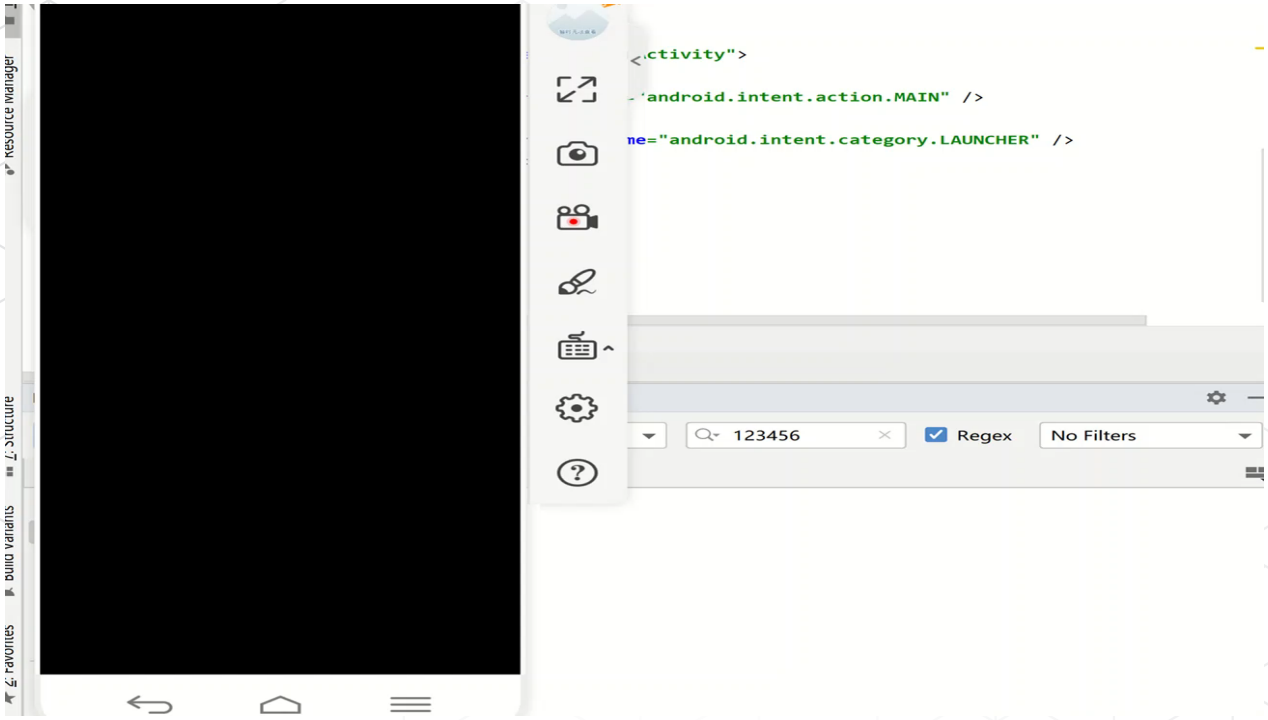
CVE-2017-8152



to be assigned soon

Vulnerabilities because of secret codes

Leak lock-screen PINs and Remove PINs



```
public Context mContext;
Runnable mRunnable = new Runnable() {
    public void run() {
        WifiAdmin unused = ConnWifiReceiver.this.wifiAdmin = new WifiAdmin(ConnWifiReceiver.this.mContext);
        ConnWifiReceiver.this.wifiAdmin.connect(ConnWifiReceiver.this.ssid, (String) null, ConnWifiReceiver.this.pwd, 1, (String) null);
        String unused2 = ConnWifiReceiver.mMacAddr = ConnWifiReceiver.this.wifiAdmin.getMacAddress();
        while (ConnWifiReceiver.this.stopBoolean.booleanValue()) {
            String unused3 = ConnWifiReceiver.mIpAddr = ConnWifiReceiver.this.wifiAdmin.getIPAddress();
            Log.d(ConnWifiReceiver.TAG, "MAC: " + ConnWifiReceiver.mMacAddr + "--IP: " + ConnWifiReceiver.mIpAddr);
            if (!ConnWifiReceiver.mIpAddr.equals("0.0.0.0")) {
                Boolean unused4 = ConnWifiReceiver.this.stopBoolean = false;
                return;
            }
            try {
                Thread.sleep(100);
            } catch (InterruptedException e) {
            }
        }
    }
};
/* access modifiers changed from: private */
public String pwd;
/* access modifiers changed from: private */
public String ssid;
/* access modifiers changed from: private */
public Boolean stopBoolean = true;
/* access modifiers changed from: private */
public WifiAdmin wifiAdmin = null;

public void onReceive(Context context, Intent intent) {
    Log.d(TAG, "enter ConnWifiReceiver......");
    String action = intent.getAction();
    Log.d(TAG, "action = " + action);
    this.mContext = context;
    if (action.equals("bbk.intent.action.CONNWIFI")) {
        this.ssid = intent.getStringExtra("ssid");
        this.pwd = intent.getStringExtra("pwd");
        Log.d(TAG, "ssid = " + this.ssid + "pwd = " + this.pwd);
        new Thread(this.mRunnable).start();
```

