Future Generation Computer Systems 174 (2026) 108010

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

# A secure framework for containerized IoT applications in integrated edge–cloud computing environments

Qifan Deng [a],*, Mohammad Goudarzi [b], Arash Shaghaghi [c], Majid Sarvi [d], Rajkumar Buyya [a]

[a] The Quantum Cloud Computing and Distributed Systems (qCLOUDS) Lab, School of Computing and Information Systems, The University of Melbourne, Australia
[b] The Faculty of Information Technology, Monash University, Clayton, Australia
[c] School of Computer Science and Engineering, University of New South Wales, Sydney, Australia
[d] Department of Infrastructure Engineering, The University of Melbourne, Australia

ARTICLE INFO

*Keywords:*
Edge and cloud computing
Containerization
IoT security
Lightweight framework
Threat modeling
Secure by design
Economic risk assessment

ABSTRACT

The integration of edge and cloud computing combines low latency with high computational power, addressing the constraints of edge resources and high access latency inherent in cloud environments. This is essential for deploying Internet of Things (IoT) applications, which are mainly developed by Containers within these heterogeneous environments. However, the open, multi-user nature of edge computing, compounded by a lack of standardized practices, introduces substantial security challenges with severe economic implications. In response, we propose SecConEC, an economically driven framework designed to secure the deployment and execution of containerized IoT applications. We conducted systematic threat modeling using the STRIDE framework, explicitly incorporating quantitative economic risk assessment to identify and prioritize security threats based on their potential economic impacts. We particularly focus on tampering and resource hijacking threats. SecConEC implements robust yet lightweight mitigation and detection mechanisms informed by the MITRE ATT&CK framework through a Security Information and Event Management (SIEM) system. Also, SecConEC introduces a dynamic, security-aware scheduling mechanism that balances performance and security considerations, proactively mitigating economic risks associated with potential security threats. Extensive performance evaluation shows that SecConEC significantly mitigates prioritized threats, effectively securing IoT application deployment and execution in edge-cloud environments, while maintaining low service latency with a minimal performance overhead of 1.7%.

## 1. Introduction

Edge computing has emerged as a complementary paradigm to cloud computing, offering distinct advantages such as lower latency and higher bandwidth due to its proximity to end users. Compared to the vast computing power available in cloud computing, edge computing is usually characterized by limited resources [1,2]. The integration of edge and cloud computing aims to combine low latency with high computational power, addressing the limitations of edge's constrained resources and the cloud's high latency [2–4].

A significant challenge in these integrated environments is their heterogeneity, encompassing varying hardware architectures and operating systems. Container technology can address this issue by streamlining resource integration [5]. A container encapsulates an application and its dependencies into a single, lightweight, and portable unit that executes in an isolated environment [6]. This approach involves preparing a container image that includes the codebase and its environment, allowing seamless execution. By leveraging container technology, resource management frameworks [5,7–9] can efficiently schedule and deploy IoT applications in an edge–cloud environment. It manages resources across both layers to provide low-latency services to IoT end users. However, there is a notable gap in the literature [8,10,11] regarding lightweight and secure resource management solutions for such integrated environments using container technology.

Ensuring the secure deployment and execution of IoT application containers in the edge–cloud integrated environment is paramount. The open nature of IoT and edge networks, along with their multi-user

* Corresponding author.
*E-mail address:* qifan.deng@outlook.com (Q. Deng).
[1] **S**poofing, **T**ampering, **R**epudiation, **I**nformation Disclosure, **D**enial of Service, and **E**levation of Privilege.
[2] They are (noted with hot links): 1. IDDIL/ATC, 2. OWASP Docker Top 10, 3. NIST Special Publication - Application Container Security Guide, 4. STRIDE, 5. PASTA, 6. LINDDUN, 7. Attack Trees, 8. PnG, 9. Security Cards, 10. hTMM, 11. Quantitative Threat Modeling Method, 12. Trike, 13. VAST, 14. OCTAVE, 15. DREAD, 16. TARA (MITRE), 17. TARA (Intel), 18. CAPEC.

https://doi.org/10.1016/j.future.2025.108010
Received 21 November 2024; Received in revised form 14 June 2025; Accepted 30 June 2025
Available online 13 July 2025

access, makes them susceptible to attacks [12–15]. A systematic threat analysis is required to identify, rank, and prioritize potential attacks on the system in such an environment [16–19]. Accordingly, we conduct this systematic analysis using threat modeling [20]. We select STRIDE[1] framework [21] after reviewing 18 threat modeling methods and their associated documents.[2] It is selected for its broad acceptance, open-source nature, and high reputation in both academia and industry. This analysis prioritizes tampering and resource hijacking as critical attacks to defend against in the execution of IoT applications in the integrated edge–cloud environment. Tampering leads to costly data remediation, while resource hijacking drains critical resources, thereby diminishing Quality of Service (QoS) and violating service-level agreements (SLAs). These attacks degrade system performance and incur severe economic losses [22]. Thus, addressing those threats is imperative to reduce operational expenditures and potential economic loss. To mitigate these threats, the integration of security mechanisms and policies is essential in the early design phase [23].

