Computer Fraud and Security
ISSN (online): 1873-7056

Modern Stateless Authentication for Legacy Stateful Systems: Enabling OAuth/OIDC User Experience Without Rebuilding the Core Product

Bhanuprakash Naidu Basani
Independent Researcher, USA

Abstract

Legacy enterprise infrastructure products such as file and print servers typically use stateful directory-based authentication such as identity sources supported by LDAP and service principals backed by Kerberos, both of which are deeply integrated into the core runtime. In contrast, enterprise customers increasingly want modern browser-native administrative experiences using OAuth 2.0 and OpenID Connect (OIDC) based on authentication state encoded in cryptographically signed JSON Web Tokens (JWTs). A popular myth is that to provide such a modern experience, a large-scale rewrite of the legacy system is required. Describe a bridging architecture that rejects that myth and only requires small additions to be made at the edge of the system as an identity translation layer between modern token-based identities and OS-native principals, while also leveraging the battle-tested legacy core. Implementation connects the modern management plane to the existing Linux-based legacy server via a token-verifying PAM-based authentication module and a trust registry for SSO handler instances, stored in the existing directory infrastructure and managed by the administrator. This enables modern SSO flows while preserving the existing filesystem and internal service security models. It accomplishes a realistically achievable path to Zero Trust-based administrative governance without the risks and costs associated with system redevelopment.

Keywords: Legacy Modernization, Stateless Authentication, OAuth 2.0, OpenID Connect, JWT, LDAP, Kerberos, PAM, Identity Bridging, Zero Trust

1. Introduction

Enterprise infrastructure products successfully build trust by being around for decades and because their architectures tightly integrate security features. Enterprise identity, authorization, and access control systems are often tightly coupled to OS user primitives and enterprise directory services. While this is an ideal scenario in terms of security, it is not aligned with token-based identity architectures, and enterprise IT's expectations have shifted to be able to manage web-based management interfaces in the browser, federated single sign-on, and browser-based authentication flows to modern identity providers according to the OAuth 2.0 Authorization Code Flow.

According to documentation from Microsoft for the Entra identity platform, the OAuth 2.0 Authorization Code Flow consists of four messages: the client makes an authorization request to the authorization endpoint, the user is authenticated and gives consent, the authorization server issues an authorization code, and the client exchanges the authorization code at the authorization server's token endpoint for an access token and an ID token. The Entra identity platform uses this flow by default for its web-based administrative tooling and grants short-lived access tokens without exposing client credentials. This means that access tokens issued by the Entra platform are set to expire in 60-90 minutes by default, though shorter lifetimes (e.g., 10 minutes) can be enforced for high-sensitivity administrative scopes [1]. The OIDC Core 1.0 specification builds on top of this to define the id_token, a JWT that carries the identity of the authenticated user, with claims providing the issuer (iss), subject (sub), audience (aud), expiration/expiry (exp), and issued-at time (iat) claims. The specification further defines an API call (the UserInfo endpoint) for claims not carried in the id_token [2]. The three types of token most frequently used in OIDC are the ID token for identity assertion, the access token for resource authorization, and an optional refresh token [2].

For more difficult applications, legacy systems cannot consume any of these constructs. This puts teams into a false dilemma: to continue to serve the legacy stateful model and accept obsolescence or to reconstruct business logic into a new stack and accept the loss of decades of well-established behavior. This article shows how a third path - bridging stateless tokens into the existing identity model at the boundary - is consistently superior in risk, cost and delivery profile.

2. Problem Statement and Requirements

2.1 Legacy Identity Model Constraints

For legacy file or print servers, stateful directory-based identity resolution can be used to authenticate users, resolve group memberships and attributes over LDAP, and provide mutual network authentication with Kerberos in domain-joined deployments. OS-native principles, such as Unix UID/GID for Linux-based implementations, are the primitives on which all filesystem and service access policy is ultimately resolved. This is a mature model yet completely incompatible with browser-centric token flows.

The NIST Zero Trust Architecture (ZTA) framework identifies this class of systems as a perimeter-dependent legacy architecture, in which trust is implicitly derived from network location and possession of credentials, as opposed to being continuously established and verified via explicit, policy-driven means [4]. NIST SP 800-207 describes the seven key tenets of the Zero Trust security model. In particular, the requirement that all communications be secured regardless of network location, the requirement that resource access be granted on a per-session basis with least-privilege enforcement, and the requirement that authentication and authorization be dynamic and continuously reevaluated are directly violated by legacy stateful authentication [4]. Bridging to token-based authentication begins to address these violations, without a complete rewriting of the core.

