New Framework for Securing Web APIs Token-based Authentication / Authorization with Auto Expire Auto Refresh (AEAR) Features

Mohamed Amer
Helwan University
Cairo, Egypt

Tarek S. Sobh
El Shorouk Academy
Cairo, Egypt

ABSTRACT
Token-based authentication for Web APIs allows users to verify their unique identity. In return, they receive a unique token that grants access to specific resources for a limited period of time. These tokens are stored on the client's browser with expiration properties [1, 2], making them vulnerable to cyber-attacks such as Stealing Tokens through Redirection, Cross-Site Request Forgery (CSRF), and Cross-Site Scripting (XSS) [3]. The algorithms themselves can also be a source of vulnerabilities, including Weak Symmetric Keys and Incorrect Composition of Encryption and Signature [4]. Various authentication protocols like Open Authorization (OAuth), Security Assertion Markup Language (SAML), OpenID Connect (OIDC), Client Initiated Backchannel Authentication (CIBA), and JSON Web Token (JWT) and their associated attacks are examined. A new framework that incorporate Multi-Factor Authentication (MFA) and One Time Password (OTP) is proposed to address these vulnerabilities, along with detailed analysis and guidelines for its implementation.

General Terms
Web Development. Secure web apps. Token protocols

Keywords
Tokens; OAuth; SAML; OIDC; CIBA; JWT; XSS, CSRF

1. INTRODUCTION
The Internet is accessible to everyone, allowing anyone to access various services and resources. To safeguard these from unauthorized access, legitimate users are required to verify identities. However, repeatedly requesting legitimate users to verify their identity for stateless applications can be frustrating and may result in decreased customer satisfaction. This can have a significant impact on the business, for example, on a social media platform, if a user is constantly asked for credentials when posting, they may interact less and potentially deactivate their account, resulting in loss of customers and ultimately the entire business. Token-based authentication for Web APIs endpoints offers a smooth authentication process by creating a unique access token for each server request, eliminating the need to re-enter credentials. This ensures a seamless user experience while maintaining data security. **[1, 2]** Attackers often target client endpoints during authentication, waiting for the token to be stored in the browser before launching attacks through poor implementation or exploiting weaknesses in cryptographic algorithms. Token expiration is designed to mitigate the risk of token theft or reuse from browser caches, similar to CSRF attacks **[5]**.Token-based authentication is stateless, providing detailed access control, scalability, efficiency, and flexibility with configurable expiration times. Although efficient, it does come with drawbacks such as data overhead, shorter lifespan, token forgery, reply, disclosure, and redirect vulnerabilities **[6, 1, 5]**.

This paper presents a secure framework for automatically expiring tokens and implementing a seamless auto-renewal scheme without user intervention.

2. TOKEN AUTHENTICATION AUTHORIZATION SCHEMATIC STEPS
As explained in Fig 1, Token authentication-authorization uses 6 steps:

1. Access Request: this step involves client requesting access to a protected resource hosted by the Service Provider (SP).
2. Identity Check: Service Provider verifies the identity of the user by sending a request to the Identity Provider (IdP) [7].
3. Login Request: The Identity Provider redirects the user to a Login Page for authentication purposes.
4. Verification: After the user is authenticated by providing credentials to the Identity Provider, they are authorized and authorization claims against the protected resource are created [7].
5. Token Generation: The authorization claims and identity information are combined into a token and provided to both the client and service provider. This token contains the user's identification and permissions for future requests, along with details on its active or expiry time [2]. Client stores token locally on their user agent [8] for future use.
6. Resource Access: client includes the token in all subsequent requests to access the protected resource until it expires. When the token expires, the client must either refresh it or request a new one by following the same steps again.

