The Secret Codes Tell the Secrets

Zhang Qing@ByteDance and Bai Guangdong@UQ
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?

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

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

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

QUICK-POLL

How has your budget for security pry changed post-COVID-19?
- Increased
- Decreased
- Stayed the same

Do you have a vulnerability manager solution in place?
- Yes
- No

If yes, where are the biggest gaps in your current solution?
- Lack of vulnerability content
- Too many false positive alerts
- Lack of risk-based prioritization
- Lack of remediation guidance
- Poor reporting capabilities
- N/A - no current solution

Elliot Alderson
@ts0c131y

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...
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
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 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.

```java
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

How to create your own secret code?

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

```xml
<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");
  }
}
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.

```bash
$ 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.smal: com
./framework/EngineerMode.apk1a/AndroidManifest.xml:
./framework/oppo/engineermode/CustomizedReset.
./framework/PowerMonitor.apk1a/AndroidManifest.xml:
./framework/Settings.apk1a/AndroidManifest.xml: <action a
./framework/Telco.odex/com/android/server/telecom/DialerCodeReceiver.s
./framework/Telco.odex/com/android/server/telecom/TelecomSystemService.smal
```
# Why would secret codes breach Android’s permission system?

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

- **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
Vulnerabilities because of secret codes

Leak lock-screen PINs and Remove PINs
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

About engineering mode's fingerprint issues
2017-12-11 10:47 | 链接

杨海

Using 9527 in engineering mode to test fingerprint results in deletion, disabling device and using a new one, how to recover @小老鼠 @

번 호

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

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

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

Recent users have encountered the problem of fingerprint configuration being erased in factory mode, disabling fingerprint authentication. Customer service was unable to resolve the issue. It is recommended not to use factory mode for fingerprint calibration!
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

- Android OS
- APPList
- Context Resources
- Xml parsing Manifest
- Searching android.provider.Telephony.SECRET_CODE
- Using the host to make POC
- Testing
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

Based on our investigation, this EngineerMode app was not authored by @Qualcomm. There may be bits of QC source code there, and we believe others built upon a past testing app used to display device info. This EngineerMode app no longer resembles the original code we provided.
Thank You!

Zhang Qing@ByteDance and Bai Guangdong@UQ