Token Lifetimes and Security in OAuth 2.0: Best Practices and Emerging Trends

By Heather Flanagan (Spherical Cow Consulting)

To comment on this article, please visit our GitHub repository and submit an issue.

Table of Contents

Abstract

In networked and federated systems using OAuth 2.0 and OpenID Connect, tokens are essential for securely communicating between human and non-human entities without requiring the constant revalidation in every request. The improper management of these tokens, however, can expose systems to serious threats, such as token replay attacks.

While the most secure practices involve real-time risk-based token revocation and sender-constrained tokens, organizations may not be ready for that, given the maturity of their security infrastructure. Developers and architects should consider the security benefits of using short-lived, narrowly scoped client-bound tokens designed to minimize the risk of misuse. By limiting the lifespan of tokens or binding them to specific clients (as in sender-constrained tokens), organizations can significantly reduce the attack surface and thwart token replay attempts. Identity practitioners must be aware of the limitations of long-lived tokens as well as the best practices for securing token-based systems.

Introduction

Identity systems and the use of tokens go hand in hand; their existence is ubiquitous, and the need to manage them carefully is at the core of identity security practice today. But what is a token?

A token is a digital object that can represent a set of claims or attributes about an entity, such as a user, device, or process, typically used in authentication and authorization protocols.[i] Tokens are often cryptographically signed and/or encrypted to ensure their integrity and confidentiality, preventing tampering and unauthorized access. Tokens are commonly employed in various identity and access management systems, enabling a secure mechanism for proving identity, delegating access, or asserting permissions across networked systems.

Tokens may be differentiated based on whether they require the server to maintain session information. A token may be stateless and only carry encoded information (e.g., claims, user roles, and expiration times) within the token itself, such as JSON Web Tokens (JWTs). This enables a more scalable architecture by reducing demand on the server. Alternatively, a token may be stateful, at which point the contents are always opaque to the client and do not inherently “contain” information. (A stateless

token can be opaque as well, if encrypted.) Instead, their meaning is tracked by the issuing server, which maintains the state (e.g., session data, user permissions) associated with the token.

Tokens are also described by how they are used. Tokens can be classified as bearer tokens, which are usable by anyone in possession of them, or client-bound tokens (sometimes called sender-constrained tokens (e.g., Demonstrating Proof of Possession (DPoP)),[ii]), which are bearer tokens cryptographically tied to a specific client. In the OAuth 2.0 Framework, both client-bound tokens and bearer tokens can be used for authorization. Tokens may also be short-lived (expiring within a few minutes or hours) to minimize the risk of misuse or long-lived (remaining valid for extended periods), with the former offering stronger protection against token replay attacks. However, while the OAuth 2.0 framework allows these options, implementation decisions have a meaningful impact on security – which is why various Working Groups have emerged to develop profiles of OAuth 2.0 that are appropriate for use cases requiring heightened security.[iii]

One of the most effective strategies when issuing tokens is to narrowly scope their permissions. A narrowly scoped token is designed to grant access to only a specific resource or a limited set of actions rather than offering broad, unrestricted access. Narrowly scoped tokens limit access to specific resources or actions, reducing the risk of unauthorized access if a token is compromised. This limitation contrasts with broader scopes, which can provide access to multiple services or actions with a single token. Using narrowly scoped tokens, combined with short expiration times, ensures that even if a token is compromised, the damage is limited to a small portion of the system, reducing the risk of unauthorized access to critical resources.

The term ‘credential’ is often used interchangeably with ‘token’; unfortunately, that’s technically incorrect. While tokens and credentials are often related, they serve distinct roles. Credentials are used to authenticate the identity of a user or system, while tokens in OAuth 2.0 are issued after authentication to authorize specific actions. Before the broad adoption of token-based protocols, you may recall that websites would simply ask for (and store) your credentials: they would log in on your behalf and access any data they deemed necessary. This is called ‘screen-scraping.’ By limiting the scope, context, and duration of the tokens they grant, authorization servers protect users (and their credentials) from unscrupulous actors on the internet.