To mitigate these security challenges and their economic consequences, we propose SecConEC, a framework that secures the deployment and execution of IoT applications in integrated edge–cloud environments. The quantitative economic risk assessment of SecConEC prioritizes threats related to tampering and resource hijacking. SecConEC integrates security-aware mechanisms directly into the container management and scheduling process. We implement our contributions in the FogBus2 [5] framework to demonstrate their effectiveness, leveraging its modular design and containerization support to focus on enhancing security. The contributions of this work are:

- **Systematic Threat Analysis:** A systematic threat analysis that combines the STRIDE framework and quantitative economic risk assessment to identify and prioritize threats in the open and multi-user edge–cloud environments. The analysis prioritizes tampering and resource hijacking as critical threats.
- **Security-Enhanced Container Management:** An approach that ensures container image and runtime integrity by integrating a Security Information and Event Management (SIEM) system, which leverages the MITRE ATT&CK framework for real-time threat mitigation.
- **Security-Aware Resource Allocation:** A security-aware scheduling algorithm that allocates resources by dynamically balancing performance and security. It incorporates both latency requirements and the potential economic impact of threats.
- **Comprehensive Evaluation:** A comprehensive evaluation demonstrating that SecConEC prevents high-risk attacks while maintaining low service latency with a minimal performance overhead of 1.7%.

The rest of the paper is organized as follows: Section 2 presents the related works. Section 3 discusses the threat analysis. Section 4 describes the architecture design. Section 5 covers the experiments and analysis. Finally, Section 6 summarizes the work and provides future directions.

## 2. Related work

The integration of edge and cloud computing using container technology has garnered significant attention, leading to numerous frameworks and models aimed at enhancing resource management and computational efficiency. [24] proposed Aura, which focuses on scalable localized computation using container technology, albeit without addressing container security issues, providing incentive-driven, ad-hoc computation offloading for mobile IoT devices [24]. [25] developed an actor-based programming model for dynamic deployment using containers, but their approach lacks security measures, facilitating fault-tolerant and parallel task execution through the Akka toolkit for Fog environments [25]. [10] presented FOGPLAN, which minimizes service latency through adaptive container allocation, though it does not include security features, emphasizing a QoS-aware dynamic service provisioning to handle latency-sensitive IoT applications [10]. [11] created a framework for managing edge computing infrastructure using containers, yet it lacks mechanisms for responding to security incidents, dynamically orchestrating distributed edge resources into ad-hoc computing infrastructures [11]. [7] leveraged containers for creating a cost-effective cloud solution but overlooked inherent security concerns, aiming to utilize IoT-based containers to form a distributed, affordable cloud infrastructure [7]. [8] proposed TinyEdge, which focuses on extensibility but lacks a concrete security implementation, introducing a low-code, module-based approach for rapidly customizing and deploying edge systems [8]. On the other hand, [5] introduced FogBus2, which integrates IoT systems with edge, fog, and cloud servers and offers scheduling and scalability but falls short on privacy and robust security measures, focusing specifically on optimized scheduling, resource discovery, and scalability mechanisms for heterogeneous IoT workloads [5]. [26] managed resources across edge layers using virtual private networks for communication security but lacked execution isolation, providing OpenStack-based middleware for elastic and transparent resource orchestration across edge, fog, and cloud [26], while [9] allocated distributed edge resources without considering security, proposing a Function-as-a-Service (FaaS) extension to OpenStack for adaptive IoT deployments [9].

The importance of security in integrated edge–cloud environments cannot be overstated, particularly given the vulnerabilities associated with multi-user access and open IoT networks. Most of the referenced works either lack security considerations or only involve simple communication security, leaving critical gaps in container security and incident response. Given the susceptibility of these environments to tampering and resource hijacking, robust security measures are essential. Existing frameworks such as [5,10] could significantly benefit from enhanced data protection and privacy preservation mechanisms to ensure secure deployment and execution of IoT applications.

