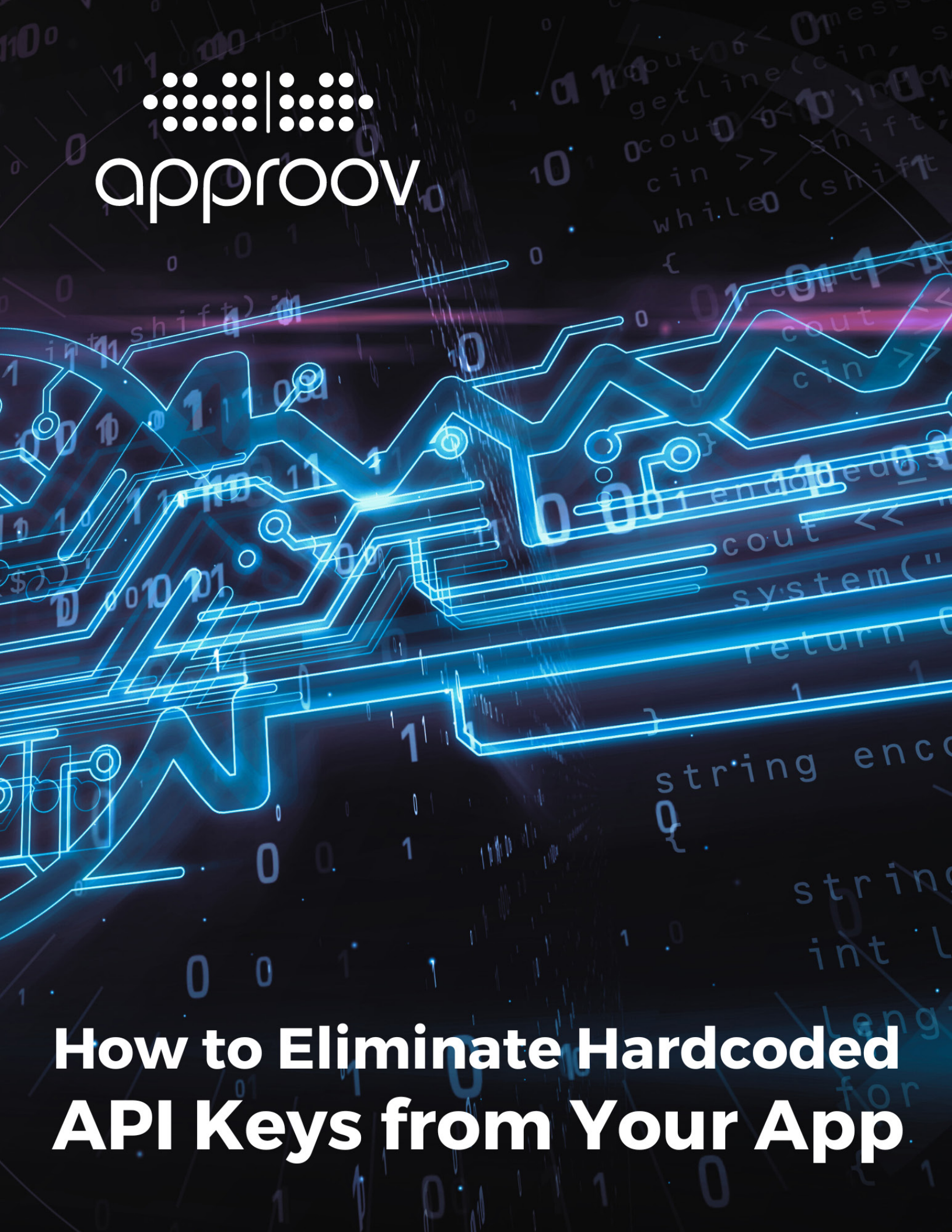
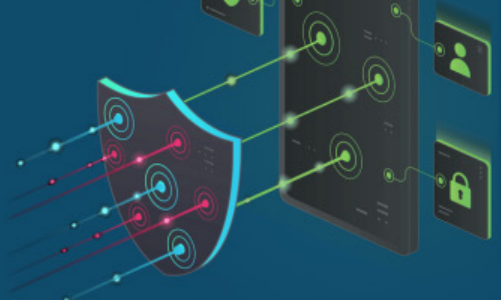