While tokens provide a convenient and flexible way to authorize users, they can also introduce significant security risks if not properly managed. Ideally, organizations are implementing real-time risk-based token revocation.[iv] However, considering the correct lifetime for a given token is necessary for organizations that do not have integrated security management tools.

This brings us to this article and the complicated family of tokens in the OAuth 2.0 framework. OAuth 2.0 is a set of specifications, defined in the Internet Engineering Task Force (IETF) that allow a client to request a set of scoped tokens to enable access to resources such as APIs. Understanding the different types and uses of tokens will help developers and identity and access management professionals understand the implementation considerations they need to consider in their environments.

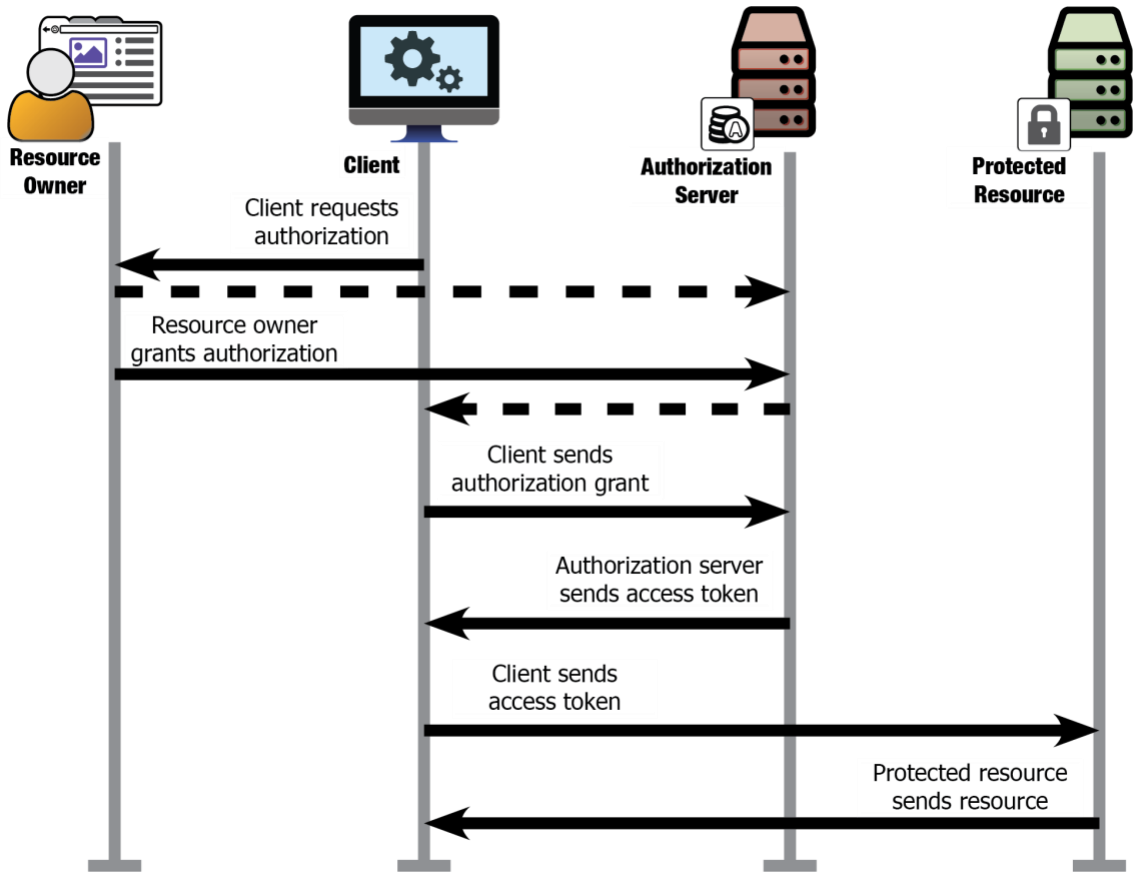

Figure 1: "The OAuth process, at a high level" – reproduced with permission from OAuth 2 in Action[v] by Justin Richer and Antonio Sanso.

