DUMPSBOSS.

Android Security Essentials

Android AND-402

Version Demo

Total Demo Questions: 15

Total Premium Questions: 151

Buy Premium PDF

https://dumpsboss.co

support@dumpsboss.co

support@dumpsboss.co
dumpsboss.co

QUESTION NO: 1

Which flag should be used when creating a PendingIntent so that the recipient cannot modify the wrapped Intent?

A. FLAG_IMMUTABLE

B. FLAG_MUTABLE

C. FLAG_ONE_SHOT

D. FLAG_UPDATE_CURRENT

ANSWER: A

Explanation:

FLAG_IMMUTABLE is correct. A PendingIntent gives another application or system component authority to perform an operation using the identity and permissions of the application that created it. When `PendingIntent.FLAG_IMMUTABLE` is specified, the recipient cannot alter the underlying Intent when sending the PendingIntent.

This is an important security control when no legitimate recipient needs to add, replace, or modify Intent data, extras, actions, or target components. Android requires developers targeting modern Android versions to explicitly specify either immutable or mutable behavior for most PendingIntent objects. `FLAG_MUTABLE` should be used only when modification is genuinely required. See the PendingIntent.FLAG_IMMUTABLE reference and Android guidance on PendingIntent security risks.

QUESTION NO: 2

Which of the following is NOT true when setting attribute “ManageSpaceActivity” to true? (Choose two)

A. It is placed inside application tag.

B. Its value is the name of package inside which the activity that manages space exits.

C. It is placed inside the activity tag

D. The activity should also be declared with an activity tag.

ANSWER: B C

Explanation:

The statements “Its value is the name of package inside which the activity that manages space exits.” and “It is placed inside the activity tag” are the two statements that are not true. The `android:manageSpaceActivity` manifest attribute belongs on the application element, not on an individual activity declaration. Its purpose is to identify an activity that the system can launch to let the user manage the app’s storage usage. The attribute does not take a Boolean value such as `true`; instead, it takes the name of the activity class. Android documentation specifies that this value must be the fully qualified name of an `Activity` subclass, such as `com.example.app.ManageStorageActivity`, or a name abbreviated relative to the application package. Providing only a package name does not identify the activity Android must start. Because the referenced component is an actual activity, it must also be declared separately using an `<activity>` element in the manifest. This structure allows Android to locate the storage-management screen while preserving the activity’s normal manifest configuration, including its label, permissions, and exported behavior where applicable. See the Android manifest reference for android:manageSpaceActivity and the documentation for the activity manifest element.

QUESTION NO: 3

What is the recommended use of BaseColumns._ID in SQLite database?

A. It should always be used to maintain unique ID.

B. It should be used to include unique ID when content provider is being used.

C. It is deprecated now.

D. All are correct.

ANSWER: B

Explanation:

It should be used to include unique ID when content provider is being used. `BaseColumns._ID` defines Android's standard column name, `"_id"`, for a unique integral row identifier. A content provider commonly exposes database rows through a `Cursor`, and using this conventional column makes the provider's result sets interoperable with Android framework components that need to identify individual rows reliably. In particular, cursor-based UI components and adapters have traditionally expected a column named `_id` to serve as the stable item identifier. Defining a table contract that implements `BaseColumns`, then creating an `_id` column as the primary key, provides a consistent schema shared by the database layer, the provider, and clients querying it. This convention also supports item-specific provider URIs, where a caller accesses one row by its unique identifier. Android's training guidance explicitly recommends implementing `BaseColumns` for a table contract so the table can inherit the `_ID` field, while the API reference defines it as the unique ID field name used by many Android classes. See the Android SQLite data storage guide and the BaseColumns API reference.

QUESTION NO: 4

Which of the following is NOT true about the "Process" attribute of the tag?

A. Its default value is same as the package name and is set by the system itself.

B. It must be included in the manifest tag with the value of package name application.

C. If the name assigned to this attribute begins with a colon (':'), a new process, private to the application, iscreated when it's needed.

