# Sandboxing Building OS: Enabling Isolated Development Environment for Smart Buildings

Yoshiki Kitatani
*Graduate School of Informatics*
*Osaka Metropolitan University*
Osaka, Japan

Manato Fujimoto
*Graduate School of Informatics*
*Osaka Metropolitan University*
Osaka, Japan

Shingo Ata
*Graduate School of Informatics*
*Osaka Metropolitan University*
Osaka, Japan

***Abstract*—In Building Operating Systems (BOS), developing new applications often poses a risk of unintentionally affecting live environments, making safe and efficient development difficult. This challenge hinders the realization of a development approach that integrates development, testing, and operations within the BOS environment. To address this issue, it is essential to establish an access control mechanism that allows application behavior to be defined and validated under production-equivalent conditions, without impacting the live system. This paper proposes an authorization gateway architecture that meets this requirement by integrating OAuth 2.0 with a sandbox mechanism implemented as an overlay structure. The proposed Overlay based sandbox enables temporary and revocable permission assignment directly on the production infrastructure without modifying permanent rules. The architecture is implemented using a container-based deployment model and evaluated through performance profiling and comparative analysis. Experimental results confirm that the proposed method achieves flexible and secure access control with minimal overhead, providing a practical foundation for safe application development and continuous deployment on BOS platforms.**

***Index Terms*—access control, smart buildings, OAuth 2.0, Reverse Polish Notation, Sandbox, dynamic authorization**

## I. Introduction

In recent years, the importance of *smart buildings*, aimed at enhancing the operational efficiency and value of buildings, has been increasing. Traditionally, there has been a demand for centralized management of in-building facilities such as HVAC, lighting, and security. In response, integrated building management systems have been developed. However, these systems are often tied to specific vendors' products, resulting in limited scalability and high initial setup costs.

To overcome these limitations, the concept of a Building Operating System (Building OS) has gained attention [1]. A Building OS integrates various devices and equipment within a building to optimize operations and enable the development of new services by leveraging accumulated data. In practice, its adoption has improved management efficiency and created new value for both building managers and occupants.

Nevertheless, current Building OS implementations mainly serve building managers and/or maintainers, and face significant issues related to interoperability across platforms and limited application development by other stakeholders. Many systems are tailored to their own services and lack sufficient compatibility with others. Furthermore, non-administrative users face considerable restrictions in freely developing and utilizing applications.

Given this, there is an increasing need for an architecture that not only enhances individual building operations and management but also supports coordination across buildings and integration with smart city infrastructure. The key lies in establishing a *collaboration domain* to enable secure, efficient data sharing among different Building OS platforms. This makes it possible to share device information and functionality as needed, and empowers a broad range of stakeholders—including tenants, service providers, and general users—to create new services.

Furthermore, in this context, a major obstacle to the advancement of Building OS platforms is the risk of developing and testing applications directly on production environments. Such activities may unintentionally interfere with active equipment or other critical applications, making it difficult to establish a development process that ensures both safety and efficiency. This hinders the realization of integrated development, testing, and operation, thereby limiting new application creation.

Additionally, to ensure privacy and security for users accessing various zones and facilities, a centralized access control mechanism via the API Gateway is essential. For instance, in shared spaces like meeting rooms, different users require different permissions depending on time and context. Centralized management of such conditional and flexible access enables a balance between security and usability.

To meet these requirements, a secure, isolated development and testing environment that replicates production conditions without affecting live operations is necessary.

This paper proposes a sandbox framework composed of three core components: (1) an overlay layer for temporary and revocable environments, (2) a rule-based authorization table using Reverse Polish Notation (RPN), and (3) OAuth 2.0 for authentication and scope-based control.

By integrating these components, stakeholders can safely define, test, and apply policies on production infrastructure without compromising integrity. This framework supports secure, flexible application development in BOS, promotes

This work was partially supported by JSPS KAKENHI Grant Number 24K02934, and is also based on results obtained from a project, JPNP22006, subsidized by the New Energy and Industrial Technology Development Organization (NEDO).

978-3-903176-75-1 ©2025 IFIP

DevOps practices, and serves as a foundation for future smart city integration.

## II. Related Work

Various access control mechanisms have been proposed to date [2], [3], and centralized access control for IoT devices in smart homes and smart buildings has been actively studied [4]–[13]. Below we categorize prior studies, review their strengths and limitations, and position our sandbox-based contribution.