approov
How to Eliminate Hardcoded
API Keys from Your App

approov

WHITE PAPER



How to Eliminate Hardcoded API Keys from Your App

Contents

Mobile apps typically communicate with several backend services over the Internet. In many cases these services will be developed and operated by a 3rd party. It is typical for access to these services to be authenticated by the use of an API key. This requires the API key to be embedded inside the released app itself. This whitepaper discusses the risks of such hardcoded API keys and how Approov's Runtime Secrets Protection feature enables such keys to be easily eliminated from apps to avoid all the associated risks.

The Problem with API Keys

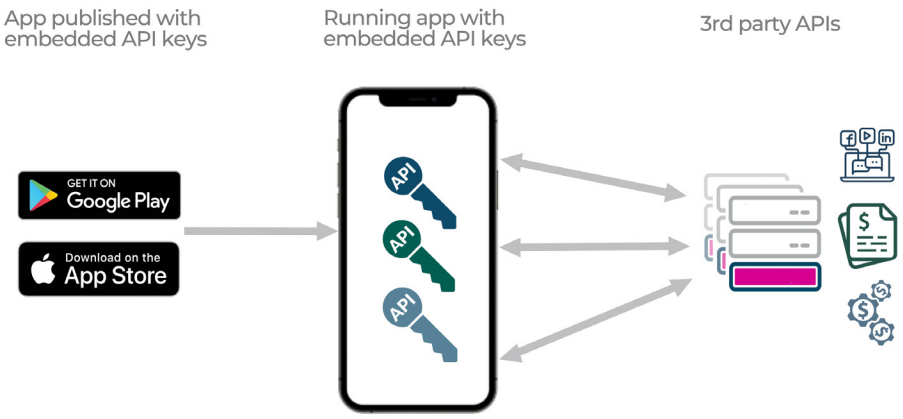
API keys are commonly used to authenticate your app to a backend API and, as such, operate like a password. API keys block any anonymous traffic and can be used to limit the rate of requests from any one particular app. So although API keys are used as an identification mechanism you also have to consider them to be a secret. If that API key becomes known then your app can be impersonated and API calls can be made by other parties, just as if your mobile app was making them.

Keeping API Keys Out of Source Code Isn't Sufficient

In the case of mobile apps the compiled code is in the public domain and must be released through the app store. So even if you have followed the good practice of using a secret management tool to keep your API keys out of your source code, this isn't sufficient. Those keys will still be compiled into or linked into your release package in some way and will appear in the app store.

Third-Party API Access from Mobile

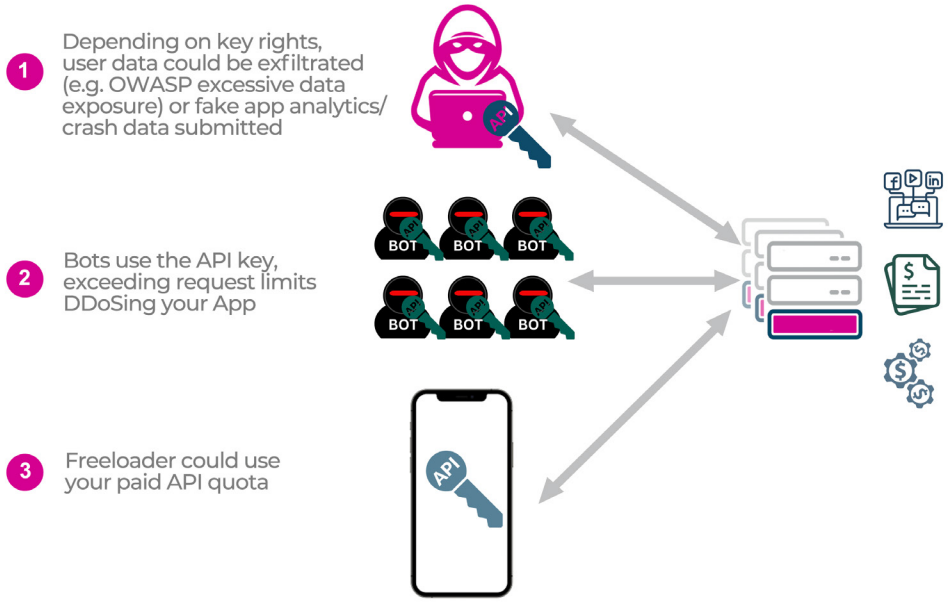
Consider the case where you have compiled your app and published it to the app store. As the app runs it will use the API keys embedded within it to access the services it is dependent upon:



We've recently completed a survey that shows in general that individual apps are dependent on a large number of 3rd party APIs. Some 45% of apps store 3rd party API keys, with an average of 40 APIs per app for those using in-house development. Whenever it accesses that individual API, that API key is copied from the code as it runs and then transmitted over the network.

The Risks of Stolen API Keys

API keys can be reverse engineered from the code, or captured in transit from the mobile app to the backend. Any attacker has complete control of the device and the network it uses, so such attacks are relatively straightforward.



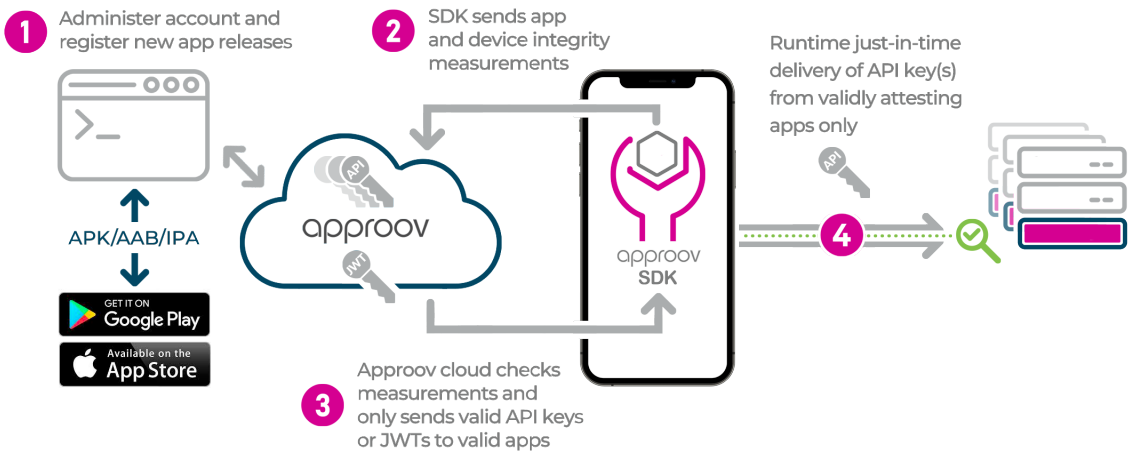
The risk obviously depends on the capabilities of the particular API whose key has been compromised. It's possible that having access to an API outside of the app allows some data exfiltration; either directly or indirectly through vulnerabilities like OWASP excessive data exposure. This is a classic API implementation bug whereby an API may present more information than it really should, but relies on the client (mobile app) to filter what is made available. Once you have the API key you can access the API directly, perhaps running it in Postman or similar, and therefore see all the data.

Another possibility is that there are API keys associated with crash or installation analytics. You can use that to impersonate the app and post incorrect and misleading data. That undermines the analytics, or causes support teams to waste time tracking down seemingly widespread phantom bugs.

Use of a 3rd party API will be typically governed by a quota, so that an individual app (identified by an API key) can't make so many API calls that the quality of service of other users is degraded. If that API key is extracted and distributed then scripts using it will rapidly deplete your quota. In effect this is a Denial of Service (DoS) attack with a low cost for the attacker. Calls from the real app will be rejected and it will cease to function correctly.

If you use some 3rd party APIs that are on a pay-per-use basis then the attacker's motivation might be to be able to make API calls without paying. But you will be paying for them!