2.2 Modern Management Expectations and Zero Trust Alignment

CISA Zero Trust Maturity Model (ZTMM) Version 2.0 organizes ZT adoption along five pillars: Identity, Devices, Networks, Applications and Workloads, and Data. Within the five pillars, organizations can achieve Customary, Initial, Advanced, and Optimal maturity stages based on capabilities and implementations [3]. Identity maturity stages are customary identity validation based on perimeter access, initial multi-factor authentication for privileged users, advanced phishing-resistant MFA along with continuous identity validation, and optimal fully automated policy-driven identity governance along with real-time risk scoring [3]. Legacy LDAP/Kerberos-only authenticated systems fit in the Customary identity maturity stage. If OAuth/OIDC token-based management authentication is added to such a system, it can leap to the Initial-to-Advanced identity maturity stage without fundamental rewrites.

2.3 Key Design Requirements

Requirement	Description	ZT Pillar (CISA ZTMM)	Priority
No core rewrite	Preserve legacy C/C++ stack and authorization semantics	—	Critical
Token-based authentication	OAuth/OIDC JWT acceptance at the managed server boundary	Identity	Critical
Explicit trust anchoring	Directory-backed registry governing legitimate token issuers	Identity	Critical
Operational scalability	Centralized management plane for large estate rollout	Applications	High
Minimal coupling	Management plane evolves independently of legacy core	Applications	High
Directory compatibility	Continued support for Active Directory, Kerberos environments	Identity	High
Continuous audit logging	Per-event token validation records with provenance	Data	Medium

Table 1: Summary of Key Design Requirements for the Bridging Solution

3. Architecture Overview: Stateless-to-Stateful Identity Bridging

3.1 High-Level Components

This architecture separates these into four components. For the web UI, the management server launches the browser-based login, handles the OAuth/OIDC identity provider, gets the tokens, and passes those as evidence of administrative identity to the legacy managed server. It supports a trust registry, a managed directory maintained by trusted administrators, to which JWK-set URLs and issuer IDs of registered management servers are added. The token

verification module is the minimum alteration on the legacy server that is a PAM-based module that verifies tokens and maps identities to the local accounts. The legacy Core remains; NIST SP 800-63B also introduces three Authenticator Assurance Levels (AAL). AAL2 requires multi-factor authentication and some cryptographic evidence of possession, whereas AAL1 requires single-factor authentication. For AAL3, phishing-resistant authenticators must be hardware-backed [5]. Relays require registered identity provider token signatures to prove cryptographic possession for AAL2 administrative access within the bridging architecture without the need to rewrite the core.

