OIPM: Access Control Method to Prevent ID/Session Token Abuse on OpenID Connect

Junki Yuasa[1], Taisho Sasada[1,2], Christophe Kiennert[3], Gregory Blanc[3], Yuzo Taenaka[1] and Youki Kadobayashi[1]

[1]*Graduate School of Science and Technology, Nara Institute Science and Technology, Ikoma 630-0192, Japan*

[2]*Research Fellow (PD) of the Japan Society for the Promotion of Science, Tokyo 102-0083, Japan*

[3]*SAMOVAR, Télécom SudParis, Institut Polytechnique de Paris, 91120 Palaiseau, France*

Keywords: Access Control, User Authenticity, Single Sign-On, OpenID Connect, FIDO, Session Hijacking.

Abstract: In recent years, the adoption of Single Sign-On (SSO) has been progressing to reduce the burden of user account management in web services. In web services using OpenID Connect, a primary SSO protocol, the user is authenticated using an ID Token (IDT) issued by the identity provider. The Session Token (ST) generated after authentication is often used to authenticate subsequent requests. However, attackers can acquire victims' IDT/ST through Cross-Site Scripting (XSS) or malicious browser extensions, enabling them to hijack sessions and impersonate victims. Related studies have proposed countermeasures against impersonation attacks using IDT/ST. Still, their effectiveness is limited against user-level malware (e.g., malicious browser extensions), making it impossible to prevent impersonation entirely. This study proposes OIPM (OpenID Connect Impersonation Prevention Mechanism) as a countermeasure to address the issue of impersonation using IDT/ST. Specifically, a unique private key is generated during user registration using FIDO, a passwordless authentication technology. This private key's signature is verified during authentication to prevent impersonation, and a temporary private key generated at authentication is used for subsequent request verification. Additionally, post-authentication high-confidentiality operations require user verification through FIDO-based gestures such as fingerprints to ensure security against user-level malware.

1 INTRODUCTION

Single Sign-On (SSO) has become widespread to reduce the burden of account management in recent years. OpenID Connect is one of the SSO protocols and is supported by large companies like Google and Microsoft. With OpenID Connect, users only need to log in once to the Identity Provider (IdP) such as Google or Microsoft; the IdP provides the user's authentication and attributes information to other web services, the Relying Party (RP). This enables users to utilize multiple services with a single account.

In OpenID Connect, the IdP issues ID Token (IDT) that contains user attributes post-authentication. After verifying the IDT's authenticity, the RP authenticates the user and issues a Session Token (ST) to manage the session. However, attackers can acquire users' credentials via phishing at the IdP or obtain IDT/ST through XSS and user-level malware (e.g., malicious browser extensions). This allows them to impersonate users, access sensitive information, or conduct financial transactions.

Openpubkey (Heilman et al., 2023) has proposed countermeasures against impersonation attacks using IDT. These attacks can stem from vulnerabilities like XSS or local threats like user-level malware. Openpubkey links a public key to the IDT for signature verification to prevent unauthorized use. However, this approach falls short against local threats where malicious browser extensions could access and use a victim's private key to forge a valid signature.

Studies have proposed methods beyond OpenID Connect to combat session hijacking by protecting ST from remote attacks (Nikiforakis et al., 2011; Tang et al., 2011; Bugliesi et al., 2015). However, their effectiveness is limited against user-level malware. Moreover, some approaches prevent impersonation by signing ST with secret information (Johns et al., 2012; Dacosta et al., 2012; De Ryck et al., 2015). However, these approaches also fall short of user-level malware that can access secrets and forge valid signatures.

This study addresses the stolen IDT and ST threats from XSS and user-level malware. We introduce an OIPM that ensures user authenticity by requiring sig-

Yuasa, J., Sasada, T., Kiennert, C., Blanc, G., Taenaka, Y. and Kadobayashi, Y.
OIPM: Access Control Method to Prevent ID/Session Token Abuse on OpenID Connect.
DOI: 10.5220/0012757900003767

In *Proceedings of the 21st International Conference on Security and Cryptography (SECRYPT 2024)*, pages 674-679
ISBN: 978-989-758-709-2; ISSN: 2184-7711

nature verification with a private key created during account setup. This mechanism verifies email ownership and employs Fast Identity Online (FIDO) for robust identity confirmation throughout the authentication processes. It also generates an unpredictable 'secret' during authentication and is used for continuous hash value verification and IP and device checks. To combat local threats, our method includes a re-authentication feature for web applications, which protects sensitive operations like accessing confidential data or conducting critical transactions. Our OIPM is designed for websites with SSO that demand high reliability.

