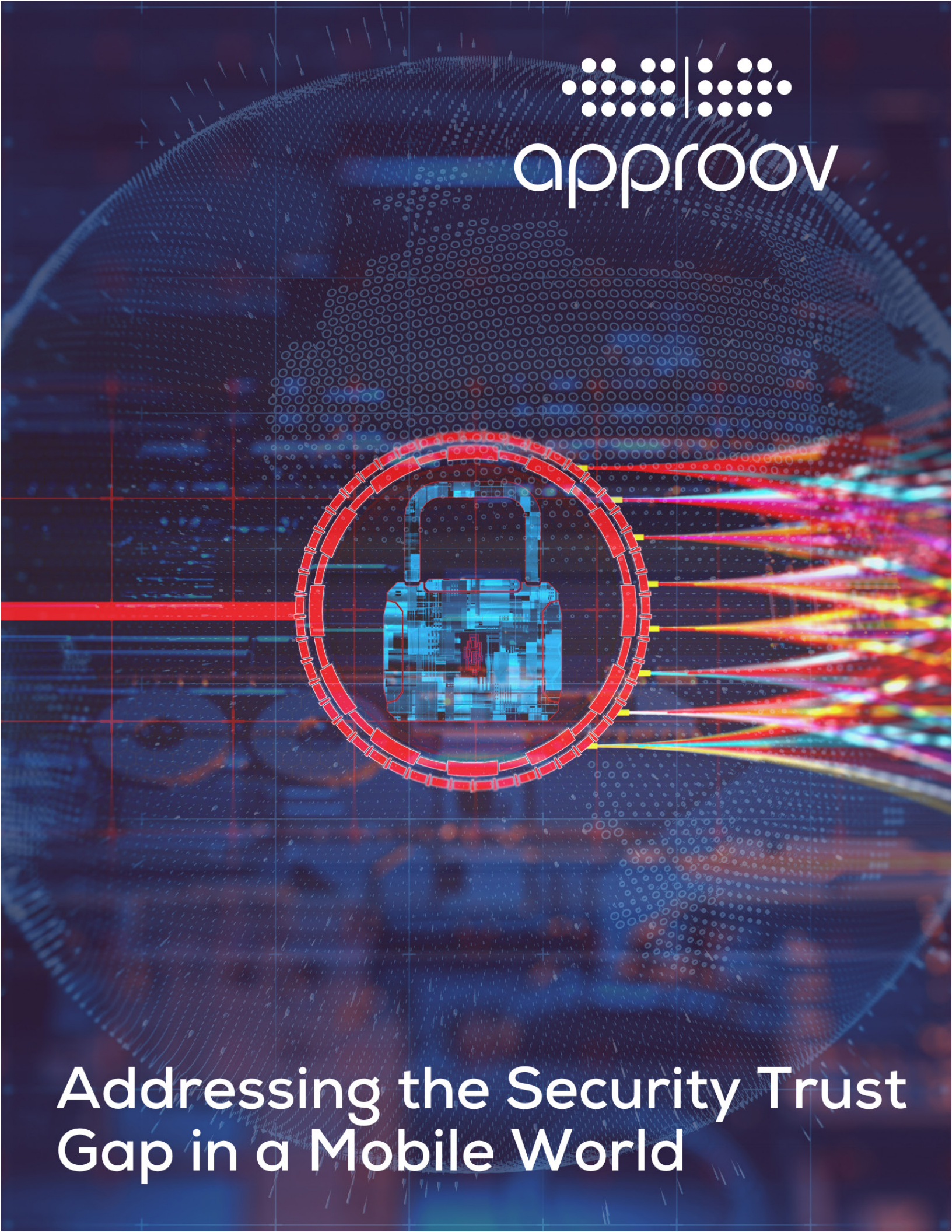
approov



Addressing the Security Trust Gap in a Mobile World

Addressing the Security Trust Gap in a Mobile World

Current security and anti-fraud measures do not adequately address the needs of the mobile app world. Sensitive data that is being shared through APIs are still subject to exploits such as app impersonation, reverse engineering of API protocols, spoofing transactions and using bots and emulators to access backend servers.

Approov creates a trusted environment that protects your APIs and your business by providing additional authentication — authenticating app instances, not users. By ensuring that the mobile app connecting through an API is a genuine, untampered instance, fraudulent transactions, malicious scripts, and bot attacks are blocked at the source.

Traditional Measures Don't Provide Enough Protection

Mobile apps have become the digital touchpoint of choice, overtaking web browser usage. This mobile, app-first, engagement model and the increased sophistication of the hackers opens up new vectors for security risk.

Many security approaches originally developed for the desktop web channel are not sufficient for the mobile API economy. With web browsers, the client side platform is inherently insecure since code must be provided in the clear and run in an untrusted environment. Therefore, security sensitive logic has always been run in the web backend. However, mobile app users expect a responsive, frictionless experience so there has been a migration of business logic and security sensitive code to the apps themselves. This has resulted in a corresponding rise in the complexity and sensitivity of the data flow between the mobile app and the APIs of supporting backend servers.

