LANDSCAPE

OVERVIEW

Mobile apps have become the digital touchpoint of choice, overtaking time spent in the web browser. This exposes a valuable new attack surface which is rapidly attracting the attentions of hackers and fraudsters. Security and anti-fraud mechanisms must be reviewed and updated in the light of the move to a mobile, app-first, engagement model and the increased sophistication of the criminals.

Many security approaches originally developed for the desktop web channel are no longer fit for purpose in the emerging world of the mobile API economy. In the web browser world 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 and there has been a migration of business logic and security sensitive code into 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 new and worrying trust gap to emerge in the mobile app world. End to end encryption provides no real 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 revealed through reverse engineering. Bad actors can then leverage these, often alongside other stolen credentials, to gain access to valuable digital assets or disrupt normal operation by scaling malicious

access using botnets and mobile emulators hosted in the cloud.

Communication traffic from a remote client to a server cannot be trusted to emanate from a genuine mobile app, even if it presents its credentials and initial behavior as such. It may be malicious and should not be trusted until proved to be genuine. Protection focused on only the app often provides little protection against this type of exploit. If app hardening or anti-tamper approaches are breached then this typically happens without server's knowledge so it is powerless to block the bad traffic. What is needed is a dynamic authentication protocol that checks the veracity of the client software itself and communicates this status live to the server in an unspoofable way. This approach closes the very real trust gap in today's mobile first world.

CURRENT APPROACHES

App security efforts have been focused on signature based behavioral approaches and anti-tampering solutions applied to the app code. Much is written about the need to use Transport Level Security (TLS) to encrypt communications between the mobile app and the server. This does prevent trivial tampering and eavesdropping of the data. With the use of certificate pinning in the app this also 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 one-way TLS does not allow a server to trust the authenticity of the client software it is communicating with. Any attacker script is able to launch a TLS session with the server and, even though such a channel is encrypted, it may be fundamentally malicious in intent.

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 issue with this approach is its inability to initially 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. What is really needed is a positive security model that provides definitive authentication of the traffic source.

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. Furthermore, current anti-tamper technologies can be difficult to integrate into existing app code, can be quite invasive in the development process and can cause measureable 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 real untampered app can access those assets by allowing the app to prove its authenticity. Trying to prevent the tampering is not sufficient because if the tampering succeeds, or if a system is developed that can spoof the app communication, then this is undetectable by the server.

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 someone with access to such a client side certificate is 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.

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 an attacker application 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 on the client side to show that the key is known, but since a static key still exists then this is still subject to reverse engineering approaches. 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 themselves may have access to sensitive corporate information and it is important that appropriate security measures are put in place around their access. In the web channel it is well known that code executing in the browser can be read by an adversary and can be manipulated. Thus the web channel tends to minimize the amount of business logic and access available to the web site, even if it is quite rich in the use of Javascript. Instead, this is handled by the web server. APIs, however, typically push more deeply into the infrastructure of the enterprise with more business logic located in the mobile app itself. This is done in the mistaken assumption that the mobile app is more strongly fortified. In reality it is the APIs in mobile apps which constitute an attack surface that has the potential to penetrate more deeply into an enterprise and thus increase the need for more sophisticated protection methods.

THE APPROOV DIFFERENCE

The breakthrough is to realize that the key issue 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 to allow a server to establish the veracity of a client app that it is connecting to. The approach is not dependent upon any secret embedded inside the app and thus fundamentally different in approach to prior solutions. The integrity of the app is dynamically measured. In other words the security rests on the app being able to establish what it is, not what is has in the form of a static secret. Approov can be described as an additional authentication factor, with some important differences: it authenticates app instances, not users, and is entirely invisible to end users.

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

rejected. This is illustrated in figure 1.

