논문 2024-4-33 http://dx.doi.org/10.29056/jsav.2024.12.33

A Study on Blockchain-Based API and WAAP Level Assignment Using Lifecycle as a Value Assessment Model

Minchul Kim*†

Abstract

In this paper, we propose a novel framework for the lifecycle management and value assessment of APIs and Web Application and API Protection (WAAP) using blockchain technology. By integrating blockchain, the framework ensures transparency, security, and traceability, enabling a robust value assessment model based on the interactions and updates logged throughout the lifecycle of APIs and WAAP. The proposed system also introduces a recursive verification process, enhancing security by continuously monitoring API and WAAP integrity. This recursive approach facilitates the verification and recovery processes by utilizing identical mechanisms, ensuring seamless API validation and WAAP restoration when vulnerabilities are detected. The research is motivated by the increasing reliance on APIs in modern application ecosystems and the limitations of traditional API gateways in addressing complex lifecycle and security challenges. Existing approaches often fail to provide the transparency and traceability required for robust security management. Our framework addresses these gaps by employing blockchain to maintain immutable records of API interactions, leveraging cryptographic hashing for integrity verification, and ensuring that only validated APIs meet operational standards. This approach not only enhances security but also establishes a foundation for systematic lifecycle management and value assessment.

keywords : Blockchain, API Security, WAAP, Lifecycle Management, Value Assessment, Integrity Verification

1. Introduction

In the ever-evolving digital landscape, enterprises increasingly rely on intranets to safeguard and manage critical business operations. These internal networks, commonly referred to as intranets, house sensitive data, such as customer information, intellectual property, and confidential business strategies. As external threats multiply in both number and sophistication, ensuring the security of these internal networks has become paramount for organizations across industries.

Historically, firewalls served as the first layer of defense, protecting the internal network perimeter by blocking unauthorized access from outside. Firewalls evolved as a fundamental component of network security, providing baseline protection for enterprises. However, as web technologies and Internet

* Pentasecurity Inc
† Corresponding Author: Minchul Kim
(email: minchul@pentasecurity.com)
Submitted: 2024.10.25. Accepted: 2024.12.05.
Confirmed: 2024.12.20.

usage expanded, web-based attacks targeting vulnerabilities in applications and protocols began to emerge. These challenges exposed the limitations of traditional firewalls, leading to the development of Web Application Firewalls (WAF) [1-3].

WAFs were designed to address vulnerabilities at the application layer by monitoring and filtering incoming and outgoing web traffic based on a set of predefined security rules. WAFs played a crucial role in mitigating attacks such as SQL injection, cross-site scripting (XSS), and other malicious payloads that could exploit application vulnerabilities. For many years, WAF was considered a reliable solution for protecting web applications that interfaced with internal systems, including those that operated within an organization's intranet.

Fig. 1 illustrates how traditional WAFs were primarily responsible for safeguarding the internal networks (intranets) of enterprises. The WAF acts as a barrier between external traffic, such as users, IoT devices, and external institutions, and the internal API servers and databases. By filtering potentially malicious traffic, WAFs ensured that only safe requests reached critical components of the organization's network infrastructure.

However, as the use of APIs expanded in modern web architectures, WAF alone was no longer sufficient to address the growing number of sophisticated threats targeting both web applications and APIs. The rise in API-based attacks highlighted the need for a more comprehensive security approach that could extend beyond web traffic filtering to include API protection as well.

This led to the evolution of Web Applications and API Protection (WAAP) [4-6]. WAAP integrates the traditional capabilities of WAF with additional features such as API security, bot mitigation, and DDoS protection. Unlike WAF, which primarily defends against common web application vulnerabilities, WAAP provides advanced mechanisms to address the growing API attack surface. The OWASP API Security Top 10 [7-10], which includes threats like broken object-level authorization and inadequate security configuration, has underscored the importance of securing both web applications and APIs.

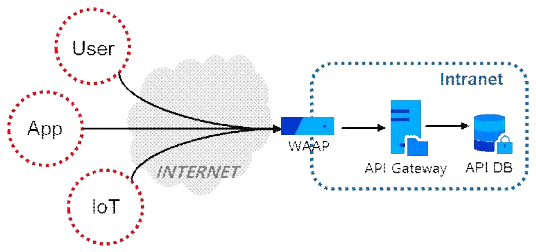

Fig. 1. Traditional WAAP for Protecting Intranet

The API Gateway [11,12] handles requests from clients, routing them to the appropriate backend services while managing tasks such as rate limiting, authentication, and load balancing. It is essential for controlling and monitoring API traffic, ensuring that external users interact with APIs securely and efficiently.