### A. General Access Control Models

Access control models can be broadly classified into four categories. **Mandatory Access Control (MAC)** enforces strict confidentiality but is inflexible in dynamic environments. **Discretionary Access Control (DAC)** provides fine-grained permissions but becomes unmanageable at scale. **Role-Based Access Control (RBAC)** simplifies administration via roles, but lacks flexibility in frequently changing contexts [14]–[16]. **Attribute-Based Access Control (ABAC)** enables context-aware control by evaluating user, resource, and environmental attributes, but suffers from policy complexity and scalability challenges [17]. Hybrid approaches have attempted to combine RBAC's manageability with ABAC's expressiveness [18]–[21], yet face difficulties in managing temporary, revocable, and conflict-free rules.

### B. BOS-Oriented Architectures

With the rise of smart buildings, BOS platforms have attracted research attention [4]–[8]. A framework such as **XBOS** [22] address platform unification, data modeling, and standardized device integration, but they assume trusted applications and do not provide mechanisms for temporary, revocable permissions or safe iterative development. **PlayGround** [5] advances safety by introducing semantic authorization and containerized isolation for untrusted applications. However, it does not offer an isolated DevOps oriented sandbox in which such updates can be tested while leaving other environments unaffected.

By contrast, our approach integrates directly with existing BOS deployments. Through an **overlay sandbox**, developers can test applications in production-equivalent environments while applying temporary and revocable permissions that do not alter baseline policies. This unique design addresses the unmet need for policy-preserving, safe DevOps in BOS.

### C. Related Concepts in Other Domains

Several related models from other domains are also relevant. **Temporal RBAC / GTRBAC** [23], [24]introduces time-bound constraints on roles. **capability-based access** such as CapBAC allows issuance of constrained, revocable tokens. These methods demonstrate important ways to handle temporary or conditional access. However, they typically operate by creating or attaching new roles/capabilities on demand. In contrast, BOS environments require systematically managing a wide variety of dynamically changing conditions (e.g., time, context, tenant, device hierarchy) in a **unified database representation**. Our Reverse Polish Notation (RPN)-based rule engine provides this capability by storing conditional expressions in a structured and evaluable form, ensuring efficient runtime evaluation without fragmenting the policy space.

### D. Remaining Challenges and Contributions

In summary, existing work either emphasizes integration (XBOS, BOSS), isolation through new runtimes (PlayGround), or conditional extensions (Temporal RBAC, CapBAC). None provide a generalized, BOS-specific mechanism to:

1) Safely manage temporary, revocable permissions without altering permanent rules,
2) Support policy-preserving DevOps directly in production-equivalent environments, and
3) Unify dynamic conditional policies in a scalable, database-driven representation.

This study addresses these gaps by proposing a comprehensive access control architecture that combines (i) an RPN-based rule engine, (ii) an overlay sandbox for temporary and revocable policies, and (iii) OAuth2.0 [25]for scope-based application-level control. Together, these components enable both fine-grained authorization and secure experimentation, laying a foundation for interoperable and adaptive Building Operating Systems.

## III. Proposed Method

This section presents the access control mechanism implemented in the BOS, clarifies design requirements, and outlines the proposed method.

As smart building technologies advance, facilities and sensors are increasingly managed through BOS, requiring more flexible access control for collaborative environments. A key challenge is enabling safe application development and testing without affecting live systems, since existing BOS tightly couple building resources with production environments. This hinders development agility and risks unintended operations.

To address this, BOS must allow developers to construct production-equivalent environments where temporary policies can be defined, evaluated, and revoked independently of permanent environment. Such mechanisms should also support testing directly on the production environment with a temporary and revocable environment.

Furthermore, the access control system must satisfy two core requirements: authorize users to access only the necessary equipment functions based on least privilege, and support efficient evaluation of complex, dynamic conditions (e.g., time, day, user attributes) without compromising performance.

To realize these requirements, the proposed design integrates three complementary mechanisms: a virtual authorization layer that grants and revokes temporary or conditional rights without affecting permanent policies, permanent role- and organization-based permissions maintained in a central authorization table, and OAuth2.0-based application-level assignment that restricts external applications to the minimum necessary resources.