2 OpenID CONNECT

OpenID Connect[1], based on OAuth 2.0[2], is an authentication protocol enabling ID federation across applications. It involves interactions among three entities: the IdP, the RP, and the user. The IdP authenticates the user and shares attribute information with the RP, providing services based on this authentication. OpenID Connect supports three authentication flows: Authorization Code Flow, Implicit Flow, and Hybrid Flow. This study focuses on the Implicit Flow[3], where the IDT is directly received through the browser. This flow is particularly used for simple login implementations without managing RP credentials, making it popular among RPs.

Fig. 1 shows the targeted use case using Implicit Flow. First, the user starts the authentication by clicking a login button in the browser (Process 1). The RP then requests authentication via browser redirection to the IdP (Process 2). The IdP presents a login page where the user inputs credentials and a consent page for user approval to share attribute information with the RP (Process 3). Upon consent, the IdP issues an IDT containing the user's identifier 'sub,' the RP's identifier 'aud,' and the user's email 'email' and sends it back via redirection (Process 4). Since processes 1 to 4 are Implicit Flow steps, the rest of the use case will be explained in Section 3.1.

3 PROBLEM DEFINITION

In this section, we describe the OpenID Connect use case and threat model targeted in this study and address the authenticity problems upon authentication and post-authentication in RP. We also define access patterns by attackers when user authenticity cannot be guaranteed.

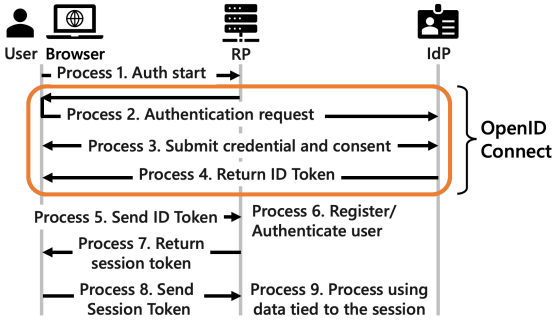

Figure 1: The Target Use Case of This Study.

3.1 Target Use Case

Fig. 1 illustrates this study's OpenID Connect use case, focusing on user registration and authentication via SSO. During SSO, the IdP verifies the user's credentials (e.g., email address, password). The user then receives an IDT through the browser and forwards it to the RP for authentication (Processes 5 and 6). After authentication, an ST to identify the user's session is issued and returned to the user (Process 7). The ST is included in a cookie for subsequent RP requests to facilitate session-related data processing (Processes 8 and 9).

3.2 Threat Model

This study assumes a threat model based on the web attacker model. The threat model includes three threats: a phishing threat, an XSS threat, and a user-level malware threat. Each threat operates distinctly without one being a prerequisite for another.

Phishing Threat: Attackers can lure victims to their hosted websites (e.g., attacker.com) and steal their credentials by having them enter the information on the website.

XSS Threat: Attackers can also exploit XSS vulnerabilities on the RP, executing arbitrary JavaScript on the victim's browser. This allows them to bypass browser-based XSS countermeasures (e.g., the HttpOnly attribute for cookies, Content Security Policy) and steal IDT/ST.

User-level Malware Threat: We assume attackers can infect the client with malware to consider robust countermeasures against compromising user devices. We specifically consider malware with only user-level access, such as malicious browser extensions, excluding malware with root-level access. We

[1]https://openid.net/specs/openid-connect-core-1_0.html

[2]https://datatracker.ietf.org/doc/html/rfc6749

[3]Note that while the Implicit Flow is deprecated in OAuth 2.0, it is not deprecated in OpenID Connect.

assume that the OS protection mechanisms are fully operational against user-level malware. This threat is realistic due to numerous malicious browser extensions installed on many users' devices in the Chrome web store. Our threat model does not include attacks against the FIDO2 protocol itself, assuming that it is secure. Users are presumed to use physical authenticators (not virtual) certified at L2 or above by the FIDO Alliance.