These trends have caused a concerning trust gap to emerge in the mobile app world. End-to-end encryption provides little security when one of the ends may have been compromised. A significant and growing vulnerability is the ability of criminals and fraudsters to reverse engineer API protocols and then spoof transactions as if they had been generated by a genuine mobile app. This can be achieved by tampering with the app code itself or by engineering new applications that impersonate the real app. Static credentials, encryption keys, API keys or other secrets embedded inside the app can be discovered through reverse engineering. Bad actors can then leverage them, often alongside other stolen credentials, to gain access to sensitive digital assets or disrupt normal operations by scaling malicious access using botnets and mobile emulators hosted in the cloud.

Communication traffic from a remote client to a server should not be trusted to emanate from a genuine mobile app, even if it presents its credentials and initial behavior as such. It needs to be verified as genuine. Protection focused only on the app provides little protection against this type of exploit. If app hardening or anti-tamper approaches are breached without then typically the server doesn't know about it and is powerless to block the bad traffic. A dynamic authentication protocol is needed to check the veracity of the client software itself and then communicate this status live to the server in an unspoofable way — closing the very real trust gap in today's mobile first world.

Current Approaches are not Sufficient

App security efforts have been focused on signature-based behavioral approaches and anti-tampering solutions applied to the app code. The use of Transport Level Security (TLS) to encrypt communications between the mobile app and the server is also a common security measure which does help prevent trivial tampering and eavesdropping of the data.

Certificate pinning in the app allows a mobile app to trust that it is communicating with a genuine server without a Man-in-the-Middle (MitM) eavesdropping attack. However, use of this one-way TLS does not provide a server with authentication that the client software it is communicating with is genuine. Attacker scripts are able to launch TLS sessions with the server even through encrypted channels.

Signature-based Approaches

Signature-based approaches rely on capturing large amounts of data and then applying rules based on known patterns. This attempts to differentiate between real data traffic from apps as opposed to bots or scripts attempting to spoof the traffic. The failing in this approach is its inability to prevent new styles of attack for which signatures have not been captured. Constant manual analysis and maintenance is required to keep up to date with the latest attack vectors. Moreover, certain attacks performed at a low velocity are intrinsically very difficult to distinguish from genuine traffic. In other words, this is a negative security model that enables traffic by default and attempts to detect irregular or unusual usage. A better solution is a positive security model that provides definitive authentication of the traffic up-front and at source.

Anti-Tampering Efforts

Anti-tamper techniques may be employed to make it more difficult to tamper with or reverse engineer the operation of a mobile app. The use of anti-tamper with code guards does harden the app, but cannot prevent an attacker impersonating the traffic of a real app, especially if they have been able to mount MitM analysis of the traffic being communicated. Current anti-tamper technologies can also be difficult to integrate into existing app code, can be quite invasive in the development process and can cause measurable performance degradation. Often the use of these technologies is somewhat misdirected. The digital assets of value are on the server, not in the app, so the focus should surely be on ensuring that only a genuine untampered app can access those assets by allowing the app to prove its authenticity. Trying to prevent tampering is insufficient because if it succeeds, or if a system is developed that can spoof the app communication, it is undetectable by the server.

Transport Level Security (TLS)

Mutual TLS attempts to establish two-way trust between the client mobile app and the server. This is achieved by installing a custom certificate (with a private key) on the mobile device itself. Only clients with access to the client side certificate are able to successfully initiate the mutual TLS session. Unfortunately, this approach has the same drawback as other attempts to conceal secret credentials inside the app. The app is subject to analysis and reverse engineering by attackers who are able to extract the certificate from the mobile app and then embed it into a malicious application that can successfully initiate a mutual TLS session.

API Keys

Existing approaches embed security credentials, such as an API key, into the app itself and these are relatively simple to reverse engineer by an attacker. In many cases, no attempt is made to conceal the key and widely available open source analysis tools can easily extract it. Once the key is available, it can be reused in a malicious application or script to gain access to the API. In some cases the API key or access credentials can be trivially extracted by using a MitM proxy software suite to observe the communication between the mobile app and the server. More sophisticated implementations never communicate the key directly, but use an HMAC implementation (message authentication code involving a cryptographic hash function) on the client side to show that the key is known. However since a static key still exists, it is still subject to reverse engineering. Server side mitigations may rate-limit the number of requests per second that use the same API key, but this does not fundamentally stop data exfiltration or other attacks – it simply slows down the rate at which it can happen. Hackers may probe the API endpoints to find vulnerabilities in your security logic, or to find potential code injection vulnerabilities, and this may be even more serious in terms of gaining access to your network and customer data.

Mobile apps typically access digital assets on your servers using APIs. The backend API servers may have access to sensitive corporate information and it is critical that appropriate security measures are put in place to keep them

secure.

How Approov Works

The secret to creating a secure API channel is for genuine apps to be able to identify themselves to backend servers, i.e. a positive trust model to authenticate genuine apps. In other words, the server needs to be able to trust that it is communicating with a true client mobile app, rather than with something else that is trying to impersonate typical app communication.