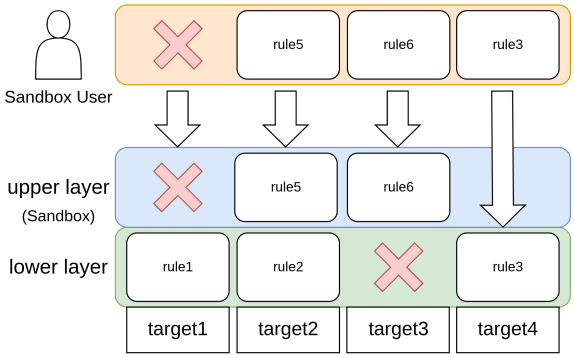


Fig. 1. Example of Overlay based Sandbox

### A. Details of the Proposed Method

The proposed method consists of the following components: the introduction of an Overlay based Sandbox [26] as an isolated environment layer, the construction of an authorization table using Reverse Polish Notation (RPN), and scope-based access control for external applications using OAuth 2.0 access tokens.

1. **Isolated Environment Using Overlay based Sandbox**

A key challenge in BOS is enabling safe application development and testing without affecting live systems, since existing implementations often couple access policies directly with the production environment. This makes it difficult to isolate experimental effects and introduces risks of unintended operations on real devices.

To address this, we propose the *Overlay based Sandbox*, which creates an isolated environment layered on top of the production structure. From the perspective of users or applications operating inside the sandbox, the environment behaves as if it were part of the production system, allowing realistic testing without requiring a separate testbed. However, the sandbox remains logically independent, so administrators can collectively remove all temporary rules and settings at once, restoring the system to its original state.

By introducing this overlay mechanism, BOS can replicate production-equivalent conditions for testing while maintaining strict separation from permanent policies. This isolation not only ensures safety but also enables additional capabilities such as temporary or time-limited permission assignment, controlled evaluation of new rules, and discarding of experimental policies. Moreover, the sandbox can also isolate device information and building resources, allowing developers to test updates or operations without affecting the live environment. Access conditions within the sandbox are described using the same structure as permanent rules, ensuring consistent evaluation and efficient implementation (Figure 1).

2. **Authorization Table using Reverse Polish Notation (RPN)**

To handle logical conditions such as date and time without dynamic parsing, we propose storing authorization rules in Reverse Polish Notation (RPN) within the database. RPN enables evaluation through simple stack operations without handling parentheses or operator precedence, allowing efficient processing of complex conditions.

With RPN-based settings, the evaluation order is explicitly managed and access decisions are made efficiently. Centralizing rules in the database also allows immediate updates and additions, enhancing both security and manageability in the Building OS.

Administrators define conditional expressions (e.g., by role, equipment type, day, or time), convert them into RPN, and store them in the authorization table. When an API request is made, the gateway references request details and context (e.g., current time or day), evaluates the RPN expression using a stack, and grants access only if the condition is satisfied.

This approach enables efficient runtime evaluation of complex conditions while ensuring flexibility through easy rule updates directly in the authorization table.

3. **Authentication and Scope Management Using OAuth2.0**

In this study, OAuth2.0 is adopted as the authorization mechanism for assigning permissions on a per-application basis. Widely used in the web domain, OAuth2.0 enables secure access by limiting each user's authorized scope.

An access token is issued following the OAuth2.0 flow. This token includes a *scope* that specifies the resources and functions the user is permitted to access. By utilizing scopes, access can be restricted to only the necessary areas. In the Building OS, scopes define control levels over facilities or access to specific data, preventing unnecessary access and ensuring that only required operations are allowed.

This mechanism enhances both security and operational efficiency in collaborative building management, making it well-suited to the objectives of this research. In practice, when accessing the Building OS API, user authentication is performed via the OAuth2.0 authorization code flow, and an access token is issued with permissions limited to the authorized functions.

Careful scope design is essential. Unlike typical service APIs with limited scopes, the Building OS handles a wide range of resources, potentially resulting in many scope entries. To manage this, scopes are expressed in path-based formats reflecting the building's hierarchical structure (e.g., building/floor/room/device). Regular expressions are also used to efficiently specify groups of equipment.

For example, allowing access to all devices on a given floor would be cumbersome if each device were listed individually. Instead, a path such as `/building1/2F/.*` compactly represents *all devices on the second floor of Building 1*. As shown in Figure 2, this method supports flexible and clear scope definitions, from individual devices to entire rooms, floors, or buildings.

By leveraging scope management in this way, the Building OS achieves flexible and efficient access control, balancing system-wide security with usability.