D. If the process name begins with a lowercase character, a global process of that name is created.

ANSWER: B

Explanation:

`android:process` is an optional attribute of the `<application>` manifest element; it is not required to be included in the `<manifest>` element, nor must it be explicitly assigned the application package name. When no process is specified, Android uses the application's default process, whose name is the application package name. Developers specify `android:process` only when they need the application or particular components to execute in a different process. This can be useful for process isolation or for separating work with different lifecycle and memory characteristics, but it also introduces interprocess communication and resource-management considerations. The value belongs on the application element (or, where appropriate, an individual component element), rather than being a mandatory manifest-level declaration. Therefore, the statement that it "must be included in the manifest tag with the value of package name application" is the statement that is not true. Android's manifest documentation describes the default process behavior and the optional `android:process` attribute in the application element reference. Additional process behavior is covered in the Processes and app lifecycle guide.

QUESTION NO: 5

Which of the following in NOT true about setting the attribute installLocation to value "internalOnly"? (Choose

two)

A. The default behavior of installation of application is same as that of giving value "internalOnly" to the installLocation attribute.

B. The application is installed on the internal device storage only.

C. If there is no space on the internal memory then the application gets installed on the external storage and moves back to internal storage as soon as the space is available. This is default behavior.

D. The application can be moved to external device storage if required.

ANSWER: C D

Explanation:

Setting `android:installLocation` to `"internalOnly"` directs Android to install the application only in the device's internal storage. This setting is also the default installation behavior when the attribute is omitted. As a result, an application using this value is not eligible to be installed on external/shared storage as a fallback when internal storage is full, nor is it eligible for the user or system to move it to external storage later. Therefore, the statements describing automatic installation on external storage when internal space is unavailable and the ability to move the application to external device storage are the two statements that are not true for `internalOnly`. Android's manifest documentation defines `internalOnly` as installing the application only on internal device storage and identifies it as the default value. See the Android manifest installLocation attribute documentation and the Android guidance on app install location.

QUESTION NO: 6

Which of the following classes is used to find out the DRM plug-in available in a device?

A. DrmManagerClient

B. DrmServer

C. DrmConstraints

D. DrmRights

ANSWER: A

Explanation:

`DrmManagerClient` is correct because it is the Android DRM framework client class that an application uses to interact with DRM services and installed DRM engines. After constructing a `DrmManagerClient` with an Android `Context`, the application can call `getAvailableDrmEngines()`. This method returns the UUID values for the DRM plug-ins (DRM engines) that are available on that device, allowing the app to discover the DRM capabilities exposed through the framework. The class also provides operations for obtaining DRM constraints, acquiring rights, processing DRM content, and handling DRM metadata, but its engine-discovery method is specifically what makes it appropriate for this question. Android's API reference documents `getAvailableDrmEngines()` as returning the unique identifiers of available DRM engines and identifies `DrmManagerClient` as the client interface for the DRM framework. Although this legacy DRM API has been deprecated in modern Android releases, it is the API represented by the class names in this question. See the Android DrmManagerClient reference and its getAvailableDrmEngines() documentation.

QUESTION NO: 7

Which of the following is NOT true about sharedUserId attribute?

A. It is placed inside manifest tag.

B. By default, Android assigns each application its own unique user ID.

C. Applications can share the same user ID.

D. It is placed in the activity tag that needs to share the data with other application.

ANSWER: D

Explanation:

“It is placed in the activity tag that needs to share the data with other application.” is the statement that is not true. The `android:sharedUserId` attribute, when used, is an attribute of the root `<manifest>` element in an app’s Android manifest. It defines an application-level identity; it is not configured separately on an `<activity>` component. Consequently, all package-level behavior associated with a shared UID applies to the application package rather than being enabled only for one activity. Historically, applications signed with the same certificate could declare the same shared user ID and run under the same Linux user ID, enabling access to each other’s app-private data and shared permissions under the applicable platform rules. Android’s manifest documentation lists `android:sharedUserId` under the `<manifest>` attributes and notes that the feature has been deprecated since Android 10. New applications should generally avoid it, because it creates installation, update, and compatibility constraints. The Android security model instead normally isolates each app with its own UID. See the official Android manifest element documentation and Android security guidance on shared user IDs.

QUESTION NO: 8

Which of the following is NOT true about the “killAfterRestore” attribute of the <application> tag?

A. Normally, third-party applications will not need to use this attribute.

B. It means that whether the application should be terminated after its settings have been restored during anapplication restore operation.