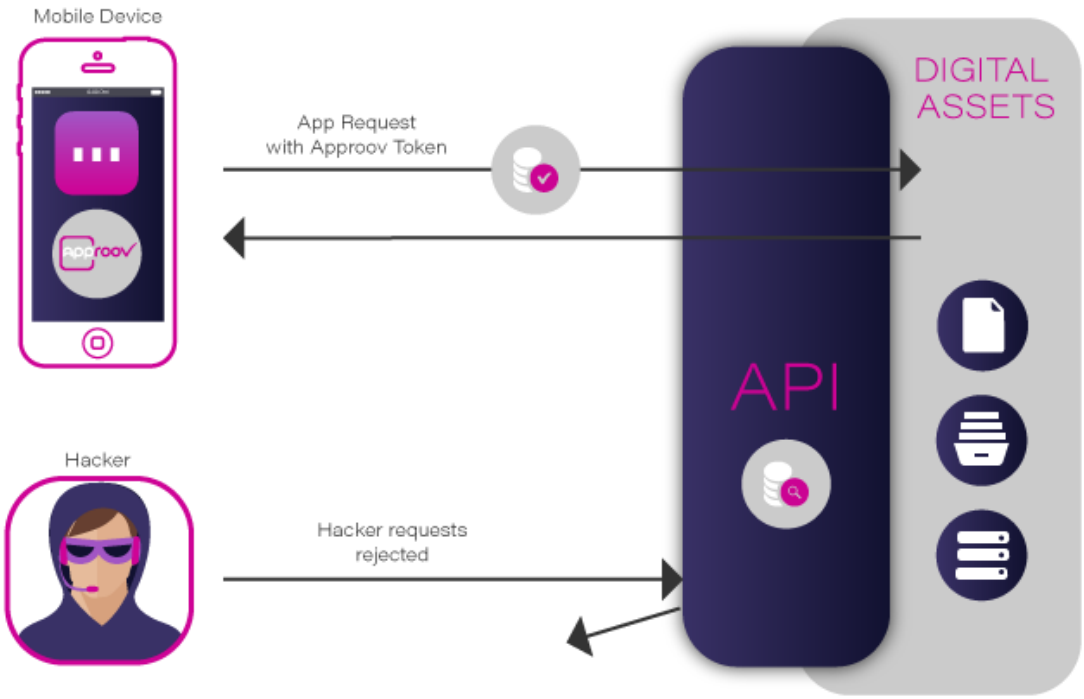

Figure 1 – Approov Protects Digital Assets Exposed by APIs

APPROOV COMPONENTS

The Approov system comprises three main components:

- A cloud service operated by CriticalBlue 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 library built into the app itself. This is easily added to the app development project and support is provided for a wide range of different app development frameworks. The SDK provides a method to obtain an Approov token that can be added to the headers of API requests that are to be protected. Behind the scenes, the SDK automatically connects to the cloud service when necessary and performs the integrity check. If the integrity check passes then the app is issued with a time limited token by the cloud service. The integrity process only needs to be repeated if the token is about to expire.
- A small amount of code to be integrated into 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.