Finally, we do not assume that parties other than the user in OpenID Connect (RP, IdP) can be compromised. For instance, attackers cannot compromise the RP to alter the authenticity verification results or the IdP to steal the secret keys for signing IDT.

3.3 Authenticity Problems

Authentication in RP: The IDT includes a nonce to mitigate unauthorized acquisition and prevent replay attacks. However, if an attacker acquires a valid, unused IDT, e.g., using malicious browser extensions, they can impersonate the user by submitting it to an RP that recognizes the RP identifier (aud). Additionally, attackers may acquire a victim's credentials at the IdP through phishing, allowing them to obtain a new IDT. By submitting these tokens to the RP, attackers can compromise user authenticity and log in as the victim.

Post-Authentication in RP: STs are managed as cookies in browser storage. Attackers can exploit the XSS vulnerability on the RP to execute JavaScript in a victim's browser or use malicious browser extensions to acquire the ST from the victim's browser. So, attackers can use the stolen ST to perform post-authentication operations as legitimate users.

Potential Access Patterns: During authentication in the RP, attacker access patterns with IDT include: (IT-1) stealing and submitting the IDT; (IT-2) using stolen credentials to obtain and submit a new IDT from the IdP. Both scenarios allow requests from either the attacker's or the victim's browser.

In the post-authentication phase, attacker access patterns with ST are: (ST-1) stealing the ST via XSS and making requests from the attacker's browser; (ST-2) using the ST from a compromised victim's browser due to XSS or browser hijacking.

4 RELATED WORKS

Studies on impersonation attacks using IDT (Mainka et al., 2017) show that attackers can tamper with or forge tokens using a victim's identifier. This attack can be detected by verifying the signature of the

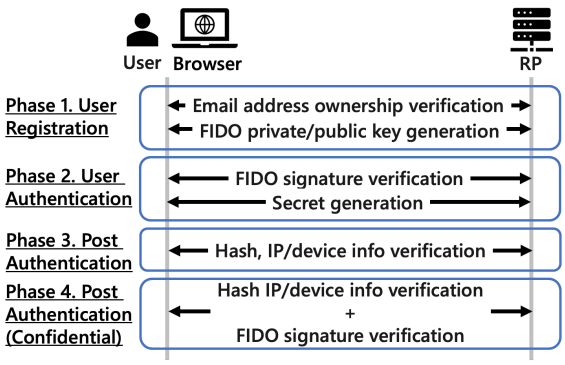

Figure 2: Overview of OIPM.

IDT. However, if legitimate IDT can be stolen, the signature value would be correct, making impersonation undetectable. The Openpubkey method (Heilman et al., 2023) addresses this by embedding users' public keys in IDTs for signature verification, enhancing security against impersonation. It also introduces the MFA-cosigner feature, which requires additional authentication beyond the IdP, increasing security but also adding complexity. Nonetheless, malicious browser extensions might still enable attackers to forge valid signatures.

Session hijacking, where STs are stolen through exploits like XSS or malicious browser extensions, has led to several defensive solutions. SessionShield (Nikiforakis et al., 2011), Zan (Tang et al., 2011), and Cookiext (Bugliesi et al., 2015) protect STs by managing them outside the browser or by setting secure cookie attributes, though their defense against browser extensions is limited. Alternatives like BetterAuth (Johns et al., 2012), One-time cookies (Dacosta et al., 2012), and SecSess (De Ryck et al., 2015) enhance security by generating unique hash values for each request using HMAC authentication, which confirms the possession of secret user information. One-time cookies use TLS for secret exchange, while BetterAuth and SecSess utilize Diffie-Hellman for secret sharing without TLS. However, these methods cannot prevent impersonation if malicious extensions access the secret to forge valid hashes.

This study highlights the limitations of current defenses against impersonation using IDT and session hijacking, especially concerning local threats like malicious browser extensions. We propose a new countermeasure to prevent impersonation involving IDT and ST.