Furthermore, by combining OAuth2.0 with Reverse Polish Notation (RPN), the system supports not only user and function-specific permission assignment but also access con-

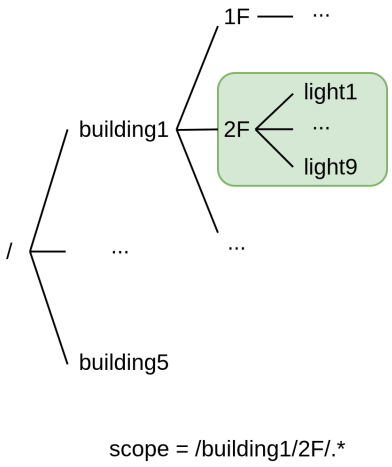


Fig. 2. Example of hierarchical structure for devices in a building

trol based on complex conditions. This enables both enhanced security and greater flexibility in building management.

## IV. Implementation

This chapter describes the implementation method for constructing the proposed API gateway. In this implementation, we adopt an architecture where the API gateway, which centrally manages requests to the building OS, and a server that provides authentication and authorization based on OAuth 2.0 are combined and deployed as containerized services using Docker.

### *A. Table Design for Isolation and Access Control*

Access control is implemented through several database tables that manage authentication, authorization rules, and protected API endpoints. Among them, three tables are central to our proposed method:

- **Authorization Rule Definition Table (auth_rule_definitions)**
  Stores logical access control conditions in Reverse Polish Notation (RPN). This enables efficient evaluation of complex conditions without parsing overhead.

```
CREATE TABLE auth_rule_definitions (
    rule_id      INTEGER,
    step_order   INTEGER,
    attribute    TEXT,
    op           TEXT,
    value        TEXT,
    PRIMARY KEY (rule_id, step_order)
);
```

- **Authorization Rule Assignment Table (auth_rule_assignments)**
  Maps rules to roles and API endpoints. The column `layer` distinguishes between permanent (“lower”) and temporary (“upper”) rules, and `sandbox_id` allows targeted temporary assignments. The `is_whiteout` flag supports overriding lower-layer rules.

TABLE I
Example of rule definition (`rule_id=1001`)

| rule_id | step_order | attribute | op | value |
|---|---|---|---|---|
| 1001 | 1 | Day_of_Week | == | Mon |
| 1001 | 2 | Day_of_Week | == | Wed |
| 1001 | 3 | logical_op | := | or |
| 1001 | 4 | Time | > | 12:00 |
| 1001 | 5 | Time | < | 17:00 |
| 1001 | 6 | logical_op | := | and |
| 1001 | 7 | logical_op | := | and |

```
CREATE TABLE auth_rule_assignments (
    id           SERIAL PRIMARY KEY,
    role_id      INTEGER NOT NULL,
    target_id    INTEGER NOT NULL,
    rule_id      INTEGER NOT NULL,
    sandbox_id   TEXT DEFAULT NULL,
    layer        TEXT CHECK
        (layer IN ('lower', 'upper'))
        DEFAULT 'lower',
    is_whiteout  BOOLEAN DEFAULT FALSE,
    expired_at   TIMESTAMP DEFAULT NULL
);
```

- **Endpoints Table (targets)**
  Defines metadata for API endpoints, including HTTP method and path, with regular expression support for flexible scope definitions.

Other supporting tables such as user management (users, roles, user_role) and OAuth 2.0 token storage follow conventional designs and are omitted here for brevity, since they do not directly affect the novelty of our proposal.

This streamlined schema focuses on the essential structures that enable fine-grained, temporary, and revocable access control while integrating seamlessly with OAuth 2.0.

### *B. Configuration and Operation of Permanent Authorization Rules*

Permanent access permissions are configured through the `auth_rule_definitions` and `auth_rule_assignments` tables.

Access conditions are defined in Reverse Polish Notation (RPN) and registered in `auth_rule_definitions`. Each step specifies an attribute (e.g., day, time, user), operator, and value. These rules are linked to roles and API endpoints (targets) in `auth_rule_assignments`, with permanent rules marked as `layer = 'lower'` and no `sandbox_id`.

When a user accesses the BOS, the gateway retrieves the user’s role and endpoint, identifies matching rules, and evaluates the RPN expression using a stack-based algorithm with runtime context values (e.g., current time). Access is granted only if the condition evaluates to true.

This table-driven design ensures rules are consistently enforced and easily updated. Table I shows an example permitting access only on Monday or Wednesday between 12:00–17:00. Figure 3 illustrates stack transitions during evaluation.