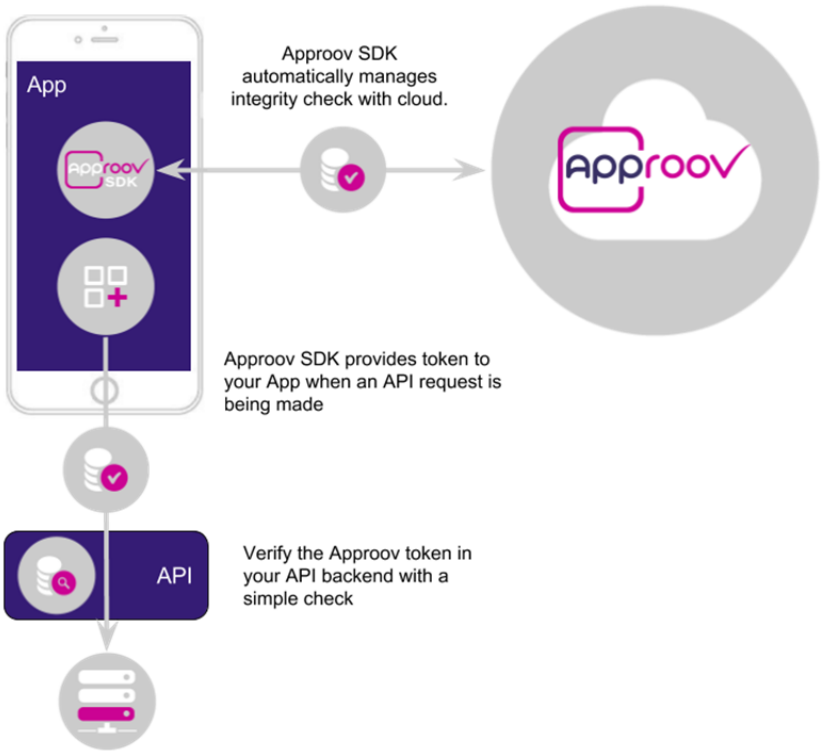

Figure 2 - The Components of Approov

An administration dashboard portal is made available for your engineers to install, control and monitor the service. A customer portal is also provided for you to verify your service plan subscription and make changes to it whenever you want. Ticket-based online support is also included in the service.

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, there is nothing to be reverse engineered by an attacker.

HOW IT WORKS

OVERVIEW

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 tokens are very quick to validate,

ensuring minimal impact on the latency of API requests made by the app.

The developer interface to the Approov SDK is a method to obtain a token. These tokens have a short lifespan of a few minutes in order to mitigate against any potential of theft. If there is currently no valid token then the SDK initiates the process of communicating with the cloud service to perform the integrity check to obtain a new token. Subsequent calls to the method made while there is still a valid token returns the cached version, resulting in very little overhead.

When the SDK requires a new token 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. The cloud service then responds with a signed JSON Web Token (JWT).

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 pass, then the issued JWT will be signed with a secret associated with the customer account. Otherwise the JWT is not signed correctly. Since the app itself does not know the secret it does not know whether it passed or failed the integrity check. This makes life difficult for anyone trying to reverse engineer the protocol.

The token is returned to your app whenever one of the `fetchApproovToken` calls is made to the SDK. Your app should then add the JWT base64 encoded string to the headers of any request to the API. Your backend API implementation an then check the validity of the token using one of the standard JWT libraries available for almost all platforms and the secret provided when you signed up to the Approov service.

APP INTEGRATION

Adding Approov API protection 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.

Once you have set up your Approov Cloud Service account, you will be able to download a copy of the Approov SDK. The SDK handles all of the interactions with the Approov Cloud Service required to establish validity of your app once deployed.

Once the SDK is built into your app, tokens can be fetched using one of the token fetch calls available via our simple API. A token can then be inserted into any suitable HTTP header field for each API request dependent on the framework you are using to communicate with your API.

