

Sun Certified Mobile Application Developer (311-110)

Sun Microsystems, Inc.

Exam Notes

Sathya Srinivasan

02/19/2004

Table of Contents

CLDC Application Level Security.....	3
Untrusted MIDlet Suites.....	4
Trusted MIDlets Authorization.....	5
Trusted MIDlets Permission.....	5
Granting Permissions to trusted MIDlets.....	6
MIDlet Signing Process.....	6
Create a Certificate.....	6
Mention the Certificate in the JAD.....	6
Create a JAR signature and add to JAD.....	6
Verify the Signer Certificate upon installation.....	6
Verify the JAR Signature.....	7
Trusted MIDlet suite installation.....	8
PKI Authentication in MIDlets.....	9

Chapter 5 Security

Given a set of requirements, design and build applications given CLDC-specified application-level security, including the sandbox model.

CLDC Application Level Security

- Application-level security means that a Java application can access only the libraries, resources, and other components that the device the application is installed in and the Java environment that it runs in allows it to access.
- Sandbox model means that the device cannot use libraries or resources other than those defined by the configurations, profiles, and other classes supported by the device (say, optional packages like Bluetooth API).
- Requirements by CLDC Sandbox model
 - Classes must be verified to be valid by the CLDC pre-verifier and runtime verifier.
 - Application programmer cannot modify the class-loading mechanism. Custom class-loading mechanisms are not allowed in CLDC.
 - Only a closed set of pre-defined profiles (like MIDP, WMA) are available to the programmer (you cannot use third-party libraries willy-nilly unless supported by the device!).
 - Application programmer cannot write or use native code that is not part of the pre-defined packages (there is no `Runtime.exec()` method). Libraries containing native code cannot be downloaded into a device (unless it is a part of J2ME or that supplied by the device manufacturer).
- Application programmer cannot override the `java.*`, `javax.microedition.*`, or other profile-specific or manufacturer-specific packages. That is, you cannot spoof a Java package (say, you cannot have a class called `java.lang.String`, which you wrote, in your package).
- Application programmer cannot modify the class lookup order. This is equivalent of rearranging the jar files and paths present in the classpath variable (you typically cannot set the classpath of a CLDC!).
- By default, a Java application can load application classes only from its own JAR file.
- No end-to-end security mechanism (like encryption) is defined in CLDC. That responsibility is left to the profile implementations.

Identify correct and incorrect statements or examples about untrusted MIDlet suites.

Untrusted MIDlet Suites

- All MIDP 1.0 applications in a MIDP 2.0 environment are considered as untrusted MIDlets.
- An untrusted MIDlet suite is a MIDlet suite whose origin or JAR file integrity cannot be verified.
- The following APIs are accessible to untrusted MIDlets, **without explicit user confirmation**.

API	Description
javax.microedition.rms	Record Management Store APIs
javax.microedition.midlet	MIDlet lifecycle APIs
javax.microedition.lcdui	User Interface APIs
javax.microedition.lcdui.game	Game APIs
javax.microedition.media javax.microedition.media.control	Audio playback API

- The following APIs are accessible to untrusted MIDlets, **with explicit user confirmation**.

API	Description
javax.microedition.io.HttpConnection	http protocol handler
javax.microedition.io.HttpsConnection	https protocol handler

- If the user does not give permission, then a `SecurityException` will be thrown.

Explain trusted MIDlet suite security authorization and permissions, including the process for MIDlet suite signing.

Trusted MIDlets Authorization

- The concept of trusted MIDlets was introduced in MIDP 2.0.
- A MIDlet is considered trusted if it is signed and verified using the X.509 PKI mechanism.
- Authorization to a trusted MIDlet can happen in one or more of these stages.
 - A set of *Allowed* and *User* permissions granted to a MIDlet suite.
 - A set of permissions requested by the MIDlet suite in its `MIDlet-Permissions` and `MIDlet-Permissions-Opt` attributes.
 - A set of permissions for each protected API.
 - The user who may be asked to grant permissions.
- The MIDlet does not need to be aware of security permissions. It need to be only worried about whether a `SecurityException` is thrown or not while executing a method.
- How the user is asked for permission is implementation-dependent.
- The permission information in a device cannot be viewed or modified.
- A `SecurityException` cannot be thrown because of unimplemented functionality. It should only be used to throw errors because of lack of permission or authorization. For example, if a protocol is not implemented, it should throw a `ConnectionNotFoundException` and not `SecurityException`.

