

HUAWEI Whitelist Technical Details

Prepared by Mark, 28 July 2021

Source code of Huawei's OS (Version 10)

```
273 public static boolean checkCallingOrSelfReadDeviceIdentifiers(Context context, String callingPackage, String message) {
274     return checkCallingOrSelfReadDeviceIdentifiers(context, -1, callingPackage, message);
275 }

296 public static boolean checkCallingOrSelfReadDeviceIdentifiers(Context context, int subId, String callingPackage, String message) {
297     return checkReadDeviceIdentifiers(context, TELEPHONY_SUPPLIER, subId, Binder.getCallingPid(), Binder.getCallingUid(), callingPackage, message);
298 }

318 public static boolean checkCallingOrSelfReadSubscriberIdentifiers(Context context, int subId, String callingPackage, String message) {
319     return checkReadDeviceIdentifiers(context, TELEPHONY_SUPPLIER, subId, Binder.getCallingPid(), Binder.getCallingUid(), callingPackage, message);
320 }

@VisibleForTesting
334 public static boolean checkReadDeviceIdentifiers(Context context, Supplier<ITelephony> telephonySupplier, int subId, int pid, int uid, String callingPackage, String message) {
335     int appId = UserHandle.getAppId(uid);
336     if (appId == 1000 || appId == 0 || context.checkPermission(Manifest.permission.READ_PRIVILEGED_PHONE_STATE, pid, uid) == 0 || checkCarrierPrivilegeForAnySubId(context, telephonySupplier,
336         return true;
337     }
338     if (callingPackage != null) {
339         long token = Binder.clearCallingIdentity();
340         try {
341             if (((AppOpsManager) context.getSystemService(Context.APP_OPS_SERVICE)).noteOpNoThrow(AppOpsManager.OPSTR_READ_DEVICE_IDENTIFIERS, uid, callingPackage) == 0) {
342                 return true;
343             }
344             Binder.restoreCallingIdentity(token);
345             DevicePolicyManager devicePolicyManager = (DevicePolicyManager) context.getSystemService(Context.DEVICE_POLICY_SERVICE);
346             if (devicePolicyManager != null && devicePolicyManager.checkDeviceIdentifierAccess(callingPackage, pid, uid)) {
347                 return true;
348             }
349         } finally {
350             Binder.restoreCallingIdentity(token);
351         }
352     }
353     if (!IS_DOMESTIC_VERSION || !isPackageNameInDeviceIdWhiteList(callingPackage)) {
354         return reportAccessDeniedToReadIdentifiers(context, subId, pid, uid, callingPackage, message);
355     }
356     return true;
357 }
```

HUAWEI Whitelist Technical Details

Source code of AOSP 10.0.0_r1

https://cs.android.com/android/platform/superproject/+android-10.0.0_r1:frameworks/base/telephony/java/com/android/internal/telephony/TelephonyPermissions.java

Android Code Search

Search for code or files

platform/superproject > android-10.0.0_r1 > frameworks/base/telephony/java/com/android/internal/telephony/TelephonyPermissions.java

FilesOutline

telephony

CarrierInfo.java

CarrierInfoAsyncQuery.java

CarrierAppUtils.java

CellNetworkScanResult.aidl

CellNetworkScanResult.java

DctConstants.java

EncodeException.java

ExponentialBackoff.java

GsmAlphabet.java

IApnSourceService.aidl

ICarrierConfigLoader.aidl

IntegerConsumer.aidl

IMms.aidl

INumberVerificationCallback.aidl

OnSubscriptionsChangedListener.aidl

IOms.aidl

IPhoneStateListener.aidl

IPhoneSubInfo.aidl

ISetOpportunisticDataCallback.aidl

ISms.aidl

ISmsImplBase.java

ISub.aidl

ITelephony.aidl

ITelephonyRegistry.aidl

IUpdateAvailableNetworksCallback.aidl

IWapPushManager.aidl

IccCardConstants.java

NetworkScanResult.java

OperatorInfo.aidl

OperatorInfo.java

PhoneConstantConversions.java

PhoneConstants.java

RILConstants.java

Sms7BitEncodingTranslator.java

SmsAddress.java

SmsApplication.java

SmsCbCmasInfo.java

SmsCbEtwsInfo.java

SmsCbLocation.java

SmsCbMessage.java

SmsConstants.java

SmsHeader.java

SmsMessageBase.java

SmsRawData.aidl

SmsRawData.java

TelephonyIntents.java

TelephonyPermissions.java

TelephonyProperties.java

package.html

TelephonyPermissions.java

```
299
300 /**
301  * Checks whether the app with the given pid/uid can read device identifiers.
302  *
303  * @returns true if the caller has the READ_PRIVILEGED_PHONE_STATE permission or the calling
304  * package passes a DevicePolicyManager Device Owner / Profile Owner device identifier access
305  * check.
306  */
307 @VisibleForTesting
308 public static boolean checkReadDeviceIdentifiers(Context context,
309     Supplier<ITelephony> telephonySupplier, int subId, int pid, int uid,
310     String callingPackage, String message) {
311     // Allow system and root access to the device identifiers.
312     final int appId = UserHandle.getAppId(uid);
313     if (appId == Process.SYSTEM_UID || appId == Process.ROOT_UID) {
314         return true;
315     }
316     // Allow access to packages that have the READ_PRIVILEGED_PHONE_STATE permission.
317     if (context.checkPermission(android.Manifest.permission.READ_PRIVILEGED_PHONE_STATE, pid,
318         uid) == PackageManager.PERMISSION_GRANTED) {
319         return true;
320     }
321     // If the calling package has carrier privileges for any subscription then allow access.
322     if (checkCarrierPrivilegeForAnySubId(context, telephonySupplier, uid)) {
323         return true;
324     }
325     // if the calling package is not null then perform the DevicePolicyManager device /
326     // profile owner and Appop checks.
327     if (callingPackage != null) {
328         // Allow access to an app that has been granted the READ_DEVICE_IDENTIFIERS app op.
329         long token = Binder.clearCallingIdentity();
330         AppOpsManager appOpsManager = (AppOpsManager) context.getSystemService(
331             Context.APP_OPS_SERVICE);
332         try {
333             if (appOpsManager.noteOpNoThrow(AppOpsManager.OPSTR_READ_DEVICE_IDENTIFIERS, uid,
334                 callingPackage) == AppOpsManager.MODE_ALLOWED) {
335                 return true;
336             }
337         } finally {
338             Binder.restoreCallingIdentity(token);
339         }
340         // Allow access to a device / profile owner app.
341         DevicePolicyManager devicePolicyManager =
342             (DevicePolicyManager) context.getSystemService(
343                 Context.DEVICE_POLICY_SERVICE);
344         if (devicePolicyManager != null && devicePolicyManager.checkDeviceIdentifierAccess(
345             callingPackage, pid, uid)) {
346             return true;
347         }
348     }
349     return reportAccessDeniedToReadIdentifiers(context, subId, pid, uid, callingPackage,
350         message);
351 }
352
353 /**
354  * Reports a failure when the app with the given pid/uid cannot access the requested identifier.
355  *
356  * @returns false if the caller is targeting pre-Q and does have the READ_PHONE_STATE
357  * permission or carrier privileges.
358  * @throws SecurityException if the caller does not meet any of the requirements for the
359  * requested identifier and is targeting Q or is targeting pre-Q
360  * and does not have the READ_PHONE_STATE permission or carrier
```