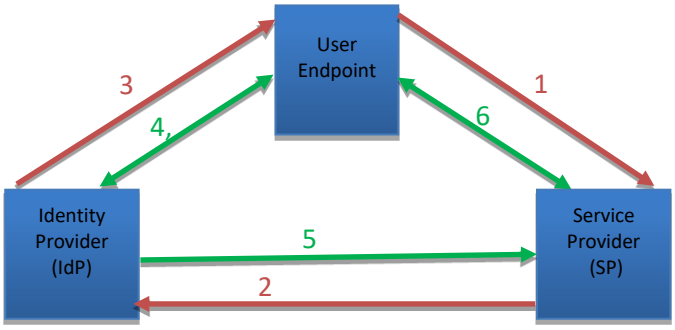

Fig 1: API Endpoint Token Authentication-Authorization

3. OVERVIEW OF TOKEN-BASED AUTHENTICATION FOR WEB APIS PROTOCOLS

Authentication refers to the act of identifying and confirming the digital identity of an entity, such as a user [9]. Token-Based Authentication for web APIs involves authenticating the entity once and then providing it with a token that must be included in each API request. This token is utilized to either identify the entity or supply details about its authorized resources.

3.1 Open Authorization (OAuth)

The OAuth authorization protocol allows third-party apps to access a resource owned by a user without needing to share the user's account credentials [7, 1, 2]. The key players in the OAuth protocol are the Resource owner (Ro), the Client (Cl), the Resource server (Rs), and the Authorization server (As). The Client (Cl) aims on behalf of the Resource owner (Ro) to access a resource hosted at the Resource server (Rs), the Authorization server (As) grants or denies client requests and issues tokens upon approval.

3.2 Security Assertion Markup Language (SAML)

SAML, while considered the ancestor of authentication protocols, remains a key component of web-based single sign-on (SSO). It specifically facilitates identity federation [10, 11], allowing identity providers (IdPs) to securely transmit authenticated identities and attributes to service providers (SPs) seamlessly and transparently. [12] This XML-based framework created by "Security Services Technical Committee of the Organization for the Advancement of Structured Information Standards" (OASIS) [13], for transmitting user authentication, entitlement, and attribute information. It serves as a language for assertion, enabling its use with various authentication and authorization protocols such as OAuth [14].

3.3 OpenID Connect (OIDC)

OpenID Connect is a public authentication protocol that uses OAuth 2.0. Clients can authenticate an end user's identity with the authorization server. **[15]**. It serves as a unified protocol for securing mobile applications, web applications, and APIs **[16]**. OIDC introduces a new component known as the UserInfo Endpoint **[17]**, a safeguarded resource that provides authenticated End-User claims **[18]**.

3.4 Client-Initiated Backchannel Authentication (CIBA)

Client-Initiated Backchannel Authentication (CIBA) is an extension to the OpenID Connect system [17]. CIBA decouples the client application and authentication server, avoiding redirection through the user's browser [19]. In order to decouple the client application and authentication server, CIBA introduces two devices the Consumption Device and Authentication Device, as well as two new endpoints: the backchannel authentication endpoint and backchannel client notification endpoint [16]. A consumption device is a device used by the user to access the service, such as a mobile device. The authentication device is used by the user to authenticate and give consent. The authentication endpoint is responsible for starting an out-of-band authentication of the end-user by sending a direct HTTP POST from the Client to the OpenID Provider's Backchannel Authentication Endpoint. The notification endpoint is responsible for informing the end-user about the success or failure of the out-of-band authentication.

3.5 JSON Web Token (JWT)

JWT is a claims [20] container represented as JSON [21] object, encoded using Base64-URL-safe [22] encoding, ready to be transmitted between two parties [23]. These claims can be authenticated and relied upon as it has been digitally signed.

3.6 Cross-interoperability of Token-Based authentication protocols.

As explained earlier some token based protocols are either extensions to other schemes, i.e. OAuth is extended by OIDC which is in turn extended by CIBA, or payload to another one i.e. SAML and JWT work as packing carrier for OAuth, ODIC and CIBA. Finally, SAML could be converted to JWT and act as a payload for it. Table 1 represents summery of relations between token protocols and how can protocols are used interchangeably.