C. The default value is true.

D. The true value of this attribute means that the application will be terminated once its settings are restoredduring a full-system restore

E. The default value is false.

ANSWER: E

Explanation:

The statement “The default value is false.” is not true. Android defines `android:killAfterRestore` on the `<application>` element as a control for whether the app process is terminated after its settings have been restored during a full-system restore. Its documented default is `true`, meaning Android normally terminates the application after restoration so that it can start again with the restored data and settings in effect. This behavior helps avoid leaving a running process in a state where it may still hold old in-memory configuration or data while its persistent application state has been replaced by restored content. An app can explicitly set the attribute to `false` only when it is prepared to continue running safely after restore and does not require the normal restart behavior. Therefore, describing `false` as the default contradicts the Android manifest documentation. See the Android reference for the `<application>` manifest element and its `killAfterRestore` attribute: Android Developers: <application>. Related platform guidance on app backup and restore is available at Android Developers: Back up user data with Auto Backup.

QUESTION NO: 9

Which of the following is correct about Android permission PROCESS_OUTGOING_CALLS? (Choose two)

A. Allows an application to access call logs.

B. Allows an application to monitor or abort outgoing calls.

C. Allows an application to record outgoing calls.

D. Allows an application to divert incoming calls.

E. Allows an application to modify an outgoing call’s dialed number before the call is placed.

ANSWER: B E

Explanation:

"Allows an application to monitor or abort outgoing calls." is correct because `PROCESS_OUTGOING_CALLS` historically enabled an application to receive information about a number being dialed and intervene before the outgoing call was placed. This capability supported call-screening-style behavior, including stopping an outgoing call. "Allows an application to modify an outgoing call's dialed number before the call is placed." is also correct because the permission's documented behavior included the ability to redirect the dialed number to another number. In older Android versions, this was commonly implemented through the outgoing-call broadcast flow. Android has since deprecated `PROCESS_OUTGOING_CALLS` (API level 29), and modern applications should use the role- and telecom-based APIs intended for call screening and redirection rather than relying on this legacy permission. The permission should therefore be understood as an outgoing-call monitoring and intervention permission, not as a general-purpose audio-recording or call-log permission. See the Android permission reference for PROCESS_OUTGOING_CALLS and Android's guidance on CallRedirectionService.

QUESTION NO: 10

What does the following line of code do?

FileOutputStream fOut = openFileOutput("MyFile.txt", MODE_WORLD_READABLE);

A. The file MyFile.txt will be created in the /data// files/ directory.

B. The file MyFile.txt will be created in the /data/data// files/ directory.

C. None are correct

D. The file MyFile.txt will be created in the /data/data// directory.

ANSWER: C

Explanation:

None are correct is the correct current answer because `MODE_WORLD_READABLE` is not supported on modern Android. Although `openFileOutput()` normally creates or opens a file in the calling app's private internal `files` directory, Android deprecated the world-readable mode because it can expose sensitive application data to other apps. Beginning with Android 7.0 (API level 24), attempting to create a file with `MODE_WORLD_READABLE` causes a `SecurityException`. Consequently, this statement does not successfully create `MyFile.txt` at either of the listed filesystem locations on supported modern platform versions. Android's documentation explicitly identifies the mode as deprecated and states that it throws a security exception as of Android N. Secure Android applications should retain the default app-private file permissions and share content only through controlled mechanisms such as a content provider and URI permissions when cross-app access is required. See the Android Context.MODE_WORLD_READABLE reference and the openFileOutput() API reference.

QUESTION NO: 11

Which of the following is correct about BiometricPrompt? (Choose two)

A. It displays a system-managed authentication prompt.

B. A CryptoObject can bind successful authentication to a cryptographic operation.

C. It gives the application access to the user's raw biometric data.

D. It requires the application to implement its own fingerprint matching algorithm.

ANSWER: A B

Explanation:

`BiometricPrompt` provides a system-managed authentication prompt for biometric authentication and, where configured, device credentials. Using the system prompt helps applications use a consistent authentication flow while leaving biometric enrollment, matching, and biometric-template management under Android system control.

A `BiometricPrompt.CryptoObject` can be associated with a cryptographic operation, such as a Cipher, Signature, or Mac. This allows a cryptographic key operation to proceed only after successful user authentication when the key has been configured with the appropriate authentication requirements. The application receives an authentication result, not the user's biometric template or raw fingerprint data. See the Android biometric authentication documentation and the BiometricPrompt.CryptoObject reference.