In our approach, rather than directly modifying the existing API Gateway, we propose managing APIs through a WAAP. The

WAAP acts as a security layer that oversees all APIs passing through the gateway. By utilizing WAAP, we can standardize the management of APIs and introduce a tagging system that categorizes APIs based on security level, lifecycle status, and value assessment criteria. This ensures that not only are the APIs being routed securely, but they are also systematically organized and protected, making it easier to monitor, verify, and update them as necessary.

Registering an API for external use is a crucial process to ensure security. Fig. 2 illustrates the key distinction between our proposed model and the traditional WAF-based approach shown in Fig. 1. While Fig. 1 focuses primarily on protecting the internal network (intranet), Fig. 2 expands the security model to protect both the internal network and external users. By ensuring that only secure APIs are accessible to external users and devices, such as IoT, our model guarantees that users interact with APIs securely.

A significant aspect of Fig. 2 is the integration of blockchain into the API lifecycle management. Blockchain is employed to securely track the registration, verification, and status of APIs, ensuring that all changes are transparently recorded in a decentralized and tamper-proof ledger. This adds an extra layer of integrity, making it easier to monitor and audit API interactions. By leveraging blockchain, any updates or modifications to APIs are securely logged, ensuring that only validated and secure APIs are made available for external use.

This approach not only safeguards the APIs themselves but also enhances the overall security for external users by providing comprehensive, end-to-end protection through WAAP. By establishing a lifecycle for APIs and WAAP, this approach also creates a value assessment model, aligning with the overall theme of this paper. By providing security throughout the API lifecycle—from registration to verification—our model ensures that the value and security of each API are continuously monitored and maintained.

This paper is structured as follows: Section 2 discusses the related works, covering existing API security solutions and previous advancements in WAAP. Section 3 provides background on key topics such as the OWASP API Security Top 10. Section 4 outlines our proposed method for registering APIs and the hash-based verification process to ensure integrity. In Section 5, we explain how blockchain is used to create an API and WAAP lifecycle, allowing us to establish a clear value assessment framework. Section 6 concludes the paper.

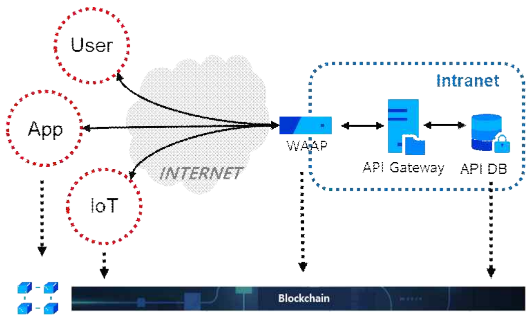

Fig. 2. Proposed WAAP with Blockchain-Integration

2. Related Works

There are several research efforts that utilize blockchain to enhance API security.

N. Moosavi et al. [13] reviewed the application of blockchain across various fields, particularly in IoT, identifying its potential for improving transparency and decentralization in security frameworks like API management. J. Govea et al. [14] explored how blockchain can strengthen cyber-resilience in critical infrastructure, focusing on its immutable and decentralized nature to reduce security incidents by 40%. Nihala Basheer et al. [15] developed a deep-learning model to manage threats in API calls by integrating transparency obligations to ensure system resilience. The study emphasizes improving API security through machine learning, particularly focusing on threat detection and management in API communications. Durre Zehra Syeda et al. [16] proposed a dynamic malware classification system that also categorizes APIs based on their usage in Windows executable files. This method enhances security by detecting malicious API behavior through machine learning, particularly addressing the categorization of API calls in malware detection.

3. OWASP API Security Top 10 [8]

The Open Web Application Security Project (OWASP) is a globally recognized organization dedicated to improving the security of software and web applications. One of its most influential contributions is the OWASP API Security Top 10, which identifies the most critical security risks specific to APIs. As modern web architectures increasingly rely on APIs for functionality, securing these APIs has become paramount. The OWASP API Security Top 10 provides a comprehensive guide to help developers and security professionals identify and mitigate the most common API vulnerabilities.