Terminology

- Token: A digital object that can represent a set of claims or attributes about an entity, such as a user, device, or process, typically used in authorization protocols
- Credential: Credentials are used to authenticate the identity of a user or system, while OAuth 2.0 tokens are issued after authentication to authorize specific actions.

- Bearer Token: According to RFC 6750, “The OAuth 2.0 Authorization Framework: Bearer Token Usage,” a bearer token is a security token with the property that any party in possession of the token (a "bearer") can use the token in any way that any other party in possession of it can.[vi]
- Client-bound Token: Tokens that are tied to a specific client or device, ensuring that only the client to which the token was issued can use it. This is not a formally defined term in the specifications, but you can learn more in RFC 8705, “OAuth 2.0 Mutual-TLS Client Authentication and Certificate-Bound Access Tokens”,[vii] and RFC 9449, “OAuth 2.0 Demonstrating Proof of Possession (DPoP)”.
- Refresh Token: According to RFC 6749, “The OAuth 2.0 Authorization Framework,” a refresh token is a credential used to obtain access tokens without requiring the resource owner to reauthenticate.[viii]
- Sender-constrained Token: Tokens that require the sender to prove that they are the authorized holder of the token when making a request.
- Token replay attack: A cybersecurity attack that occurs when an attacker intercepts valid tokens—such as tokens or session IDs—during transmission and reuses them to impersonate the legitimate user or system.

Defining “Short” versus “Long”

In OAuth 2.0, two types of tokens are issued during the authorization process. The access token tends to be short-lived and is scoped for the resources where it will be used. The refresh token tends to be longer-lived and is only used with the authorization server to obtain new short-lived access tokens. Access tokens are typically created during the authorization process after the user successfully authenticates. In contrast, refresh tokens are issued alongside access tokens and allow new short-lived access tokens to be generated without requiring the user to reauthenticate. Access tokens are meant for authorizing specific API calls, while refresh tokens are used only with the authorization server to obtain new access tokens. This mechanism ensures that users don’t need to log in again for subsequent access, but periodic reauthentication might still be necessary, depending on the security requirements.

How you define “short” versus “long” token lifespans will vary depending on the sensitivity of your use case. For example, NIST SP 800-63B specifies that for Authenticator Assurance Level (AAL) 1, “reauthentication of the subscriber SHOULD be repeated at least once per 30 days during an extended usage session, regardless of user activity.” This guideline suggests that even with mechanisms like refresh

tokens, reauthentication should occur periodically to maintain the security of the session."[ix]

The Role of Short-Lived Tokens in Security

Short-lived tokens are a fundamental tool in enhancing the security of modern authentication and authorization systems. These tokens are designed to expire after a brief period—usually minutes or hours—thereby minimizing the window of opportunity for attackers to exploit them if they are intercepted or compromised. By limiting their lifespan, short-lived tokens reduce the risk of token misuse and improve the overall security posture of a system. Short-lived tokens significantly mitigate risks associated with token replay attacks and unauthorized use, but they must be part of a broader security strategy, including token binding and revocation mechanisms.

Although this article focuses on short-lived tokens as a security best practice, emerging standards like Continuous Access Evaluation Profile (CAEP) aim to extend token lifetimes in controlled ways, reflecting a shift toward risk-based token management. [x]

The use of short-lived tokens is recommended in numerous security standards and guidelines. For instance, NIST SP 800-63C emphasizes the importance of using short-lived, narrowly-scoped tokens for federated identity systems to minimize risks.[xi] Similarly, BCP 225 outlines best practices for JSON Web Tokens (JWTs), including the recommendation to limit the lifetime of tokens to reduce their exposure to attacks.

The standards also encourage the combination of short-lived tokens with other security mechanisms, such as client-bound tokens or token binding, which ensures that tokens can only be used by specific clients, further enhancing their security, or sender-constrained tokens, which require the sender to prove that they are the authorized holder of the token when making a request.

