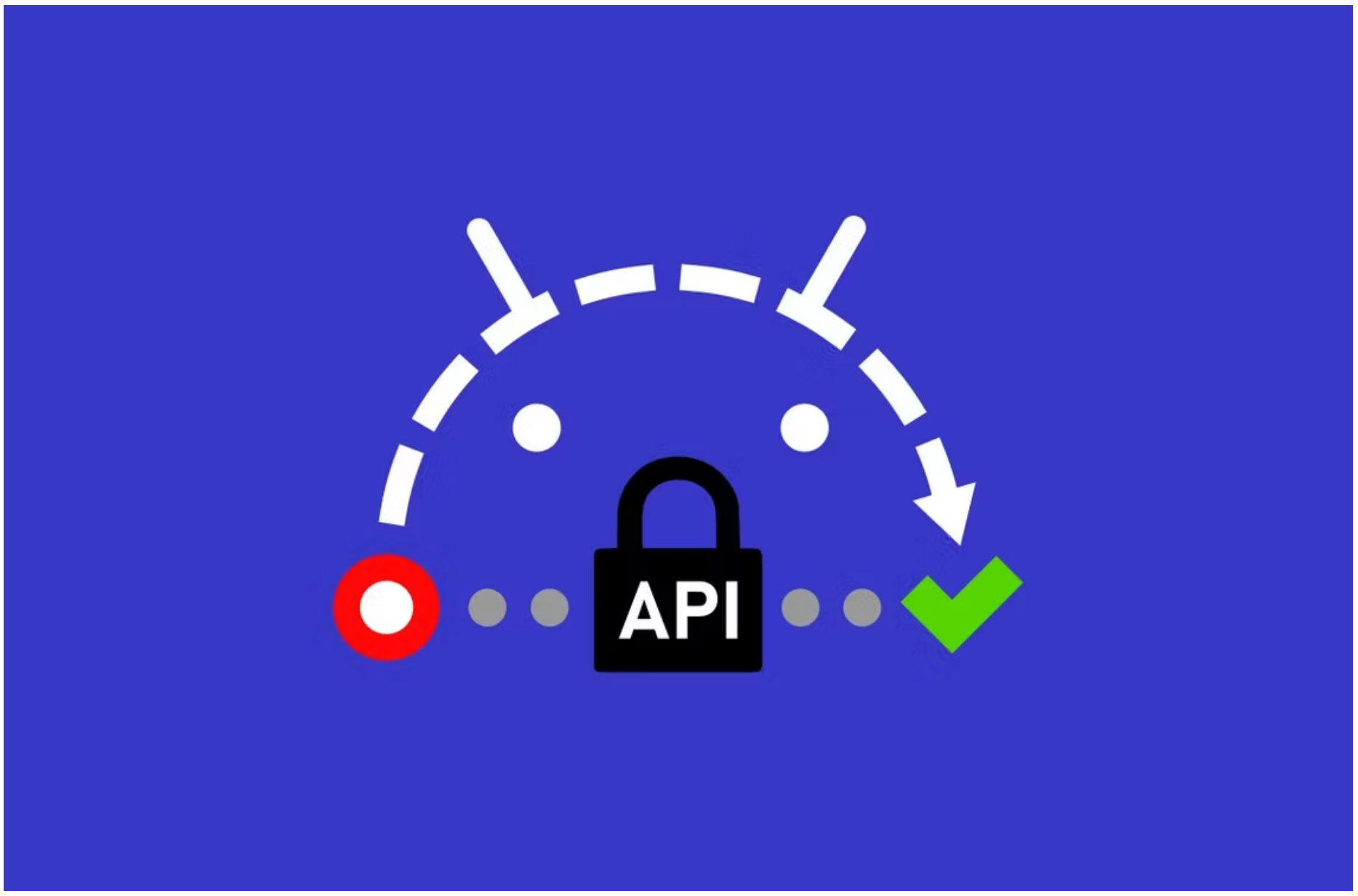
Advanced Android Development: How to bypass the hidden API blacklist

Google introduced restrictions in Android 9 on which APIs can be accessed by developers. Here's how to bypass those hidden API restrictions.

BY **ZACHARY WANDER** PUBLISHED OCT 8, 2021



Way back in 2018, Google released Android Pie. Among the UI changes and new features, there were also some developer-side changes. Those changes included new APIs, bugfixes for existing APIs, and also restrictions on access to hidden APIs.