The latest OWASP API Security Top 10 for 2023 list includes the following threats:

- **Broken Object Level Authorization (BOLA)**: This vulnerability arises when APIs do not properly enforce authorization checks at the object level, allowing attackers to access unauthorized data by manipulating object identifiers.
- **Broken Authentication**: Weak or improperly implemented authentication mechanisms may allow attackers to impersonate legitimate users or hijack user accounts, leading to unauthorized access to APIs and their resources.
- **Broken Object Property Level Authorization**: This risk involves insufficient checks at the object property level, allowing attackers to view or manipulate sensitive properties they shouldn't have access to. For example, an attacker could change a user's role from 'user' to 'admin' without proper checks.
- **Unrestricted Resource Consumption**: APIs that do not limit resource

consumption may be overwhelmed by high traffic or resource-intensive requests, causing denial of service (DoS) attacks or elevated operational costs.

- **Broken Function Level Authorization:** In this vulnerability, APIs fail to correctly verify users' privilege levels, allowing unauthorized users to access sensitive functions or data that they shouldn't be permitted to.
- **Server-Side Request Forgery (SSRF):** SSRF vulnerabilities occur when APIs are tricked into sending malicious requests to internal services or third-party servers, potentially leading to data leakage or system compromise.
- **Security Misconfiguration:** This threat stems from improper configuration settings, such as leaving sensitive endpoints exposed or not patching known vulnerabilities, leaving APIs vulnerable to attack.
- **Lack of Protection From Automated Threats:** APIs lacking adequate protection against automated threats (e.g., bots or credential stuffing attacks) are susceptible to being exploited by malicious automated scripts.
- **Improper Inventory Management:** This risk arises from poor management of API endpoints, which can result in forgotten, outdated, or insecure APIs being exposed and susceptible to exploitation.
- **Unsafe Consumption of APIs:** APIs that consume data from untrusted or unverified sources without proper validation pose a risk, as attackers can inject malicious data, compromising the API and the broader system.

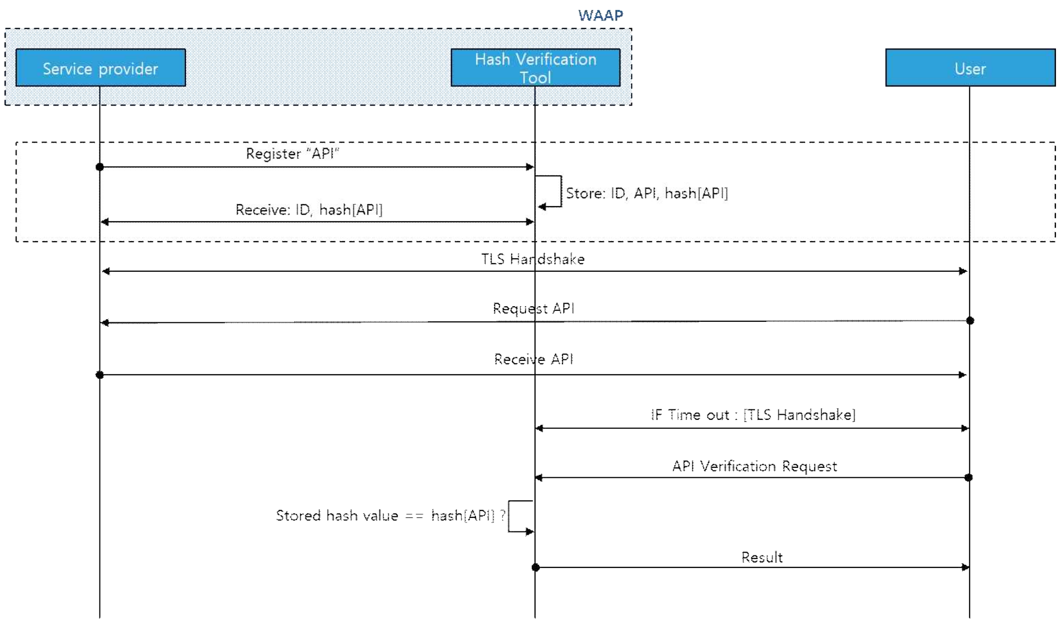

Fig. 3. Proposed API Registration and Hash-Based Integrity Verification Process

4. Proposed API Registration and Integrity Verification

Fig. 3 illustrates the process of API registration and hash-based integrity verification within a WAAP. The diagram depicts the interactions between the service provider, the WAAP, and the user, emphasizing how the API is secured and maintained throughout its lifecycle.