Trusted MIDlets Permission

- Permissions are typically specified at an API level.
- Permission names have a hierarchy similar to Java package names.
 - The permissions of an API must have the same prefix as that of the package name.
 - If a permission is for a specific class, then the permission name prefix must be same as that of the class name. For example, if you want to define permissions for a class `com.company.module.Foo`, then the permission name must begin with `com.company.module.Foo`, say, like `com.company.module.Foo.read`.
- If a permission is specified for an application, its permission names must be defined in its package or class-level API documentation.
- Permissions defined in the `MIDlet-Permissions` attribute are those permissions without which the MIDlet suite cannot operate.
- Permissions defined in the `MIDlet-Permissions-Opt` attribute are those permissions with or without which the MIDlet suite can operate, but probably with a reduced functionality (for example, a single-player game instead of a network-game).
- Permissions defined in these attributes are separated by a comma.
- *Allowed* permissions are those which do not need explicit user confirmation.
- *User* permissions are those which need explicit user confirmation.
- A permission can be a *User* permission or an *Allowed* permission, but not both.
- *Allowed* permission has the value "allow".
- *User* permission can have one of three values.
 - "blanket" means once the permission is defined by user, it remains that way for the life of the MIDlet application (from install to uninstall) till explicitly changed.
 - "session" means once the permission is defined by user, it remains that way for that run of the application (from start of MIDlet suite to stop).
 - "oneshot" means the permission has to be defined by user for every invocation of the API.

- The implementation should (but not a must) provide these options to the user. Option to deny is a must.

Granting Permissions to trusted MIDlets

- The `MIDlet-Permissions` and `MIDlet-Permissions-Opt` attribute values must be identical in the JAD file and the JAR file manifest. If not, the MIDlet suite must not be installed.
- If requested critical permissions are not known to the device, the MIDlet suite must not be installed.
- If requested permissions are not present in the protection domain but are marked critical, then the MIDlet suite must not be installed.
- If a permission requested is a *User* permission, then the user must explicitly provide permission.
- Permission granted to a midlet is
 - $(\text{Requested Permissions}) \cap (\text{User Permission} \cup \text{Allowed Permission})$.
- A `SecurityException` must be thrown if the user did not grant permission to an API.

MIDlet Signing Process

Signing a MIDlet suite involves the following steps.

Create a Certificate

- The device should have zero or more root certificates that can be used to validate the certificate associated with the MIDlet suite. It must at least support X.509 certificates.
- The signer (the application programmer) requests a certificate from a Certificate Authority (CA) using the standard signing procedure. The CA creates an RSA X.509 (v3) certificate and sends it back to the signer.

Mention the Certificate in the JAD

- The certificate path (except the root certificate portion) should be inserted into the application descriptor using base64 encoding standard (without line breaks).
 - The attribute name for the path is of the form `MIDlet-Certificate-<n>-<m>`
 - `n` = 1 for the first certification path and increases for extra certificates.
 - `m` = 1 for the signer certificate or and increases for intermediate certificates.

Create a JAR signature and add to JAD

- The signature of the JAR is created with the signer's private key.
- The signature is encoded using base64 algorithm and added as value to the attribute `MIDlet-Jar-RSA-SHA1` in the JAD file.

Verify the Signer Certificate upon installation

- The attribute values are checked for presence and verified.
- Appropriate certificate definition is matched with the root certificates installed in the device.

Verify the JAR Signature

- The public key of the verified certificate is used along with the signature from the `MIDlet-Jar-RSA-SHA1` attribute value and the SHA-1 digest of the JAR file to verify the signature.

Trusted MIDlet suite installation

Initial State	Installed?	Trusted?
JAD not present, JAR is unsigned or signed	Yes	No
JAD present, JAR is unsigned	Yes	No
JAR signed but no root certificate present in device	No	-
JAR signed, certificate expired	No	-
JAR signed, certificate rejected	No	-
JAR signed, certificate valid, but signature fails	No	-
JAR signed, certificate valid, signature verified	Yes	Yes

Explain requirements and process of using X.509 PKI authentication for MIDlet suites.

PKI Authentication in MIDlets

See [MIDlet signing process](#).