Reduced Attack Surface

One of the big benefits of short-lived tokens is the reduction of the attack window (i.e., the length of time that an attacker could exploit endpoints). In traditional token-based systems, tokens with long expiration times are particularly vulnerable to token replay attacks, where an attacker captures a token and reuses it to impersonate the legitimate user. Short-lived tokens mitigate this threat by limiting the time in which

a token can be used. Even if an attacker manages to steal a token, its utility is severely restricted because it will soon expire, rendering it invalid.

For example, in the context of OAuth 2.0, access tokens should be issued with a short lifespan, after which the client must request a new token using a refresh token.[xii] This approach ensures that even if an attacker intercepts an access token, they cannot use it for long.

Enhanced Security Through Short-Lived Token Rotation

Short-lived tokens play a crucial role in reducing the impact of token compromise by enforcing frequent token rotation. With short expiration times—typically ranging from minutes to hours—these tokens minimize the window of opportunity for attackers. If a short-lived access token is intercepted, its limited lifespan means that it will soon become invalid, rendering it useless to the attacker. This frequent renewal of access tokens forces attackers to be more persistent and repeatedly intercept new tokens, which increases the likelihood of detecting suspicious activity.

In contrast, long-lived tokens remain valid for extended periods, such as days or even months. If compromised, they provide attackers with prolonged access to protected resources, increasing the risk of unauthorized activity. By reducing token lifespan, organizations can significantly limit the potential damage of compromised tokens, ensuring that access is tied to a shorter, more controlled time frame.

However, it is important to consider the role of refresh tokens, which are used to obtain new access tokens without requiring the user to reauthenticate. While refresh tokens often have longer lifespans, they must be protected with robust security measures. If an attacker gains access to a refresh token, they can continue to generate new short-lived access tokens, potentially maintaining unauthorized access over time. There must be additional protections to prevent or mitigate refresh token replay, such as refresh token rotation upon every use, client binding, real-time risk-based token revocation, and secure storage of refresh tokens. Together, these protections prevent misuse and ensure that short-lived token strategies remain effective.

Security of Refresh Tokens

While access tokens are short-lived and may be in API requests or during authentication flows, refresh tokens are stored securely on the client. This safe storage is critical because refresh tokens are valuable; if they are compromised, the attacker can maintain access far beyond the expiration of the original access token.

Systems that use refresh tokens should implement several security measures to mitigate the risk:

1. Refresh Token Rotation: Upon each use, the refresh token should be reissued to limit the threat of any one token being captured.
2. Client Binding: Refresh tokens should be cryptographically bound to a specific client or device. This secure binding means that even if an attacker steals both the access token and the refresh token, they cannot use the refresh token from a different device or application. The refresh token will only work on the client it was originally issued to.
3. Reauthentication: Some systems require users to periodically reauthenticate before issuing new refresh tokens. This action adds a layer of security by forcing legitimate users to prove their identity again, limiting the window during which a stolen refresh token can be used.
4. Token Revocation: If suspicious activity is detected—such as refresh tokens being used from an unfamiliar device or location—the system can revoke both the refresh and access tokens. This revocation forces a reauthentication and cuts off any further unauthorized access.

Summarizing the Security Implications

- If only the access token is compromised and not client-bound, the impact is limited to the token's short lifespan. The attacker can use the token for a limited time, after which it becomes useless.
- If both the access token and the refresh token are compromised, the attacker can maintain long-term access by continuously refreshing the access token. In this case, additional security measures such as client binding, reauthentication, or token revocation become crucial for mitigating the damage.

Support for Scalable and Stateless Architectures

Short-lived tokens are particularly well-suited for stateless architectures, such as RESTful APIs, because they reduce the need to maintain the server's session state. When a token is short-lived, the server may not need to track session data over an extended period, which simplifies the architecture and reduces the potential attack vectors related to session management.

In microservices architectures, where services communicate frequently over APIs, short-lived tokens play a critical role in authenticating and authorizing requests between services. Their limited lifespan ensures that, even if an attacker manages to access a token from one microservice, the time available for them to use it is

minimal. The scope of the token is also critical to its security impact. A token representing only the client's identity can expose a broader range of resources. However, if the token carries user-specific information with a narrowly defined scope, the potential attack surface is significantly reduced.