The Approov service uses a challenge-response cryptographic protocol allowing a server to establish the veracity of a connecting client app. This approach is not dependent upon any secret embedded inside the app and thus fundamentally different to other solutions. The integrity of the app is dynamically measured to establish what it is, not what it has in the form of a static secret. Approov authenticates app instances, not users.

App authentication provides an additional layer of protection that can be used alongside your existing authentication and authorization methods. By integrating the Approov SDK into your mobile app, the API requests it generates can include a special token in the header. A simple change in your backend API endpoint will check the presence of valid tokens for each API request and any requests not containing a valid token are simply rejected. With this protection in place a hacker can't launch scripts or individual probing requests against your API. All their requests will be immediately rejected. This is illustrated in figure 1.

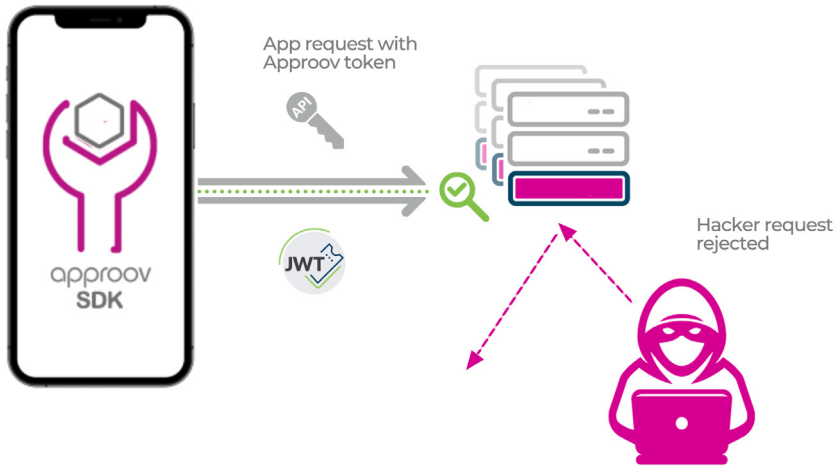


Figure 1. *Approov Protects Digital Assets Exposed by APIs*

The Approov platform is comprised of three main components:

- **A cloud service that handles requests from apps and determines the authenticity of the app.** It does this using a challenge-response protocol, built upon established and trusted underlying cryptography, which ensures a live interaction and prevents any attempts to replay a previous response.
- **A simple SDK built into the app itself.** This is easily added to the app development project and quickstart support is provided for a wide range of different app development frameworks. The quickstarts automatically substitute placeholder API key values for the real API keys (which are only delivered securely at runtime) or add an Approov token to the headers of API requests. This can be done with almost no code changes. Behind the scenes, the SDK automatically connects to the cloud service when necessary and performs the integrity check and obtains the token. This process is repeated automatically on a regular basis. The integration also pins the connections to ensure that it is not possible for a MitM attacker to insert themselves to steal the tokens or other app data.
- **If you are using Approov tokens, a simple verification check in your backend API server responsible for checking the token authenticity.** Examples are provided for a range of server side languages. The token itself is a standard JSON Web Token (JWT) format to allow straightforward integration. The authenticity of the token is guaranteed by

signing it with a secret only known to the cloud service and the backend API server.

These components are illustrated in figure 2.

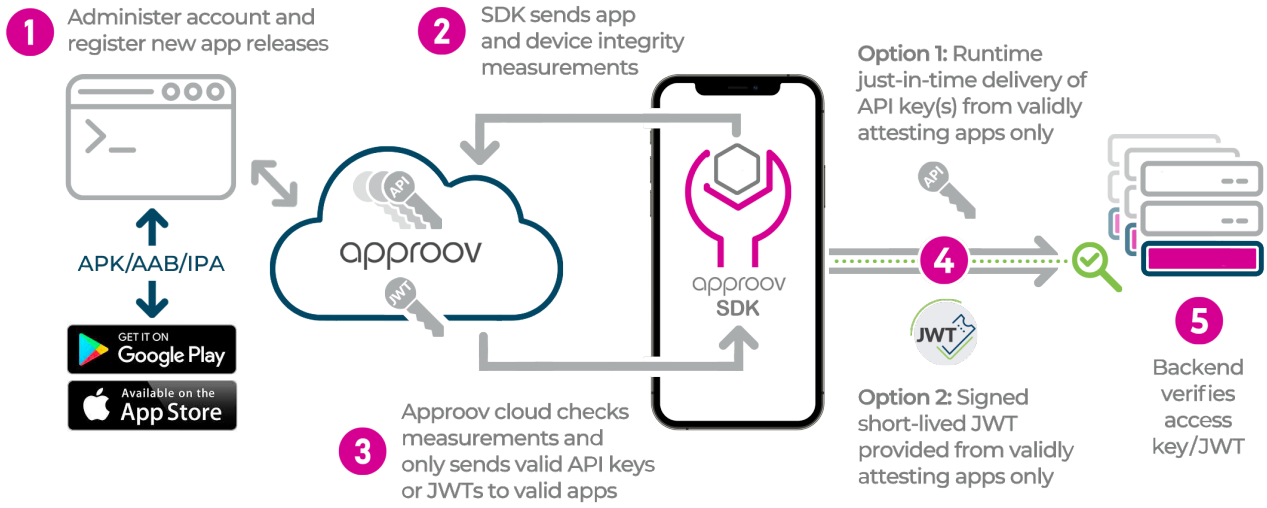


Figure 2. *The Components of Approov*

An Approov command line tool is made available for your engineers to install, control and monitor the service. This also allows registrations of valid app versions before they are submitted to the app store, so that Approov can recognize them as being valid.