Table 1: Cross-interoperability of Token-Based authentication protocols

	OAuth			
SAML	Integrated using access and refresh tokens	SAML		
OIDC	OIDC expands OAuth 2.0 by including an extra layer for identity verification.	Access, refresh, and ID tokens are packed as SAML objects.	OIDC	
CIBA	Extends OIDC Which Extends OAuth. CIBA decouples client applications from the authentication server	Access, refresh, ID, and notification tokens Packed into SAML Objects	CIBA Extends OIDC by decoupling client applications from the authentication server	CIBA
JWT	Access and refresh tokens packed into JWT objects	SAML could be converted To JSON objects and loaded into JSON payload	Integrated by packing access, refresh tokens, and ID Tokens into JWT	Access, refresh, ID, and notification tokens Packed into JWT Tokens

3.7 Common Vulnerabilities in Token Authentication Protocols

These protocols, essential for ensuring secure digital communication, possess several vulnerabilities that can be manipulated by malicious individuals. Recognizing these flaws and the potential security risks they present is essential for formulating tactics to safeguard confidential data and systems [6, 1, 5].

3.7.1 Stealing Tokens through Redirection and URL Manipulation

Various token authentication protocols such as OAuth and OIDC share a prominent weakness related to insecure redirection and URL manipulation [24, 25]. This vulnerability can be leveraged by attackers to intercept or redirect users during the authentication procedure in order to pilfer credentials or tokens. An example of this is demonstrated in OAuth, where the reliance on redirection can be exploited if the redirect URIs lack stringent validation, enabling attackers to redirect responses and seize authorization codes or tokens.

3.7.2 Cross-Site Request Forgery (CSRF)

The OpenID Connect (OIDC) protocol, derived from OAuth 2.0, is susceptible to Cross-Site Request Forgery (CSRF) attacks, particularly when the state parameter is absent or improperly utilized. In such cases, malicious actors can deceive users into performing unauthorized actions within a web application where they are logged in. [25] It is imperative to adopt robust methods for generating and validating CSRF tokens to counteract this threat, and this precaution must be consistently implemented across various protocols to safeguard the continuity of user sessions.

3.7.3 Reusability of Access Token

Tokens with long lifetimes, has no expiration field or Stored in Browser History can be hijacked and reused by attackers [24] to gain access to unauthorized resources [26]. OAuth, SAML are sensitive to this class of attacks.

3.7.4 Signature and Assertion Flaws

Both JWT and SAML are vulnerable to issues related to signatures and assertions. In the case of JWT, inadequate implementation or validation of signature algorithms can enable attackers to manipulate the token or evade signature verification [27]. Likewise, SAML can be compromised through the manipulation of assertion statements, enabling attackers to assert false identities or attributes if these statements are not adequately protected.

3.7.5 Endpoint Security and Misconfiguration

CIBA and OAuth are susceptible to compromise due to vulnerabilities in endpoint security and misconfigurations. These weaknesses frequently stem from inadequate security protocols established at the endpoints managing authentication requests, or from improperly configured services that inadvertently make sensitive data or features accessible to unauthorized entities [27, 28].

4. MULTIFACTOR AUTHENTICATION

Multi-factor authentication (MFA) is a robust security mechanism that necessitates the use of multiple authentication methods from different categories of credentials to validate a user's identity during a login or other transaction [29]. This approach merges two or more separate credentials including knowledge-based (password), possession-based (security token), and intrinsic factors (biometric verification) to enhance security measures [10].MFA provides increased security as it establishes numerous security layers beyond single sign-ons to mitigate potential single point failures, also it strengthen protection against attacks such as brute force, dictionary, malware, key loggers, and others [30]. On the other hand, it has Challenges in usability as (MFA) tools may present various usability challenges, including user apathy, difficulty comprehending risk trade-offs [31], and the presence of user interfaces that are not intuitive, also researchers have found a lower adoption rate to be inevitable for MFAs, while avoidance was pervasive among mandatory use [32].