Code snippet found in EngineerMode app

Vulnerabilities because of secret codes

reboot to Qualcomm's Kernel FFBM mode

1、FFBM: Fast Factory Boot Mode

To facilitate fast reboot and testing in factory, Qualcomm developed FFBM mode which provides minimal user interaction

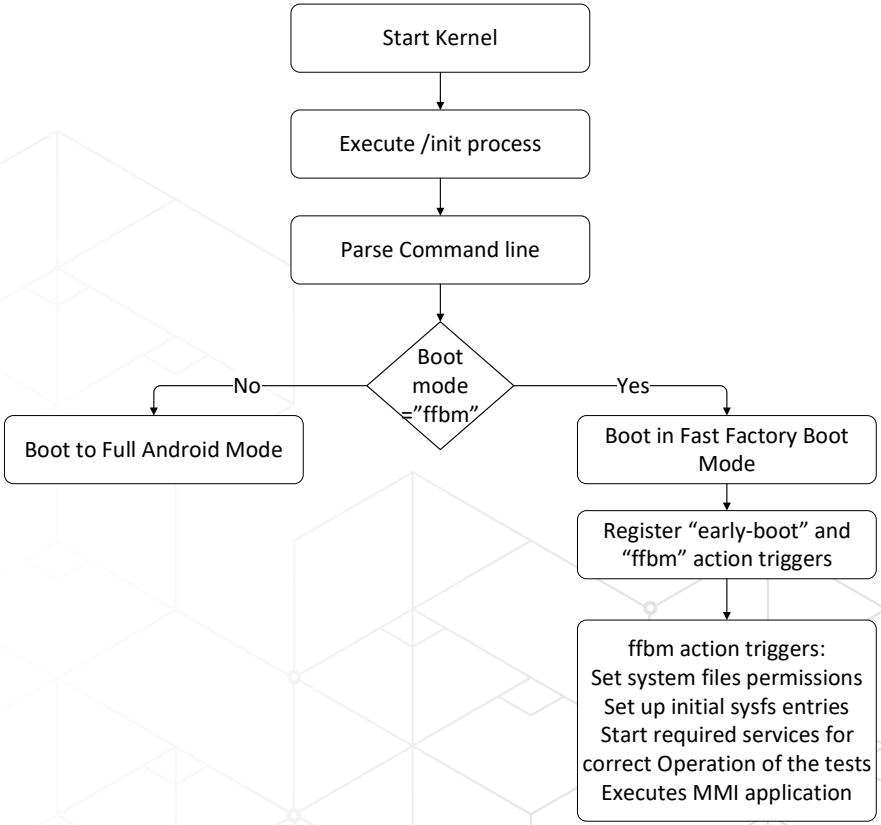
2、Flow chart on the right: boot into Kernel FFBM

3、To quit FFBM mode and enter Android mode

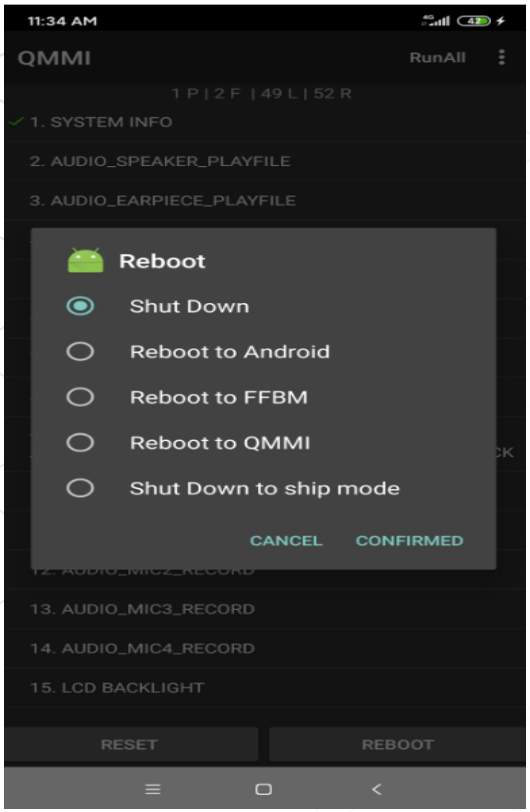
1) adb reboot bootloader
2) fastboot erase misc