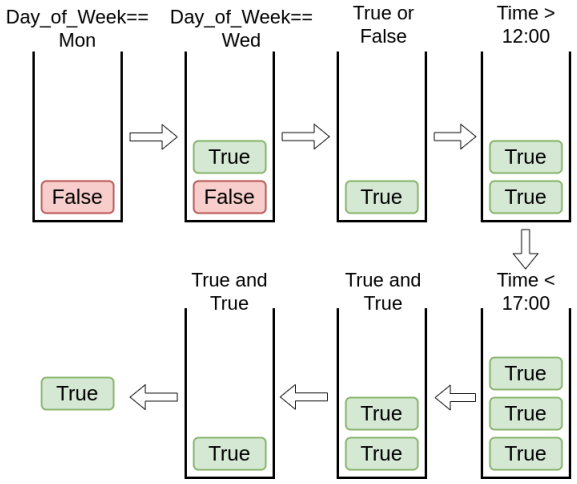


Fig. 3. Stack state transition diagram (with Overlay support)

TABLE II
EXAMPLE OF TEMPORARY RULE ASSIGNMENTS

| id | role_id | target_id | rule_id | sandbox_id | layer | is_whiteout |
|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1001 | NULL | lower | false |
| 2 | 1 | 2 | 1002 | sandbox01 | upper | false |
| 3 | 1 | 1 | null | sandbox02 | upper | true |

### C. Temporary Authorization with Overlay based Sandbox

Temporary access is often needed in testing or for short-term visitors. To support this, the Overlay based Sandbox adds a supplementary layer independent of permanent rules.

Temporary rules are also written in RPN, registered in `auth_rule_definitions`, and linked in `auth_rule_assignments` with `layer = 'upper'` and a `sandbox_id`. Optional fields (`expired_at`, `is_whiteout`) allow timed expiration or suppression of permanent rules.

Evaluation proceeds in a layered manner: for users operating within a sandbox, only the upper-layer rules associated with their `sandbox_id` are evaluated, effectively overriding the corresponding permanent rules. If `is_whiteout = TRUE`, the linked permanent rules are explicitly suppressed. In this way, sandbox participants are governed exclusively by temporary rules, ensuring that experimental access policies are applied without altering or combining with the baseline policies. This design allows safe testing and temporary overrides while guaranteeing that permanent policies remain intact for all other users. (Table II).

### D. Token Issuance and Scope Assignment Using OAuth2.0

Access control for external applications is implemented using OAuth 2.0, which grants limited permissions (scopes) based on user consent, ensuring secure and least-privilege access.

Before issuing tokens, applications are registered as clients in the `oauth2_clients` table with client ID, secret, and redirect URI. When a user accesses the BOS through such a client, the authorization server presents a consent screen for scope selection. Scopes are expressed as hierarchical paths (e.g., `/building1/3F/meetingroom1/light_control`), and regular expressions allow broader definitions such as `/building2/.*`.

After approval, an access token is issued and stored in the `oauth2_tokens` table, containing user ID, expiration, and permitted scopes. Applications attach this token to subsequent API requests.

Upon receiving a request, the API gateway verifies the token, then matches its scopes against the requested endpoint using the `targets` table. If matched, corresponding `auth_rule_assignments` are retrieved and evaluated. Thus, scopes first filter candidate endpoints, and final permission is determined by rule evaluation.

In summary, the proposed system integrates OAuth 2.0 authentication and scope filtering with permanent authorization and temporary overlays (via Overlay based Sandbox). The use of Reverse Polish Notation enables efficient processing of complex conditions, supporting secure and minimally privileged access across the Building OS's diverse functionality.

## V. EVALUATION

This chapter evaluates the proposed access control system in a containerized virtual environment, focusing on effectiveness, flexibility, and operational feasibility. We examined authorization behavior based on the rule table and OAuth 2.0 scopes, as well as the temporary control enabled by the Overlay based Sandbox. Results are summarized below.

### A. Verification of Effectiveness

By storing authorization conditions in Reverse Polish Notation (RPN) within structured tables, the system combines multiple contextual factors (e.g., day, time, user attributes) while ensuring efficient evaluation without runtime parsing. Conditions can be easily updated at the SQL level, and scope-based pre-filtering effectively restricts access and eliminates unnecessary requests.

### B. Verification of Flexibility with Overlay based Sandbox

The Overlay based Sandbox enables temporary access control independent of permanent rules. Permissions are defined with a `sandbox_id` and revoked as needed without altering baseline policies. For example, visitor access to a meeting room light can be granted for a limited time. Both permanent and temporary rules share the same RPN evaluation logic, ensuring consistency and negligible performance impact.

