Android Security

Daniel Cibrão, 070509149
Rui Gonçalves, 070509070
Contents

1. Security architecture
2. Installing applications
3. Interprocess communication
4. Authorizing operations
5. Problems and vulnerabilities
6. Practical work: PermissionExplorer
A quick overview
The Linux *kernel*

- Thoroughly used and updated for years
- Combines stability with functionality
- Provides Android some key features
  - User-based permission model
  - Process isolation
  - Well-defined IPC mechanisms
  - Modularity
- Multi-user operating system philosophy
Application sandbox

- Android is a single user system!
- The user model is applied to applications:
  - A different UID is assigned to each installed application
  - Each app runs in the context of that user
  - The app resources are owned by the user
Android filesystem

- Unix permissions model
- Each node in the filesystem has:
  - An owner (a single user)
  - A group (any number of users)
  - Three rwx tuples
Full filesystem encryption

- Introduced in Android 3.0
- Kernel level, using dm-crypt module
- AES128 with CBC and ESSIV:SHA256
  - AES with 128 bit key
  - Cipher Block Chaining
  - Encrypted Salt-Sector Initialization Vector, using SHA (256 bit) as hash function
Normal IV calculation

\[ \begin{align*}
P_0 & \rightarrow E \rightarrow C_0 \\
P_1 & \rightarrow E \rightarrow C_1 \\
P_2 & \rightarrow E \rightarrow C_2 \\
P_3 & \rightarrow E \rightarrow C_3 \\
\end{align*} \]
ESSIV IV calculation

Sector

H(C) → E → + → P₀ → E → + → P₁ → E → + → P₂ → E → + → P₃

Key

H → E → + → C₀ → E → + → C₁ → E → + → C₂ → E → + → C₃

Encryption box

...
Device rooting

- Process of obtaining root privileges (kernel level)
- Total control over the device! (for good and for bad...)
- Not usually permitted by stock ROMs
Device rooting

- Three "entry points" to consider when rooting
  - Bootloader and Recovery
  - ADB
  - System services
Contents

1. Security architecture
2. Installing applications
3. Interprocess communication
4. Authorizing operations
5. Problems and vulnerabilities
6. Practical work: PermissionExplorer
What do you need?

Application Signing

Keytool

Jarsigner

Coding...
Keytool

...is a key and certificate management utility
...works on any file-based keystore
Keytool Purpose?

- **Asymmetric key pair generation**
  - Digital Signature Algorithm (DSA)
  - Ron Rivest, Adi Shamir and Leonard Adleman (RSA)

- **Signature Algorithm**
  - DSA algorithm with SHA1 for hashing
  - RSA algorithm with MD5 for hashing

- Provide your own private key and certificate
Jarsigner

- Uses key and certificate in *keystore*
  - generate digital signatures for JAR files
  - verify digital signatures for JAR files

- Can only sign JAR files

- Signature Algorithm
  - DSA algorithm with SHA1 for hashing
  - RSA algorithm with MD5 for hashing

- What's the outcome of this signature?
What's the outcome of the signature

- The Jar file has two new files
  - A file with .SF extension
    - File name;
    - Name of the digest algorithm
    - Digest value
  - A file with .DSA extension
    - Certificate which authenticates the public key corresponding to the private key
You're good to go...

- Okay... Why did I sign my application?

- Google gives four main reason to do so:
  - Establish trust relationship between applications
  - Application upgrade
  - Application modularity
  - Code/data sharing through permissions

How can we benefit from?
**android:protectionLevel**
Characterizes the potential risk implied in the permission requesting it. The value can be set to one of the following:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;normal&quot;</td>
<td>The default value. A low-risk permission; it is safe for the system to execute the user's explicit app.</td>
</tr>
<tr>
<td>&quot;dangerous&quot;</td>
<td>A higher-risk permission. Because this type of permission is dangerous, these permissions may be taken to avoid.</td>
</tr>
<tr>
<td>&quot;signature&quot;</td>
<td>A permission that the signature of the certificate must match.</td>
</tr>
<tr>
<td>&quot;signatureOrSystem&quot;</td>
<td>A permission that the signature of the certificate must match the system. Please note that applications built into a system image do not require a certificate match.</td>
</tr>
</tbody>
</table>
Contents