Approov Architecture



The diagram above illustrates the four-step flow of how Approov Runtime Secrets Protection works. This protection can be easily dropped into an existing app, that currently embeds API keys, with the minimum of changes:The steps are as follows:

1. Approov account administration is simple, using a single CLI tool. The API keys can be added into the Approov account where they are stored securely in the Approov cloud. The Approov SDK can be dropped into the app with only a small number of straightforward code changes needed to support its usage. These code changes eliminate the API keys from the shipped app code. When a new app is released to the appropriate app store the Approov CLI is employed to add that particular version of the app so that Approov recognizes it as being valid.
2. At runtime, when the app makes a request that will need an API key, the SDK automatically gathers app and device environment integrity measurements and sends them to the Approov cloud. Various protections are in place to ensure that these measurements cannot be spoofed.
3. The Approov cloud evaluates the measurements and, only if they are consistent with an official App running in an appropriate environment, are the API keys transmitted to the app. You have complete control of the security policy applied and thus which environmental detections will cause a rejection.
4. The API keys are added to the requests as required so that the mobile app can authenticate itself to the backend. If the original request was rejected then an error message can be shown to the user. The API keys are only ever held temporarily in memory and communication with the backend is protected to ensure that the keys cannot be intercepted via a Man-in-the-Middle (MitM) attack.

With Approov protection in place, only valid app instances running in uncompromised environments can access the API keys stored in the Approov cloud. Absolutely no changes to the backend APIs themselves are necessary.

As an additional benefit, API keys can then be easily rotated simply by administering the Approov account without the need to issue a new version of the app. All running app instances repeat their integrity checks at least every 5 minutes, so new API keys can be propagated to all app users within this period.

Comprehensive Environment Checks

Approov performs a wide range of app and environment checks as part of the measurement and subsequent attestation process:

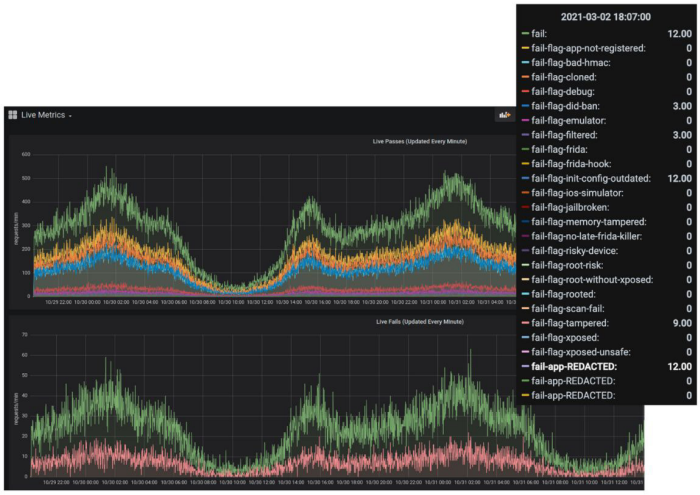
- Debug detection
- Root and jailbreak detection, including Android Magisk detection
- App tampering
- Emulator and simulator detection
- Running inside a cloner app
- Detection of Android automated app launch
- Detection of function hooking
- Advanced detection of runtime app instrumentation frameworks such as Frida
- Protections against late attachment of Frida and debuggers

In most cases several different detection methods are employed to provide high resiliency against attack. The core measurement gathering and detection logic is heavily defended, and it can also be updated over-the-air without the need for a new app release.

Approov is also under continual development to protect against the latest attacks seen across the full range of our customers.

The interaction with the Approov cloud has a side benefit. Approov sees all the connections from different devices and Approov account holders get live and cumulative metrics showing what's actually happening in their apps.

Adjust your security position based on real time data from your app installed base