### C. Performance Evaluation and Bottlenecks

To confirm practical response times, we measured processing latency during API calls and identified potential bottlenecks. In the experiment, 1000 pseudo authorization rules (RPN format) were registered in both permanent and sandbox tables. JWT tokens were issued, and requests were sent ten times to an authorization-enabled endpoint. The average response time was **985.3** $\mu$s per request.

- **Retrieving Authorization Conditions from the Database** Based on the user's role and the target API endpoint, this process retrieves authorization conditions from the `auth_rule_assignments` and `auth_rule_definitions` tables. It simultaneously references both the upper and lower layers, including the sandbox, and selects applicable conditions based on the presence or absence of `sandbox_id`. This process took an average of **905.2** $\mu$s.
- **Evaluating Authorization Conditions Using a Stack** This process evaluates the retrieved rules written in Reverse Polish Notation sequentially. For each step, it compares against the runtime context (such as time, day of the week, or user attributes), and uses stack operations to derive a boolean result. This process took an average of **1.8** $\mu$s.

Profiling results showed that database access accounted for approximately **92%** of the total processing time. On the other hand, the condition evaluation based on Reverse Polish Notation was efficiently processed through stack operations and had only a minor impact on overall latency.

Therefore, the primary bottleneck is attributed to database queries, which may cause performance degradation under high access loads.

### *D. Performance Improvement through Caching and Evaluation of Overlay based Sandbox Impact*

To reduce database access overhead, this study introduces a caching mechanism using a Key-Value Store (KVS), in which all or part of the authorization rules are stored in memory.

With this mechanism, the set of authorization rules corresponding to a specific combination of user role, `target_id`, and `sandbox_id` is stored in the cache at the time of first access. Subsequent accesses retrieve the rules directly from memory for a fixed period, thereby avoiding repetitive database queries.

According to profiling results, the average response time per request when using KVS caching was reduced to **19.2** $\mu$s, representing an approximately **98%** improvement over the previous implementation.

To determine whether the introduction of the Overlay based Sandbox mechanism introduces any new performance bottlenecks, a comparative evaluation was conducted against a baseline configuration without the Overlay based Sandbox.

Specifically, the baseline configuration employed a simplified design in which authorization rules were directly assigned to `target_id` and `role_id` without any sandbox layer. The schema is as follows:

```
CREATE TABLE auth (
    id           SERIAL PRIMARY KEY,
    role_id      INTEGER NOT NULL,
    target_id    INTEGER NOT NULL,
    attribute    TEXT,
    op           TEXT,
    value        TEXT,
    rule_order   INTEGER
);
```

TABLE III
COMPARISON OF AUTHORIZATION PROCESSING TIME WITH AND WITHOUT OVERLAY BASED SANDBOX

| Configuration | With Cache ($\mu$s) | Without Cache ($\mu$s) |
|---|---|---|
| No Overlay | 24.0 | 963.2 |
| Overlay Upper | 19.2 | 985.3 |
| Overlay Lower | 26.1 | 1140.2 |

Using both the baseline and the proposed configurations, authorization processing times were measured under two conditions:

- **Cache Hit:** Authorization rules are found in the KVS.
- **Cache Miss:** Rules are retrieved from the database on each request.

Additionally, within the Overlay-enabled configuration, the performance difference between the **upper layer** (temporary authorization) and the **lower layer** (permanent authorization) was also analyzed to assess the processing impact of temporary rules.

Table III summarizes the average authorization response times under each configuration and condition.

As shown in Table III, caching had a significant effect on processing time, while the impact of the Overlay based Sandbox, or the difference between upper and lower layers, was minimal.

With caching enabled, all configurations achieved response times of 20–26 $\mu$s, maintaining high-speed processing regardless of the Overlay. Even without caching, latency stayed within about 1 ms, showing no major degradation from the added sandbox layer.

To verify that caching overhead is negligible in practice, we compared three endpoints: with authorization and caching, with authorization only, and without authorization. Concurrent requests were gradually increased, with each request including a 400 ms sleep interval to emulate realistic API use. Results are shown in Figure 4. The uncached endpoint saturated at about 400 concurrent requests due to database connection limits, while the cached endpoint sustained throughput comparable to the non-authorized endpoint and showed less saturation under high concurrency.

These results confirm that caching effectively reduces I/O and connection bottlenecks, maintaining stable performance even under heavy load. Overall, the Overlay based Sandbox provides flexible temporary access control with only minimal overhead.