Another option for microservices, currently under discussion within the IETF, is the use of transaction tokens.[xiii] These tokens allow workloads in a trusted domain to maintain the user identity and authorization details of an external request, such as an API call, across all workloads involved in processing that request.

Security Risks of Long-Lived Tokens

While short-lived tokens are not inherently more secure in stateless architectures, their design aligns with the scalability and efficiency needs of modern systems, reducing the attack surface by limiting token lifespans. The persistence and vulnerability of tokens can open doors for various attack vectors, particularly token compromise and replay. Let's look at a few critical security drawbacks associated with tokens whose lifetimes are long enough to be more easily used by an attacker.

Long-Lived Tokens

Common types of long-lived tokens, such as API keys and session tokens used in mobile apps, often remain valid for extended periods—sometimes days, weeks, or even indefinitely. API keys are often used for backend services that need to authenticate to external services, while session tokens maintain user sessions across mobile app interactions. More to the point, API keys are used to authenticate programmatic requests, while session tokens help maintain user sessions across multiple interactions. These tokens reduce the burden on users or applications to re-authenticate or refresh their credentials frequently, offering convenience and stability.

However, this also creates a significant security risk. If these long-lived tokens are compromised, attackers can use them to gain persistent unauthorized access to systems or resources without being detected for an extended period.
In environments where tokens are not frequently reissued or refreshed, the chances of interception or theft increase. This possibility makes long-lived tokens particularly attractive targets for attackers, as they allow for sustained access once compromised. Without mechanisms like token revocation or detection of replayed tokens, such breaches can go unnoticed, giving attackers ample time to explore and exploit vulnerabilities within the system.

Token Replay Vulnerabilities

One of the most concerning security drawbacks of some tokens is their susceptibility to token replay attacks. In a token replay attack, an attacker intercepts a token (through manipulator-in-the-middle attacks, session hijacking, or other methods) and reuses it to impersonate a legitimate user.[xiv] Tokens that are not tied to a specific client or have long expiration times are easy targets for replay attacks.

For example, in a system using long-lived tokens, an attacker who successfully intercepts an access token could reuse it repeatedly until it expires, giving them ongoing access to sensitive resources. Without mechanisms like token binding or short expiration times, little can prevent these replay attacks from succeeding. According to NIST SP 800-63C, tokens not bound to a specific client or have inadequate expiration times are a critical weakness in federated identity systems.

Challenges with Revocation

Revocation is another significant weakness of traditional long-lived tokens. Once issued, long-lived tokens, session cookies, or API keys are often difficult to revoke in real time. If a token is compromised, administrators must manually revoke it or wait for it to expire, creating a time window in which an attacker can continue to exploit the token.

Many systems that rely on older OAuth token specifications do not have effective mechanisms for real-time token revocation or monitoring of credential usage, which can lead to prolonged security incidents. While traditional long-lived tokens pose revocation challenges, frameworks like CAEP are redefining how tokens can be managed dynamically, enabling longer lifespans under strict monitoring and policy enforcement. Organizations that are taking advantage of CAEP may extend token lifetimes, requiring a refresh only after an event that triggers policy enforcement.

Increased Attack Surface

Long-lived tokens can increase the attack window for an organization, especially when they are not tightly scoped or frequently refreshed. Because these tokens remain valid for extended periods and may be used across multiple devices or services, a compromised long-lived token can give attackers more time to explore and exploit vulnerabilities within a network. This is particularly true when the token allows access to multiple services or resources without requiring frequent revalidation.

However, refresh tokens are typically not issued in most server-to-server interactions, and short-lived access tokens are preferred to minimize the time a token remains valid. This restriction limits the window of opportunity for attackers to misuse a compromised token. In contrast, long-lived tokens, like those used in user sessions or in environments where periodic reauthentication is not enforced, could be reused by an attacker until they are revoked or expire naturally.