5. ONE TIME PASSWORD

One-Time Password (OTP) is a password that remains valid for a singular login session or transaction, commonly utilized in MFA protocols to enhance security, especially in critical operations such as online banking and digital payments [33]. OTPs offer robust security by virtue of their singe use, greatly minimizing the chances of password theft and replay attacks. These passwords are usually produced through cryptographic methods that guarantee unpredictability, enhancing their resistance to prediction. Time-based and Hash-based OTPs (TOTP and HOTP) represent a prevalent manifestation of this technology, wherein the password remains valid for a brief duration, thereby heightening the security of the authentication procedure. Nevertheless, despite the advantages they offer [34], OTP systems may have weaknesses. In the event that a confidential code utilized to create the OTP is leaked, an unauthorized individual could potentially create a legitimate OTP.

6. ONE TIME JWT (OTP-JWT) WITH AUTO EXPIRATION AUTO RECOVERY (AEAR) FEATURES PROPOSED MODEL

This proposed model has been introduced to mitigate vulnerabilities and attacks related to token-based mechanisms. This model requires a correctly signed JWT and involves three essential stages: registration, authentication, and service utilization. JWT is used as in implementation Example for the model which could be also utilized for other protocols such as OAuth, SAML, OIDC, CIBA as all are interoperable as described in Table 1.

6.1 Registration stage

During this stage, the client enrolls with an Identity Provider (IdP) [10]. The registration process typically involves providing profile details and access credentials such as a username or email address and password. An extra component is included in the access information, serving as the shared key (K) for the OTP algorithm OTP(K.C) where C represents a counter for HOTP or a Time (T) for TOTP. Following a successful registration, the user's information and login details, including (K), are securely stored by the CSP [35].

6.2 Login phase

The user utilizes their credentials for the purpose of logging in. Once authentication is completed successfully, the client-side application will receive an initial JWT for use in subsequent requests, with this initial JWT being denoted as JWT^0.

6.3 Service utilization (Consuming) phase

Client application per server request (Ri) uses a shared key (K) obtained in the registration phase to generate OTPi and use it to sign the entire initial JWT (JWT^0) including its signature part.

$$S^i = HMACSHA256(JWT^0.OTP^i)$$

Equation 1: Shared Key Generation

then the signature part of the original JWT0 is replaced by the new signature (Si). The newly generated JWT is called One Time JWT (OTJWT) and noted as (JWT^i) as it changes with the change of the OTP^i.

$$JWT^i = Base64(Header) + "." + Base64(Payload) + "." + Base64(S_i)$$

Equation 2: JWT Generation

On the server side when it receives JWT^i it uses the same steps as the client side described above to calculate the signature (Sic) then the calculated signature (Sic) is compared with the received Si to validate the request R_i. Because OTP^i signs the entire JWT^0 including the signature part then if JWT^i is validated this means that the original signature of the original JWT (JWT^0) is validated as it is included in S_i.