## VI. CONCLUSION

In this paper, we designed and implemented a sandbox environment for Building Operating Systems (BOS) that enables secure, flexible, and efficient application development and deployment. Specifically, we developed an access control mechanism integrated into an API Gateway, which facilitates

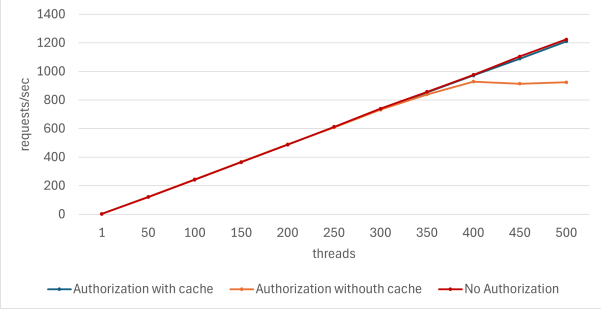


Fig. 4. Request throughput under increasing concurrent accesses: with and without authorization.

unified communication between heterogeneous BOS platforms and external applications.

The proposed system combines OAuth 2.0 for scope-based application authorization, a rule-based authorization table using Reverse Polish Notation (RPN) for efficient policy evaluation, and a layered overlay mechanism that allows temporary and revocable permissions to be applied independently of permanent rules. These components together form a sandbox framework that supports safe, condition-based access control without compromising system integrity.

To validate the system, we evaluated performance through profiling and comparative analysis. The RPN-based rule evaluation was shown to be highly efficient, completing in microseconds, while the main bottleneck was found in database access. By introducing a key-value store (KVS) cache, we significantly reduced average response times from nearly 1 ms to under 20 $\mu$s.

Further testing confirmed that the overlay layer adds negligible overhead, and caching maintains high throughput even under increased load.

## References

[1] S. Dawson-Haggerty, A. Krioukov, J. Taneja, S. Karandikar, G. Fierro, N. Kitaev, and D. Culler, "BOSS: Building operating system services," in *10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, Lombard, IL, Apr. 2013, pp. 443–457.

[2] J. D. Bokefode, S. A. Ubale, D. G. Modani, and S. S. Apte, "Analysis of dac mac rbac access control based models for security," *International Journal of Computer Applications*, vol. 104, no. 5, pp. 6–13, Oct. 2014.

[3] R. Ausanka-Crues and H. Mudd, "Methods for access control: Advances and limitations," *Harvey Mudd College*, vol. 301, p. 20, * Month Missing * 2006.

[4] L. Bindra, K. Eng, O. Ardakanian, and E. Stroulia, "Flexible, decentralised access control for smart buildings with smart contracts," *Cyber-Physical Systems*, vol. 8, no. 2018, pp. 1–35, Jul. 2021.

[5] X. Fu, Y. Liu, J. Koh, D. Hong, R. Gupta, and G. Fierro, "Playground: A safe building operating system," in *in Proceedings of the 15th ACM/IEEE International Conference on Cyber-Physical Systems (ICCPS)*, Hong Kong, China, May 2024, pp. 111–122.

[6] N. Xue, L. Liang, J. Zhang, and X. Huang, "An access control system for intelligent buildings," in *in Proceedings of the 9th EAI International Conference on Mobile Multimedia Communications (MobiMedia)*, Beijing, China, Jun. 2016, pp. 11–17.

[7] J. Koh, D. Hong, S. Nagare, S. Boovaraghavan, Y. Agarwal, and R. Gupta, "Who can access what, and when?: Understanding minimal access requirements of building applications," in *in Proceedings of the 6th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation (BuildSys)*, Nov. 2019, pp. 121–124.

[8] A. K. Sikder, L. Babun, Z. B. Celik, H. Aksu, P. McDaniel, E. Kirda, and A. S. Uluagac, "Who's controlling my device? multi-user multi-device-aware access control system for shared smart home environment," *ACM Transactions on Internet of Things*, vol. 3, no. 4, pp. 1–39, Sep. 2022.

[9] B. A. Mozzaquatro, C. Agostinho, D. Goncalves, J. Martins, and R. Jardim-Goncalves, "An ontology-based cybersecurity framework for the internet of things," *Sensors*, vol. 18, no. 9, Sep. 2018.

[10] R. Godha, S. Prateek, and N. Kataria, "Home automation: Access control for IoT devices," *International Journal of Scientific and Research Publications*, vol. 4, no. 10, p. 1, Oct. 2014.

