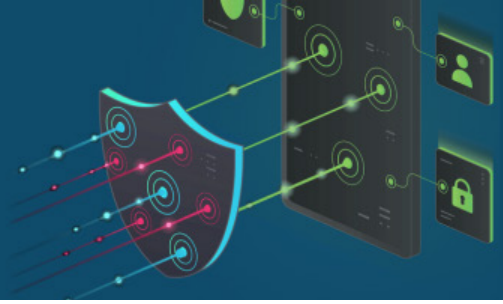
How to Prevent MitM Attacks Between Mobile Apps and APIs

approov

approov

WHITE PAPER



How to Prevent MitM Attacks between Mobile Apps and APIs

Contents

The massive deployment of mobile apps is presenting new attack surfaces to bad actors. The channel between apps and APIs presents a rich target for hackers via Man-in-the-Middle (MitM) attacks. This white paper explains why MitM attacks are a particular issue for mobile apps and explains why using Transport Level Security (TLS) alone is not sufficient to stop them. After an in-depth analysis of the problem we will look at how certificate pinning can help thwart mobile MitM attacks, and the risks involved in setting pins statically within an app. We also look at the advantages of being able to set the pins dynamically, and the steps you should take to protect your organization's data and revenue from these types of exploits.

Introduction

Mobile app usage has been increasing year on year and that seems unlikely to change. As shown in Figure 1, direct revenue derived from mobile apps is also showing impressive growth. Most consumer facing enterprises now have a mobile app since it is the preferred touchpoint for their customers and even if those apps don't generate revenue directly for the company, trust in the mobile app platform is vital for brand reputation.

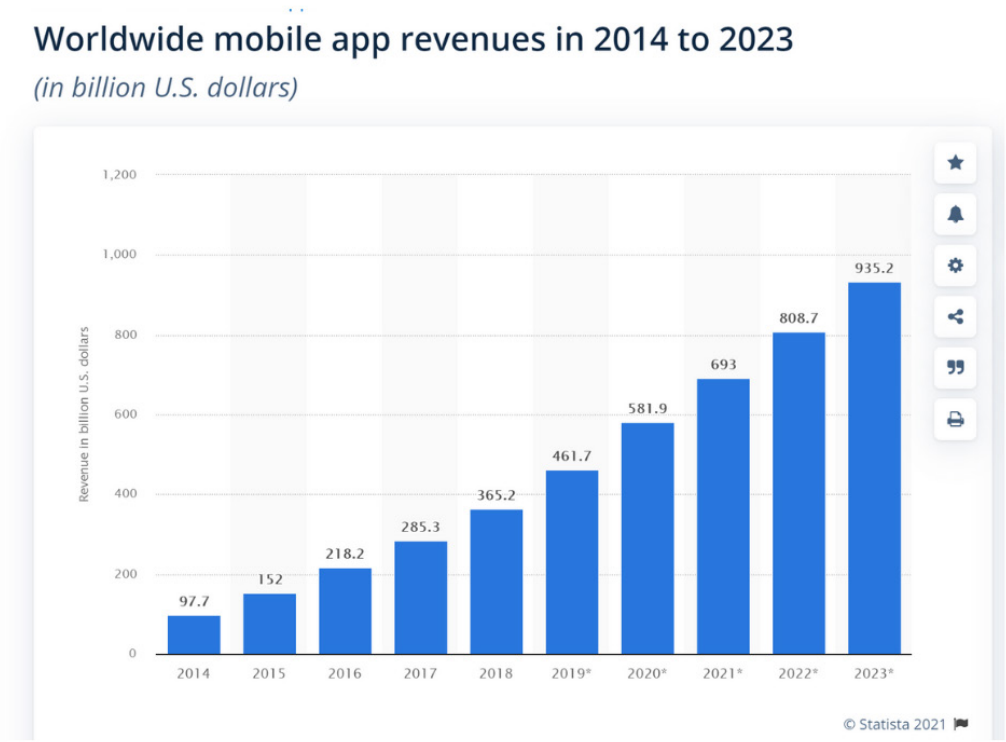


Figure 1: Worldwide Mobile App Revenue Forecast

(Image source: Statista)

Unfortunately the deployment of mobile apps and the proliferation of APIs that serve them present some new security challenges, offering novel opportunities to bad actors to access sensitive data and derail your business. This is because mobile apps are downloaded to unmanaged devices and there are a battery of available tools to allow hackers to dissect and manipulate them at their leisure.

The API channel between the apps and backend services is one of the 5 defined attack surfaces in the mobile ecosystem.

Attacking the API channel between mobile apps and their backend servers through Man-in-the-Middle (MitM) attacks are a growing threat for mobile users. The ability to intercept and manipulate communications between mobile devices and servers is a particular problem in mobile because of the explosive growth in mobile app usage, and is now becoming a common attack vector. It is an issue that has been known for some time but in spite of this, many enterprises are not clear on effective and efficient ways to combat these attacks. This paper is intended to help address this issue and help mobile-first enterprises eliminate this threat.

Man in the Middle Attacks

Man-in-the-middle attacks occur when an attacker intercepts or manipulates mobile device communications to gain access to sensitive information. The bottom line is that they give attackers the ability to see any communications, modify messages using the channel, steal login details or certificates from encrypted traffic, intercept sensitive commercial/personal data, or even easily launch a denial of service attack against the service being accessed via a mobile app.