By adding Approov to your security architecture, you can be sure that mobile app instances in the wild will be authenticated, on top of your regular user authentication and connection encryption, further reducing risks of application layer attacks. The Approov positive authentication model allows genuine app requests to go through while allowing you to focus on actual threats. Untrusted software agents, such as attacker scripts or modified apps, are unable to generate valid tokens and are immediately rejected. Since Approov does not rely on hiding a secret, such as a static API key, traditional reverse engineering techniques used by attackers are ineffective.

SDK and Tokens

Approov is based on the concept of software attestation. It allows your apps to uniquely identify themselves as the genuine, untampered software images you originally published. In exchange for this proof the app is granted a token which can then be presented to your API with each request. Your server side implementation can then differentiate between requests from known apps, which will contain a valid token, and requests from unknown sources. Approov does not interfere with any of the other traffic between the client mobile app and the server.

The developer interface to the Approov SDK is via an integration with the networking stack for a particular app development platform. This automatically requests runtime secrets or an Approov token, as needed, from the underlying SDK. When the SDK first requires this it begins the attestation process by requesting a random value (nonce) from the Approov Cloud Service which it uses to seed the signature hash of the integrity check. Running heavily obfuscated and defended code, the integrity of the Approov SDK itself is measured along with the rest of the app content. The combined cryptographic hash is sensitive to any change in the app or the code used to measure it. Since the hash is seeded with the nonce value it will always be unique and cannot be replayed by an attacker.

This resulting dynamic app signature is sent to the Approov Cloud Service as part of a token request along with other information about the app and the device it is running on. Since the Approov Cloud Service knows the set of registered good apps it is able to perform the same calculation as the Approov SDK. If the dynamic app signature reported by the SDK is consistent with a registered app and the other security checks are acceptable then the attestation has passed. The cloud service then responds with runtime secrets if the attestation has passed and with a signed JSON Web Token (JWT) signed with a secret associated with the customer account.

If using Approov tokens, the backend API implementation can quickly check the validity of the token using one of the standard JWT libraries available for most platforms and verify the secret provided when you signed up to the Approov service.

App Integration

Adding Approov to your iOS or Android app is a straightforward process and can be easily integrated into most development flows. Figure 3 shows the steps in the app integration process.

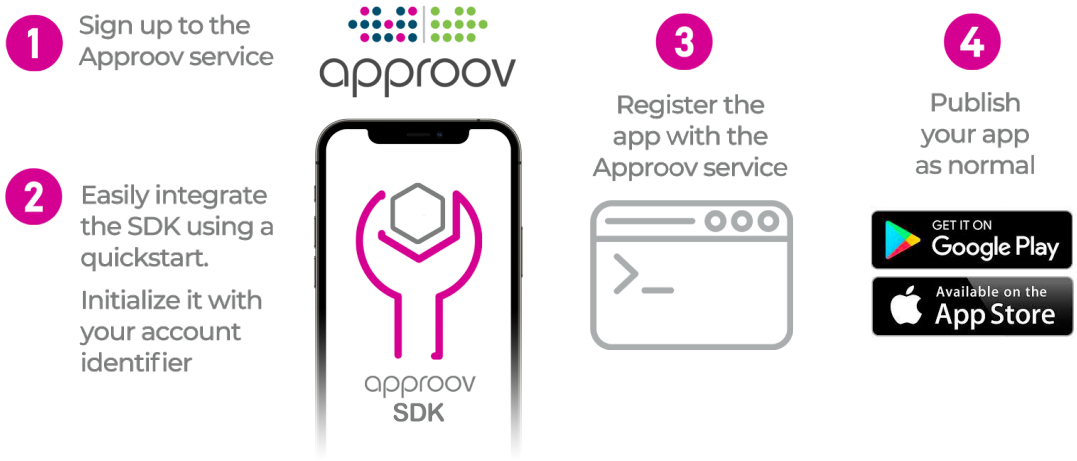


Figure 3. *Integration into an app*

When you sign up for your Approov account you will be given an initialization code to access the Approov CLI to administer your account. You will also be provided with an account identifier used to initialize the Approov SDK inside your app.

Next you integrate the Approov SDK into your app using one of our mobile app quickstarts. These operate by either hooking into or providing a modified networking handling layer that is fully compatible with the standard on. This automatically substitutes API keys or adds the Approov JWT to requests on a header and pins connections to ensure they cannot be subject to MitM attacks. Typically you will only need to change a few lines of your app's code to integrate Approov.