[11] A. K. Malik, N. Emmanuel, S. Zafar, H. A. Khattak, B. Raza, S. Khan, A. H. Al-Bayatti, M. O. Alassafi, A. S. Alfakeeh, and M. A. Alqarni, "From conventional to state-of-the-art IoT access control models," *Electronics*, vol. 9, no. 10, Oct. 2020.

[12] M. Alramadhan and K. Sha, "An overview of access control mechanisms for internet of things," in *Proc. 26th Int. Conf. Comput. Commun. Netw. (ICCCN)*, Vancouver, BC, Canada, Jul. 2017, pp. 1–6.

[13] W. He, M. Golla, R. Padhi, J. Ofek, M. Dürmuth, E. Fernandes, and B. Ur, "Rethinking access control and authentication for the home internet of things (iot)," in *Proc. 27th USENIX Security Symposium (SEC)*, Baltimore, MD, USA, Aug. 2018, pp. 255–272.

[14] N. Xue, C. Jiang, X. Huang, and D. Liu, "A role-based access control system for intelligent buildings," in *in Proceedings of the 11th International Conference on Network and System Security (NSS 2017)*, ser. Lecture Notes in Computer Science, J. Lopez, J. Zhou, and M. Soriano, Eds., Jul. 2017, vol. 10394, pp. 710–720.

[15] N. Li and M. V. Tripunitara, "Security analysis in role-based access control," *ACM Transactions on Information and System Security*, vol. 9, no. 4, pp. 391–420, Nov. 2006.

[16] D. F. Ferraiolo and D. R. Kuhn, "Role-based access controls," in *in Proceedings of the 15th National Computer Security Conference*, Baltimore, MD, USA, Oct. 1992, pp. 554–563.

[17] L. Karimi, M. Abdelhakim, and J. B. D. Joshi, "Adaptive abac policy learning: A reinforcement learning approach," *arXiv preprint arXiv:2105.08587*, May 2021.

[18] S. Ameer, J. Benson, and R. Sandhu, "Hybrid approaches (abac and rbac) toward secure access control in smart home iot," *IEEE Transactions on Dependable and Secure Computing*, vol. 20, no. 5, pp. 4032–4051, Sep. 2023.

[19] D. R. Kuhn, E. J. Coyne, and T. R. Weil, "Adding attributes to role-based access control," *Computer*, vol. 43, no. 6, pp. 79–81, Jun. 2010.

[20] B. Bezawada, K. Haefner, and I. Ray, "Securing home iot environments with attribute-based access control," in *Proc. 3rd ACM Workshop Attribute-Based Access Control (ABAC)*, Tempe, AZ, USA, Mar. 2018, pp. 43–53.

[21] S. Ameer, J. Benson, and R. Sandhu, "An attribute-based approach toward a secured smart-home iot access control and a comparison with a role-based approach," *Information*, vol. 13, no. 2, Jan. 2022.

[22] G. Fierro and D. E. Culler, "Xbos: An extensible building operating system," in *Proc. 2nd ACM Int. Conf. Embedded Systems for Energy-Efficient Built Environments (BuildSys)*, Seoul, Republic of Korea, Nov. 2015, pp. 119–120.

[23] J. B. D. Joshi, E. Bertino, U. Latif, and A. Ghafoor, "A generalized temporal role-based access control model," *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 1, pp. 4–23, Jan. 2005.

[24] E. Uzun, V. Atluri, S. Sural, J. Vaidya, G. Parlato, A. L. Ferrara, and M. Parthasarathy, "Analyzing temporal role based access control models," in *in Proceedings of the 17th ACM Symposium on Access Control Models and Technologies (SACMAT 2012)*, Jun. 2012, pp. 177–186.

[25] B. Gao, F. Liu, S. Du, and F. Meng, "An oauth2.0-based unified authentication system for secure services in the smart campus environment," in *in Proceedings of the 18th International Conference on Computational Science – ICCS 2018*, ser. Lecture Notes in Computer Science, Y. Shi, H. Fu, Y. Tian, V. V. Krzhizhanovskaya, M. H. Lees, and D. Sloot, Eds., Jun. 2018, vol. 10862, pp. 752–764.

[26] Y. Sun, J. Lei, S. Shin, and H. Lu, "Baoverlay: A block-accessible overlay file system for fast and efficient container storage," in *Proc. 11th ACM Symp. Cloud Comput. (SoCC)*, Virtual Event, Oct. 2020, pp. 90–104.