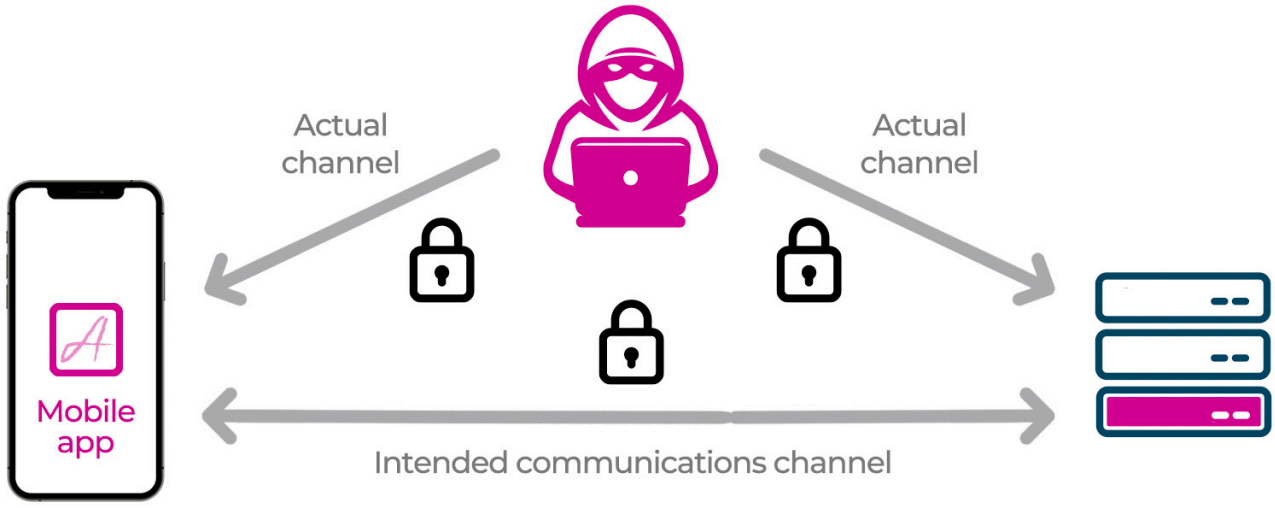


Figure 2: A Man-in-the-Middle Attack

Image source: Approov

TLS and Encrypted Traffic

You might be wondering about the fact that API traffic is normally encrypted using TLS (Transport Level Security). This is now ubiquitous and typically enforced by the mobile platforms themselves. TLS uses PKI (Public Key Infrastructure) and trusted Certificate Authorities to ensure a mobile app can verify it is communicating with a legitimate backend server. However, there are still ways that a MitM can intercept traffic in the channel so that the mobile app communicates with the MitM actor over an encrypted channel thinking that it's actually the backend server. In this way the MitM can see all the traffic, potentially modify the traffic, and then transmit it onward, again over an encrypted channel to the backend service. Let's look at how TLS is supposed to work and how it can be manipulated.

When a communication is made from the app to the backend service, the certificate issued for your server and associated with a particular domain name, forms part of an overall trust chain to prove its legitimacy. During the TLS negotiation a chain of digital certificates are presented and are verified by the client to prove the legitimacy of the server. The certificates form a trust chain that ultimately needs to lead to a root Certificate Authority (CA) certificate. The trust anchor point is established via the pool of root certificates which are pre-installed (and subsequently updated) on the mobile device itself. A chain of trust from a leaf server certificate is only accepted if the chain leads to one of those root certificates.

The Chain of Trust

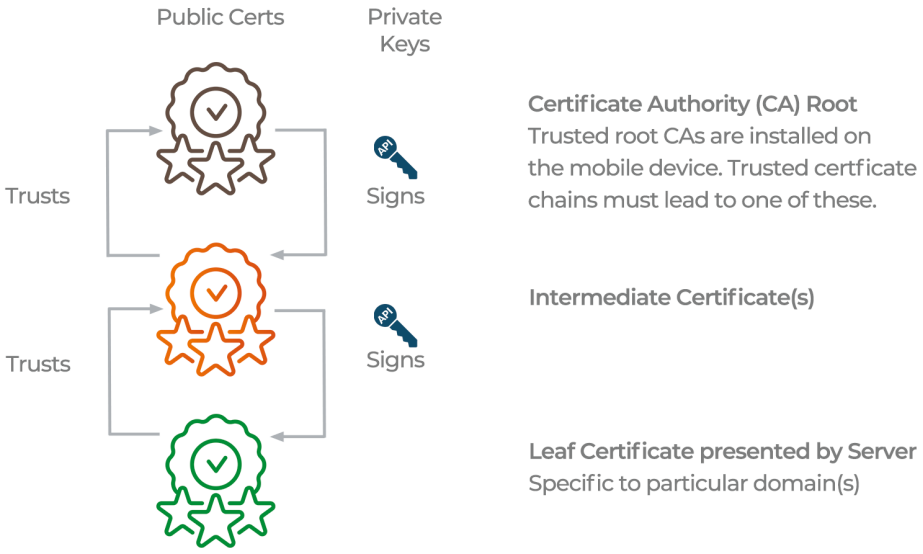


Figure 3: The Chain of Trust

Image source: Approov

In order to be issued with a valid certificate for a particular domain you need to prove to the certificate authority and prove to them that you actually own or control this domain. This is typically done by setting a specific record on the Domain Name System (DNS) record for the domain or, in some cases, placing a specific file token on a web server. For some types of certificate, a more in-depth process requiring phone verification may be employed. If ownership is proven, then the CA will use the private key corresponding to their certificate to sign your certificate. This forms the trust chain that ultimately proves that you own the domain. This allows the certificate to be used by a server associated with the domain and for it to be recognized as valid by TLS. In the general case there may be one or more intermediate certificates in the chain between the leaf certificate issued for a specific domain and the root certificate held in the trust store of a device.

So how is a MitM attack executed for a TLS encrypted channel requiring a verified trust chain in order to allow the connection to be established?

In general there are two approaches. Firstly, there's a variant where the MitM attacker is in the channel and has no

access to the mobile app. Secondly, there's a variant where the MitM attacker is also controlling the app as well and the environment that the app is running in.

Let's look in more detail at two common ways an attacker can break the chain of trust and execute a MitM attack.