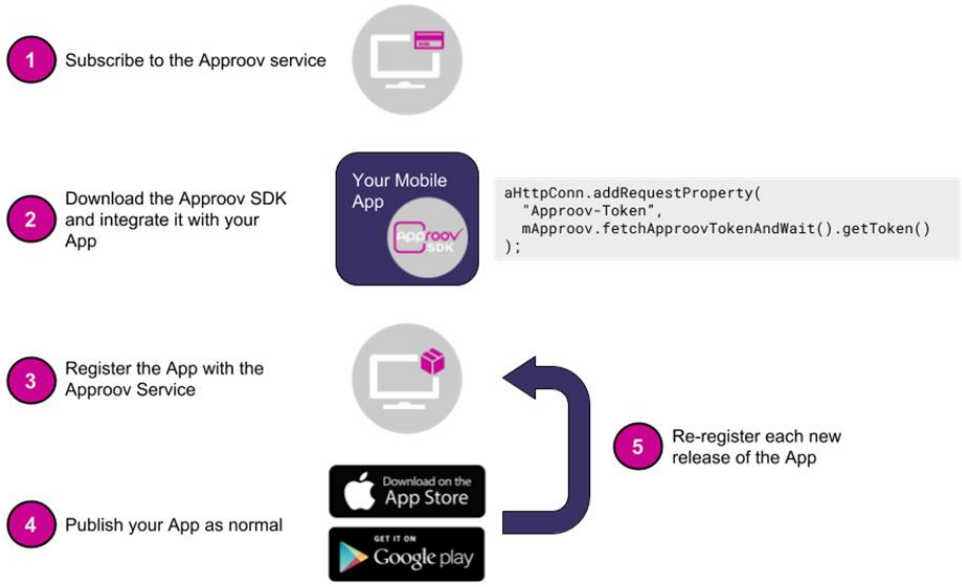

Figure 3 – Integrating Approov into an App

Once the SDK has been hooked into your app you must register it with the Approov Cloud Service. This is done by running the registration tool downloaded from your Approov account with your library. 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 with Approov, the app is 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 versions of the app being in circulation. You can remove app signatures from the Approov service at any time via your account, allowing you to tightly control which versions of the software can access your API.

BACKEND API INTEGRATION

Once 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.

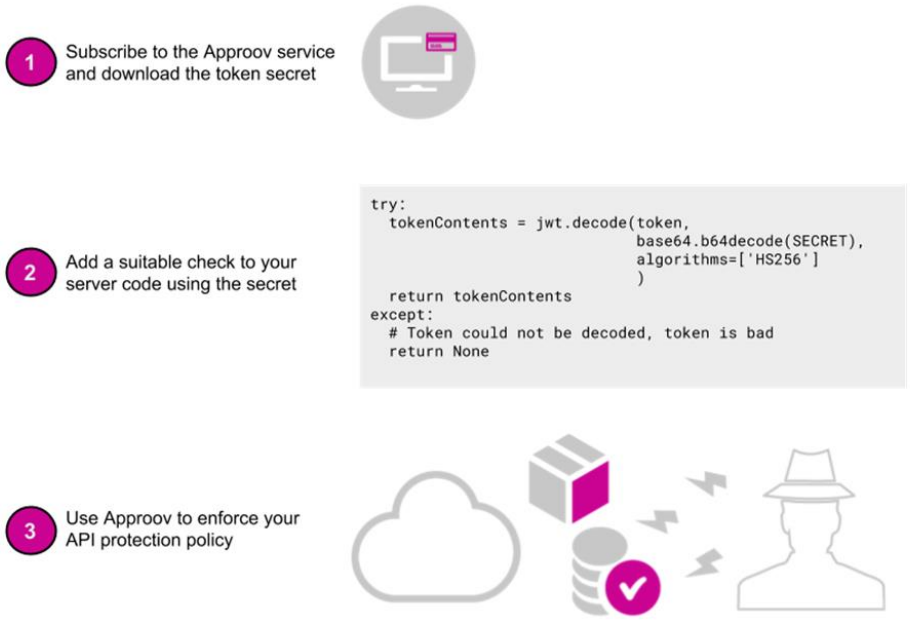

Figure 4 – Integrating Approov into the Backend API

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. Regardless of the action taken this token check is straightforward. Since the Approov is in JWT format it can be interpreted using libraries available for the vast 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 Approov account.

APPROOV PROTOCOL IN DETAIL

The protocol used by a mobile app to receive a valid token is as follows:

- A call is made to the `fetchApproovToken` method in the SDK (note that there are synchronous and asynchronous variations of this call, but the underlying protocol is identical). A check is made to see if a token has been previously fetched that has not yet expired. If not (and therefore also 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. The cloud service sends a challenge value that is unique to this particular session.
- The Approov SDK in the app replies with a response based on its runtime app image in memory. This response checks that none of the SDK code itself has been tampered with in any way and also that the overall app is one that has been registered. The cryptographic protocol and extensive code hardening techniques prevent a correct response being spoofed.
- The Approov Cloud Service validates the app response with the signature information obtained during the generation of the original unique SDK library and during the app registration phase.
- The Approov Cloud Service sends the app an access token, which can be valid or invalid, depending on the result of the verification; there's no way for the app to distinguish between a valid and an invalid token. The token is signed with a secret known only to the Approov Cloud Service and your servers. This secret information is never embedded in the app SDK library itself so no amount of reverse engineering could possibly reveal it. Since 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, perhaps alongside other access tokens such as OAuth.
- On receipt of the token the server runs the 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 lifetime this procedure must be repeated at a regular interval while the app is running. This happens automatically as required when `fetchApproovToken` calls are made. A typical token lifetime is 5 minutes.