The process begins with the service provider registering the API in the WAAP. This includes submitting the API's details to the Hash Verification Tool within WAAP. The primary goal of this submission is to secure the API before making it available to users, ensuring that any future modifications or tampering can be detected through verification. Once the API registration request is submitted, the Hash Verification Tool generates a cryptographic hash for the API. This unique identifier reflects the API in its current state. Should any unauthorized changes occur, the hash will differ, serving as a means of detecting these modifications. After the hash is generated, the WAAP stores the API's ID, the API itself, and the corresponding hash value. This storage ensures that future versions of the API can be validated by comparing the newly generated hash with the stored value. Following this, the service provider receives a response from the WAAP, which includes the API's ID and the corresponding hash. This provides a reference point for verifying the API's integrity in future interactions or during potential changes. When a user interacts with the API, they can verify its integrity by comparing the hash of the current API version with the stored hash in the WAAP. This verification ensures that no unauthorized modifications have occurred. This entire process, as illustrated in Fig. 3, underscores the importance of utilizing a hash-based method to maintain the API's integrity and security. The use of this system helps ensure that unauthorized changes are detected promptly, securing the API from registration to end-user interaction.

Algorithm 1 demonstrates the detailed process of API registration and hash-based integrity verification in a WAAP, following the conceptual steps outlined in Fig. 3. The algorithm consists of three main functions, which collectively handle the API's registration, hashing, storage, and integrity verification.

Algorithm 1: API Registration and Integrity Verification

```
function registerAPI(api):
    api_hash = generateHash(api)
    storeInDatabase(api.id, api, api_hash)
    return api.id, api_hash

function verifyAPI(api):
    stored_hash = getStoredHash(api.id)
    current_hash = generateHash(api)

    if current_hash == stored_hash:
        return "API integrity verified"
    else:
        return "API integrity compromised"

function generateHash(api):
    return cryptographicHash(api)
```

Table 1 outlines the different API security levels and their corresponding criteria. At Level

Table 1. API Security Levels and Criteria

Level	Description
1	Secure and trusted API, fully approved by WAAP Level 1, no vulnerabilities detected
2	Partially secure, pending verification checks, minor vulnerabilities detected
3	Unverified or risky API, not approved for general use, multiple vulnerabilities detected

1, the API is fully approved by WAAP Level 1 and is considered secure and trusted, with no vulnerabilities detected. Level 2 APIs are partially secure and still pending verification, with minor vulnerabilities identified that are yet to be addressed. Finally, Level 3 includes APIs that are unverified or risky, and therefore not approved for general use due to multiple vulnerabilities being detected.

Table 2 presents the security levels of the WAAP itself. Level 1 WAAPs are fully operational and certified to manage secure APIs, with no vulnerabilities reported. Level 2 systems are under review due to security concerns, with minor vulnerabilities that are being addressed, leading to a slightly reduced confidence in their ability to manage API security. Level 3 WAAPs have severe vulnerabilities detected and require urgent action to restore their integrity, making them unsuitable for handling APIs at this level.

Table 2. WAAP Levels and Security Measures

Level	Description
1	Fully operational, certified to manage secure APIs, no reported vulnerabilities
2	Under review for security concerns, minor vulnerabilities being addressed, reduced confidence in API security
3	Severe vulnerabilities detected, require urgent action to restore integrity, not recommended for handling APIs at this level

5. Blockchain–based API and WAAP Lifecycle for Value Assessment

This chapter focuses on the lifecycle management of APIs and WAAP using blockchain technology, emphasizing its role in securing the system and maintaining transparency through immutable records. By integrating blockchain, the framework not only ensures that each interaction or change is tracked and logged for accountability but also enables a recursive verification process. This process unifies API validation and WAAP restoration mechanisms, facilitating seamless lifecycle management while enhancing the system's overall integrity. Furthermore, the proposed approach supports a value assessment model by systematically evaluating the security and functionality of APIs and WAAPs at each stage of their lifecycle.

Fig. 4 to 7 illustrate various scenarios in which the lifecycle of APIs and WAAPs are managed, highlighting how blockchain is utilized to ensure the security, integrity, and value of each component within the system.

Fig. 4 illustrates the initial registration of an API and its management through a WAAP (Level 1). The process begins with the user submitting an API, which is registered and verified through the WAAP. During this process, the blockchain ensures that the registration is tamper-proof, with the API's ID and integrity securely stored on the blockchain. If the API has a developer tag (sample code), it is filtered out and not allowed to be used at WAAP Level 1. Only APIs that pass this