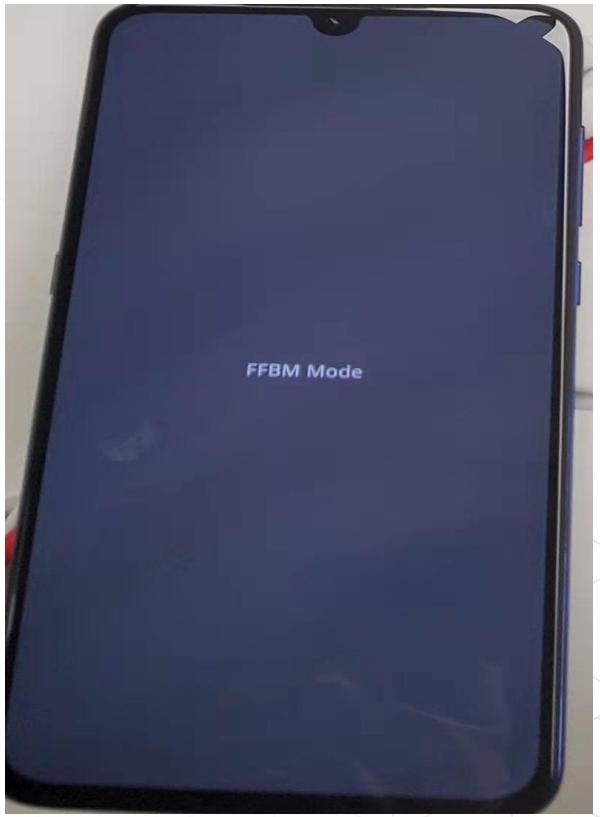
Many mobile devices erase /misc partition to quit from FFBM. /misc includes crucial system configuration, CID (Carrier or Region ID), USB configuration, hardware configuration, etc. Once lost, the device doesn' t function normally

3) fastboot reboot



Vulnerabilities because of secret codes





Vulnerabilities because of secret codes

Reset your fingerprint using engineering mode

关于工程模式的指纹问题

2017-12-11 10:47 | 复制链接

已答复

用9527进了工程模式的指纹测试后指纹被删除，进不了设置和加密应用，怎么恢复@小老鼠 @俊康

Fingerprint in factory mode erased my fingerprint, so I can't enter settings

工程模式别乱测，把我指纹什么的都测没了

[讨论求助] | 发表于 2019-12-3 16:37 | 只看大图 | 倒序浏览 | 864 | 10 | 阅读模式 直达

Factory mode erased my fingerprint configuration

好尴尬呀。

手滑进工程模式弄的指纹识别没法用了怎么办 去售后修了还是没完 只看楼主 收藏

楼主

手滑进工程模式弄的指纹识别没法用了怎么办 去售后修了还是没完全好

Factory mode disabled my fingerprint authentication, and customer service couldn't fix it

[讨论] 千万别用工程模式校准指纹 [复制链接]