Figure 5 depicts the process in its various phases.

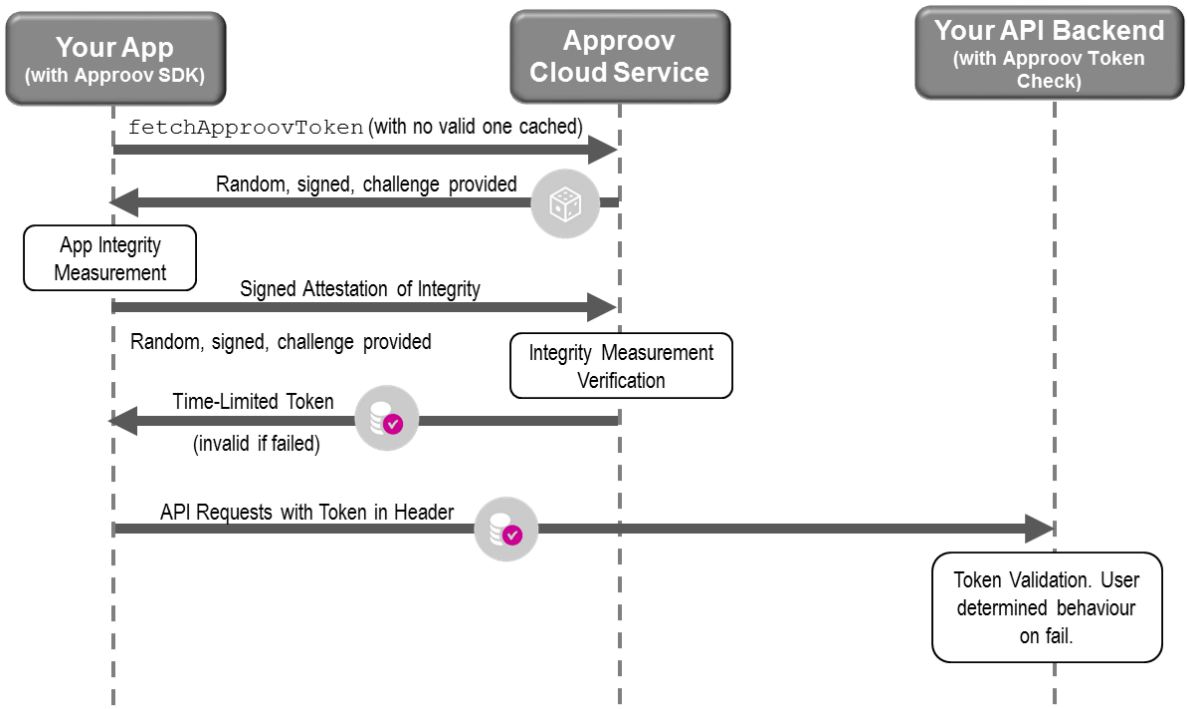

Figure 5 - The Approov App Authentication Protocol

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 as well as being highly tamper resistant. A separate thread is launched in the app to perform the integrity checking process so that it can run independently of any other activity in the app and so that the user reactivity of the app is not impacted. When an authenticity check needs to be performed this takes less than 100ms of compute time on current mobile devices so that the battery consumption of the app is not adversely

affected.

Each authenticity check requires two endpoint transactions to our cloud servers. The messages and their responses have been highly optimized for length so that the impact on both battery life and data usage is minimized.

Minimal Backend Latency Impact

The checking of the validity of an Approov token presented to a server has been made extremely efficient. We have employed the widely used JWT format for this purpose. 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 servers. The fact that the token is correctly signed provides the guarantee of authenticity that it was issued by the Approov servers as a result of a successful app attestation.