Luckily though, there are ways to get around those restrictions. Before I get into how to bypass the restrictions, I should explain a bit about what hidden APIs are, why they were restricted in the first place, and why you might want to access them.

What are hidden APIs?

Hidden APIs are the APIs in Android that app developers normally can't see. If you take a look at AOSP's code, you'll see a whole bunch of classes, variables, and methods that have a `@hide` annotation inside a comment block above them.

This annotation instructs whatever tool Google uses when compiling the SDK to exclude the item under it. That SDK is then distributed to developers inside the SDKs downloaded through Android Studio. Unless you use a modified SDK, Android Studio will think any of those hidden items just don't exist. If you try to use one directly, it will show it in red and refuse to compile.

Why are APIs hidden?

There are a lot of reasons why an API might be hidden. Some things are only meant to be used by internal or system apps and won't work if used by a third-party app. Others are experimental or unstable, and might be removed or changed in the future. Some are even just APIs Google just doesn't want to apple the normal deprecation cycle to if they're ever removed.

Why use hidden APIs?

While the standard Android SDK has a *lot* in it, sometimes it's not enough. Sometimes there's something you want to do that already exists in Android, but just isn't publicly exposed.

For instance, a lot of the apps I make, including SystemUI Tuner and Lockscreen Widgets, make use of a bunch of different hidden APIs. SystemUI Tuner needs to access some to properly track, change, and reset options. Lockscreen Widgets uses some to show the wallpaper under it, among other things.

Most developers don't need to access hidden APIs, but sometimes they can be pretty useful.

How are hidden APIs restricted?

With the release of Android 9 (Pie), Google introduced the hidden API blacklist. Not every hidden API was included, and there were different levels of lists. Hidden APIs on the whitelist could be accessed by anyone. Hidden APIs on the light-greylist could be accessed by any app, but might be inaccessible in future versions of Android. Anything on the dark-greylist could only be accessed by apps targeting API levels before Pie (i.e., before API level 28). Apps targeting Pie and later would be denied access. Finally, hidden APIs on the blacklist couldn't be accessed by any non-system (or non-whitelisted) app, no matter the target API.

Android 10 changed how the lists were organized, and simplified them slightly, but the idea stayed the same. Certain hidden APIs could be accessed by apps while others were blocked. Android 11 strengthened the access detection to block a bypass used for Pie and 10.

In all Android versions, any time a third-party app attempts to access a blacklisted hidden API, Android will throw the appropriate "not found" error.

How to bypass the hidden API blacklist

There are actually quite a few ways to get past the hidden API blacklist. Depending on your needs, you can choose ones that work for all Android versions, ones that work for only Android 9 and 10, ones that use native C++ code, and ones that are fully Java-based. There's even a development-only workaround using ADB.

ADB Workaround

If your device is running Android Pie, run the following two ADB commands to enable hidden API access.

```
adb shell settings put global hidden_api_policy_pre_p_apps 1
adb shell settings put global hidden_api_policy_p_apps 1
```

If your device is running Android 10 or later, run the following ADB command to enable hidden API access.

```
adb shell settings put global hidden_api_policy 1
```

To revert to the default behavior, just replace `put` with `delete` and remove the `1`.

Obviously, these commands aren't exactly useful for a production app. I can tell you firsthand that properly instructing users on how to use ADB is incredibly difficult. But they can be useful if you need to update an old app to comply with the new restrictions.

Native/JNI workaround

There are two ways you can bypass the hidden API blacklist using JNI in your Android app. One works for Android 9 and 10, and the other works for Android 9 and later.

Android 9 and 10

If you already have a native portion of your app, this will be easy to implement. Just use the `JNI_OnLoad()` function.

static art::Runtime* runtime = nullptr;

```
extern "C" jint JNI_OnLoad(JavaVM *vm, void *reserved) {
      ...
      runtime = reinterpret_cast<art::JavaVMExt*>(vm)->GetRuntime();
      runtime->SetHiddenApiEnforcementPolicy(art::hiddenapi::EnforcementPolicy::kNoChecks);
      ...
}
```

Be aware that this method only works on Android 9 and 10.