5 OIPM DESCRIPTION

5.1 Verifying Authenticity of ID Tokens

This section proposes a method to verify user authenticity during authentication. It begins with checking the user's IDT and verifying email ownership at registration, preventing attackers without email access from registering. Users generate private and public keys during registration. Based on FIDO2, signature verification ensures security as the private key is stored in a FIDO device that is resistant to attacks, requiring a PIN or biometric verification. FIDO2's features, like KHAccessToken and authenticatorClientPIN, are safeguarded against user-level malware, preventing attackers from forging signatures(Kuchhal et al., 2023).

Key Pair Generation During User Registration: User registration (phase 1 in Fig. 2) involves generating a key pair, starting with the user registering at the RP, followed by IdP authentication and receiving an IDT. The IDT is sent to the RP for validation, and the RP emails a confirmation URL to the user. Upon clicking this URL, a personal confirmation token is validated by the RP, which then issues a session-specific challenge to the authenticator through the browser. User authenticity is confirmed by biometric or PIN verification on the authenticator, generating a key pair with the private key stored securely in the authenticator. The public key, sent to the RP along with the IDT, is saved for future login authenticity verification.

Authenticity Verification Using Private Key at Authentication: During authentication, the signature created by the user is verified (phase 2 in Fig. 2). Specifically, the RP uses a private key to generate a signature that includes a random string, known as a challenge value. This signature, along with the IDT, is then transmitted to the RP, which uses the public key to verify the validity of the signature. The authenticity verification process using a signature during authentication is detailed in Fig. 3.

The user authentication process at the RP starts with the user logging in and receiving an IDT from the IdP, which is then sent to the RP for verification. The RP issues a session-specific challenge to the authenticator to prevent replay attacks and ensure session uniqueness. The user confirms their identity through biometric or PIN authentication on the authenticator, which then uses the private key to create a signature. This signature, along with the IDT, is sent to the RP, where the public key, provided during registration, is used to validate the signature and authenticate the user. Although authentication can be

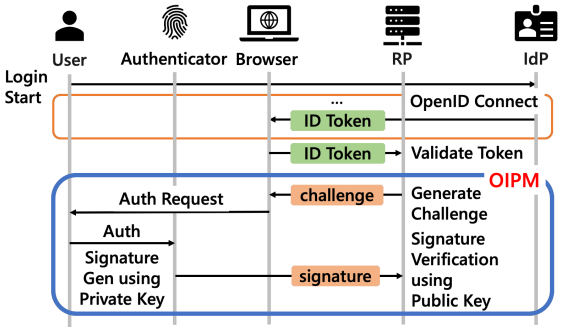

Figure 3: Authenticity Verification Using Private Key at Authentication.

achieved solely through FIDO-based signatures, it is crucial for SSO that RPs continuously reference and update IdP user information for personalized experiences. Therefore, transmitting the IDT alongside FIDO-based signatures remains essential.

5.2 Verifying Authenticity of Session Tokens

This section proposes a method for verifying user authenticity in post-authentication. Specifically, when users have been logged in, a random string (secret) shared exclusively between the user and the RP is generated. This allows the verification of user authenticity at the time of post-authentication to be substituted and verified through the maintenance of the secret. The following explains the generation and verification method of the secret at the time of authentication and describes a re-authentication method to address the threat of secret theft.

Authenticity Verification Using Secret at Post-Authentication: During authentication, verifying authenticity with FIDO private keys requires a gesture. Requiring gestures for each post-authentication request would hinder convenience, so we use a shared secret between the user and RP instead. This 128-bit random string (secret) secures request authenticity and resists brute-force attacks. For post-authentication requests, the user-generated hash value is verified (phase 3 in Fig. 2). Specifically, a hash value is generated using a nonce value and the secret created by the RP, and this hash value, along with the ST, is sent to the RP. The RP then generates a hash value using the nonce value and secret and compares it with the received one.