Breaking Trust - Trust Store Poisoning

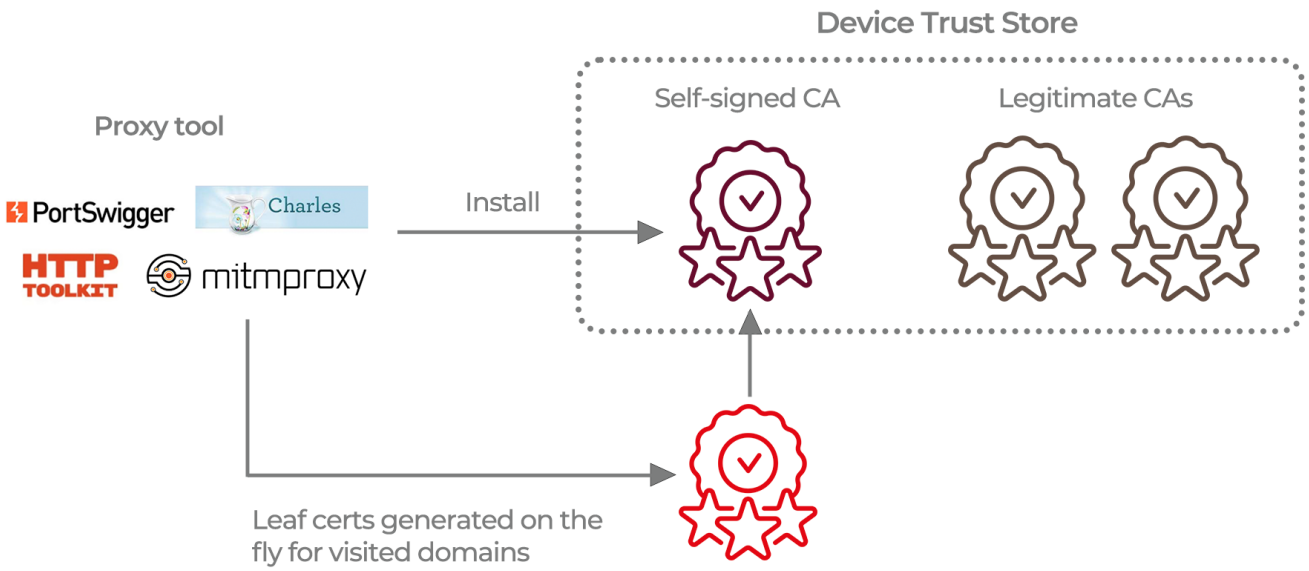


Figure 4: Trust Store Poisoning

(Image source: Approov)

If you have access to the device, you can use a MitM tool, of which there are many, such as mitmproxy. In such cases you'll be analyzing traffic from a mobile app and you're also controlling the device. Tools such as mitmproxy create a certificate that is installed onto the end user device. Rather than being a certificate from a root certificate authority, it is actually a self-signed certificate that the tool has itself created (see Figure 4). You install it into the trust store on the device and then when the tool intercepts traffic it will, on the fly, also create leaf certificates for the particular domains that you are visiting, which have a chain of trust back to the self-signed certificate authority that is installed on the device. Thus the trust chain will be verified and the traffic is successfully redirected to the MitM rather than going to the real server. From there the MitM will then connect to the real server, relaying the traffic but with the proxy in the middle able to observe all the traffic. The proxy can potentially modify the traffic in flight. Moreover, it can also record and potentially replay the traffic later.

In recent years various protections have been added to mobile platforms to actually make it very difficult for an end user to be tricked into installing one of these self-signed certificates on your device.

The attacker really needs to control the end device, and may require it to be rooted or jailbroken, in order to install one of these certificates in a way that will be trusted by most apps. This is typically a technique used for traffic analysis when the attacker controls the end user device, and wishes to perform a MitM to analyze an API protocol.

Breaking Trust - CA Breach

Figure 5: CA Breach or Bad Issuance

(Image source: Approov)

Another way that the trust chain can be broken is if there is a breach of a certificate authority or a bad issuance of a certificate.

One of the weaknesses of PKI architecture is manifested in the large number of root certificates that are installed on the device. There could be a data breach at any one of the certificate authorities or, more likely, a failure or circumvention in the process that's used to prove that you are actually the owner of a domain. This would allow an attacker to be issued with a valid certificate for your domain. It only requires a breach at any one single certificate authority for this to be a threat.

Alternatively there could be a breach within your own systems. If an attacker is able to control your DNS records or web content then they could pass the verification to enable a valid certificate to be issued by a certificate authority to them.

It is then possible to utilize DNS poisoning, especially on public WiFi, for an attacker to redirect requests to their servers with the bogus certificate. As far as the app is concerned it would readily connect with TLS since the certificate being served would look completely legitimate. The attacker could then perform full interception of end-user traffic (see Figure 5).

The Benefits of Pinning

Certificate pinning enables developers to protect mobile apps from the MitM attacks described. However, despite its effectiveness, it isn't widely used except in some highly security conscious sectors such as financial services.

In Figure 6 you can see the relationship between the device on the left-hand side and the back-end service on the right. As we have seen, the server provides a particular certificate and then during the negotiation a trusted chain is established from that leaf certificate to some certificate authority. The certificate authority needs to be trusted and this is verified by checking the device's trust store. This means that the app is relying on information in the trust store on the device, but it doesn't entirely control this, especially if the app is running in a compromised environment.

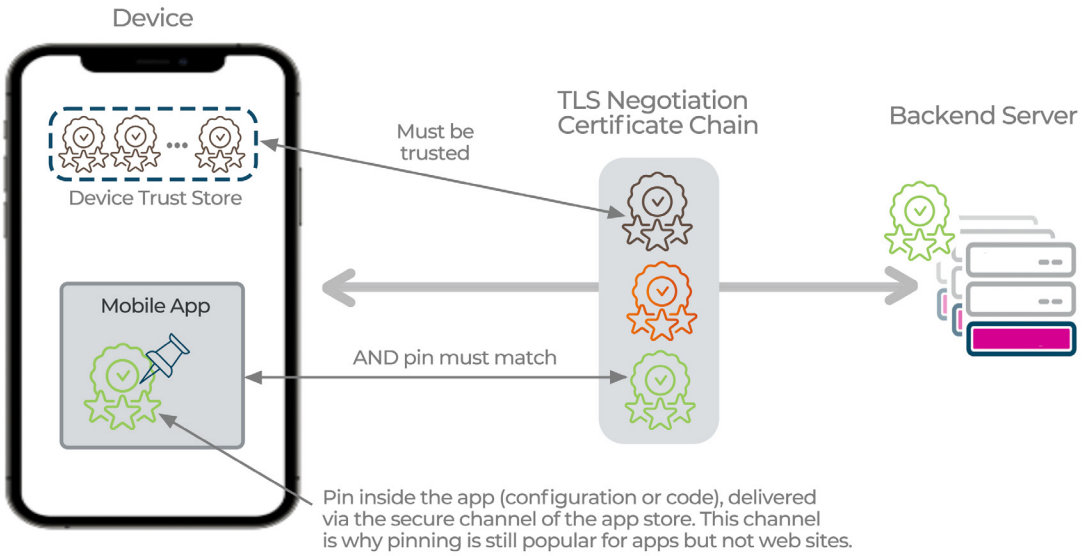


Figure 6: Certificate Pinning

(Image source: Approov)