Android 9 and later

For any version of Android, you have your choice of two libraries to bypass the hidden API restriction: FreeReflection and RestrictionBypass.

Both are easy to implement and use.

Link copied to clipboard

To implement FreeReflection, add the dependency to your module-level build.gradle.

```
implementation 'me.weishu:free_reflection:3.0.1'
```

Then override `attachBaseContext()` in your Application class.

```
@Override
protected void attachBaseContext(Context base) {
      super.attachBaseContext(base);
      Reflection.unseal(base);
}
```

If you don't have an Application class, you can add it pretty easily. Create a new class that extends `Application` and then point to it in your AndroidManifest.xml.

Example:

```
public class App extends Application {
      ...
      @Override
      protected void attachBaseContext(Context base) {
            super.attachBaseContext(base);

            Reflection.unseal(base);
      }
}
```

```
<manifest>
      ...
      <application
            ...
            name=".App">
            ...
      </application>
</manifest>
```

To implement RestrictionBypass, add the JitPack repository to your project-level build.gradle.

```
allprojects {
      repositories {
            ...
            maven { url "https://jitpack.io" }
      }
}
```

Then add the dependency to your module-level build.gradle.

```
implementation 'com.github.ChickenHook:RestrictionBypass:2.2'
```

And that's it. This library automatically removes the blacklist restrictions.

Java workaround

While the JNI solutions are effective, there are times you might not want to use native code. If you aren't already doing things in C++, it can add unnecessary size, along with platform restrictions, to your app. Luckily, there are ways to bypass the hidden API blacklist using only Java.

Android 9 and 10

In Android 9 and 10, you can use what can be called double-reflection or meta-reflection to bypass the hidden API blacklist. Because the system only checks what third-party apps are calling, double-reflection tricks it into thinking the system is making the hidden API calls.

This trick can be used to call a method to give your app hidden API exemptions, aptly named `setHiddenApiExemptions()`. Simply add the following code somewhere early on in your app's lifecycle (such as Application's `onCreate()` method), and it'll handle bypassing the blacklist.

```
lass.class.getDeclaredMethod("forName", String.class);
Method getDeclaredMethod = Class.class.getDeclaredMethod("getDeclaredMethod", String.class, Class[].class);

Class vmRuntimeClass = (Class) forName.invoke(null, "dalvik.system.VMRuntime");
Method getRuntime = (Method) getDeclaredMethod.invoke(vmRuntimeClass, "getRuntime", null);
Method setHiddenApiExemptions = (Method) getDeclaredMethod.invoke(vmRuntimeClass, "setHiddenApiExemptions", new Class[] { String[].class} );

Object vmRuntime = getRuntime.invoke(null);
setHiddenApiExemptions.invoke(vmRuntime, new String[][] { new String[] { "L" } });
```

If your app is compatible with versions of Android lower than 9, remember to wrap this in a version check.

Android 9 and later

To bypass the hidden API blacklist on Android 9 and any later version, you can use LSPosed's library. This library uses Java's Unsafe API, so it's unlikely to ever break.

To implement it, just add the dependency to your module-level build.gradle.

```
implementation 'org.lsposed.hiddenapibypass:hiddenapibypass:2.0'
```

Then use it to bypass the blacklist.

```
HiddenApiBypass.addHiddenApiExemptions("L");
```

If your app is compatible with versions of Android lower than 9, remember to wrap this in a version check.

Conclusion and More Info

There are plenty of options for bypassing the hidden API blacklist on Android, no matter which platform version you target or use. If you're curious to learn more about how these methods and libraries work, be sure to check out the following links.

- My Stack Overflow Q&A.
- LSposed's Hidden API Bypass library on GitHub.
- ChickenHook's RestrictionBypass library on GitHub.
- tiann's FreeReflection library on GitHub.
- Google's documentation on the hidden API blacklist.

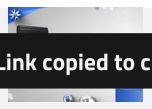
— **Related Topics**

XDA TUTORIALS | TUTORIAL | APP DEVELOPMENT | DEVELOPMENT

— **About The Author**

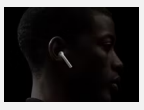
Zachary Wander

(77 Articles Published)



Samsung's early Black Friday deals bring deep discounts on monitor and SSDs

Link copied to clipboard



AirPods drop to $69 in phenomenal early Black Friday deal

10 hours ago

See More