For instance, if an attacker gains access to a compromised long-lived token, they can use it to maintain access to specific services or potentially move laterally within a network to escalate privileges. While the token itself does not directly extend its own lifespan (unlike a refresh token), the extended time during which it remains valid provides the attacker with more opportunities to exploit the network before detection.

In microservices environments, where APIs are constantly communicating, short-lived tokens are crucial for securing service-to-service interactions. These tokens ensure that access is limited in time and scope, making it harder for attackers to leverage a compromised token to pivot to other services. However, each location where tokens are stored or transmitted in such environments represents a potential attack vector, emphasizing the need for secure storage and transmission practices.

Difficulty in Enforcing Least Privilege

While long-lived tokens can be scoped to enforce least privilege, their existence represents standing privilege;[xv] they still pose a higher risk if not carefully managed, as compromised tokens can grant attackers prolonged access to specific resources.[xvi] If the scope is too broad at the time of issuance, long-lived tokens might unintentionally provide access to resources beyond what is necessary.

In contrast, short-lived, narrowly scoped tokens allow administrators to enforce fine-grained access control, ensuring that tokens are only valid for specific operations or resources for a limited time. By applying narrow scopes, organizations can reduce the potential damage from a compromised token, as an attacker would only gain access to a limited subset of resources. This approach restricts unauthorized access to a small part of the infrastructure, making it an essential control for protecting sensitive systems.

When Long-Lived Tokens May Make Sense

There are scenarios where long-lived tokens do offer certain benefits, particularly in specific use cases where organizations have good reason to prioritize convenience and reduced token management overhead. For example:

Reduced Overhead in Token Renewal

Long-lived tokens minimize the need for frequent token refreshes, reducing the number of calls to the authentication server. This limitation is particularly useful in applications where users or services need continuous access over extended periods without interruptions.

- **Use case**: In systems where uptime and uninterrupted access are critical (such as in background services, batch processing, or long-running data analytics jobs), long-lived tokens can prevent delays or failures caused by expired tokens.
- *Note: In cases where the application involves workload identities, then protocols like SPIFFE enable dynamic, instance-based tokens.*[xvii] *Long-lived tokens are not necessary.*

Improved Performance in Stateless Systems

In stateless architectures, such as RESTful APIs, where the session state is not maintained on the server side, long-lived tokens eliminate the need to constantly issue new tokens. This option can enhance the system's overall performance by reducing the load on authentication and authorization servers.

- **Use case**: For applications requiring minimal server-side interaction for performance reasons, such as APIs serving high requests, long-lived tokens can reduce the number of database or server interactions needed to reissue tokens.

Service-to-Service Communication

In certain system-to-system communications where services need to interact with each other in a highly trusted environment (e.g., within the same enterprise or data center), long-lived tokens can reduce the need for frequent authentication and authorization exchanges. This makes service communication more efficient.

- **Use case**: Within microservice architectures in secure internal networks, long-lived tokens can facilitate inter-service communication without requiring frequent re-authentication.
- *Note: This use can also be solved by using short-lived tokens that leverage client_secret_jwt or private_key_jwt as defined in the OpenID Connect Core specification*.[xviii]

Emerging Trends and Future Directions

The field of token management is evolving rapidly as organizations seek to balance security, scalability, and usability. While this article focuses on the current best practices for short-lived tokens, ongoing advancements in standards and frameworks are introducing new approaches to token lifetimes and security.

Continuous Access Evaluation Profile (CAEP)

One of the most significant emerging developments is the Continuous Access Evaluation Profile (CAEP), a framework designed to dynamically assess and adjust access permissions in real time. CAEP enables access tokens to have longer lifespans without compromising security. This is achieved by combining periodic risk assessments with real-time revocation capabilities. For example, if a security event, such as a user's location change or device compromise, is detected, access can be revoked immediately, even if the token has not yet expired. While still in draft form, CAEP is gaining traction among vendors as a promising solution for risk-based token management.