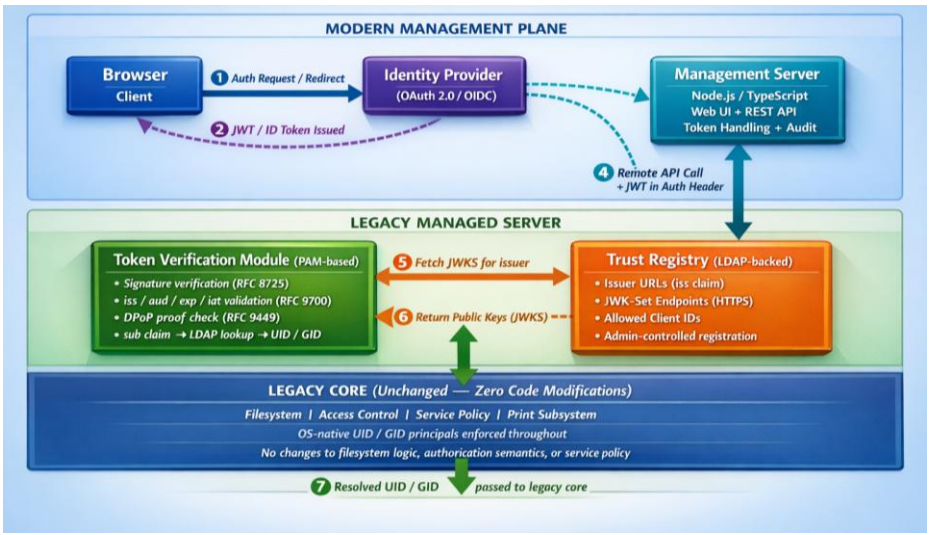

Figure 1: Stateless-to-Stateful Authentication Bridging Flow

3.2 Trust Registry as the Security Bridge

This architecture separates these into four components. For the web UI, the management server launches the browser-based login, handles the OAuth/OIDC identity provider, gets the tokens, and passes those as evidence of administrative identity to the legacy managed server. It supports a trust registry, a managed directory maintained by trusted administrators, to which JWK-set URLs and issuer IDs of registered management servers are added. The token verification module is the minimum alteration on the legacy server that is a PAM-based module that verifies tokens and maps identities to the local accounts. The legacy core remains; NIST SP 800-63B also introduces three Authenticator Assurance Levels (AAL). AAL2 requires multi-factor authentication and some cryptographic evidence of possession, whereas AAL1 requires single-factor authentication. For AAL3, phishing-resistant authenticators must be hardware-backed [5]. Relays require registered identity provider token signatures to prove cryptographic possession for AAL2 administrative access within the bridging architecture without the need to rewrite the core.