Moreover, integrating security early in the software development lifecycle reduces remediation costs by addressing vulnerabilities before deployment [23]. However, current frameworks are inadequate for this “shift-left” approach. Existing quantitative risk assessment methods, such as the bow-tie model [27], evaluate threat probability and impact but fail to guide secure system design. This gap is particularly acute for Internet of Things (IoT) systems, which lack systematic methodologies for proactively engineering defenses against emerging cyber threats [19]. While threat modeling frameworks like MITRE ATT&CK [28] can partially address this, edge–cloud IoT systems require a more comprehensive solution. Therefore, a systematic methodology is needed to unite quantitative risk assessment with proactive, security-by-design principles from the initial design phase.

To address these gaps, we present SecConEC, a framework developed through a threat-driven, security-by-design methodology. We began by applying threat modeling to a representative edge–cloud architecture to identify and prioritize critical risks, specifically data tampering and resource hijacking. Based on this analysis, we design and implement SecConEC to integrate security throughout the system lifecycle. As summarized in Table 1, our framework introduces three key security enhancements absent in prior work: it enforces secure, authenticated communication between all distributed components; it incorporates security-aware scheduling policies for resource allocation on both the edge and cloud; and it hardens the deployment and execution of containerized IoT applications.

## 3. Threat analysis

In this section, we conduct a systematic threat analysis to identify, evaluate, and prioritize potential security threats within an integrated edge–cloud environment for IoT applications. Section 3.1 describes the foundation framework adopted for the analysis, detailing its primary software components and their interactions. Section 3.2.1 introduces

**Table 1**
A comparison with related works.

| Work | E | C | MP | HA | Scl. | Scd. | RD | CLM | SMP | DIM | SS | MLSC | AEC | SDTM |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Zhang et al. [8] | ✓ | ✓ | | | | | | | | | | | | |
| Ferrer et al. [11] | ✓ | | | | ✓ | | | | | | | | | |
| Noor et al. [7] | | ✓ | | | ✓ | | | | | | | | | |
| Hasan et al. [24] | | ✓ | | | | | ✓ | | | | ✓ | | | |
| Yousefpour et al. [10] | ✓ | ✓ | | | ✓ | | ✓ | | | | | | | |
| Merlino et al. [26] | ✓ | ✓ | ✓ | ✓ | | | ✓ | | | | | | | |
| Srirama et al. [25] | ✓ | ✓ | ✓ | ✓ | ✓ | | | | | | | | | |
| Merlino et al. [9] | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | | | | | | | |
| Deng et al. [5] | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | | | |
| SecConEC (This work) | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |

E: Integration of Edge Computing; C: Integration of Cloud Computing; MP: Multi-Platform Support (e.g., x86_64, ARM); HA: Heterogeneous Application Support; Scl.: Scaling Mechanism and Policy; Scd.: Scheduling Mechanism and Policy; RD: Dynamic Resource Discovery; CLM: Centralized Log Management; SMP: System Monitoring with Persistence; DIM: Diverse Interaction Models; SS: Security-Aware Scheduling; MLSC: Multi-Layer Secure Communication (VPN, Isolated Networks, and TLS); AEC: Attested Execution of Container via signed metadata; SDTM: Integration of security mechanisms and policies in the design phase using Threat Modeling.

the threat modeling methodology, employing the STRIDE framework to categorize threats systematically. This subsection also elaborates on how Data Flow Diagrams (DFDs) are constructed and analyzed at multiple abstraction levels, providing a comprehensive mapping and quantification of identified risks. Section 3.2.2 discusses the threat ranking procedure, highlighting the prioritization strategy used to allocate resources efficiently for threat mitigation. Finally, Section 3.2.3 outlines the mitigation and detection strategies implemented, leveraging the MITRE ATT&CK framework to provide practical and effective defense mechanisms against the prioritized threats.

The adoption of economic risk assessment as the primary driver for threat modeling addresses the economic dimensions inherent in securing containerized IoT applications in edge–cloud environments. While the economic values attributed to likelihood and impact are inherently subjective, their explicit inclusion ensures alignment with organizational cost objectives. It enables stakeholders to justify security investments by balancing potential economic losses against the cost of security implementations. Consequently, this approach enhances both the effectiveness and feasibility of the threat analysis, particularly relevant in environments where resources are limited and cost considerations are paramount.