Risk-Based Token Lifetimes

Traditional token lifetimes often rely on static durations, but future implementations are likely to adopt adaptive token lifetimes based on real-time risk analysis. By evaluating factors such as user behavior, device trust levels, and contextual signals, organizations could dynamically adjust token expiration times. This approach seeks to reduce unnecessary reauthentication while maintaining a high level of security.

Proof of Possession and Sender-Constrained Tokens

The shift from bearer tokens to proof-of-possession (PoP) or sender-constrained tokens, mentioned earlier in the Introduction, represents another critical trend. These tokens require the client to demonstrate cryptographic proof that they are the legitimate holder of the token. Standards such as OAuth 2.0 DPoP (Demonstration

of Proof-of-Possession) and Mutual TLS (mTLS) are advancing this concept, reducing the risk of token replay attacks and unauthorized use.

Enhanced Revocation Mechanisms

Real-time token revocation remains a challenge, especially in distributed systems. One draft under discussion is draft-parecki-oauth-global-token-revocation, "Global Token Revocation".[xix] At the time of publication of this article, the draft has not been accepted by an IETF working group, but it remains an interesting example of ongoing work in the revocation space.

Conclusion

While long-lived tokens can offer convenience, particularly in trusted environments, the risks they present—such as token replay, session hijacking, and unauthorized access—must be carefully managed. (Of course, if you are following zero-trust principles, there is no such thing as a trusted environment.) Short-lived, narrowly-scoped tokens, client-bound tokens, and strong cryptographic standards provide an effective framework for mitigating these risks. Several guides are available on best practices in different environments, including BCP 225, "JSON Web Token Best Current Practices"; review them and consider how they can apply to your environment.

Frameworks like OAuth 2.0 and emerging standards like WIMSE (Workload Identity in Multi-System Environments) are beginning to provide the necessary infrastructure for managing short-lived tokens, particularly in cloud-native and microservices architectures.[xx] WIMSE is an interesting standardization effort for applications and multi-cloud identities. And while short-lived tokens remain a cornerstone of secure token management today, the evolving landscape of token management frameworks, such as CAEP, suggests that risk tolerance for longer token lifetimes may shift as standards and implementations mature.

Your organization's risk posture will guide your definition of "short" and "long." Regardless of whether you consider seconds, minutes, or months "short," you need to implement strict scoping, use token binding, and ensure robust revocation and monitoring processes. You are aiming for a balance between security, usability, and performance, ensuring that both user and service interactions are protected against emerging threats.

Author Bio

 Heather Flanagan, Principal at Spherical Cow Consulting, comes from a position that the Internet is led by people, powered by words, and inspired by technology. She has been involved in leadership roles with some of the most technical, volunteer-driven organizations on the Internet, including IDPro as Executive Director and Principal Editor; the OpenID Foundation as Lead Editor; the IETF, IAB, and the IRTF as RFC Series Editor; ICANN as Technical Writer and Editor; and REFEDS as Coordinator, just to name a few. If there is work going on to develop new Internet standards, or discussions around the future of digital identity, she is interested in engaging in that work. You can learn more about her at <https://www.linkedin.com/in/hlflanagan/>

---

[i] Y., Hardt, D., and M. Jones, "JSON Web Token Best Current Practices", BCP 225, RFC 8725, DOI 10.17487/RFC8725, February 2020, <https://www.rfc-editor.org/info/rfc8725>. *Note that some readers may find reference to RFC 8471, "The Token Binding Protocol" in their search for more information on token binding. That specification has been largely abandoned. See https://groups.google.com/a/chromium.org/g/blink-dev/c/OkdLUyYmY1E/m/w2ESAeshBgAJ for the thread that ultimately ended widescale adoption of the Token Binding Protocol.*

[ii] Fett, D., Campbell, B., Bradley, J., Lodderstedt, T., Jones, M., and D. Waite, "OAuth 2.0 Demonstrating Proof of Possession (DPoP)", RFC 9449, DOI 10.17487/RFC9449, September 2023, <https://www.rfc-editor.org/info/rfc9449>.