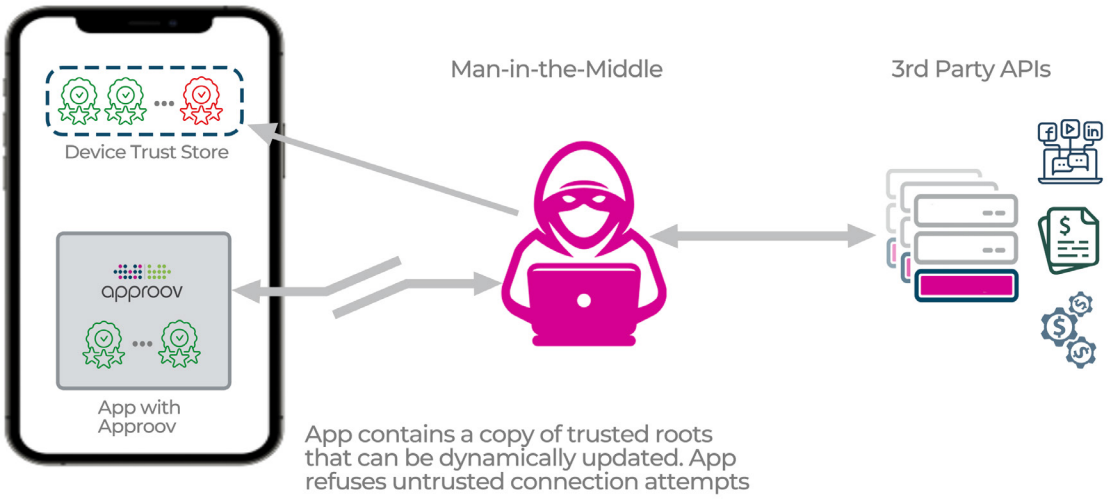


Protecting API Keys in Transit

The app and device integrity measurements ensure that the API keys are only ever transmitted to valid instances of the apps. It is also necessary to ensure that there is no possibility that the API keys could be stolen in transit to the backend API by a Man-in-the-Middle (MitM) attacker. This is challenging, since pinning cannot be employed because the certificates used on a backend API server could be changed at any time in a manner not controlled by the app developer.

We know that the trust store on the device itself could be compromised if the device has been rooted or jailbroken. It is not possible to guarantee that this can be detected if all other evidence of a prior rooting or jailbreak has been erased from the device.

To solve this, Approov provides a facility called Managed Trust Roots:

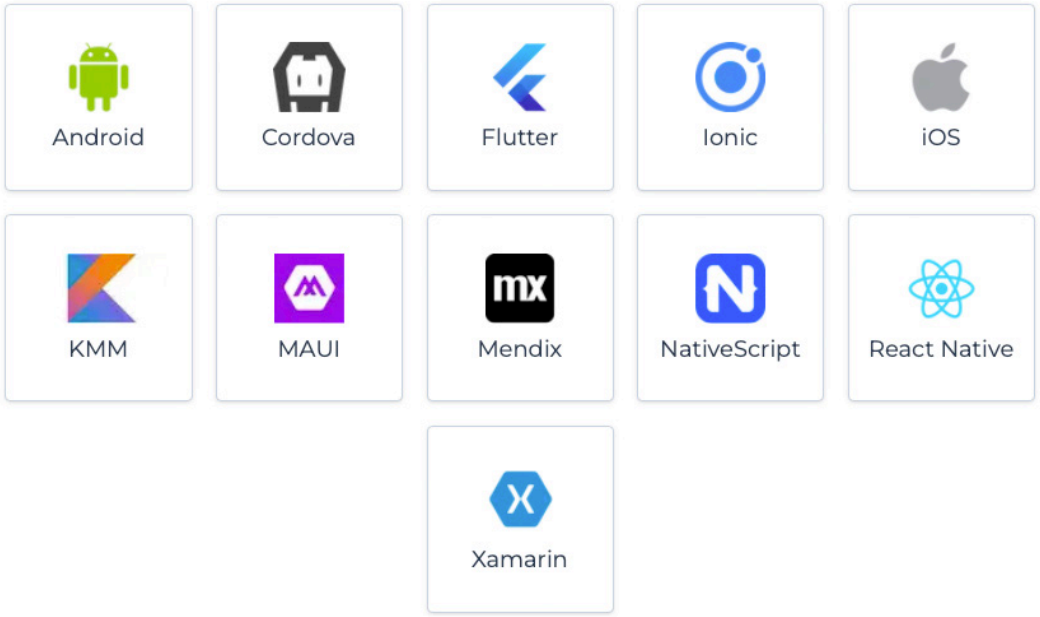


This effectively provides the capability for another trust store to be held within the app itself. The app will only make the connection if the certificate chain root is in the device trust store **and** it is also in the Approov managed trust roots inside the app. MitM attacks are blocked because there is no way for an attacker to overwrite the app's managed trust roots without this being detected by Approov. Moreover, other methods that might be used to bypass detection, such as Frida, are also detected.