The tokens themselves are digitally signed using a SHA256 signature. This is a widely used algorithm with many highly efficient implementations and even hardware assistance instructions 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.

Scalability

The Approov Cloud Service that your mobile app communicates with is hosted in the Amazon Web Services (AWS) cloud platform. This gives us high availability worldwide with low network latency. Our services are designed to automatically scale horizontally. If we see an increased level of traffic, new servers are brought online within a matter of minutes as demand increases, thus ensuring that requests for app authentications and token issuing remain within acceptable performance parameters. We have tested our service to deal with tens of thousands of token issuing requests per second and are confident that our solution can scale to even the most popular and widely used apps.

Security

We have used 3rd party independent experts to penetration test the Approov client library and analyze the cryptographic protocol at the core of the system. This testing has simulated 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 library and adapt it to pass the correct results for anything other than the authentic app.

Ease of Integration

We have planned our service so that it is easy to integrate into new or existing apps. The SDK provides a simple interface for obtaining a token that can be added to the headers of API requests. The developer does not need to know about the details of how the app integrity measurement is done. When a new version of the app is to be released to the app store the developer simply needs to register it with the Approov Cloud Service. There is no other impact on the app development or build process. On the backend server side the Approov 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 really coming from a genuine mobile app. The following section provides various customer deployment scenarios in security and fraud contexts. Approov is a generic capability and so if you see other potential use cases appropriate to your business then we would be happy to discuss them with you.

ANTI-AUTOMATION

APIs provide an information rich and easy to traverse 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 which are evolving into highly 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 ever more sensitive, increasing the false positive rate so reducing effectiveness. Figure 6 illustrates the threat posed by such automation.

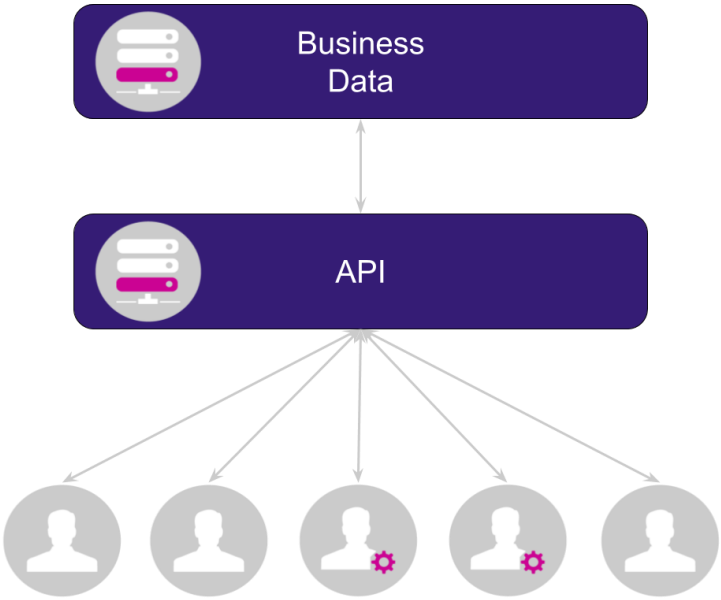

Figure 6 – Threat from Attack Automation

Enforcing access control with user accounts protected with reCAPTCHA style anti-bot protections may hamper fully automated exploitation of a system, but systems already exist for the bot to co-opt a human as required to get them past this obstacle. The friction introduced for genuine users though

might incite them to abandon a transaction or simply go to a competitor.

Traditional API keys start to provide the right kind of protection but these are vulnerable to reverse engineering once the app containing them is published.

Approov provides a robust method for apps to positively identify themselves to your API allowing you to filter traffic to your service. Valid apps which are registered with the Approov Service are dynamically issued a short lived JSON Web Token (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. The Approov service will only grant tokens to unaltered, pre-registered Apps preventing scraping by automated scripts.

This positive identification approach, built on standard JWT technology, removes the possibility of false positives that behavioral approaches suffer from and is transparent to users. 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 the API protected using Approov.