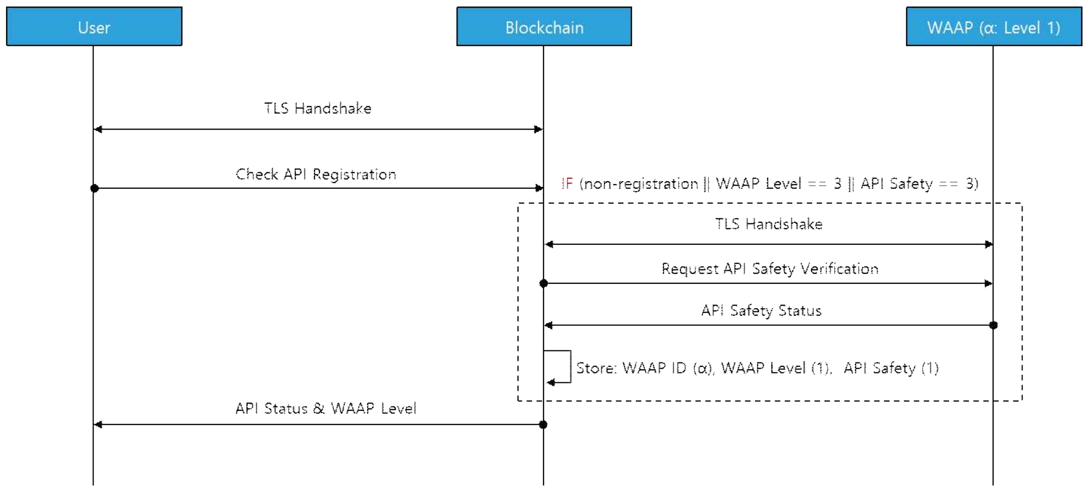
User
Blockchain
WAAP (α: Level 1)
TLS Handshake
Check API Registration
IF (non-registration || WAAP Level == 3 || API Safety == 3)
TLS Handshake
Request API Safety Verification
API Safety Status
Store: WAAP ID (α), WAAP Level (1), API Safety (1)
API Status & WAAP Level

Fig. 4. Initial API and WAAP Registration Process

verification are approved for usage. This step is critical in establishing the foundation of the API's lifecycle, ensuring that it can be securely monitored and assessed throughout its usage.

Fig. 5 demonstrates the vulnerability management process. When a vulnerability is discovered within an API, the blockchain is updated to reflect this issue. The WAAP acknowledges the vulnerability and begins the process of patching or mitigating the issue. If the WAAP fails to detect the vulnerability, its level is downgraded. Blockchain securely logs this change, allowing transparent tracking of vulnerabilities and responses. This ensures that any vulnerability in the API lifecycle is properly documented and addressed.

Fig. 6 depicts the revalidation of APIs handled by downgraded WAAPs. When the WAAP's level is reduced, the APIs that were processed by that WAAP must undergo revalidation.

In this process, the API is validated by querying another WAAP at Level 1. If the other WAAP validates the API successfully, it is considered secure. If not, the API's status is flagged as insecure, and its usage is restricted.

Fig. 7 showcases how a downgraded WAAP can recover. If vulnerabilities in the WAAP are patched or mitigated, the system automatically updates the WAAP back to Level 1. Blockchain securely tracks this recovery, providing a clear audit trail that shows when and how the WAAP was restored to its original level. This ensures that the WAAP can return to full functionality, securing APIs at Level 1 again.

By utilizing blockchain to track the lifecycle of APIs and WAAPs, we create a value assessment model. Each interaction, update, or modification is securely logged, providing a transparent and verifiable record of the system's integrity and security. This allows for continuous monitoring and assessment of the API and WAAPs, ensuring that their value is preserved throughout their lifecycle.

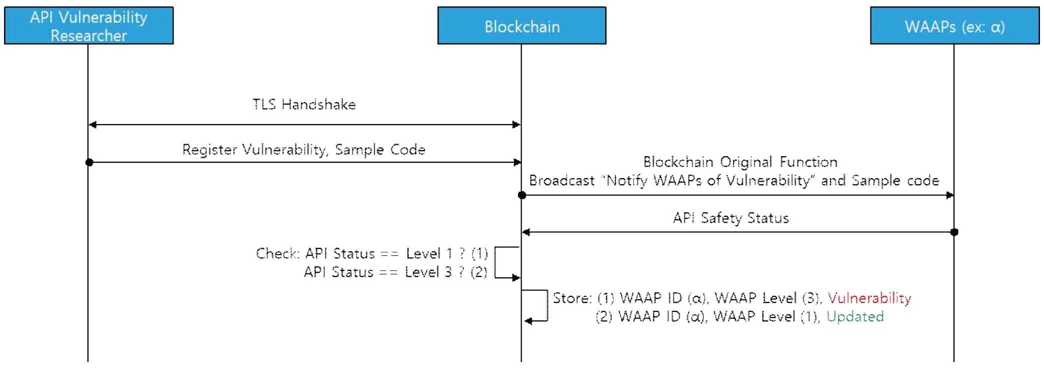
API Vulnerability Researcher
Blockchain
WAAPs (ex: α)
TLS Handshake
Register Vulnerability, Sample Code
Blockchain Original Function
Broadcast "Notify WAAPs of Vulnerability" and Sample code
API Safety Status
Check: API Status == Level 1 ? (1)
API Status == Level 3 ? (2)
Store: (1) WAAP ID (α), WAAP Level (3), Vulnerability
(2) WAAP ID (α), WAAP Level (1), Updated