### 3.1. Foundation framework

Threat analysis is applied to a foundational framework to eliminate redundant implementations of baseline resource management features. It enables a focused approach to securing IoT applications. As shown in Table 1, FogBus2 [5] includes centralized log management, persistent system monitoring, and support for diverse interaction models, which are not offered by other frameworks. These features, along with its open-source nature, containerization capabilities, modular design, and readily available container images, make FogBus2 the preferred foundational system (see Fig. 1).

1. **User**: This component runs on IoT devices, managing the interactions between physical sensors and actuators. The *Sensor* sub-component periodically captures and serializes raw data, while the *Actuator* sub-component collects processed data from the Master to trigger corresponding actions, either in real-time or periodically, based on application scenarios.
2. **Master**: The central decision-making entity that can run on hosts at edge/fog or cloud layers. It handles resource allocation and application management through its sub-components:
    - *Registry*: Manages the registration and authentication of Actors, Task Executors, and IoT devices, assigning unique identifiers for tracking and communication.
    - *Profiler*: Dynamically monitors available system resources (CPU, RAM) and network characteristics (bandwidth, latency). It updates resource profiles either periodically from the Remote Logger or directly from Actors and Task Executors when necessary.
    - *Scheduler & Scaler*: Employs dynamic scheduling using the Optimized History-Based Non-dominated Sorting Genetic Algorithm (OHNSGA) to allocate tasks efficiently, considering historical data and system states. Additionally, it initiates scalability actions by starting new Master components when current resources are insufficient or overloaded, thus maintaining responsiveness.
    - *Resource Discovery*: Automatically discovers new Master and Actor components within the network to maintain an updated list of available resources, promoting dynamic adaptability.
3. **Actor**: Runs on hosts in edge/fog or cloud layers, responsible for profiling local resources and managing Task Executors. It contains the following sub-components:
    - *Profiler*: Similar to the Master's Profiler, it maintains resource status (CPU, RAM, network parameters) and regularly reports to the Remote Logger.
    - *Task Executor Initiator*: Starts Task Executor containers upon receiving task assignments from the Master.
    - *Master Initiator*: Activates new Master containers on demand, facilitating horizontal scalability and load distribution.
4. **Task Executor**: Performs the actual execution of IoT application tasks within containers, ensuring rapid deployment and efficient resource utilization. Each Task Executor component comprises:
    - *Executor*: Runs assigned tasks and manages inter-task data flow, especially in applications with dependent tasks. Completed tasks enter a reuse state to expedite subsequent requests, significantly reducing deployment overhead.
5. **Remote Logger (RL)**: Centralized logging component, which persistently collects and stores logs for performance analysis, resource monitoring, and auditing. It comprises:
    - *Logger Manager*: Receives and maintains logs from all components in persistent storage, utilizing multiple databases for system performance metrics, hardware resources information, and container image availability. Logs can be stored locally or distributed, depending on the scenario requirements.