After user authentication, the RP stores an unpredictable secret, the user's IP address and device details. Additionally, the RP issues a ST and returns it with the secret. The secret is stored in non-persistent

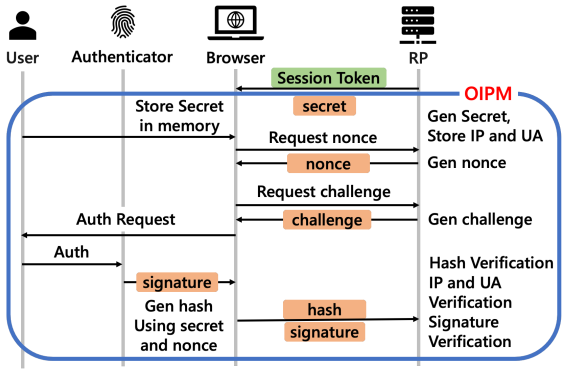

Figure 4: Authenticity Verification for Confidential Requests after Authentication.

in-memory storage using JavaScript closures[4]. Since variables defined within the scope of a function cannot be referenced from outside the scope, the secret cannot be referenced from external JavaScript provided via XSS. Users also obtain a nonce from the RP to prevent hash reuse. A JavaScript program on the browser combines this secret and nonce to generate a hash. The RP verifies the user by comparing this hash with its generated one and checks the request's IP and device details, denying access to attackers even with a valid ST. Access is granted only if both verifications pass. In addition, if a user's IP changes, they must log in again.

Re-authentication Methods to Address Secret Theft Threats: The post-authentication verification process, outlined previously, checks IP and device information to mitigate impersonation from stolen secrets or hashes. However, attackers could still capture the secret through malicious browser extensions that access communication logs. To address this, sensitive actions like accessing confidential data or making financial transactions require FIDO re-authentication, as phase 4 of Fig. 2 illustrates. This involves local authentication on the user's device with FIDO and verification of hash values, IP, and device info by the RP, including signature checks using a public key.

6 EVALUATION

Implementation Method: The experimental setup uses a Ubuntu 22.04 LTS server with a 6-core processor and employs Docker and Docker Compose for virtualization and container management. The OIPM is built with Node.js, Express as the web server, and MongoDB as the database, with its code hosted on GitHub[5]. We simulated the attacker, victim, RP, and IdP as containers on the same network for access pattern evaluation.

Table 1: Control Results for Each Access Pattern.

User	Information	Request Origin	Result
Victim	Credential	Victim	Allowed
	ST, Secret	Victim	Allowed
Attacker	Credential	Attacker	Denied
	Credential	Victim	Denied
	IDT	Attacker	Denied
	IDT	Victim	Denied
	ST, Secret	Attacker	Denied
	ST, Secret	Victim	Allowed
	ST, Hash	Attacker	Denied
	ST, Hash	Victim	Denied
	ST, Secret	Victim	Denied

6.1 Access Pattern Evaluation

This study evaluates OIPM's response to legitimate user and attacker-induced access patterns, as outlined in Section 3.3. Table 1 shows control outcomes for victims and attackers during RP login and post-login actions. Access is granted after verification if the ST and secret are transmitted from the victim's browser. However, access can be safeguarded against malicious browser extensions by mandating re-authentication using FIDO for sensitive operations. These findings demonstrate OIPM's effectiveness in preventing impersonation at both the authentication and post-authentication stages.

6.2 Performance Evaluation and Discussion

This study measures OIPM's impact on system performance by analyzing response times and computational resource usage before and after deployment.

Response Time: We assess the response time for authentication and post-authentication requests to the RP's server using the PlayWright browser automation tool, executing 100 requests at 500-millisecond intervals. Pre-OIPM authentication requests averaged 550 milliseconds, with a post-deployment increase of 100 milliseconds. Post-authentication requests went from a 50- to a 5-millisecond delay, and confidential requests needing FIDO re-authentication experienced an additional 10-millisecond delay (Fig. 5).

Computational Resource Load: We used Docker stats to monitor the resource usage of Docker containers for the RP, measuring average memory and

[4]https://developer.mozilla.org/en-US/docs/Web/JavaScript/Closures

[5]https://github.com/oidc-access-control/OIPM

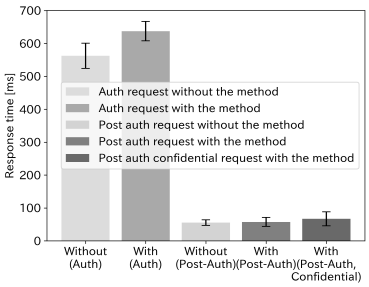

Figure 5: Response Time.