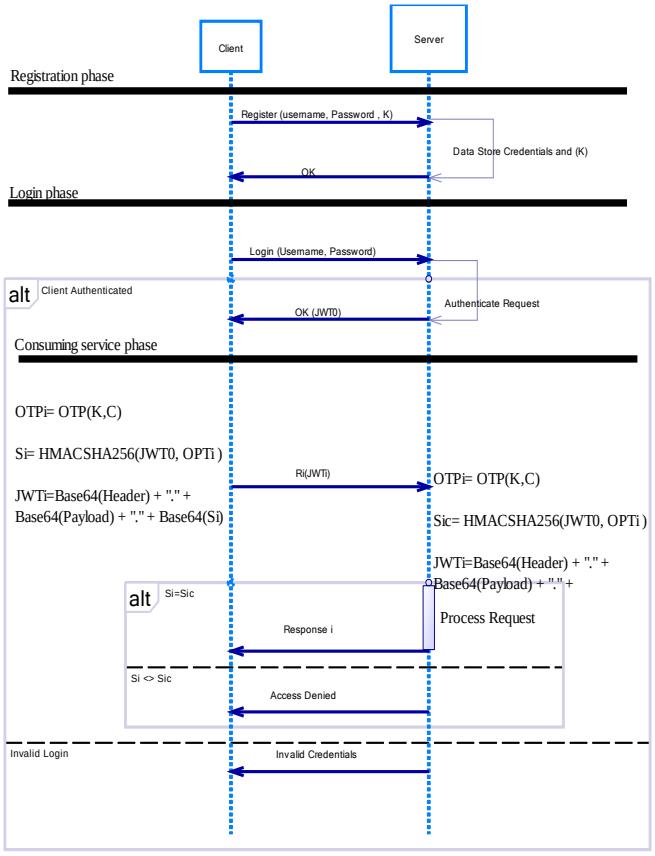

Fig 2 Proposed Model Flow

6.4 Implementation Guidelines

Implementing a One-Time JSON Web Token (JWT) with auto-expiration and auto-recovery features involves setting up a secure and efficient token management system. Next guidelines outline the steps and considerations necessary to implement these features effectively.

6.4.1 Setting Up JWT with Auto-Expiration.

Lifespan of tokens should be defined. JWTs support the "exp" claim, which specifies the expiration time. The value must be a number containing a Numeric Date value the shorter the expiration value the more secure from security prospective and more error prone from reusability prospective as any delta time difference between client user agent and server causes the token to be expired and should be recovered. Auto Expiration mitigates the risk of Access Token Reusability. As any stored Access token as it becomes useless because of using OTP Key,

6.4.2 Auto-Recovery Feature for Expired Tokens

Detection and Response: server side should be able to identify instances when an expired token is utilized. This usually requires intercepting the precise error generated when validating JSON Web Token (JWT) during the validation of an expired token. Token renewal endpoint should be capable of receiving expired tokens. Upon verification of the token's integrity and the identity of the user, a new token should be generated. Implementing rate limiting on the token renewal endpoint is essential in order to avoid misuse and abuse.

6.4.3 Security Considerations

Secret Key Management: Utilize a robust, OTP generated key that is stored securely. Base URL should be included as part of OTP seed key to mitigate any redirect attack that when client is redirected to fake URL the generated OTP will not match the next JWT^i according to Equation 2. Regularly change this key and ensure it is not fixed within your application code. It is crucial to utilize HTTPS for all communication involving JWTs to safeguard against interception by unauthorized individuals. Token Storage: Safely store tokens on the client's device. When developing web applications, it is advisable to utilize the HttpOnly flag in cookies in order to restrict access from JavaScript.

6.4.4 Conformity to regulations and the implementation of optimal methods

Regular updates are essential in order to ensure that JWT library and other dependencies are up to date, thereby safeguarding against potential security threats posed by known vulnerabilities. Compliance should be a top priority to ensure that your token handling practices align with the appropriate regulations, including but not limited to GDPR, HIPAA, and other relevant laws specific to industry.

7. IMPLEMENTATION PERFORMANCE COMPARISON BETWEEN NORMAL JWT AND OTP-JWT

The proposed algorithm has been executed utilizing the NodeJs Open source "MPL 1.1/GPL 2.0/LGPL 2.1" JWT standard Package [36] on both the server and client aspects. A genuine JWT package was utilized to assess regular functionality without alterations, with the execution time being noted. Subsequently, the recommended adjustments were integrated, and the test was performed once more, capturing the execution time. The evaluation took place within an environment comprising a Windows 10 machine having 32GB RAM Core i7 processor, serving as both client and server to eliminate any potential propagation or network lags. Each package (original and modified) was tested in 30 separate runs using fixed payload. Maximum execution time in case of Original JWT was 20 milliseconds while minimum was 11milliseconds. In case of proposed OTP-JWT maximum execution time was 22 and minimum was 12 milliseconds. Average overhead time delay due to modification was 1.8 milliseconds with average 14.15% increase in execution time.