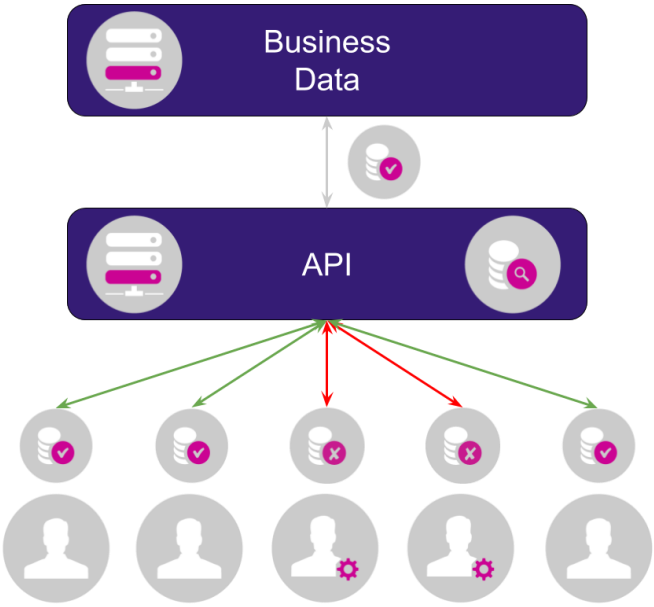

Figure 7 – Blocking Automation with Approov

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 sort 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 your apps can fall into the wrong hands in a number of ways. They may simply be extracted from the distribution package and redeployed in scripts, but it is not uncommon for keys to be accidentally 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, a common solution is to centralize use of keys into a single proxy server as shown in figure 8.

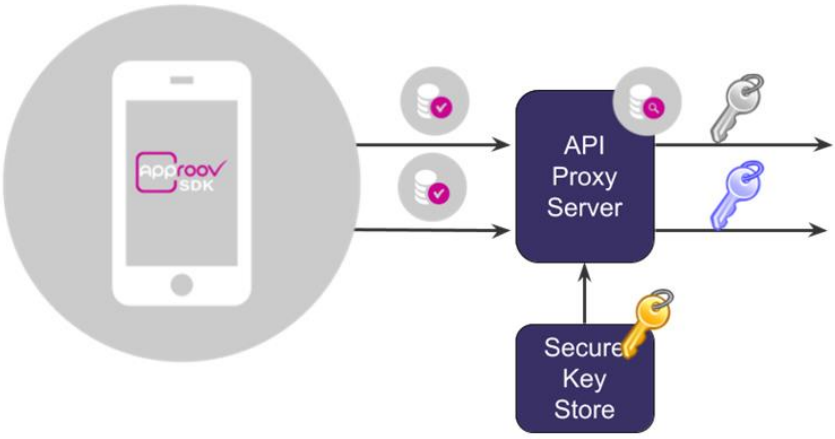

Figure 8 – Using a Key Store to Protect 3rd Party API Keys

Introducing a proxy server for all API requests across your apps can simplify deployment significantly. All API calls from all apps are directed to the proxy server instead of going directly to 3rd party (and internal) APIs. The proxy service looks up the relevant API key and adds it to the request with the appropriate headers before forwarding it onto the required API.

By removing API keys from the app you make it impossible for attackers to reverse engineer the secrets. This is far more effective than using code obfuscation or app hardening techniques. Server side assets are easier to control and manage. For heightened security keys can be encrypted at rest or even stored in a Hardware Security Module (HSM).

Development is also simplified with the use of a unified API proxy. App developers, including 3rd parties and contractors, no longer need access to API keys directly, removing issues around security and key renewal and revocation. Where only a subset of a 3rd party API is used, a simplified version of the API can be presented to developers, reducing App complexity and increasing API security.

The only weakness in this strategy is that you still need to protect the API of your proxy server. Since the issue is protecting API keys by not deploying them in an app, it does not make sense to use a traditional API key.