Pinning adds an additional check by adding a pin of a certificate inside the mobile app itself. In addition to all the standard verification which is performed on the certificate chain during the negotiation, a check is made that a certificate corresponds to the expected pinned value. Typically, pinning is done against a leaf certificate. However, pinning to a root or intermediate certificate is also possible, as discussed in the next section.

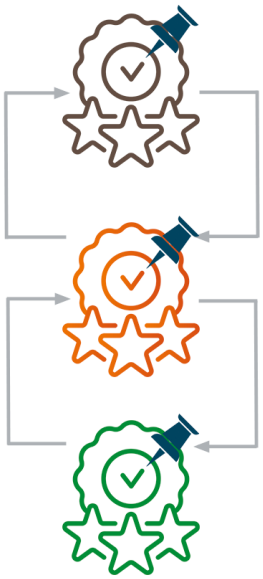
Certificate pinning allows mobile applications to restrict communication only to servers with a valid certificate matching the expected pin value. The connection is terminated immediately if communication is attempted with any server that doesn't match this "expected" value.

The pin is usually not a copy of the entire certificate. In fact it is typically a hash of the certificate, or some key attributes from the certificate. The app is shipped including the pin and will only connect if it sees the expected certificate.

In this way, you're not just relying on the trust chain: The app is saying 'I want to see this particular certificate which I know is my certificate, anyone else who generates a certificate for my domain is not going to get through'.

One reason this is easier to implement on mobile apps is that there is a secure channel through the app store for releasing the app code including the configuration. There have been attempts in the past to have certificate pinning on the web, for example HTTP Public Key Pinning (HPKP), but this suffers from multiple problems and is no longer used. One key problem was that the only channel to receive the information for the pins was the channel that needed to be pinned, and this led to potential for attacks. In the mobile context the pins are included in the app code, which are delivered by a different, secured, route.

Public Key Pinning versus Certificate Pinning



Pin Certificate Authority (CA) Root
- You only trust anything from this CA - substantially reduces attack surface.
- Likely to be very long lived.
- But pin may change if you move CA, or they use a new trust root.

Pin Intermediate
- Most relevant if you have your own intermediate you sign from.
- Allows common pinning, for faster rotated server specific certificates

Pin Leaf
- Specifically your certificate for your endpoint - most secure.
- Most subject to rotation though.
- Only do this if you're sure public key retained through rotations.
- Beware of different certificates for different geographic locations.

Figure 7: Pinning Options

(Image source: Approov)

The term "certificate pinning" is employed here, although technically what is usually used is "public key pinning".

Certificates have many fields, including the signature of the certificate authority as created by the next authority up in the chain. A one-way cryptographic hash function takes a selection of the content and generates a derived value which cannot be feasibly reversed. It is possible to produce a hash of all the certificate content and check that a specific certificate is being presented by verifying that.

However, the approach typically employed in mobile apps (and supported by development frameworks) takes the algorithm and public key information and hashes only these elements. This is termed Subject Public Key Information (SPKI) hashing. This produces a hashed pin value with the main advantage that it is not a specific certificate that is being pinned, but its public key. This means that if you renew the certificate, as long as the public/private keypair is retained, then the pin won't actually change. Thus it is possible to keep renewing certificates without constantly having to change pins in mobile apps.

If you are pinning then you do have a choice of what to pin. Please refer to Figure 7.

One option is that you can pin to a certificate authority or the root certificate. This says 'I will only accept certificates which have come from this particular certificate authority, signed by this particular root certificate'. That has the advantage that it's much less likely to change because these certificates are much longer lived. What it's actually doing is restricting connections to be from a particular certificate authority, not necessarily tied to a particular domain. In security terms it's not quite as strong as pinning to a leaf certificate but this may be an acceptable trade-off depending on your circumstances. Of course, you may still need to change the pin if you obtain future certificates from a different authority, or they change the root certificate they are using in a future certificate issuance to you.

You can pin to an intermediate certificate; some larger organizations have intermediate certificates internally from which they sign their own server certificates. Putting in an intermediate means that you're pinning something which may be longer lived than a leaf certificate.

Ultimately you may choose to pin to the leaf certificate. This is definitely the most secure because it ensures the mobile app only accepts your certificate, with your private key, issued for your domain. Good practice says that you should rotate your leaf certificates regularly. It is important to ensure that the same public/private key is retained to avoid needing to change the pins.

You also have to be aware that if you are dealing with individual servers, sometimes in particular infrastructures there might be different certificates for different geographic locations. You need to be sure that they are all using the same key pair or otherwise you need to include all those pins in your mobile app, which may then be harder to manage.

It is also recommended practice to include at least one backup pin for a certificate that is not currently in service. In an emergency you can then use one of these certificates that will immediately match the pinning you have in place.

Implementing Pinning

Android (since Android 7)

Add `res/xml/network_security_config.xml` file and reference from manifest. Specifies pins for domains. No code changes needed.