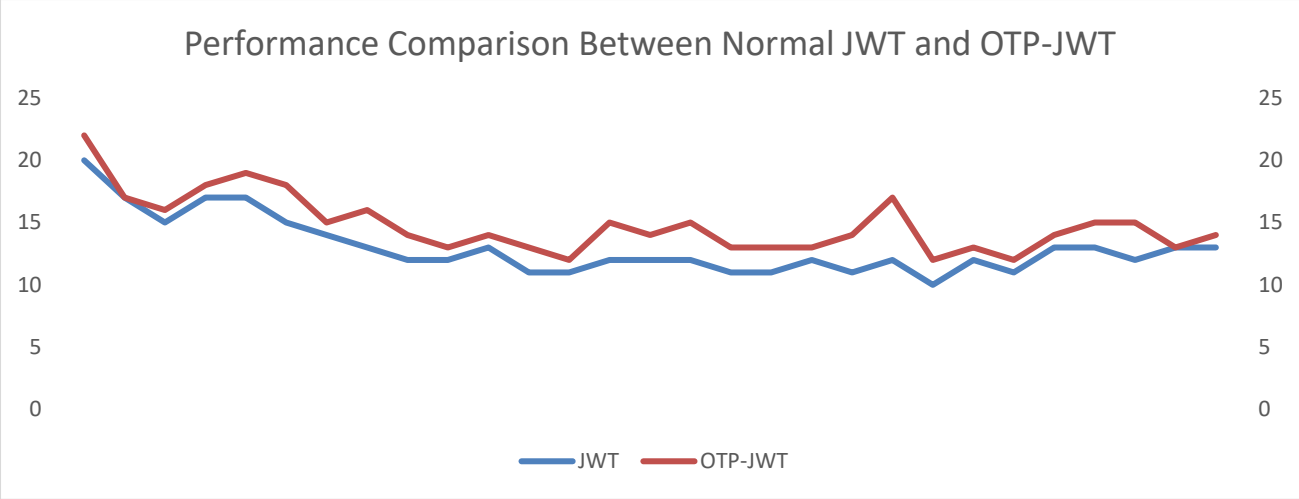

Fig 3: Performance Comparison between Normal JWT and OTP-JWT (30 Runs).

In Figure 3, it is demonstrated that the execution time ranges from 10 to 20 milliseconds, with a shift of 1.8 milliseconds upwards in the case of OTP-JWT. In practical situations, additional delays may occur due to network latency, security measures, and various other factors that impact the overall delay from the client-side to the server-side and vice versa.

8. CONCLUSION AND FUTURE WORK

In this work, a comparative analysis of various web-API token-based authentication methods is presented, along with a proposal for a new framework incorporating Multi-Factor Authentication (MFA) and One-Time Password (OTP). The proposed framework offers protection against well-known security attacks such as token theft through redirection, Cross-Site Request Forgery (CSRF), and Cross-Site Scripting (XSS). Implementation guidelines are provided, detailing the setup of auto-expiration, auto-recovery (AEAR) features, security considerations, regulatory compliance, and optimal methods. Future work should focus on incorporating enhanced encryption technique, such as post-quantum algorithms, to address emerging processing powers and new attacks. Additionally, efforts should be made to integrate the framework with emerging technologies such as the Internet of Things (IoT) and Blockchain. Developing an open-source Software Development Kit (SDK) will facilitate the integration of the framework into existing or new applications.

9. REFERENCES

[1] P. Siriwardena, Advanced API Security OAuth 2.0 and Beyond, Second Edition ed., San Jose, CA, USA: Apress, 2020.

[2] D. Hardt and M. , "Request for Comments: 6749 - The OAuth 2.0 Authorization Framework," October 2012. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc6749. [Accessed 21 4 2022].

[3] S. E. Peyrott, The JWT Handbook, Auth0 Inc, 2016-2018.

[4] Y. Sheffer, D. Hardt and M. Jones, RFC 8725 JSON Web Token Best Current Practices, Internet Engineering Task Force (IETF), February 2020.

[5] Ksenia Peguero and Xiuzhen Cheng, "CSRF protection in JavaScript frameworks and the security of JavaScript applications," High-Confidence Computing, 2021.

[6] Lodderstedt, "OAuth 2.0 Threat Model and Security Considerations RFC 6819," IETF, January 2013. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc6819.