Fig. 5. Vulnerability Management in WAAPs

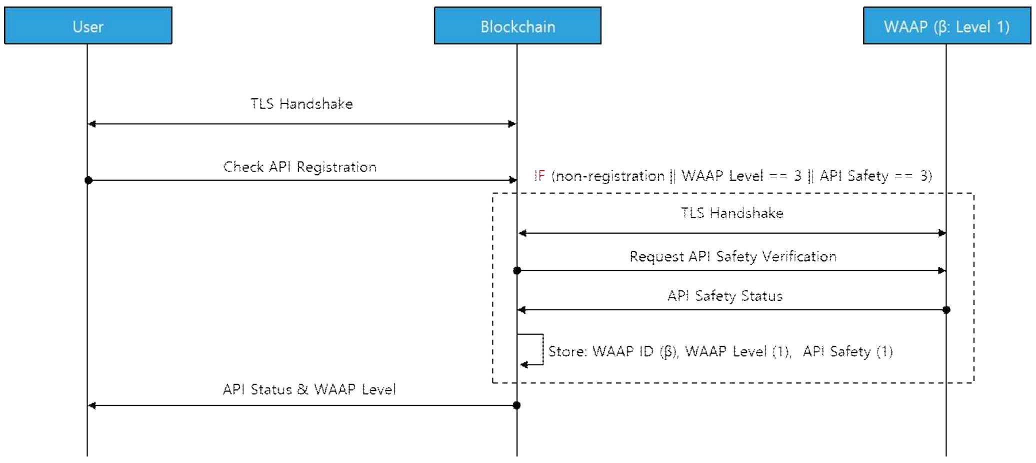
User
Blockchain
WAAP (β: Level 1)
TLS Handshake
Check API Registration
IF (non-registration || WAAP Level == 3 || API Safety == 3)
TLS Handshake
Request API Safety Verification
API Safety Status
Store: WAAP ID (β), WAAP Level (1), API Safety (1)
API Status & WAAP Level

Fig. 6. Lifecycle Update for WAAP

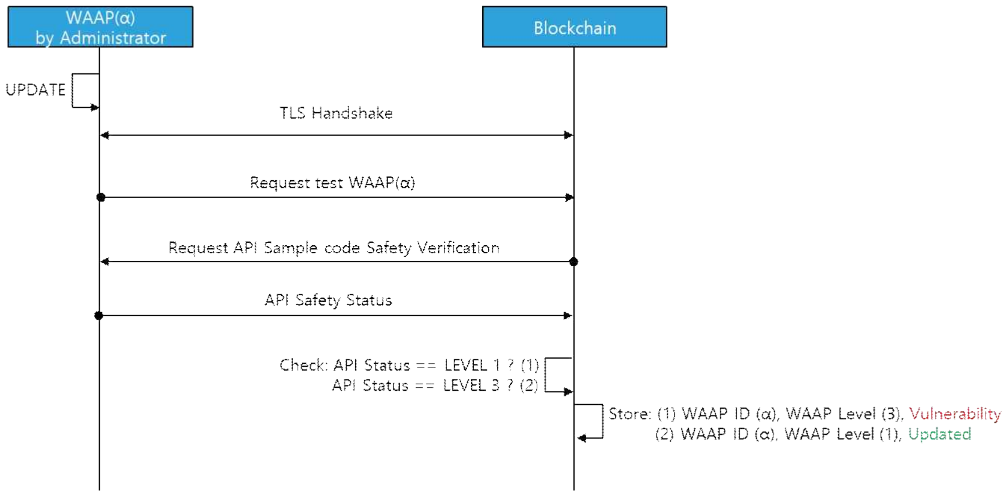
WAAP(α) by Administrator
Blockchain
UPDATE
TLS Handshake
Request test WAAP(α)
Request API Sample code Safety Verification
API Safety Status
Check: API Status == LEVEL 1 ? (1)
API Status == LEVEL 3 ? (2)
Store: (1) WAAP ID (α), WAAP Level (3), Vulnerability
(2) WAAP ID (α), WAAP Level (1), Updated