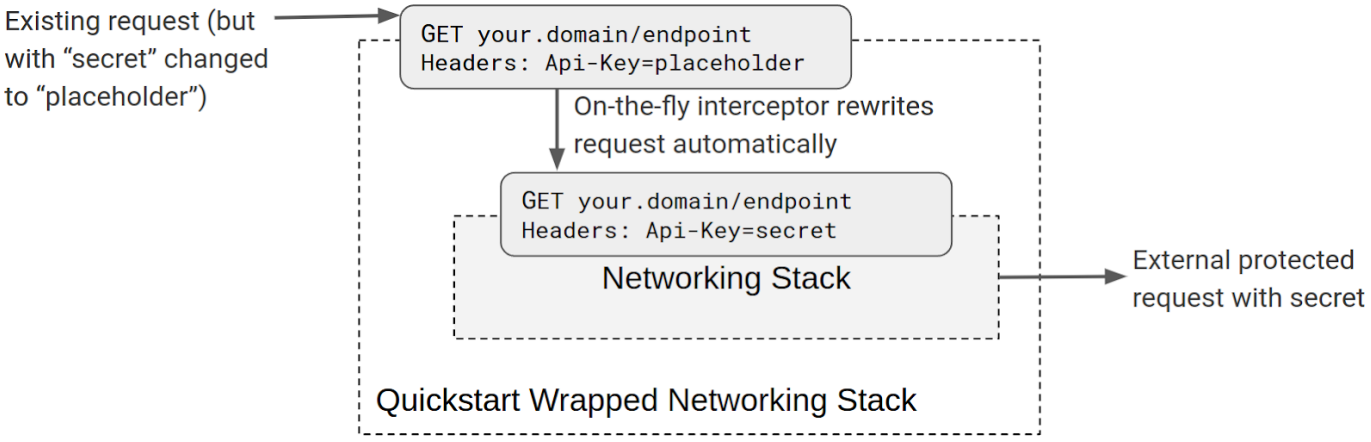
The managed trust roots within running apps can be dynamically updated by Approov as required. Thus if particular root certificates need to be removed or newly issued root certificates added then this can all be achieved without any

need to update the app itself. Crucially though, these updates are only accepted if signed by Approov so an attacker cannot use this mechanism to subvert the trust store.

Quickstart Integrations



In order to make the integration Approov straightforward there are a range of app quickstart integrations. For native Android support is provided for `OkHttp`, `Retrofit`, `Volley`, `GRPC` and `HttpsURLConnection`. For iOS native there is support for `NSURLSession`, `URLSession`, `AlamoFire`, `GRPC` and `Async Http`. New quickstarts are added continually, so please contact us if you would like support for something else.



Integration is made as simple as possible by providing a networking stack that has exactly the same interface as the standard one, except that it provides runtime API key substitution.

Simply swap out the API key value in your app for a placeholder value instead. This can just be some innocuous string that cannot be directly used for API access. The Approov networking stack then performs interception on the fly at run-time back to the actual API key value.

If you are providing an API key as a header then it will detect the placeholder API key and then perform the Approov integrity checks. If the app passes then the API key value is made available and the replacement of the placeholder

value can be completed. The API key is only provided transiently for the network request and is otherwise held in a protected form in memory. Every request causes Approov to make some authenticity checks, and a complete integrity check is performed if it has been longer than 5 minutes since the previous one.

Some APIs require API keys to be provided as a query parameter on the URL rather than as a header, and this is also supported.

Summary

Approov Runtime Secrets Protection frees you from needing to hardcode API keys in your mobile app. Concerns of reverse engineering and abuse are removed. Strong app attestation protection ensures that only an authentic app running on an uncompromised device can access them, and thus the APIs that they protect.

The TLS communication channel is protected with managed trust roots, providing an additional trust store within the app, that blocks any Man-in-the-Middle (MitM) attacks.

The API keys can be managed in the Approov cloud where they may be updated at any point, with only a 5 minute propagation time to all running app instances.

This is all achieved without any need to change the backend APIs and with a straightforward drop-in app integration approach.