API Access Management

Okta Inc.
301 Brannan Street, Suite 300
San Francisco, CA 94107

info@okta.com
1-888-722-7871

API Security from Concepts to Components

This whitepaper describes OAuth 2.0 and how it fits into the overall landscape of authentication and authorization from both the infrastructure and software development mindset. To date, most organizations have hard boundaries between system administrators and their software development teams, where they occasionally interact, rarely coordinate, and never collaborate. While this concept has worked for decades, the assumptions, constraints, and requirements of software development have quickly changed as employees bring their own devices, partners connect in new and deeper ways, and customers expect smooth, consistent user experiences across every device. To address this ever-increasing overlap, we need to reconsider the boundaries of our systems, expectations of users, and the security policies that protect both.

How Have IT and Software Development Changed?

At a practical level, IT departments are considered a cost center with the sole responsibility of "keeping the lights on" while application development teams drive new revenue, customer retention, and the resulting growth and the information security teams have responsibilities in both areas. At the same time, the IT department is charged with writing and enforcing the security policies that the development teams must implement.

Unfortunately, most software development teams are encouraged to "move fast and break things" while the IT department and the organization customers suffer the consequences of data breaches for months and years to come. Even worse, as information security teams have tried to assert their rightful role in protecting the organization, they are treated as obstacles to be avoided and worked around. In terms of APIs specifically, whether it's a backend system, partner-facing website, or customer-facing mobile application, the development teams are sharing more data, in more ways, to more users than ever before.

API Security Goals and Approaches

As companies move to secure their APIs, their goals are the same as securing any other system or software. Fundamentally, it comes down to: 1. the right people and systems having access to (authentication) 2. the right things to accomplish their task for (authorization) 3. the shortest time necessary (least privilege).

If a company fails to do any of those three, their systems will be frustrating, unreliable, catastrophically insecure, or all three. While the mindset doesn't change with regards to APIs, the tools we have available are different.

Approach #1: No Security

While this isn't a serious approach to security, it is the most common by far. As developers build mobile apps, they believe if the API is hidden within the application then it doesn't require the same care and security that a publicly available API requires. Unfortunately, that's 100% wrong. If an API is online, it is susceptible to abuse. Being "public" or "private" is a false hope because the vast majority of data breaches occur from insiders and today's trusted partner may be tomorrow's compromised system.

Approach #2: API Keys

Most API access starts with API keys. The required logic is implemented by most frameworks out of the box so they're fast and easy to implement but not sufficiently secure. API keys are created by the developer and inherit their permissions. At first glance, this makes sense but it does not take into account the end user's permissions and what they need to accomplish. Therefore, an API key may allow read/write access, even when the use case only needs read access. Further, since the keys are at the account level, generally there is only one per account so all applications share the same over-permissioned key.

Finally, since most APIs only support a single key per account, developers often reuse keys between applications which makes automatic expiration impossible and rotation challenging. If a key is compromised, a developer leaves the team, or a simple copy and paste error in the wrong place, and the owners of all impacted applications have to coordinate a simultaneous update to minimize downtime.

API keys address Authentication but rarely address Authorization or Least Privilege.

Approach #3: OAuth 2.0

OAuth 2.0 serves as a more advanced approach to granting and protecting API access. In the simplest implementation, an OAuth token looks and acts quite a bit like an API key but with two distinctions.

1. An OAuth token inherently includes the concept of 'scoping' to enable API designers to grant fine-grained permissions to applications. For example, a simple logging application could have a token for read-only access while a different application would have a different token with different access.
2. An OAuth token is designed to expire and therefore has a refresh process built into the specification.

As a result of these two aspects, if a token is compromised, we get three benefits over API keys:

- The token will have permissions limited to the specific use case potentially making it worthless for attacks on other parts of the system.
- The token automatically expires, so the timeframe for an attacker to execute an attack is limited.
- Tokens can be revoked with a simple API call, blocking access immediately.

Regardless, while OAuth 2.0 is a much better solution, it is still not a complete solution for securing APIs because while it addresses the original three points of Authentication, Authorization, and Least Privilege, we haven't considered how to protect the API itself. Just like any other system in our infrastructure, we have to defend our API from malicious users and poor software regardless of how they approach it. A great lock on the front door isn't sufficient if we leave the windows open.

Approach #4: API Gateways

No matter what, malicious users and compromised applications will attempt to misuse and abuse your API. To protect an API's infrastructure, one of the greatest tools available is an API gateway. From an enterprise architect's perspective, a gateway can serve as an organization-wide design and orchestration tool to connect any API to all other APIs. From a developer's perspective, a gateway can serve microservice-specific systems and be included directly in a continuous integration system for seamless deployment. Regardless of the vendor or project, they all serve as an API "firewall" to protect APIs from malicious data, incorrect requests, and denial of service attacks.

In general, API gateways include simple API key creation and management. A select few go further and offer embedded OAuth servers using simple user profiles. This creates a powerful combination where developers can both protect their APIs with API gateways and implement fine-grained access control for their users. Where this falls short is that it creates yet another place to store, maintain, and authenticate users. You can synchronize user profile fields but as the user's information, behavior, and potentially subscription changes, building authorization policies based on those aspects become important. For high security use cases in Open Banking, those advanced policies are vital.

Therefore, an API gateway can support our Authentication and Least Privilege requirements, but their real strength is in enforcing an Authorization policy long before a request touches your API.

Approach #5: API Gateway and API Access Management

The distinction between a trusted user and a suspicious user is not just who they are but who they are and what they are trying to accomplish. When you use an HR system's API to download your vacation history, the risk and consequences are minor. Using that same API to change your direct deposit information is risky if not potentially catastrophic and therefore should require tighter restrictions with elevated permissions. This is where an API gateway combined with API Access Management create a powerful solution.

Okta's API Access Management is built on Okta's Universal Directory which allows Single Sign-On and Authorization Policies that limit particular OAuth scopes to specific devices, a specific network, and even group membership. Further, specific scopes can require user consent to ensure the user explicitly authorized access for the application. Most importantly, a security team can manage those policies outside the API gateway while centrally logging access requests, grants, and policy changes. For additional compliance reasons, access information can also be viewed via the Okta UI or exported to a third-party system (such as SIEM/ticketing systems). This brings APIs out of the realm of "shadow IT" and back to trusted, known systems.

An API gateway combined with API Access Management ensures that the right people have access to the right resources to accomplish their task for the shortest time necessary.

Summary

According to Gartner, APIs will be the most common attack vector by 2022. Unfortunately, we're already seeing the leading edge of that as the sheer volume of business-critical capabilities are provided by underprotected APIs. Therefore, without a deliberate, focused effort on protecting your systems now, that timeline may be optimistic.

API keys are only a starting point. An API Gateway and OAuth provide a better, more powerful solution but a centralized point of control with closely monitored policies and context-aware access management is the best solution of all. Today's trusted partner may be tomorrow's compromised system letting attackers mimic legitimate users. We need the flexibility to adjust, respond, and protect our systems based on the full context of the user and their goals.

For Further Reading

- Okta's guide to Building Secure APIs [book] [website]
- OAuth 2.0 Simplified from Aaron Parecki
- Okta Integration Network: API Gateways
- Recommended Practices for API Access Management


okta