Fig. 7. Administrator-Controlled WAAP Updates

Algorithm 2: Recursive API and WAAP Lifecycle Algorithm

```
function manageAPILifecycle (api, waap_status, api_status, vulnerability_status):
    if api_status == "Not Registered":
        return registerAPIProcess(api)
    else if api_status == "Registered" and checkAPILevel(api.id) == 1 and waap_status == 1:
        if checkForTag(api):
            return "This API is a sample code (Tag found), cannot be used."
        return "API is registered and safe to use with WAAP Level 1."
    else if api_status == "Registered" and (checkAPILevel(api.id) < 1 or waap_status < 1):
        return "API or WAAP level is too low, cannot be used."
    return manageAPILifecycle(api, checkWAAPStatus(api.id), checkAPIStatus(api.id), checkVulnerabilityStatus(api.id))

function checkForTag(api):
    if "Tag" in api.metadata:
        return True
    return False

function registerAPIProcess(api):
    if queryBlockchainForWAAPLevel(1) == "Approved":
        api_hash = generateHash(api)
        storeOnBlockchain(api.id, api_hash, "WAAP 1")
        return "API registered successfully."
    else:
        return "API registration failed. WAAP Level 1 required."
```

Algorithm 2 outlines the process for managing API registration and WAAP verification recursively, as explained in Fig. 4 to 7. It begins by checking if an API is already registered. If it is not registered, the algorithm proceeds with the registration process, while also ensuring that APIs containing developer tags, such as sample code, are not approved at WAAP Level 1. Once registered, the algorithm checks the integrity of both the API and WAAP, ensuring they meet the required security levels, specifically Level 1. If the API or WAAP levels fall below the required threshold, the system revalidates the APIs handled by downgraded WAAPs. Additionally, the algorithm allows for continuous revalidation of APIs and automatic WAAP recovery, leveraging blockchain technology to securely track changes and provide decentralized auditing. This ensures that the lifecycle of both APIs and WAAPs is maintained securely, with every interaction being logged and monitored.

This recursive approach ensures that both APIs and WAAPs maintain their integrity and security throughout their lifecycle, automatically handling vulnerabilities and recovery processes.

6. Conclusion and future works

The system we developed offers a robust framework for assessing the value of APIs and WAAPs through the application of blockchain technology. This approach ensures transparency, security, and traceability, making it possible to create a trustworthy value assessment model based on each interaction and update logged on the blockchain. Our recursive verification process adds an additional layer of security, maintaining the integrity of APIs and WAAPs across their lifecycle, which is increasingly critical as APIs become central to modern application ecosystems.

However, it is important to recognize that while this model lays a strong theoretical foundation, practical testing has not yet been conducted. As APIs play an increasingly pivotal role in various industries, their security must be evaluated from multiple perspectives. Future work should focus on testing this framework in real-world scenarios to understand its operational limitations and to identify any potential challenges that may arise during implementation. This testing will allow us to refine the model and ensure that it can handle the complexities of actual environments.

Additionally, the framework's recursive verification and recovery mechanisms offer a unique advantage for managing vulnerabilities efficiently. By utilizing blockchain's immutable record-keeping and decentralized structure, these mechanisms not only streamline the validation process but also provide robust auditing capabilities for all API interactions. This ensures that even during recovery operations, the system adheres to stringent security protocols, maintaining trustworthiness across all stages.

Given that we have already outlined the experimental process for the system's lifecycle management, the next logical step involves testing the value assessment model itself. Understanding how the model functions in real-world operations is key to determining whether it accurately reflects the true value of the APIs and WAAPs it monitors. Moreover, future studies should explore scalability and integration challenges when applying the framework to large-scale enterprise networks or high-traffic API ecosystems. These investigations will help confirm the framework's applicability and identify enhancements to strengthen its deployment in diverse operational contexts.

This work is supported by the Korea Agency for Infrastructure Technology Advancement (KAIA) grant funded by the Ministry of Land, Infrastructure and Transport (Grant RS-2022-00144012)

References

[1] B. R. Dawadi, B. Adhikari, and D. K. Srivastava, "Deep Learning Technique-Enabled Web Application Firewall for the Detection of Web Attacks," Sensors, vol. 23, no. 4, 2023. DOI: https://doi.org/10.3390/s23042073.