The app containing the Approov SDK must then be registered with the Approov Cloud Service by running the Approov CLI tool. The registration process generates the same signature for the app as the SDK does at runtime and is used as a basis for verifying that an attestation request comes from a genuine app.

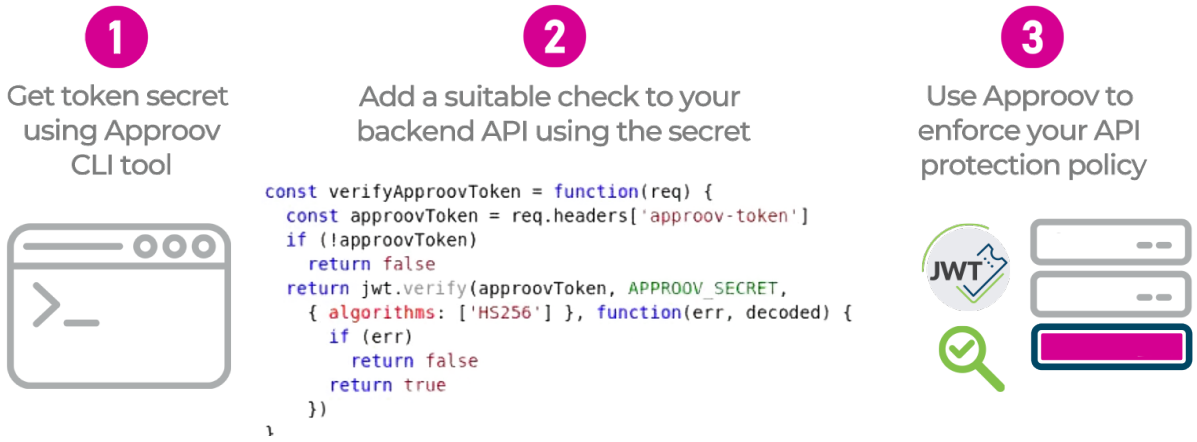
Once registered, the app can be published as normal. Since Approov checks for changes in the app's signature to detect app repackaging, it is important to repeat the registration process every time you release a new version of the app. Approov supports multiple live versions of the app and app signatures can be removed at any time via your account. This allows you to tightly control which versions of the software can access your API.

Backend API Integration

This is only needed if you wish to check a short lived Approov token in your backend. If you are only using the runtime secrets features of Approov then this is not required.

When the SDK has been integrated with your app and tokens are being added to your API request headers, you need to modify your server code to act upon this extra information as illustrated in figure 4.

.




```
const verifyApproovToken = function(req) {
  const approovToken = req.headers['approov-token']
  if (!approovToken)
    return false
  return jwt.verify(approovToken, APPROOV_SECRET,
    { algorithms: ['HS256'] }, function(err, decoded) {
      if (err)
        return false
      return true
    })
}
```

Figure 4. *Integration into the backend API*

How your API backend code treats tokens is entirely up to you. Approov gives you the flexibility to balance your security needs against API accessibility as the token check is straightforward. Since the Approov token is in JWT format it can be interpreted using libraries available for a majority of server platforms and languages. All that is required is to check that the token is signed with the secret string provided with your account.

Approov Protocol In Detail

The protocol used by a mobile app to receive a valid token is as follows. A very similar process is used to obtain runtime API keys / secrets but is not shown here:

- The app makes an API request using the Approov capable networking stack. This causes a call to be made to the `fetchApproovToken` method in the underlying Approov SDK. A check is made to see if a token has been previously fetched that has not yet expired. If not (and therefore always on the first call after an app is launched), then a new token must be obtained. To do this the Approov SDK in the app contacts the Approov Cloud Service, which then sends a challenge value that is unique to this particular session.
- The Approov SDK in the app replies with a response based on a measurement of the contents of the runtime app image in memory. This response specifically measures all of the SDK code. Data is also gathered about any hacking frameworks that may be running on the device, information indicating if the device is rooted or jailbroken, and checks for running in a debugger, emulator and various other aspects. These measurements are further protected against any tampering that may be performed on them. The cryptographic protocol and extensive code hardening techniques prevent a correct response from being spoofed.
- The Approov Cloud Service validates the app response with the various signatures that are expected from a valid app and device.
- The Approov Cloud Service sends the app an access token, which can be valid or invalid depending on the result of the verification. The app itself can not distinguish between a valid and an invalid token. The token is signed with a secret known only to the Approov Cloud Service and to your servers. This secret information is never embedded in the app or SDK so reverse engineering can never yield it. Because the token is signed, any tampering with its contents would be immediately detected.
- The app provides the token obtained via the `fetchApproovToken` call with its communication request to your server. Typically the token will be added as an additional header to API requests, alongside other access tokens such as OAuth.
- On receipt of the token, the server runs a verification check. If the token is invalid (incorrectly signed or expired) then what happens next will depend on the application area. For example, the server could drop all further communication or simply reduce the priority of responses to the client
- Since the tokens have a limited activation time (typically 5 minutes), verification must be repeated at a regular interval

while the app is running. This happens automatically as required when networking requests are made and any previously fetched token has expired, or is about to do so.