1. Security architecture
2. Installing applications
3. Interprocess communication
4. Authorizing operations
5. Problems and vulnerabilities
6. Practical work: PermissionExplorer
Android Application Components

- **Content Provider**
  - Content URI
  - insert()
  - update()
  - delete()
  - query()

- **Alarm Clock**
  - 08:00
  - Mon, Tue, Wed, Thu, Fri

- **Triggered by Service**
  - Data passed: 3
  - original Data: Data passed from Activity to Service in startService

- **Android System**
  - Registers for certain Intents

- **Broadcast Receiver**
  - Gets notified when Intent happens
How Componentes Comunícate?

- **Intents**
  - a passive data structure holding an abstract description of an action to be performed

- **Binders**
  - Similar to a Remote Proceduce Call (RPC)

- **Messengers**
  - Message-based communication across processes
Contents

1. Security architecture
2. Installing applications
3. Interprocess communication
4. Authorizing operations
5. Problems and vulnerabilities
6. Practical work: PermissionExplorer
Android Permission Model

Application

Android OS Permissions Check

Personal Information

Sensitive Input Devices

Device Metadata

Types of Sensitive User Data
Android Permission APIs

- Personal Information
  - Phone Contacts
  - SMS/MMS
- Sensitive Data Inputs
  - Camera
  - GPS
  - Phone Inputs
- Device Metadata
  - Phone Identification
How to access this APIs?

- Declare the API permission to access in
  - AndroidManifest.xml

```xml
<uses-sdk android:minSdkVersion="7" />
<uses-permission android:name="android.permission.SEND_SMS" />
<uses-permission android:name="android.permission.CALL_PHONE" />
<uses-permission android:name="android.permission.READ_CONTACTS" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.CHANGE_WIFI_STATE" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission android:name="android.permission.WRITE_SETTINGS" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.VIBRATE" />
<uses-permission android:name="android.permission.READ_CALENDAR" />
<uses-permission android:name="android.permission.WRITE_CALENDAR" />
<uses-permission android:name="android.permission.CAMERA" />
<uses-permission android:name="android.permission.GET_TASKS" />
<uses-permission android:name="android.permission.GET_ACCOUNTS" />
```
Explicit Permissions

Do you want to install this application?

- **Services that cost you money**
  directly call phone numbers

- **Your location**
  coarse (network-based) location, fine (GPS) location

- **Network communication**
  full Internet access

- **Your accounts**
  Google Maps, manage the accounts list, use the authentication credentials of an account

- **Storage**
  modify/delete USB storage contents

[Options: Install, Cancel]
Explicit Permission

- When a user grants the permissions...
  - He will never need to grant them again
  - ...unless if he uninstall ou updates the app(...)

- Typically dont know/care about permissions
  - They press ok, because they want the app...
  - Unaware of the damage extent...

- Our goal is to increase the awareness of end users.
Permissions

- Secure or not secure, that it's the question....
Contents

1. Security architecture
2. Installing applications
3. Interprocess communication
4. Authorizing operations
5. Problems and vulnerabilities
6. Practical work: PermissionExplorer
Malicious usage of permissions
Malicious usage of permissions

- No way to know how are the granted permissions being used!
- Not a bug nor a design flaw; just a call for common sense
- Possibly the source of most Android attacks
Application delivery process

- What we see
Application delivery process

- What really happens
Application delivery process

- What really happens

  The permissions are approved here.