See: https://developer.android.com/training/articles/security-config#CertificatePinning

```
<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
    <domain-config>
        <domain includeSubdomains="true">example.com</domain>
        <pin-set expiration="2018-01-01">
            <pin digest="SHA-256">7HIpactkIAq2Y49orFOOQKurWxmmSFZhBCoQYcRhJ3Y=</pin>
            <!-- backup pin -->
            <pin digest="SHA-256">fwza0LRMXouZHRC8Ei+4PyuldPDcf3UKgO/04cDM1oE=</pin>
        </pin-set>
    </domain-config>
</network-security-config>
```

iOS (since iOS 14)

Edit `Info.plist` file and add `NSAppTransportSecurity` section for pins. No code changes needed.

See: https://developer.apple.com/news/?id=g9ejcf8y

```
<key>NSAppTransportSecurity</key>
<dict>
    <key>NSPinnedDomains</key>
    <dict>
        <key>example.org</key>
        <dict>
            <key>NSIncludesSubdomains</key>
            <true/>
            <key>NSPinnedCAIdentities</key>
            <array>
                <dict>
                    <key>SPKI-SHA256-BASE64</key>
                    <string>r/mIkG3eEpVdm+u/ko/cwxzOMo1bk4TyHIlByibiA5E=</string>
                </dict>
            </array>
        </dict>
    </dict>
</dict>
```

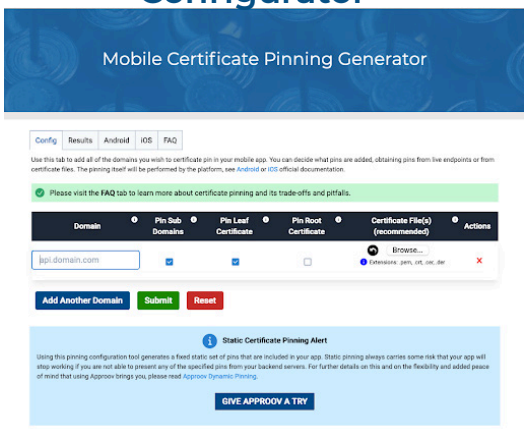
Figure 8: Pinning on Android and iOS

Historically, certificate pinning has been challenging to implement and highly reliant on the networking stack in use. DataTheorem's TrustKit has been a useful implementation library. However, Google and Apple have recently enhanced their platforms to simplify pinning further, removing any dependency on the network stack and allowing pinning via configuration across a wide range of networking stacks.

Google has supported pinning configuration since Android 7. Developers simply define pins in the network security configuration file's particular XML syntax. In Figure 8 (top), you can see the sort of syntax you need to use. With this in place, you don't need to change the code of the app to specify the particular domain or domains that you want to pin against. This creates the list of pins to use. It doesn't specify where in the chain those pins are and if a particular hash is present anywhere in the chain, it is considered to be a good connection as long as it passes all the other TLS checks.

Apple has lately followed suit and added `NSPinnedDomains` support with iOS 14. Of course iOS users tend to upgrade much more frequently than Android so that represents quite a high percentage of the user base. Developers may add pins by adding them in the `Info.plist` file for the app in the correct format. It's a completely different XML syntax to Android but essentially doing the same thing: you specify pins for particular domains (See bottom of Figure 8).

The Mobile Certificate Pinning Configurator



If you want to get started with certificate pinning, this free Pinning Generator Tool makes it simple to generate and maintain pinning configurations for mobile apps, ensuring that they are kept up to date on Android and iOS.

We made this tool available to the community in order to simplify the process of pinning in mobile apps and take as much of the guesswork out of the process. The tool is easy to use and generates exactly the information you need to cut and paste for both Android and iOS.

However, getting the pin information is quite cumbersome. It's really not very user friendly and you have to do this for every single pin that you want to include for each domain. As well as being laborious, it doesn't really help you with the process of actually generating the exact format for the XML file. There are also complications with converting between the various different formats that you might have certificates in. You might need to get a certificate and then change between different types of formats.

So even though there is now some solid platform support, the configuration part is tricky, especially if you're not familiar with PKI and certificate management. The majority of the setup is based on issuing numerous complex OpenSSL commands and managing certificate files in various formats.

Approov has made available to the community a tool to take the hard work out of generating and maintaining pinning configurations for mobile apps (See box).

Barclays Bank UK Incident

A good example of a situation where static pinning was disastrous is the 2016 Barclays Bank UK incident. The bank's mobile application had been pinning an obsolete intermediate certificate in the mobile application - making transaction authentication impossible. Hundreds of thousands of consumer payment transactions were affected due to the outage, which prevented many small and medium-sized enterprises from conducting important transactions. As a result, many companies had to close their doors at 8:30 am on 25th November 2016 (Black Friday) and for the rest of the festive period leading to immense financial losses. In addition, it had a significant negative impact on Barclays' reputation and its business customers.

The principal operational risk here is that if the pins no longer match your real endpoint then you have a problem because your app no longer works. You can no longer communicate with your servers which effectively means your app is down. This risk is the reason that some DevOps teams are reluctant to implement pinning.

You can get yourself out of this situation by issuing an app update with some changes to the pins. Unfortunately this takes time and in the meantime you will have a major problem. You also can't be absolutely sure that everyone is going to update their app. It's quite useful if you have some kind of messaging inside the app if there are pinning failures to indicate that you might want to do an update but really you need to make sure this doesn't happen at all. The thing to avoid is that you must never pin a domain that you don't control, so don't pin to an API endpoint where you're not controlling the server or you're not controlling whoever is responsible for rotating certificates, because at any point they can change their certificate chain and your app is no longer going to work.

If you have auto renewal on certificates on the endpoint then you really want to make sure the public key is maintained to save you having to change the pins. Of course you may, if there is some kind of breach of your private key, have to change pins and this is why you also need a backup pin so that you can put that into service should you need to rotate to a completely different key pair. You just need to be aware that you need to coordinate between the frontend and backend teams to make sure that the certificates don't change in a way that you didn't expect.

If you are about to rotate the certificate you can do it by putting the new pin into the app some time before you're going to rotate, ideally many days or possibly even weeks beforehand. The new pin and the old pin will both be in the app for a period of time.

In short, you need a well-defined process.

The Bad News - Pinning Can Be Bypassed in the Client

Pinning is very effective if you're just looking at Man-in-the-Middle attacks where the attacker doesn't control the end device.