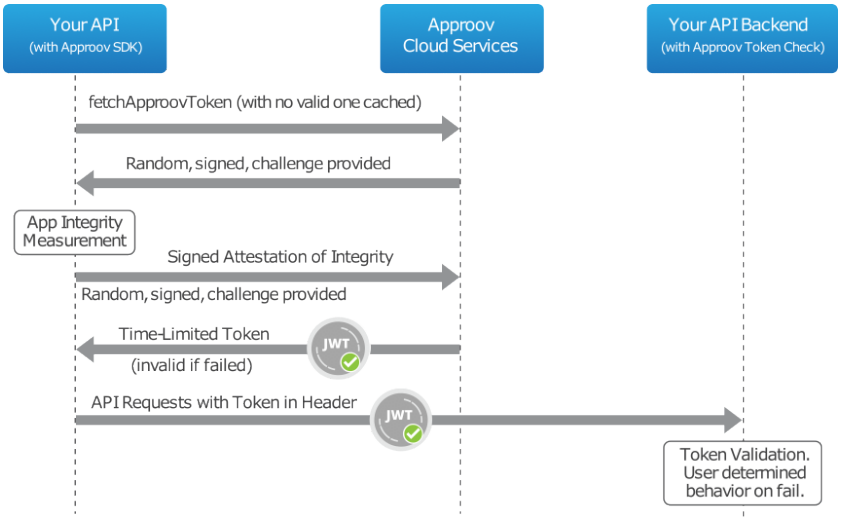


Figure 5. *App authentication protocol*

Benefits & Key Attributes

Approov has been designed for minimal impact on quality of service, user experience, and service load.

Low Overhead

The integrity checking process built into the Approov SDK is designed to be efficient and highly tamper resistant. When an authenticity check is needed, it only takes the order of 100ms of compute time, depending on the device and options used. Each authenticity check requires two endpoint transactions to the Approov cloud servers. The messages and their responses have been optimized for length so that the impact on both battery life and data usage is minimized.

Minimal Backend Latency Impact

Use of the JWT format makes checking the validity of a token presented to a server extremely efficient. Optimized libraries for dealing with these tokens are available across a wide range of server side languages. The token check does not require any communication with the Approov Cloud Services because the fact that the token is correctly signed guarantees that it was issued by the Approov Cloud Service as a result of a successful app attestation.

The tokens themselves are digitally signed using a SHA256 signature by default — a widely used algorithm with many efficient implementations available. The signature checking time is proportional to the size of the token, so we have optimized the token for length to ensure the checking can be performed very quickly. Options are also provided for other signature types, including asymmetric signing schemes using RSA and ECC.

Scalability

The Approov Cloud Service is hosted in the Amazon Web Services (AWS) cloud platform, providing high availability worldwide on a low latency network. Approov services are designed to automatically scale. If the system detects an increased level of traffic, new servers are brought online within a matter of minutes as demand increases, therefore ensuring that requests for app authentications and token issuing remain within acceptable performance parameters. We regularly test our service to ensure it can handle thousands of token issuing requests per second and can scale to support even the most popular and widely used apps.

Security

We regularly work with independent security experts to penetration test the Approov SDK and analyze the cryptographic protocol at the core of the system. This testing simulates potential attacks against the system to demonstrate that it is

only possible to obtain a valid token when using an unmodified app. This analysis has shown that it is not possible, even with considerable technical expertise and effort, to reverse engineer the Approov SDK and adapt it to pass the correct results for anything other than an authentic app.

Ease of Integration

Approov can easily integrate into new or existing apps. We provide a wide range of quickstart integrations for both the app and the backend across many popular development frameworks. When a new version of the app is released, the developer simply needs to register it with the Approov Cloud Service. There is no other impact on the app development, build or deployment processes. On the backend server side, the checking can be done with a simple JWT verification call. The entire integration process shouldn't take more than a few hours.

Application Areas

Approov can be deployed across a wide range of use cases. Various security vulnerabilities and fraud attacks have, as their root cause, an inability for a server to authenticate that the communication it is receiving is actually coming from a genuine mobile app. The following are examples of customer deployments of Approov within security and fraud contexts.

Anti-Automation

APIs provide an information rich and easy target for automated systems. The increasing size of the API economy means that more effort is being expended by the developers of these automated scraping systems to create sophisticated bots capable of executing various attacks including:

- Wholesale harvesting of any data which does not require an authentication to access
- Probing for server side vulnerabilities with malformed requests
- Automated account discovery and takeover using stolen credentials from other services
- Large scale creation of accounts to support phishing and other fraudulent activities against the service and its users

These automated systems are increasingly capable of detecting and sidestepping behavioral analysis-based approaches for protecting APIs by adapting their behavior to appear human where necessary. This forces behavioral approaches to become even more sensitive, increasing the false positive rate and reducing effectiveness. Figure 6 illustrates the threat posed by such automation.