QUESTION NO: 12

Signature permission is also known as: (Choose two)

A. System Permissions or Level three permission.

B. Special situation permissions.

C. All are correct.

D. None are correct.

ANSWER: A B

Explanation:

A signature permission is a certificate-based permission: Android grants it only when the requesting application is signed with the same signing certificate as the application or component that declared the permission. In older Android security-training terminology, this was commonly classified as a level three permission, following normal and dangerous permission levels. It was also associated with system or special-situation permissions because platform components and tightly coupled applications use signing identity to establish a trusted relationship without presenting a runtime consent prompt. This model is especially useful when an application exposes a sensitive service, provider, broadcast receiver, or activity intended only for applications produced by the same organization. Android's current documentation describes the core rule directly: matching signing certificates are required for signature-level access. It also notes the related privileged-permission model used for selected preinstalled applications, reinforcing why system-oriented terminology appears in historical Android security materials. See the Android permission protection-level reference at Android Developers: permission protection levels and the platform security overview at Android Open Source Project: permissions.

QUESTION NO: 13

Which of the following is NOT true about "allowbackup" attribute of application tag? (Choose two)

A. Its default value is true.

B. If set to false then no backup or restore of the application will ever be performed.

C. If full system backup is made the application's data is saved via adb. This is in system's control and occurs in all cases whether the allowbackup has value true or false.

D. The default value of this attribute is false.

ANSWER: C D

Explanation:

The statements asserting that a full system backup saves application data through adb regardless of the `allowBackup` value, and that the attribute defaults to `false`, are the two claims that are not true. The Android application manifest documentation defines `android:allowBackup` as the setting that controls whether an application can participate in the backup and restore infrastructure. Its documented default is `true`, so an application that does not explicitly declare the attribute is, subject to platform backup behavior and applicable backup rules, eligible for backup and restore. The attribute is specifically intended to let developers opt an application out of backup participation; it is not a setting that the backup system simply ignores whenever a full backup is performed. Historically, adb backup behavior also honored this manifest setting, and adb backup/restore has since been deprecated and removed from modern Android platform workflows. Current Android versions additionally distinguish cloud backup and device-to-device transfer in some circumstances, but this does not make

the unconditional claim that app data is backed up regardless of the manifest setting correct. See the Android manifest reference for android:allowBackup and Android's Auto Backup documentation.

QUESTION NO: 14

In DRM system, Content and Rights object are stored inside?

A. Content server

B. Rights server

C. Storage Device

D. DRM agent

ANSWER: C

Explanation:

Storage Device is correct because a DRM system must retain both the protected content object and the associated rights object locally so that content can be accessed later, including when the device is offline. The content object contains the encrypted or otherwise protected media, while the rights object contains the permissions and constraints that govern use, such as whether playback is allowed, an expiration time, or a play-count limit. These objects are persisted in device storage; the DRM agent then retrieves and evaluates the stored rights when an application requests access to the stored protected content. In practice, the DRM implementation may use protected or secure storage for sensitive keys, licenses, and DRM metadata, but the essential architectural location for retaining downloaded content and rights is the device's storage. Android's DRM framework similarly provides applications with access to DRM-protected media and relies on DRM components to manage licenses and enforce the rights associated with protected content. See the Android Digital rights management documentation and the Android Open Source Project overview of DRM in Android.

QUESTION NO: 15

Which of the following are helper classes provided by Android to backup data to Google server? (Choose two)

A. FileBackupHelper

B. DatabaseBackupHelper

C. SharedPreferencesBackupHelper

D. MapBackupHelper

ANSWER: A C

Explanation:

`FileBackupHelper` and `SharedPreferencesBackupHelper` are Android framework helper classes used with the legacy key/value backup framework. An app can extend `BackupAgentHelper`, register one or more helpers, and let the backup framework manage the backup and restore lifecycle through the device's configured backup transport, which can include cloud backup services.

`FileBackupHelper` is intended for backing up specified files from the application's private storage. The developer supplies the files to include, and the helper handles creating backup entities and restoring those files when the app's data is restored. Android documents this helper in the FileBackupHelper API reference.

`SharedPreferencesBackupHelper` serves the corresponding role for an app's named shared-preference files. It enables preferences stored through Android's shared-preferences mechanism to participate in key/value backup and restoration, without requiring the app to implement the lower-level entity processing itself. Its supported usage and constructor are documented in the SharedPreferencesBackupHelper API reference. Thus, these two classes are the provided helpers that match the requested file and preference data backup functions.