Unfortunately, certificate pinning implemented in this way does not totally prevent the threat of MitM attacks on mobile apps. If an attacker controls the end device and they're doing some kind of analysis on your app then there are still ways they can bypass pinning.

There are a number of ways to bypass pinning if you have access to the client. The following sections describe two main approaches.

Pinning Bypass by App Repackaging

Figure 9: Pinning Bypass by App Repackaging

(Image source: Approov)

See our blog: Bypassing Certificate Pinning

The first way applies to both Android and iOS, although it is typically easier on Android. This involves repackaging the app. On Android, Apktool can be used for this purpose, but other similar tools are available. With this you can edit the pins, you can change the configuration, or if the pinning is implemented inside the code, then you can change the code. Then you can rebuild the app and run the modified version of the app (See Figure 9).

How susceptible your app is to this approach depends on how hardened it is. You can't just assume because you've obfuscated your app that it will necessarily be protected against this exploit.

Pinning Bypass Using a Hooking Framework

```
// Bypass OkHTTPv3 {4}
const okhttp3_Activity_4 = Java.use('okhttp3.CertificatePinner');
okhttp3_Activity_4['check$okhttp'].implementation = function (a, b) {
    console.log('  --> Bypassing OkHTTPv3 ($okhttp): ' + a);
    return;
};
```

Figure 10: Pinning Bypass by App Repackaging

(Image source: Approov)

https://github.com/httptoolkit/frida-android-unpinning/blob/main/frida-script.js

See: How to Bypass Certificate Pinning with Frida on an Android App

Another way of attacking and circumventing pinning is at runtime. If you have a rooted or jailbroken phone there are various types of frameworks that allow runtime hooking of functions running inside an app to modify their behavior. Frida is the most widely used.

One of the major use cases for these types of frameworks is to circumvent pinning and Frida has various open source modules that you can use. If you browse Frida CodeShare, then one of the most popular modules is a universal SSL pinning bypass. It's an enormous script that looks at all the different ways pinning is actually implemented for different types of networking stacks and inserts changes to the code at runtime to make sure that the pinning isn't active.

See Figure 10 for an example of such a script. In this case a particular method inside the java class OkHttp Certificate Pinner is modified. This replaces the code at runtime with a stub that doesn't do anything. The way the function works normally is that if there's a pinning failure then it blocks or it throws an exception. So once the function is hooked out by Frida the app's pinning protection is disabled.

Certificate Transparency

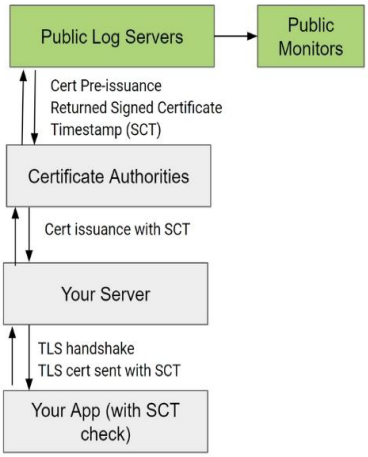


Figure 11: Blocking client-side MitM Attacks

(Image source: Approov)

There is an emerging solution that's used widely in browsers but less so in mobile apps. It's called certificate transparency and it's likely that in future years it will become quite prominent. The idea is that every time you issue a certificate you need to make sure that it's issued on a public log. This is to deal with the issue of some kind of breach around the certificate being issued against your domain. It doesn't necessarily stop a breach happening, but if you enable this particular feature then for any certificate to be valid it must probably be in the public log.

The idea is that you can monitor this public log and if a certificate turns up that you don't know anything about or it wasn't issued by you then you can put some mechanism in place to block communication using that certificate. The certificate transparency doesn't cover how that is done, it simply provides the transparency.

The way it works is that the public log servers provide a Signed Certificate Timestamp (SCT) when a certificate is issued which proves that the issuance of the certificate is now public and on the logs in a way that can't be revoked. It's a bit like a blockchain artefact where you can't revoke it after it's been issued and this SCT is then transmitted alongside the certificate information as part of the protocol. This is an interesting area which is implemented in iOS, but not yet entirely available on Android. In the future this might be a key part of the overall solution to prevent MitM attacks.

Dynamic Pinning Provides Easy Administration and Elimination of Operational Risks

To eliminate operational risks, mobile app developers need a way to pin certificates without requiring static pins. Instead, mobile applications should have access to dynamic or live pinned certificates from an online service so they can be updated automatically on the fly. This allows updates without having users download and install updates for their apps every time there's a change in security infrastructure.

Essentially, this approach allows mobile application developers to stay one step ahead of hackers by keeping up with changes in certificate authorities' keys over time while minimizing downtime due to misconfiguration, avoiding any potential reputational damage among consumers that could lead them away from using your business' mobile offering altogether.

This would allow developers and DevOps teams to avoid further incidents like Barclays' and improve customer experience over mobile. Certificate pinning must be implemented for all APIs that service mobile apps in industries which handle commercially or personally sensitive data. Trust is a major factor in mobile security, and app developers need to do everything they can to protect their customers from cyber-attacks while also maintaining trust among their users that the mobile application has been designed with privacy and data protection as top priorities.