[7] Microsoft, "RFC 6750 - The OAuth 2.0 Authorization Framework: Bearer Token Usage," October 2012. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc6750.

[8] Fielding, "Request for Comments: 2616," Network Working Group, 1999. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc2616.

[9] D. Rountree, Federated Identity Primer, Elsevier, Ed., Syngress, 2013.

[10] Paul A. Grassi, Michael E. Garcia and James L. Fenton, Digital Identity Guidelines, NIST Special Publication 800-63-3, June 2017.

[11] B. W. S. E. Hans-Jörg Vögela, "Federation solutions for inter- and intradomain security in next-generation mobile service platforms," International Journal of ELectronics and Communications, 2006.

[12] "Extensible Markup Language (XML)," [Online]. Available: https://www.w3.org/XML/. [Accessed 4 2022].

[13] "OASIS Open - OASIS Open," Organization for the Advancement of Structured Information Standards, [Online]. Available: https://www.oasis-open.org/.

[14] Campbell, "Assertion Framework for OAuth 2.0 Client Authentication and Authorization Grants RFC 7521," Internet Engineering Task Force (IETF), May 2015. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc7521. [Accessed April 2022].

[15] M. B. Jorge Navas, "Understanding and mitigating OpenID," Computers & Security, 2019.

[16] "OpenID Connect," [Online]. Available: https://openid.net/connect/. [Accessed April 2022].

[17] "OpenID Connect Core," OpenID Connect, November 2014. [Online]. Available: https://openid.net/specs/openid-connect-core-1_0.html. [Accessed April 2022].

[18] R. C. G. S. S. R. Amir Sharif, "Best current practices for OAuth/OIDC Native Apps A study of their adoption in popular providers and top-ranked Android clients," Journal of Information Security and Applications, 2022.

[19] "OpenID Connect Client-Initiated Backchannel Authentication Flow," openid.net, September 2021. [Online]. Available: https://openid.net/specs/openid-client-initiated-backchannel-authentication-core-1_0.html. [Accessed April 2022].

[20] "JSON Web Token jwt.io," jwt.io, [Online]. Available: https://jwt.io/. [Accessed April 2022].

[21] Tim Bray, "The JavaScript Object Notation (JSON) Data Interchange Format, rfc7159," Internet Engineering Task Force (IETF), March 2014. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc7159. [Accessed April 2022].

[22] S. Josefsson, "RFC 4648 - The Base16, Base32, and Base64 Data Encodings," IETF-Network Working Group, October 2006. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc4648. [Accessed April 2022].

[23] Michael B. Jones, John Bradley and Nat Sakimura, "JSON Web Token (JWT) rfc7519," Internet Engineering Task Force (IETF), May 2015. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc7519. [Accessed April 2022].

[24] D. Fett and Ralf Küsters, "A Comprehensive Formal Security Analysis," in Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna Austria, 2016.

[25] K. Munonye1 and Martinek Péter1, "Machine learning approach to vulnerability detection in OAuth 2.0," International Journal of Information Security, p. 223–237, 2021.

[26] Vasyl Bukovetskyi and Vasyl Rizak, "Developing The Algorithm And Software For Access Token Protection Using Request Signing With Temporary Secret," Eastern-European Journal of Enterprise Technologies, 2022.

[27] San-Tsai Sun and Author Picture Konstantin Beznosov, "The devil is in the (implementation) details: an empirical analysis of OAuth SSO systems," in ACM Conference on Computer and Communications Security, 2012.

[28] Marios Argyriou, Nicola Dragoni and Angelo Spognardi, "Security Flows in OAuth 2.0 Framework: A Case Study," in Computer Safety, Reliability, and Security, Trento, Italy, 2017.

[29] Aleksandr Ometov and Sergey Bezzateev, "Multi-Factor Authentication: A Survey †," mdpi Cryptography, 2017.

[30] S. Das, B. Wang, Z. Tingle and a. L. J. Camp, "Evaluating User Perception of Multi-Factor Authentication," School of Informatics, Computing, and Engineering.