Instead, Approov is deployed in the client side app. This avoids all of the issues surrounding securing static API keys while providing your proxy server with a high reliability method of identifying requests from authorized apps enabling a complete solution for API key protection.

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 but there is a very real possibility that a tampered app might be repackaged and distributed to many users. In that situation you face the possibility that a large number of your end user customer base 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 this app repackaging will depend on your particular domain, but loss of control of the client app used to access the service is not a good situation. Here are some examples of what the tampered app might do:

- Undermine your brand values
- Steal advertising revenue
- Gather confidential user information
- Remove 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
- Include assets and capabilities that are normally only unlocked by in-app purchases, thus directly impacting your bottom line
- In the context of online gaming, the tampered app might enable some cheating or automated play capabilities that undermine the experience of your legitimate customers

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 an immediate cost. More indirect effects can be on 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 web 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.

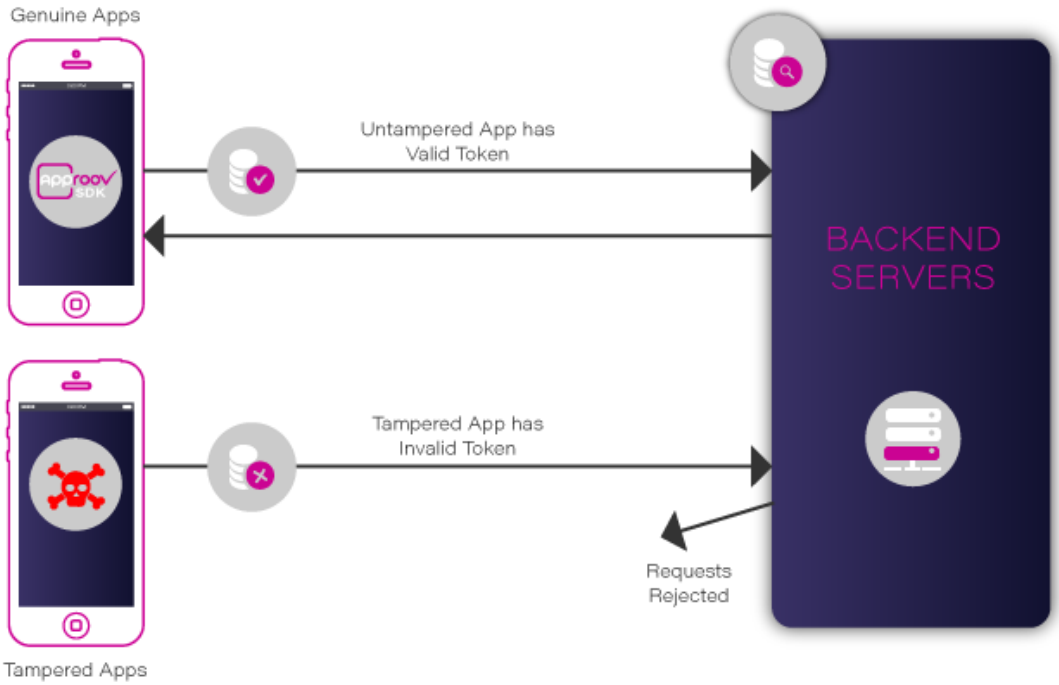

Figure 9 – How Approov Restricts Accesses to Legitimate Apps Only

OUR ADVANTAGE

With 15 years of experience in code performance optimization, code analysis, and mature binary level dynamic runtime instrumentation IP proven in the field, we know how to verify that things are working as they should across the entire software stack.

Approov enables a fundamental advance in the digital economy security ecosystem. Protect your digital assets from cyber-attacks and fraud vectors. Know that when your backend servers receive requests purporting to be from your mobile apps that this is really where the traffic is originating from. In conjunction with your existing security layers, Approov establishes the two-way trust needed to truly secure your business. We are always happy to customize our dynamic app authentication to suit your specific business needs, just contact us for the details and a free trial.

CONTACT DETAILS

info@approov.io
CriticalBlue Ltd
181 The Pleasance
Edinburgh EH8 9RU
United Kingdom