The Final Piece in the Puzzle - How to Block Client-Side MitM Attacks

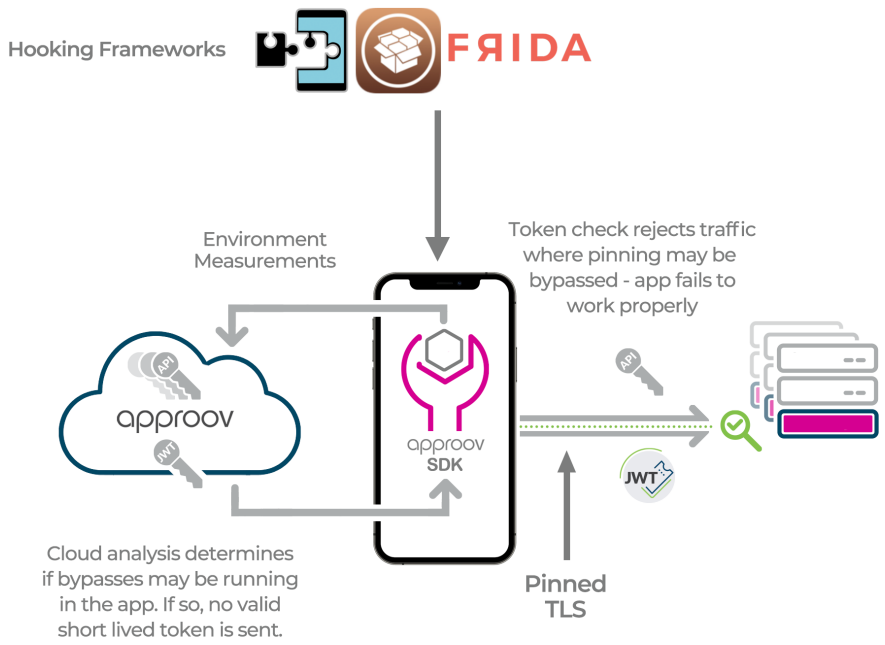


Figure 12: Blocking client-side MitM Attacks

(Image source: Approov)

MitM attacks involving client-side manipulation are very hard to detect at the back-end. It is often impossible to distinguish traffic coming from a proxy or actually coming from your app. There are some ways of fingerprinting TLS to see what type of stack is being used and this traffic may look slightly different coming from a proxy, but this is not entirely reliable, especially if the MitM is being executed by modifying an app running in the standard mobile environment.

The only way to totally prevent pinning bypass is to deploy a solution that implements app and client attestation. Application attestation verifies the integrity of the app and prevents anyone from modifying the app to change the pins.

Client device attestation detects any frameworks running in the client environment and detects any kind of hooking activity intended to change the behaviour of the app in real-time.

Approov: Complete MitM Protection with Assured Service Continuity

Approov provides a solution which addresses both the service continuity risks and the threat of client-side attack. Essentially it's an SDK and cloud service which is used to protect the integrity of the elements between the back-end service and the user. This includes proving the run-time integrity of the app, verifying device integrity especially in terms of checking if a framework like Frida is running on the device, and also ensuring the integrity of the communication channel.

Rather than this being a fixed resource file or a change to the code, the required pins are in a configuration file that can be updated live inside the app. The Approov SDK runs inside the mobile app and that SDK is responsible for communicating with a cloud service which transmits a dynamic configuration which includes the domains that your app is using and the pins for those domains. This can be updated dynamically so there is a way that you can change the pins at will, on the server side, and this is then sent back to individual apps that are running.

This approach completely avoids the situation of static pins in the mobile app where, due to circumstances beyond your control, you may end up with the app not working because the pins don't match. With dynamic pinning, you can update the pins at any point.

Also, as part of the solution, continuous end point monitoring is performed. Any domain which is set up is pinged from Approov servers on a regular basis, and the certificate chain is inspected to make sure it matches the pins which are inside the app. If they change there is an immediate notification allowing you to then push out a dynamic update to your apps.

It is essential that the dynamic update mechanism itself is well protected. Approov does not simply rely on the fact that the communication channel used for the update itself is pinned. In fact, each individual Approov account holder has a public/private key pair. The public key is embedded inside the app as part of the configuration of the Approov SDK. Approov servers hold the private key and when a configuration needs to be updated, it needs to be correctly signed with this key. An update for pins can be sent over an unpinned channel and the SDK itself will only accept this if it's correctly signed, verified with the public key. This proves that any update is actually coming from the Approov service. If so, then the app will be updated with the latest pins and start working again.

Client-side MitM attacks can be blocked because you will see that certain apps are failing the Approov checks. This may be because Frida has been detected or that the app has been modified. The Approov SDK gathers information, sending environment measurements back to the Approov cloud service. In turn this sends very short-lived tokens back to the app which then relays them on to back-end APIs. If the environment measurements don't check out because the app has been changed or there's an instrumentation framework running, then a validly signed token is not provided. This results in the backend API token check failing. The backend will then block further communication with the app.

One of the drawbacks of the static configuration using the platform integrated mechanism is that it is only available

from iOS 14, whereas the Approov implementation automatically adds the pinning for OS versions as old as Android 5 and iOS 10.

There is also very wide coverage across a large number of different frontend frameworks. The pinning implementation is open source but the pins themselves are obtained dynamically from the Approov SDK. Indeed, Approov's pinning implementation is a useful resource if you're looking to implement pinning on a particular framework but don't want to use the networking built-in mechanism.

Conclusion

MitM Attacks are a particular issue for mobile apps because the application and client environment can be manipulated by bad actors. Using Transport Level Security (TLS) alone is not sufficient to stop them. Certificate pinning is an approach which significantly reduces the attack surface but static certificate pinning can present some operational challenges.

However, as described in this Whitepaper, it is possible to prevent MitM attacks completely in a way that is both easy to administer and will ensure that service continuity is maintained. The following steps can be taken to achieve increasing levels of security, ease of operation, and ultimately to achieve full protection:

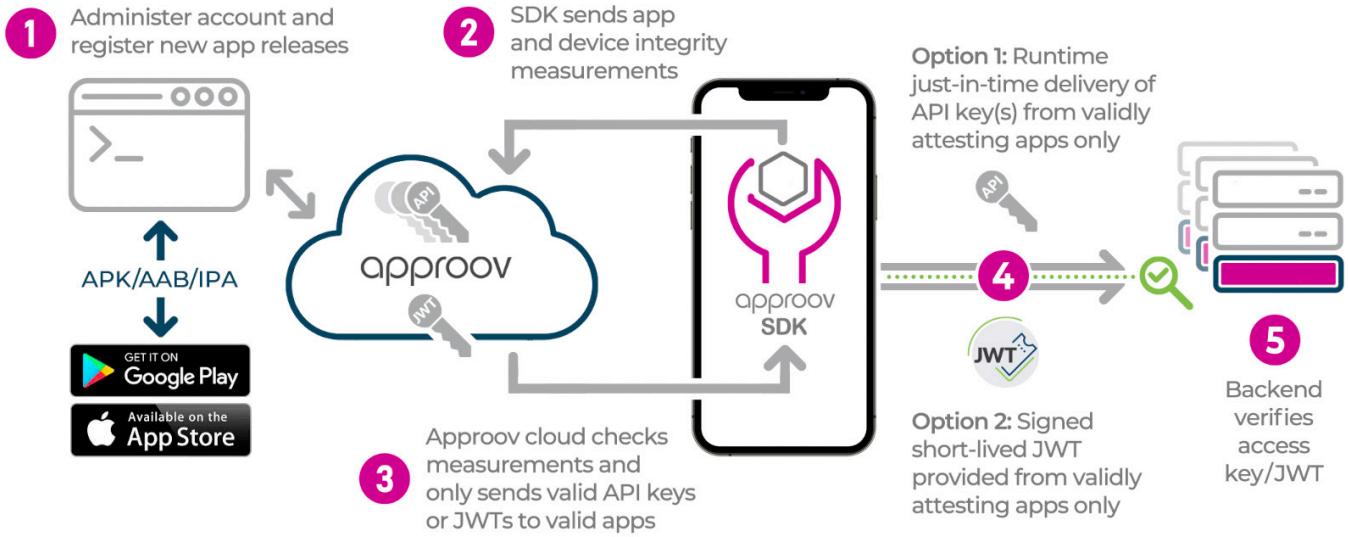
- First, implementing static certificate pinning can protect against in-channel MitM attacks and presents an enhanced level of security. Tools such as the Free Pinning Generator Tool from Approov can help make implementation straightforward. As discussed, there is some operational overhead to be managed and care must be taken to ensure service continuity when certificates change.

- Implementing a solution for dynamic pinning can eliminate the operational burden of managing static pinning. This can automate the process and make operation easier and less prone to errors. Operational risks and delays associated with updating apps when certificates change are eliminated.

- Finally, in order to eliminate MitM attacks which target the application itself or the client environment, a solution which validates both app and client environment can be deployed. Such a solution, combined with dynamic pinning can completely eliminate the risk of any MitM attacks in the mobile channel.

These approaches are tried and tested ways of reducing the threat of MitM. Some of the most-security conscious mobile application developers already use these techniques and with wider adoption, the threat of MitM attacks on mobile apps can be eliminated completely.

Appendix 1 - The Approov Solution for Mobile App and API Security



Approov provides a run-time shielding solution which is easy to deploy and protects your APIs and the channel between your apps and APIs from any automated attack. There are two options for protection. It can use a cryptographically signed and short lived "Approov token" to allow the app to provide proof that it has passed the attestation checks. Alternatively, runtime secrets can only be supplied to an app once it has passed the attestation checks, and these can include API keys for accessing 3rd party APIs without any need to modify the API backend at all.

IIntegration of the SDK into your mobile app is designed to be straightforward. A full set of frontend and backend Quick-starts are available to facilitate integration with common native and cross-platform development environments.

By ensuring only an untampered genuine mobile app running in an uncompromised environment can access the API, Approov prevents the exploitation at scale of:

- Stolen user identity credentials.
- Vulnerabilities in your apps or APIs, irrespective of whether the vulnerabilities are already known, uncovered through testing or "zero-day".
- Malicious business logic manipulation of the API.
- Man-in-the-Middle (MitM) middle attacks.

The following sections refer to the diagram above to show how Approov flow works in detail.

1. Account Administration

The Approov CLI (Command Line Interface) tool is downloaded to your development environment. It is used to access and administer the Approov account provided upon sign up. The tool is also used to register new apps that are to be released to the app stores. This is achieved by analyzing the app (in either `.apk`, `.aab` or `.ipa` format) and creating a unique signature which captures all aspects of the application and is virtually impossible to access or replicate. This unique "DNA" of the app is added to a database in the Approov cloud service for your account. No application code is stored or uploaded to the Approov service. The particular build of the app then becomes recognized as being official.

2. App Launches and Requests Attestation

At run-time prior to making the backend API call, the app requests attestation using the SDK. Normally this is done automatically as part of the quickstart integration, which will intercept calls being made to the backend API. If this is the first request for attestation, then this will initiate an integrity assessment process inside the SDK that requires communication with the Approov cloud service (see next section). Once completed successfully, an Approov token and/or runtime secrets are returned to the SDK, and may be cached by it for up to 5 minutes so that subsequent uses do not require additional network communication. However, certain events (e.g. evidence of an instrumentation framework being attached) can trigger a completely new attestation check.

3. Integrity Assessment

The integrity check process requires the SDK and the Approov cloud service to work together. The SDK analyzes the runtime environment of the app and the authenticity of the app that is being measured. These checks are implemented in hardened code and communications are protected by TLS, certificate pinning and also by a secondary level of request integrity signing. The app gathers and passes data and measurements to the Approov service. The Approov cloud service performs analysis on the data provided by the SDK and makes a decision based on this and the security policy criteria you set for your account. These policies are dynamic and can be updated in the cloud service at any time.

If the criteria are met then the Approov cloud service provides the short lived cryptographically signed Approov token and any relevant runtime secrets. For Approov tokens, options are also available to use various different signing algorithms, including those with asymmetric keys.

4. Token/Secrets Included in API Request

The obtained Approov token can be added by the Approov service to every backend API request as an additional header, such as `Approov-Token`. It is also possible to swap in runtime secrets, that are only just obtained just-in-time, into requests automatically to replace placeholder values shipped with the app. It is important that all communications made by these APIs are pinned so that no Man-in-the-Middle (MitM) interception is possible that could copy Approov tokens or runtime secrets. TLS pinning of connections is managed automatically by the Approov dynamic pinning functionality.

5. Backend Verification

The customer backend API is able to check the validity of the Approov token by checking if it has been correctly signed and has not expired. API keys that are delivered just-in-time as runtime secrets can be checked in the normal way with no need to modify the backend API at all. Invalid requests can be quickly and easily blocked, ensuring the request is coming from a genuine and unmodified instance of the official mobile app.