[iii] For examples of how specific use cases are handled, see the profile of OAuth 2.0 and OpenID Connect recommended by the OpenID Foundation's FAPI Working Group that focuses on use cases requiring high security, like Open Banking: Fett, D. "FAPI 2.0 Security Profile" Implementers Draft, December, 2022, https://openid.net/specs/fapi-2_0-security-profile-ID2.html

[iv] There is work underway in the IETF to standardized real-time token revocation, but the work is still in draft form. See Parecki, A., "Global Token Revocation," draft-parecki-oauth-global-token-revocation, https://datatracker.ietf.org/doc/draft-parecki-oauth-global-token-revocation/.

[v] Richer, Justin, and Antonio Sanso. 2017. *OAuth 2 in Action*. Manning.

[vi] Jones, M. and D. Hardt, "The OAuth 2.0 Authorization Framework: Bearer Token Usage", RFC 6750, DOI 10.17487/RFC6750, October 2012, <https://www.rfc-editor.org/info/rfc6750>.

[vii] Campbell, B., Bradley, J., Sakimura, N., and T. Lodderstedt, "OAuth 2.0 Mutual-TLS Client Authentication and Certificate-Bound Access Tokens", RFC 8705, DOI 10.17487/RFC8705, February 2020, <https://www.rfc-editor.org/info/rfc8705>.

[viii] Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, DOI 10.17487/RFC6749, October 2012, <https://www.rfc-editor.org/info/rfc6749>.

[ix] Grassi, Paul A, James L Fenton, Elaine M Newton, Ray A Perlner, Andrew R Regenscheid, William E Burr, Justin P Richer, et al. 2017. “Digital Identity Guidelines: Authentication and Lifecycle Management.” https://doi.org/10.6028/nist.sp.800-63b.

[x] Cappalli, T. and Tulshibagwale, A. “OpenID Continuous Access Evaluation Profile 1.0 – Draft 03” 2024-06-19, https://openid.net/specs/openid-caep-1_0-ID2.html

[xi] Grassi, Paul A, Justin P Richer, Sarah K Squire, James L Fenton, Ellen M Nadeau, Naomi B Lefkovitz, Jamie M Danker, Yee-Yin Choong, Kristen K Greene, and Mary F Theofanos. 2017. “Digital Identity Guidelines: Federation and Assertions.” https://doi.org/10.6028/nist.sp.800-63c.

[xii] RFC 6749, "The OAuth 2.0 Authorization Framework”.

[xiii] Fletcher, George, Pieter Kasselman, Atul Tulshibagwale, “Transaction Tokens,” last updated 2024-07-03, https://datatracker.ietf.org/doc/draft-ietf-oauth-transaction-tokens/.

[xiv] “Manipulator-in-the-middle Attack | OWASP Foundation,” n.d. https://owasp.org/www-community/attacks/Manipulator-in-the-middle_attack.

[xv] Trevino, Aranza, and Aranza Trevino. 2024. “What Are Zero Standing Privileges?” Keeper Security Blog - Cybersecurity News & Product Updates. April 29, 2024. https://www.keepersecurity.com/blog/2024/04/29/what-are-zero-standing-privileges/.

[xvi] Carter, M. K., (2022) “Techniques To Approach Least Privilege”, *IDPro Body of Knowledge* 1(9). doi: https://doi.org/10.55621/idpro.88

[xvii] “SPIFFE – Secure Production Identity Framework for Everyone,” n.d. https://spiffe.io/.

[xviii] Sakimura, Nat, John Bradley, Michael Jones, Breno De Medeiros, and Chuck Mortimore. 2023. “OpenID Connect Core 1.0 incorporating errata set 2.” OpenID Foundation. https://openid.net/specs/openid-connect-core-1_0.html.

[xix] Parecki, A. “Global Token Revocation,” draft-parecki-oauth-global-token-revocation, https://datatracker.ietf.org/doc/draft-parecki-oauth-global-token-revocation/.

[xx] “Workload Identity in Multi System Environments (Wimse).” n.d. https://datatracker.ietf.org/group/wimse/about/.