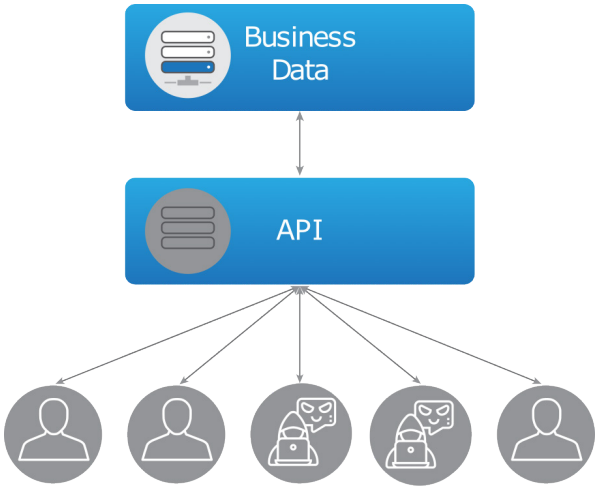


Figure 6. *Threat from attack automation*

Using reCAPTCHA style anti-bot protections to protect user accounts may hamper fully automated exploitation of a system, but systems already exist allowing them past this obstacle. Employing CAPTCHAs also creates friction and often frustration for legitimate users.

Traditional API keys are a better option, but are still vulnerable to reverse engineering once the app containing them is published.

Approov Protection provides a trusted method for apps to positively identify themselves to your API allowing better traffic filtration to backend servers. Valid apps which are registered with the Approov Service are dynamically issued a short-lived JWT which is then sent with each request to your API. Traffic with valid JWTs is from known apps and can be prioritized.

Since Approov does not rely on embedded API keys, there is nothing to be extracted and reused in automated systems and tokens will only be granted to unaltered, pre-registered apps preventing scraping by automated scripts.

This positive identification approach is transparent to users and removes the possibility of the false positives that behavioral approaches suffer from. By using a simple JWT in the header, you have the flexibility to manage traffic in whatever way best suits your business needs. You can choose to block all unidentified traffic on the API, deprioritize it or simply monitor where it is coming from. Figure 7 shows how the API is protected using Approov.

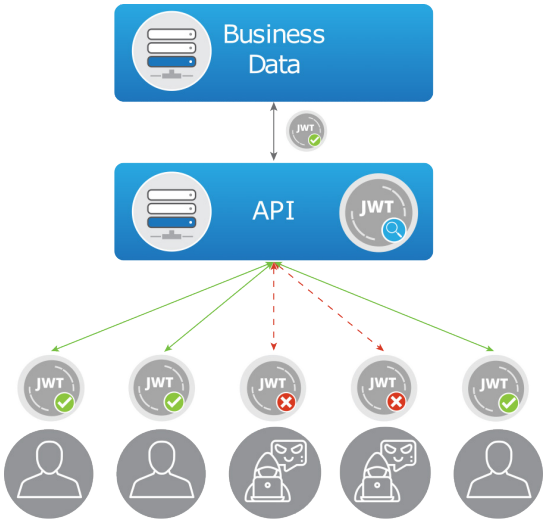


Figure 7. *Blocking automation*

API Key Protection

Useful apps are dependent on the data and services provided by multiple APIs from a range of vendors. A typical enterprise app will make use of both internal and 3rd party APIs, each with its own approach to access management and associated charges.

Most APIs require apps to present some type of valid API key with each request to allow access. Failing to protect this key from misuse can have a number of consequences:

- Paying for someone else's access to a pay-per-call API
- Key revocation due to it being used outside of terms of service
- Rate limiting due to overuse

The API keys used by apps can fall into the wrong hands by simply being extracted from the distribution package and redeployed in scripts or by accidentally being uploaded to public code sharing sites such as GitHub and Bitbucket by developers.

To address both the security and management issues around keeping API keys safe, Approov also provides a runtime secrets protection capability illustrated in figure 8.