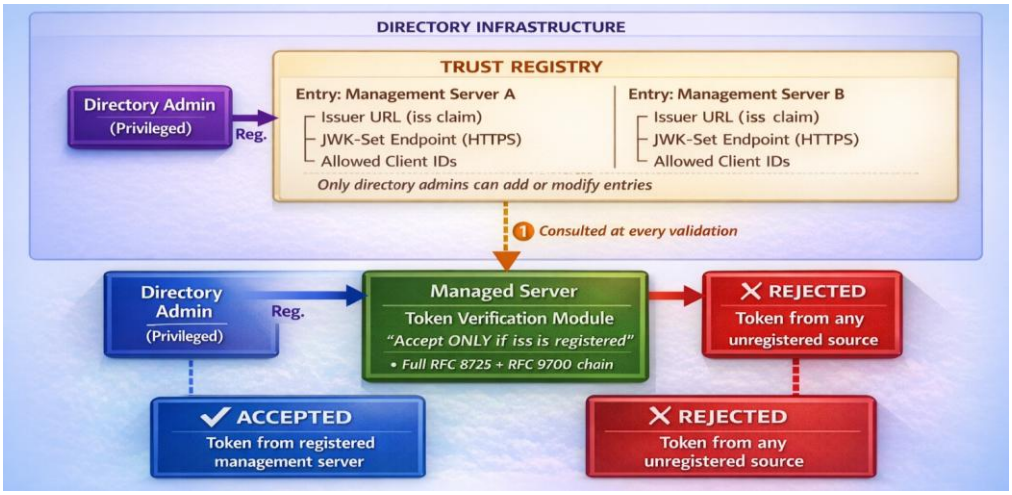

Figure 2: Trust Registry as the Security Bridge

3.3 Modernization with Minimal Change Surface

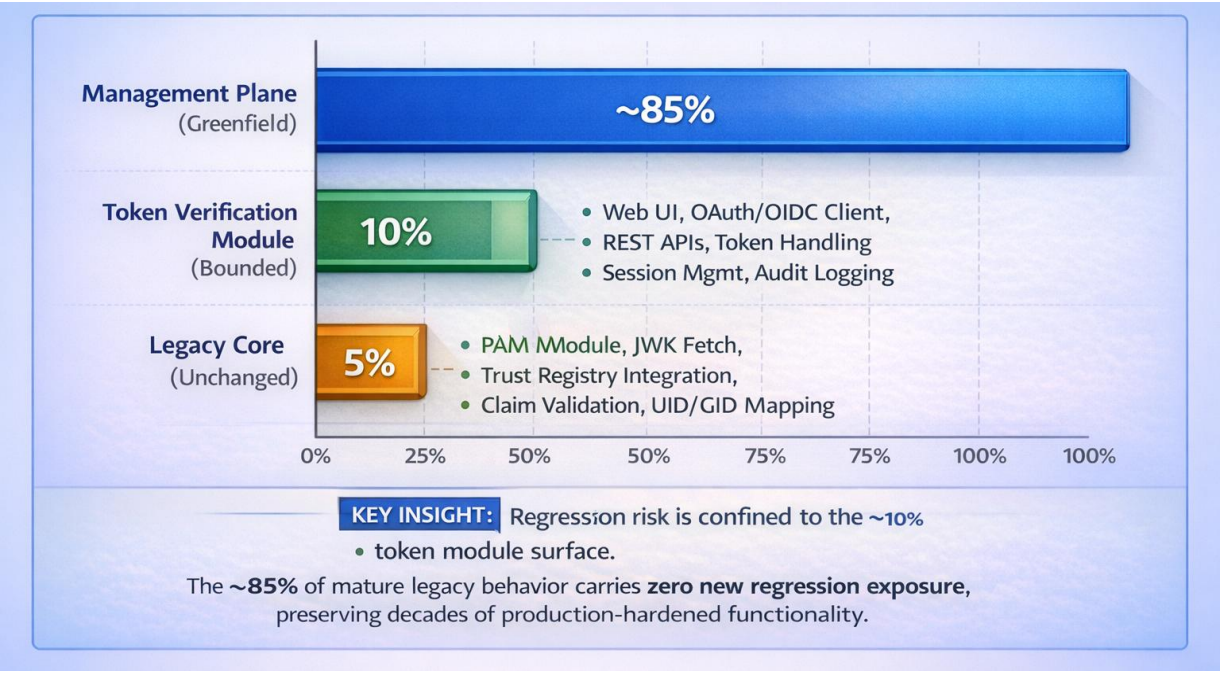

Figure 3: Modernization Effort Distribution

4. Security Model: Administrator-Controlled Trust Without Rewriting the World

4.1 Trust Registry and DPoP-Improved Token Binding

RFC 9449 describes a variant of OAuth 2.0 DPoP known as OAuth 2.0 Showing Proof of Possession (DPoP), where the client cryptographically binds the access token to an ephemeral private key. Each request is accompanied by a DPoP proof JWT signed by the client's ephemeral key pair in conjunction with the access token. The DPoP proof JWT contains the htm (HTTP method), htu (HTTP URI), iat (issued-at), and jti (JWT ID) claims, effectively making it non-replayable on behalf of a single request or endpoint [7]. Receiving a DPoP-bound token, a resource server must verify the access token's signature and the DPoP proff JWT, binding it to the same client that received it. In older management use-cases, DPoP is instead introduced at the management server to managed server API layer, considerably reducing the risk of bearer token interception and replay, since the token without the matching private key is cryptographically unusable.

4.2 Token Validation Chain and OAuth 2.0 Security Best Practices

The design of the token verification module is based on the validation requirements specified in RFC 9700, Best Current Practice for OAuth 2.0 Security [8]. The resource servers must verify the ISS claim in the access token against an allowlist, the AUD claim against the resource server, reject expired tokens, and check the scope with the least privileged access model according to the scopes that the token authorizes [8]. RFC 9700 also prohibits the implicit grant type from the authorization framework in new deployments and requires support for the Proof Key for Code Exchange (PKCE, pronounced 'pixie') extension, even for confidential clients, to reduce vulnerabilities to the authorization code interception attack [8].

Step	Validation Action	Applicable Standard	Failure Consequence
1	Retrieve JWKS from the trust registry for token issuance	RFC 8725 §3.1	Reject—issuer not registered
2	Verify JWT signature against retrieved public key	RFC 8725 §3.2	Reject—signature invalid
3	Validate the ISS claim against the directory trust registry	RFC 9700 §4.3	Reject—issuer mismatch

4	Validate the aud claim matches the managed server identifier	RFC 9700 §4.3	Reject—audience mismatch
5	Reject the token if the exp claim is in the past	RFC 8725 §3.9	Reject—token expired
6	Validate nbf/iat for token freshness within tolerance	RFC 8725 §3.9	Reject—token not yet valid
7	Verify DPoP proof JWT if DPoP binding is enabled	RFC 9449 §4.3	Reject—proof invalid or replayed
8	Validate optional scope and role claims	RFC 9700 §4.8	Reject—insufficient privilege
9	Map sub-claim via LDAP lookup to OS UID/GID	NIST SP 800-63B §6	Reject—no principal mapping found

Table 2: Token Validation Steps per RFC 9700 and RFC 8725

For protection against algorithm confusion attacks (in which an attacker can replace the algorithm header with an algorithm which the resource server allows, such as a weak algorithm or a symmetric algorithm), RFC 8725 requires that resource servers only accept signing algorithms that they know. Token verification libraries should be configured with a small number of algorithms (such as RS256 or ES256) and should not accept alternative algorithms.