  But it's this service that makes the installing request!
Application delivery process

- The structure of message 2

```protobuf
message UnknownThing {
  optional fixed64 mGoogle = 12;
}

message InstallRequest {
  optional string appId = 1;
}

message RequestContext {
  required string authSubToken = 1; // authsub token for service 'android'
  required int32 unknown1 = 2; // always 0
  required int32 version = 3; // always 1002
  required string androidId = 4; // android id converted to hexadecimal
  optional string deviceAndSdkVersion = 5; // ro.product.device ':' ro.build.version.sdk
  optional string userLanguage = 6; // ro.product.locale.language
  optional string userCountry = 7; // ro.product.locale.region
  optional string operatorAlpha = 8; // gsm.operator.alpha
  optional string simOperatorAlpha = 9; // gsm.sim.operator.alpha
  optional string operatorNumeric = 10; // gsm.operator.numeric
  optional string simOperatorNumeric = 11; // sim.gsm.operator.numeric
  optional UnknownThing unknown12 = 12;
  optional string unknown13 = 13;
}

message Request {
  optional RequestContext context = 1;
  repeated group RequestGroup = 2 {
    optional InstallRequest installRequest = 10;
  }
}
```
Application delivery process

- The structure of message 2

```protobuf
message UnknownThing {
  optional fixed64 mgoogle = 12;
}
message InstallRequest {
  optional string appId = 1;
}
message RequestContext {
  required string authSubToken = 1; // authSub token for service 'android'
  required int32 unknown1 = 2; // always 0
  required int32 version = 3; // always 1002
  required string androidId = 4; // android id converted to hexadecimal
  optional string deviceAndSdkVersion = 5; // ro.product.device ':' ro.build.version.sdk
  optional string userLanguage = 6; // ro.product.locale.language
  optional string userCountry = 7; // ro.product.locale.region
  optional string operatorAlpha = 8; // gsm.operator.alpha
  optional string simOperatorAlpha = 9; // gsm.sim.operator.alpha
  optional string operatorNumeric = 10; // gsm.operator.numeric
  optional string simOperatorNumeric = 11; // sim.gsm.operator.numeric
  optional UnknownThing unknown12 = 12;
  optional string unknown13 = 13;
}
message Request {
  optional RequestContext context = 1;
  repeated group RequestGroup = 2 {
    optional InstallRequest installRequest = 10;
  }
}
```
Privilege escalation

- Explore a software flaw to gain unauthorized access to resources
- Attacks in Android must focus one of the three mentioned mechanisms
  - Bootloader
  - ADB
  - System services
Privilege escalation

- What's the problem with this piece of code?

```c
/* Code intended to run with elevated privileges */
do_stuff_as_privileged();

/* Drop privileges to unprivileged user */
setuid(uid);

/* Code intended to run with lower privileges */
do_stuff_as_unprivileged();
```
Privilege escalation

- From `setuid(2)` man page:

**ERRORS**

**EAGAIN** The `uid` does not match the current `uid` and `uid` brings process over its `RLIMIT_NPROC` resource limit.

**RLIMIT_NPROC**
The maximum number of processes (or, more precisely on Linux, threads) that can be created for the real user ID of the calling process. Upon encountering this limit, `fork(2)` fails with the error **EAGAIN**.
Privilege escalation

- From ADB source code (adb.c):

```c
(...)
setgroups(sizeof(groups)/sizeof(groups[0]), groups);

/* then switch user and group to "shell" */
setgid(AID_SHELL);
setuid(AID_SHELL);

/* set CAP_SYS_BOOT capability, so "adb reboot" will succeed */
header.version = _LINUX_CAPABILITY_VERSION;

(...)
```
Privilege escalation

- fork() up to RLIMIT_NPROC for “shell” user
- Kill adb, fork() again, adb fails setuid()
- The opened adb shell is now a root shell!
Privilege escalation

- Exploring known Linux vulnerabilities

(from Common Vulnerabilities and Exposures website)

**CVE-2009-1185**

udev before 1.4.1 does not verify whether a NETLINK message originates from kernel space, which allows local users to gain privileges by sending a NETLINK message from user space.
Other general things to consider

● CPU and RAM limits per application
● Web access
● SQL injection
● Connectivity
Contents

1. Security architecture
2. Installing applications
3. Interprocess communication
4. Authorizing operations
5. Problems and vulnerabilities
6. Practical work: PermissionExplorer
Questions?