[2] J. Chen, "WAFMAN: Web Application Firewall Management for Next Generation Applications," in Proceedings of the IEEE Symposium on Security and Privacy (SP), 2024, pp. 129-140. DOI: https://doi.org/10.1109/SP.2024.00024.

[3] M. Sepczuk, "Dynamic Web Application Firewall Detection Supported by Cyber Mimic Defense Approach," Journal of Network and Computer Applications, vol. 212, 2023. DOI: https://doi.org/10.1016/j.jnca.2023.103596.

[4] Penta Security, "Evolving Web Security: WAAP, Web Application and API Protection," Penta Security Blog. 2024. Available: https://www.pentasecurity.com/blog/evolving-web-security-waap/. [Accessed: Oct. 2024].

[5] Andrew Hoffman, Web Application Security, O'Reilly Media, Inc., 2024. ISBN: 9781098143930.

[6] E. M. Mohamed, M. F. Mohamed, and E. M. Saad, "A Blockchain-based Security Framework for Preventing DDoS and Enhancing Privacy in Cloud Environment," Future Internet, vol. 15, no. 10, 2023. DOI: https://doi.org/10.3390/fi15100326.

[7] S. Martin and L. Ruiz, "Designing Secure APIs for Cloud-based Applications," in Lecture Notes in Computer Science, vol. 13710, Springer, 2023, pp. 452-463. DOI: https://doi.org/10.1007/978-3-031-35314-7_33
[8] OWASP Foundation, OWASP API Security Top 10 - 2023, OWASP API Security Project, 2023. Available: https://owasp.org/www-project-api-security/. [Accessed: Oct. 2024].
[9] R. Botwright, OWASP Top 10 Vulnerabilities: Beginner's Guide to Web Application Security Risks, Pastor Publishing Ltd, 2024. ISBN: 9781839386299.
[10] L. Kree, R. Helmke, and E. Winter, "Using Semgrep OSS to Find OWASP Top 10 Weaknesses in PHP Applications: A Case Study," in Detection of Intrusions and Malware, and Vulnerability Assessment, F. Maggi, M. Egele, M. Payer, and M. Carminati, Eds., DIMVA 2024, Lecture Notes in Computer Science, vol. 14828, Springer, Cham, 2024. DOI: https://doi.org/10.1007/978-3-031-64171-8_4
[11] J. Alvarado-Valiente, J. Romero-Álvarez, E. Moguel, et al., "Technological diversity of quantum computing providers: a comparative study and a proposal for API Gateway integration," Software Quality Journal, vol. 32, pp. 53 - 73, 2024. DOI: https://doi.org/10.1007/s11219-023-09633-5.
[12] X. Cao, H. Zhang, and H. Shi, "Load Balancing Algorithm of API Gateway Based on Microservice Architecture for a Smart City," Journal of Testing and Evaluation, vol. 52, no. 3, pp. 1663-1676, May 2024.
DOI: https://doi.org/10.1520/JTE20220718.
[13] N. Moosavi and H. Taherdoost, "Blockchain Technology Application in Security: A Systematic Review," Blockchains, vol. 1, no. 2, pp. 58-72, Oct. 2023. DOI:
https://doi.org/10.3390/blockchains1020005.
[14] J. Govea, W. Gaibor-Naranjo, and W. Villegas-Ch, "Securing Critical Infrastructure with Blockchain Technology: An Approach to Cyber-Resilience," Computers, vol. 13, no. 5, article 122, May 2024. DOI:
https://doi.org/10.3390/computers13050122.
[15] N. Basheer, S. Islam, M. K. S. Alwaheidi, and S. Papastergiou, "Adoption of Deep-Learning Models for Managing Threat in API Calls with Transparency Obligation Practice for Overall Resilience," Sensors, vol. 24, no. 15, article 4859, July 2024.
DOI: https://doi.org/10.3390/s24154859.
[16] D. Z. Syeda and M. N. Asghar, "Dynamic Malware Classification and API Categorisation of Windows Portable Executable Files Using Machine Learning," Applied Sciences, vol. 14, no. 3, article 1015, Jan. 2024. DOI: https://doi.org/10.3390/app14031015.

Authors

Minchul Kim

2022.2 Ph.D. in Information Security from Korea University, Seoul, South Korea
2022.9-2023.4 Research Professor in the Department of Computer Science and Engineering at Korea University
2023.4-present Senior Manager at Pentasecurity, Inc.
<Research interests> Blockchain, Data Security, Network Security, Hardware Security, Communication Security, etc