5. Implementation Strategy: Minimal Server Change, Modern Management Experience

5.1 Token-Verifying Authentication Module

On Linux legacy servers, PAM is the most natural and least intrusive extension point for token-based authentication. A PAM module that implements JWT validation receives bearer tokens from the invoking client process. The PAM module retrieves the JWT's issuer (iss) claim to get a JWK set from the trust registry, then validates tokens using the steps in Table 2, resolves the sub claim to a local UID/GID in a local directory service, and returns a principal context to the legacy subsystem. The legacy code core would work as before, receiving OS-level identities, permissions, and capabilities. However, the Node.js runtime (at this writing, v25.6.1) comes with a fully fledged crypto module built in, which can perform RSA-PKCS1-v1_5, RSASSA-PSS, and ECDSA signature verification natively. This is especially convenient as token issuing and signing done by the management plane can take advantage of these cryptographic primitives with no additional dependencies [9].

5.2 Unified Development Stack and the Strangler Fig Pattern

The use of a single Node.js/TypeScript technology stack for both backend API services and front-end web components reduces the burden on staff resources supporting different stacks. The Node.js runtime's built-in crypto module (available in the current v25.6.1 release) supports RSA-PKCS1-v1_5, RSASSA-PSS, and ECDSA signature verification natively; combined with the Fetch API, stable since Node.js v21, this enables JWK-set retrieval and token verification with no additional dependencies [9]. Finally, the Strangler Fig pattern from Martin Fowler describes building the new functionality around the old and routing more and more functionality to be processed in the new system until the old system can be abandoned (or, as here, be left as a stub and all functionality be in the new plane). He breaks this approach down into three stages: Transform, where the new component is built up in parallel; coexist, where new functionality is routed to the new and existing functionality is handled by the old; and eliminate, where the old functionality is retired. The bridging architecture is thus permanently within the Coexist stage, because the legacy core (proven and correct) cannot be removed and only the administrative authentication boundary is updated.

6. Operational Benefits

6.1 Modern User Experience Without Core Redevelopment

The bridging architecture will provide a complete OAuth/OIDC admin login experience—identity provider redirect, credential challenge, multi-factor authentication (MFA), token issuance, and browser-native session management—without requiring reimplementation of any legacy functionality. Administrators will have an integrated web platform

utilizing their organization's identity provider, without sacrificing advantages like SSO behavior, phishing-resistant MFA eligibility, or conditional access policies. The CISA Zero Trust Maturity Model describes this progress along the Identity pillar from the Customary maturity stage of static credential-based access to the Advanced stage of continuous, policy-driven identity validation across a system of identity providers: Token-based authentication tied to registered identity providers allows dynamic access policies and automated account lifecycle actions that are structurally impossible in purely directory-bound models. [3]

6.2 Security Posture and Governance Improvements

To achieve Zero Trust, NIST SP 800-207 prescribes that real-time or continuous diagnostics and dynamic policy enforcement are required. The policy engine must be fed with detailed information in real-time, such as identity posture, device security posture, and behavioral analytics [4]. This matches the trust registry architecture as it gives centralized control of trust to the directory administrators, e.g., enforced key rotation. Changes to the JWK-set endpoint are propagated to referenced servers in the next validation cycle. Token validation events produce an auditable log including not just issuer, subject, audience, and expiry, but their outcome as well. It satisfies the NIST SP 800-207 requirement for complete transaction logs to support retroactive policy analysis [4].

Evaluation Index	Full Rewrite (Baseline = 1.00)	Bridging Approach	Relative Improvement
Engineering Scope Index	1	~0.30–0.40	~60–70% reduction
Delivery Time Index	1	~0.35–0.45	~55–65% reduction
Core Regression Risk Index	1	~0.05–0.10	~90–95% reduction
Feature Gap Risk Index	1	~0.02–0.05	~95–98% reduction

Table 3: Normalized Evaluation Indices—Bridging Approach vs. Full Rewrite

CISA ZTMM Stage	Identity Pillar Characteristic	Trust Registry Capability	Token Validation Depth	Audit Coverage
Traditional	Static credentials, perimeter trust	No registry; ad-hoc trust	Absent or structural-only	Minimal
Initial	MFA for privileged access	Registry present; manually maintained	Signature + expiry validation	Per-session logging
Advanced	Phishing-resistant MFA; continuous validation	Registry with automated JWKS rotation	Full RFC 9700 + RFC 8725 chain	Per-event with token provenance
Optimal	Automated, risk-scored identity governance	Registry with real-time policy integration	DPoP binding; RFC 9449 enforced	Real-time anomaly-correlated logs