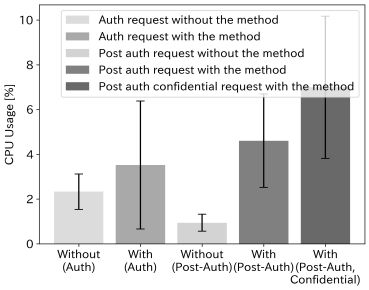

Figure 6: CPU Usage Rate.

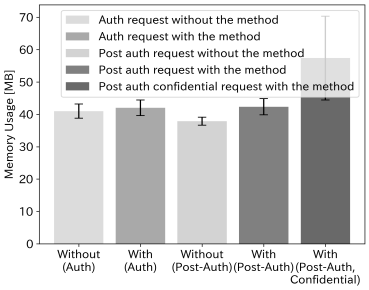

Figure 7: Memory Usage.

CPU utilization under 100 authentication and post-authentication requests before and after OIPM's implementation. Data on memory and CPU usage were collected at one-second intervals, with outliers removed using z-score for data points within +/- 3 standard deviations, as shown in Fig. 6 and Fig. 7. Post-implementation, CPU utilization doubled for authentication requests and quadrupled for post-authentication, with a five-fold increase for confidential requests. Memory usage remained stable, with a 1.5 times increase only for confidential requests.

Discussion: Verifying email ownership in OIPM is critical to prevent attackers from associating their keys with a victim's ID, a feature not present in standard SSO. It is important to create identity verification methods that maintain user convenience. Additionally, OIPM's requirement for FIDO implementation faces challenges due to its low website adoption rate. Future studies should also consider the costs of adopting FIDO.

7 CONCLUSION

This proposal attempted to ensure user authenticity in OpenID Connect at the RP. Our proposed OIPM method leverages FIDO private keys for signature verification and secret information for hash verification, protecting against user-level malware. The method shows access control can be managed effectively with little system impact. Future efforts will aim to create an easy identity verification process and assess FIDO costs.

ACKNOWLEDGEMENTS

This work was partly supported by JSPS KAKENHI Grant Number JP24K03045.

REFERENCES

Bugliesi, M., Calzavara, S., Focardi, R., and Khan, W. (2015). Cookiext: Patching the browser against session hijacking attacks. *Journal of Computer Security*, 23(4):509–537.

Dacosta, I., Chakradeo, S., Ahamad, M., and Traynor, P. (2012). One-time cookies: Preventing session hijacking attacks with stateless authentication tokens. *ACM Transactions on Internet Technology*, 12(1):1–24.

De Ryck, P., Desmet, L., Piessens, F., and Joosen, W. (2015). Secsess: Keeping your session tucked away in your browser. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, pages 2171–2176.

Heilman, E., Mugnier, L., Filippidis, A., Goldberg, S., Lipman, S., Marcus, Y., Milano, M., Premkumar, S., and Unrein, C. (2023). Openpubkey: Augmenting openid connect with user held signing keys. *Cryptology ePrint Archive*.

Johns, M., Lekies, S., Braun, B., and Flesch, B. (2012). Betterauth: web authentication revisited. In *Proceedings of the 28th Annual Computer Security Applications Conference*, pages 169–178.

Kuchhal, D., Saad, M., Oest, A., and Li, F. (2023). Evaluating the security posture of real-world fido2 deployments. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, pages 2381–2395.

Mainka, C., Mladenov, V., Schwenk, J., and Wich, T. (2017). Sok: single sign-on security—an evaluation of openid connect. In *Proceedings of the 2017 IEEE European Symposium on Security and Privacy*, pages 251–266.

Nikiforakis, N., Meert, W., Younan, Y., Johns, M., and Joosen, W. (2011). Sessionshield: Lightweight protection against session hijacking. In *Proceedings of the 3rd International Symposium on Engineering Secure Software and Systems*, pages 87–100.

Tang, S., Dautenhahn, N., and King, S. T. (2011). Fortifying web-based applications automatically. In *Proceedings of the 18th ACM conference on Computer and communications security*, pages 615–626.