[31] K. Abhishek, S. Roshan, P. Kumar and R. Ranjan, "A Comprehensive Study on Multifactor Authentication," in Proceedings of the Second International Conference on Advances in Computing and Information Technology (ACITY) , Chennai, India, 2012.

[32] T. Suleski1, M. Ahmed, W. Yang and E. Wang, "A review of multi-factor authentication," DIGITAL HEALTH, vol. 9, pp. 1-20, 2023.

[33] H. Kim and O. Yi, "Analysis of Distinguishable Security between the One-Time Password Extraction Function Family and Random Function Family," Applied Sciences, 2023.

[34] R. Dubey and J. S.Nair, "A Review on Secured One Time Password Based Authentication and Validation System," International Journal of Computer Sciences and Engineering, vol. 5, no. 6, pp. 232-236, 2017.

[35] Paul A. Grassi , James L. Fenton , Elaine M. Newton, Ray A. Perlner , Andrew R. Regenscheid , William E. Burr , Justin P. Richer , Naomi B. Lefkovitz , Jamie M. Danker , Yee-Yin Choong , Kristen K. Greene and Mary F. Theofanos , Digital Identity Guidelines Authentication and Lifecycle Management, NIST Special Publication 800-63B , June 2017.

[36] M. Robenolt, "jwt-node," npm, 2010. [Online]. Available: https://github.com/mattrobenolt/jwt-node.

[37] Y. Sheffer, Intuit, D. Hardt, M. Jones and Microsoft, "JSON Web Token Best Current Practices," Internet Engineering Task Force (IETF), February 2020.

[38] V. Radha and D. Hitha Reddy, "A Survey on Single Sign-On Techniques," Procedia Technology, vol. 134, no. 139, 2012.

[39] Tarek S. Sobh, "Identity management using SAML for mobile clients and Internet of Things," Journal of High Speed Networks, 2019.

[40] Scott Cantor, John Kemp, Rob Philpott and Eve Maler, "Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0," OASIS Standard, 15 March 2005. [Online]. Available: http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf. [Accessed April 2022].

[41] M. Jones, "RFC 7797 - JSON Web Signature (JWS) Unencoded Payload Option," Internet Engineering Task Force (IETF), February 2016. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc7797. [Accessed April 2022].

[42] M. Jones, "RFC 7518 - JSON Web Algorithms (JWA)," Internet Engineering Task Force (IETF), May 2015. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc7518. [Accessed April 2022].

[43] M. Jones, "RFC 7517 - JSON Web Key (JWK)," Internet Engineering Task Force (IETF), May 2015. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc7517. [Accessed April 2022].

[44] M. Jones and J. Hildebrand, "RFC 7516 - JSON Web Encryption (JWE)," Internet Engineering Task Force (IETF), May 2015. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc7516. [Accessed April 2022].

[45] R. Barnes, "Use Cases and Requirements for JSON Object Signing and Encryption (JOSE) -RFC 7165," Internet Engineering Task Force (IETF), April 2014. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc7165. [Accessed April 2022].

[46] authlete.com, authlete, [Online]. Available: https://www.authlete.com/developers/ciba/.

[47] Anthony Nadalin, Marc Goodner, Martin Gudgin, David Turner, Abbie Barbir and Hans Granqvist, "WS-Trust 1.4," OASIS , 25 April 2012. [Online]. Available: http://docs.oasis-open.org/ws-sx/ws-trust/v1.4/ws-trust.html. [Accessed April 2022].

[48] "Internet Assigned Numbers Authority (IANA)," IANA, [Online]. Available: https://www.iana.org/. [Accessed April 2022].

[49] "SAML Security Cheat Sheet," OWASP , [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/SAML_Security_Cheat_Sheet.html. [Accessed 2024].

[50] D. Rountree, Federated Identity Primer, Elsevier Inc, December 10, 2012. decisions", Journal of Systems and Software, 2005, in press.

10. ACKNOWLEDGMENTS

Many thanks to the Doctor Hawaf who helped me while he was alive and inspired me after he passed away.