Table 4: Security Governance Maturity-CISA ZTMM Alignment

6.3 Cost and Staffing Efficiency

Since the management plane represents the bulk of the work (85% of the scope appears to be in the management plane as illustrated in Figure 3 below), JavaScript and TypeScript are the dominant engineering languages needed to deliver the modern management experience. There is a small, bounded need for engineers experienced with PAM and LDAP integration on the legacy server side. The Strangler Fig pattern enables incremental migration of each management plane feature to the new architecture without first replicating the legacy environment, reducing feedback time and integration risk at every step [10]. The normalized indices of Table 3 suggest that the engineering scope and delivery time of this approach are 60-70% and 55-65%, respectively, of a full rewrite's scope and time. The regression risk of this approach is 90-95% because the proven legacy core never changes.

Conclusion

Legacy enterprise systems have decades of operational investment, behavioral correctness, and security hardening that cannot safely or quickly be duplicated in a wholesale rewrite. The primary contribution of the bridging architecture presented in this paper is showing that a modern OAuth 2.0 and OpenID Connect administrative experience can be enabled without abandoning operational capabilities that already work. By implementing a thin identity translation layer at the system boundary using an administrator-controlled trust registry within the existing directory infrastructure, institutions can adopt the stateless token model of modern identity provider systems while maintaining the OS-native authorization semantics that legacy systems are built on.

This is not an optimization; it's a principled design based on current practice. The trust registry is administratively controlled, issuer governed (JWT Best Current Practices), and adapts the principles of Zero Trust Architecture (ZTA) to JWT verification by ensuring all tokens are verified to be chains of trust ending in registered, administratively controlled issuers. The PAM-based verification module is a chain of trust that adheres to open standards. The optional DPoP binding can further strengthen the trustworthiness of tokens without affecting the legacy core functionality. The Strangler Fig modernization pattern provides a long-term delivery model, where the management plane evolves continuously, independently from the delivery path, and thus independently from the proven functionality.

Ultimately, this architecture challenges the false dichotomy that has driven unnecessary and expensive legacy rewrites across the enterprise software industry: when the core system and its authorization semantics are correct, the appropriate modernization is to honor that correctness in the way that system is bridged to new expectations at the boundary, rather than being rebuilt. OAuth 2.0, OIDC, Zero Trust governance, and pluggable authentication on Linux are applications of this principle. It is relevant wherever an organization has long-lived infrastructure with modern identity ecosystems.

References

[1] Microsoft, "Microsoft identity platform and OAuth 2.0 authorization code flow," Microsoft Learn. Available: https://learn.microsoft.com/en-us/entra/identity-platform/v2-oauth2-auth-code-flow

[2] N. Sakimura, et al., “OpenID Connect Core 1.0 incorporating errata set 2,” OpenID Foundation, 2023. Available: https://openid.net/specs/openid-connect-core-1_0.html

[3] Cybersecurity and Infrastructure Security Agency (CISA), "Zero Trust Maturity Model," Version 2.0, Apr. 2023. Available; https://www.cisa.gov/sites/default/files/2023-04/zero_trust_maturity_model_v2_508.pdf

[4] Scott Rose, et al., "Zero Trust Architecture," NIST Special Publication 800-207, National Institute of Standards and Technology, 2020. Available: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-207.pdf

[5] Paul A. Grassi, et al., "Digital Identity Guidelines: Authentication and Lifecycle Management," NIST Special Publication 800-63B, National Institute of Standards and Technology, 2020. Available: https://pages.nist.gov/800-63-3/sp800-63b.html

[6] Y. Sheffer, et al., "JSON Web Token Best Current Practices," IETF RFC 8725, 2020. Available: https://www.rfc-editor.org/rfc/rfc8725

[7] D. Fett, et al., "OAuth 2.0 Demonstrating Proof of Possession (DPoP)," IETF RFC 9449, 2023. Available: https://www.rfc-editor.org/rfc/rfc9449

[8] T. Lodderstedt, et al., "Best Current Practice for OAuth 2.0 Security," RFC 9700, 2025. Available: https://www.rfc-editor.org/rfc/rfc9700

[9] Node.js, "Node.js v25.6.1 documentation." Available: https://nodejs.org/en/docs

[10] Martin Fowler, "Strangler Fig," martinfowler.com, updated 2024. Available: https://martinfowler.com/bliki/StranglerFigApplication.html