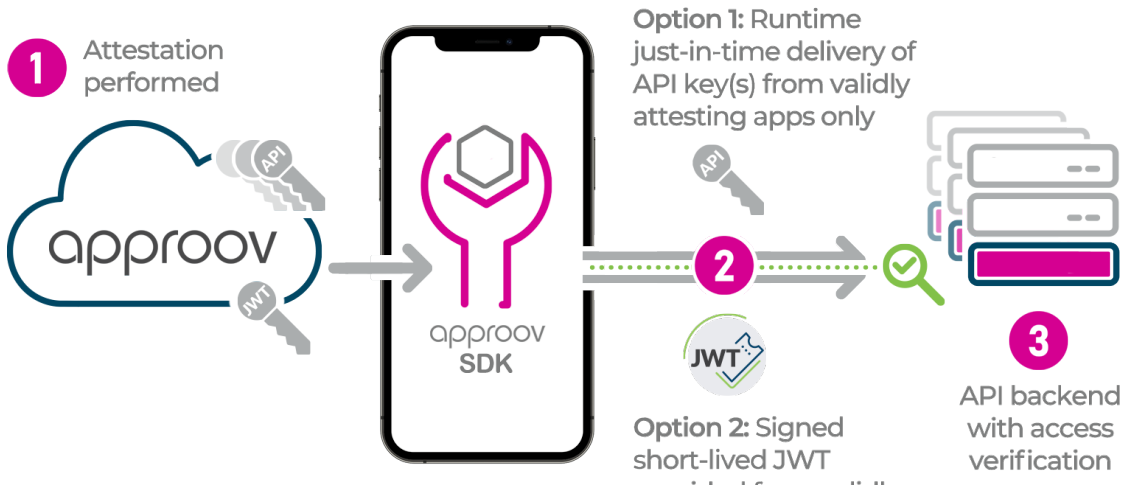


Figure 8. *Using a key store to protect 3rd party API keys*

The app undertakes the normal attestation process as described previously. This allows Approov tokens to be added to requests to API backends controlled by the same organization as the app developer. Additionally, API keys can be added into the Approov cloud using the Approov CLI. Capabilities are provided in the quickstart which allows API keys to be removed from the app package itself and replaced with a placeholder value. When an API call is made that contains the placeholder value, this is automatically substituted for the real value just in time for the call to be made. The API keys are only ever transmitted to instances of the app that are passing the authentication process.

By removing API keys from the app you make it impossible for reverse engineering. This is far more effective than using code obfuscation or app hardening techniques which aim to make keys harder to reverse engineer. This approach completely removes them from the code. As an added benefit, this approach also allows the API keys to be updated if required, without any need to update the app itself.

App Legitimacy

In some cases, an app used to access an online service may be tampered with in some way. This might be in the form of a one-off modification by an individual hacker or a tampered app being repackaged and distributed to many users. In the latter case there is the possibility that a large number of your app users are accessing your service using an unofficial client mobile app over which you have no control. Any secrets or credentials embedded inside the app will have been compromised and are being used to access the online service. Moreover, these will be the same secrets or credentials used by the official version of the app. So if you revoke these credentials to block access to the tampered app, then you will also block your legitimate users. Even if you manage this transition smoothly, the updated secrets are likely to be stolen in exactly the same way again.

The actual risks of app repackaging will depend on your particular domain, but losing control of the client app could lead to:

- Undermining brand value
- Stealing advertising revenue
- Gathering confidential user information
- Removing restrictions in the logic of the original app to allow it to do things that were never intended and are in some way damaging to your business model
- Including assets and capabilities that are normally only unlocked by in-app purchases, thus directly impacting revenue

- Tampering with the app to enable some cheating or automated play capabilities that undermine the experience of your legitimate customers (i.e. online gaming)

Some of these threats are targeted at the owners of mobile devices, but many have a significant impact on app developers. Many apps depend on ad revenue as an income stream and fake apps have a big revenue impact. More indirect effects can be on a company's reputation if user credentials are stolen or the perception is that the developers write unreliable, ad-infested or resource hungry apps. Fake apps might also gain access to services accessed by the genuine app, such as analytics, usage information or scoring for online games. This data then becomes polluted and less usable for real app users.

Approov is a way to prevent successful repackaging of apps which use web services to provide some of their capabilities. In a process analogous to user authentication, the Approov SDK integrates with the app and provides a mechanism to verify the authenticity of the code being used to access an API. By positively identifying traffic from genuine apps, attempts to use the API from repackaged apps or other unofficial clients can be blocked. Fake or tampered apps are simply unable to access any of the features provided by the app servers, as shown in figure 9.

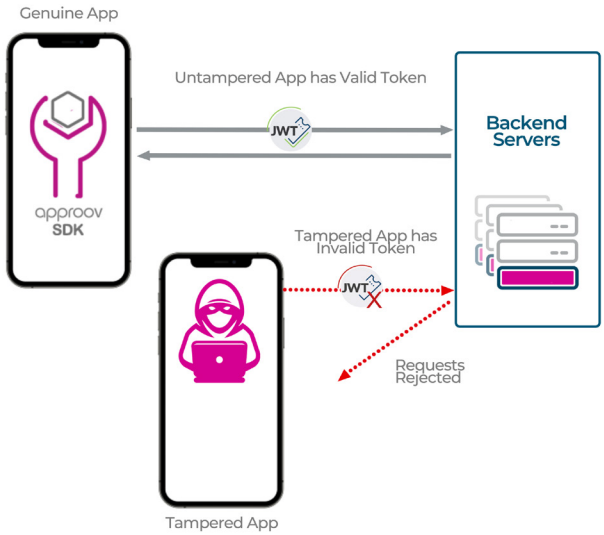


Figure 9. *How Approov restricts accesses to legitimate apps only*

Conclusion

The world is changing and traditional security measures are no longer sufficient to protect mobile API connections because they still are vulnerable to exploits such as Man-in-the-Middle attacks, reverse engineering, API key extraction, etc. They only focus on stopping an attack already in process.

Approov starts where the request originates - at the app.

Working with other security protocols, Approov helps build a true end-to-end API environment that can be trusted. By identifying, verifying and certifying that only your mobile apps, running in untampered environments and communicating over secured channels, can access your APIs and cloud services, your valuable and sensitive assets stay safe and secure.

approov
mobile security

Contact us for a free technical consultation - our security experts will show you how to protect your revenue and business data by deploying Approov Mobile Security
www.approov.io