秦高远 | 24

4392 | 11

发表于 2019-3-17 09:41:22 来自手机 | 只看楼主 | 倒序浏览 | 阅读模式 | 收藏本帖

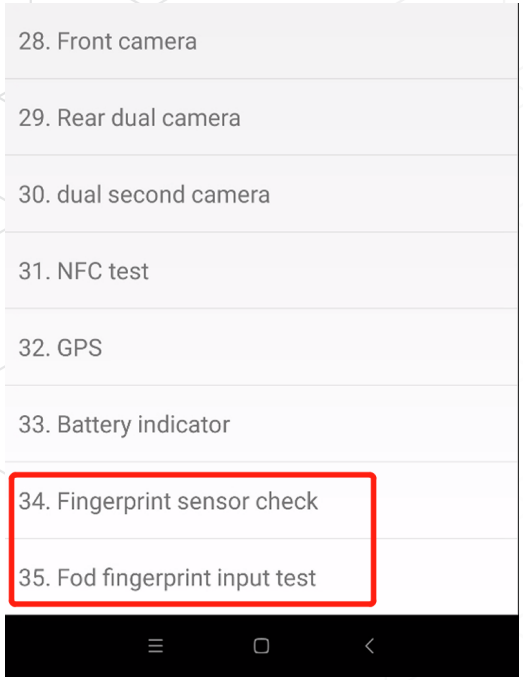
楼主 电梯直达

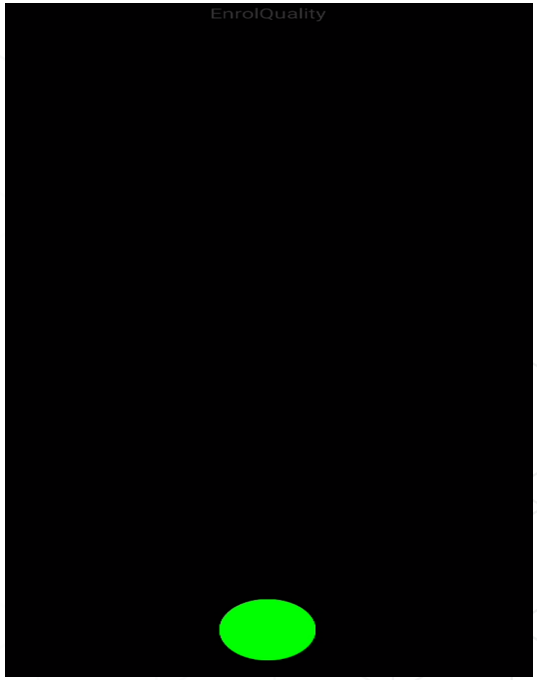
Never use factory mode to calibrate fingerprint!

我最近发现社区里面有许多人用工程模式（*#808#）校准指纹，结果全部都挂掉了，双清也无效。我的天哪，那是工程模式啊，能随便用吗？参数修改可不是闹着玩的。发生问题建议大家9008吧。。。

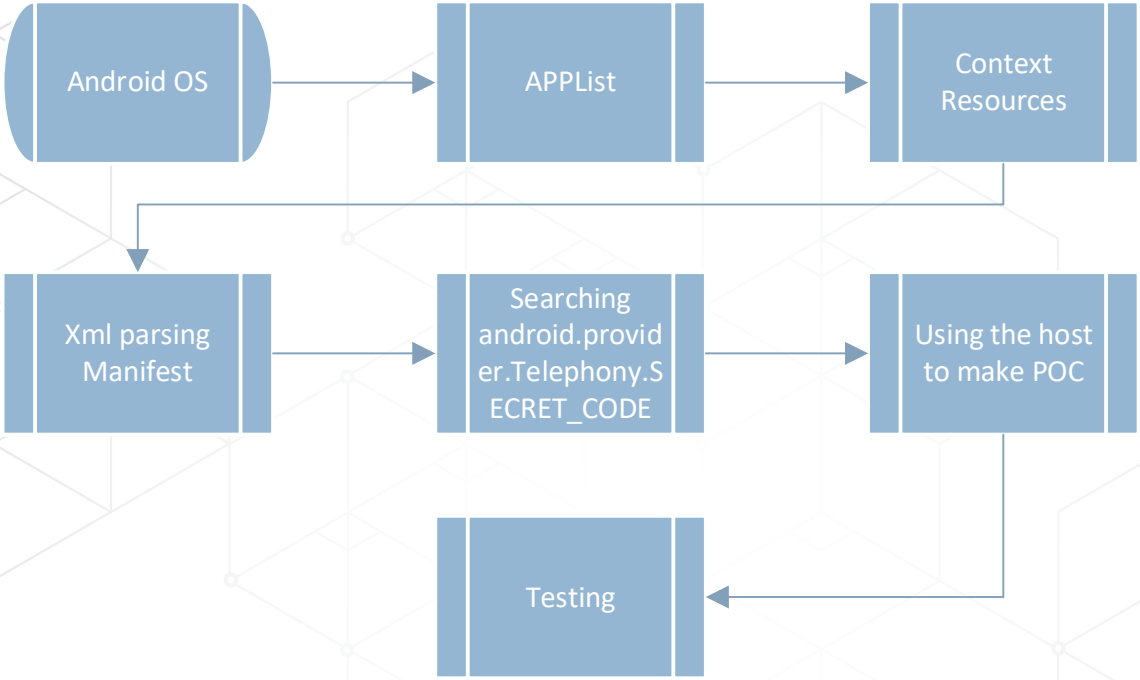
Vulnerabilities because of secret codes

We find that setting fingerprint in FFBM bypasses authentication and erases stored fingerprint





Fuzzing tool to find these vuls faster



Summary

01 **Consistency in access control system**

02 **Remove all “backdoors” as they will definitely be found**

03 **Remove all functionalities for testing only: EngineerMode APP**

Thank You!

HITB LOCKDOWN 002 livestream

Zhang Qing@ByteDance and Bai Guangdong@UQ