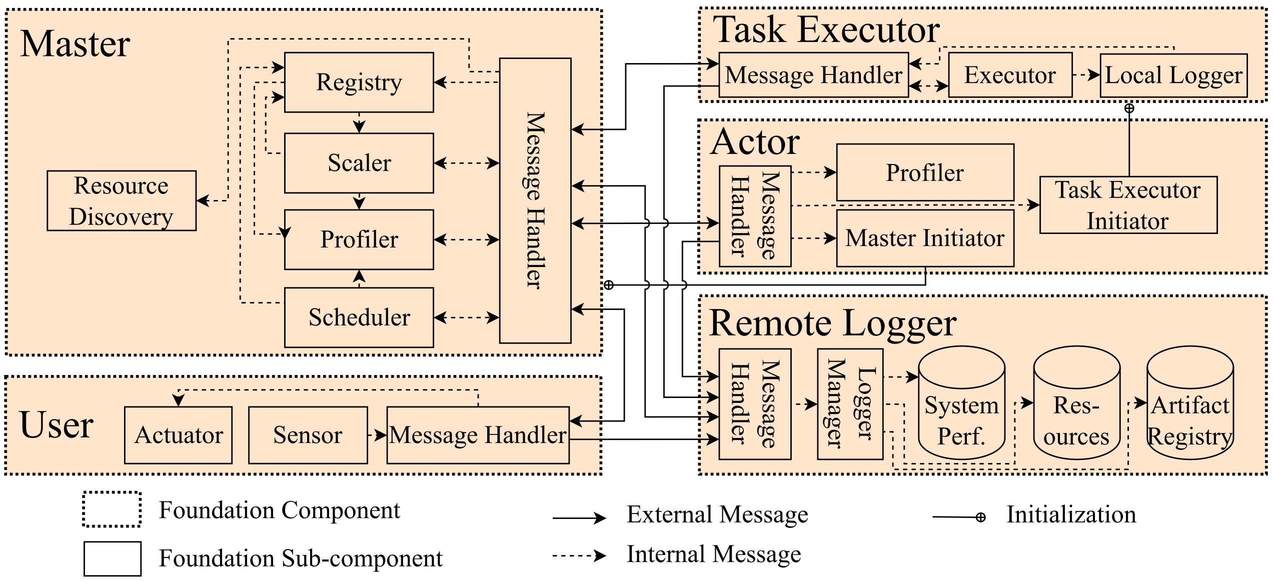

**Fig. 1.** Detailed foundation framework components and interactions.

Interactions among these components involve extensive message exchanges managed by dedicated *Message Handler* sub-components embedded in each primary component. Users initiate requests for application placement, which are received and processed by the *Master*. The *Master* dynamically schedules these requests through OHNSGA, then instructs *Actors* to instantiate *Task Executors*. *Task Executors* perform computations and return results to the *Master*, which forwards processed information back to *User* components for actuation. *Actors* regularly communicate system status updates to the *Remote Logger*, and the *Master* accesses logged data for optimized scheduling and scalability decisions.

These clearly defined components and robust interactions make FogBus2 capable of managing dynamic IoT environments effectively, maintaining both scalability and performance [5].

In summary, the process of managing requests and allocating resources begins with the *User* sending requests to the *Master* for running IoT applications that process sensor data. The *Master* then allocates the necessary resources, manages component registration, and continuously profiles the system resources to enable efficient allocation decisions. Next, the *Actor* component profiles local host resources, reports this information to the *Master*, initiates the appropriate *Task Executors* for executing specific IoT tasks, and can dynamically assume the role of a *Master* if scaling is required. The *Task Executor* performs the assigned IoT application tasks, after which the processed results are returned to the user. Finally, the *Remote Logger* centrally logs all system configurations, operational data, and execution details for performance monitoring, administrative review, and auditing purposes.

From a hardware perspective, as shown in Fig. 2, devices in the IoT layer generate data that are sent to the proximate edge layer or the remote cloud layer for processing and are scheduled by the *Master* based on resource requirements.

### *3.2. Threat modeling*

Threat modeling is a proactive process to identify, evaluate, and mitigate potential security threats during system design, enhancing security and reducing risks [20]. It systematically analyzes security threats. We select STRIDE [21], developed by Microsoft, from 18 reviewed frameworks due to its broad acceptance, open-source nature, and high reputation in both academia and industry. STRIDE categorizes potential security threats into six types: (1) **S**poofing — An adversary impersonates another entity; (2) **T**ampering — Malicious or altered data/code attacks; (3) **R**epudiation — Exploiting flaws to prevent system refusal of invalid requests; (4) **I**nformation Disclosure — Unauthorized exposure of information; (5) **D**enial of Service (DoS) — Attacks degrading or halting service; (6) **E**levation of Privilege (EoP): Users gain unauthorized privileges. Each initial of the six categories forms the acronym STRIDE.

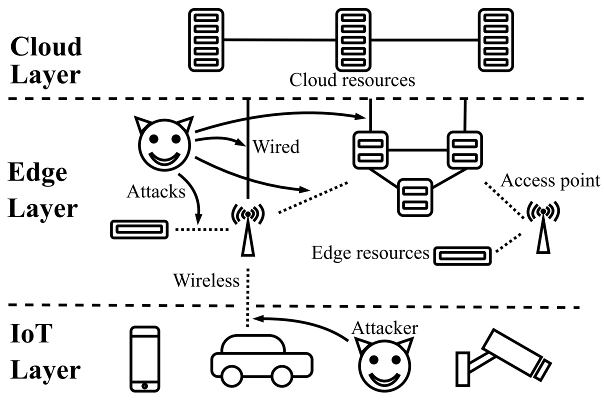

**Fig. 2.** Hardware layers and potential attacks.

These threats target the integrated edge–cloud environment, as depicted in Fig. 2. As the figure depicted, the wired and wireless connections in IoT and Edge layers are exposed to the public. As a result, attackers can more easily invade through the wireless and/or wired connections in the two layers compared with the Cloud layer, which usually has more limits with physical and digital access to resources [29].

To clearly describe the process steps aligned with secure software development practices, we follow established guidelines detailed in Microsoft's Security Development Lifecycle (SDL) [21]. Specifically, our threat modeling consists of six sequential steps:

1. Diagramming the system by constructing detailed Data Flow Diagrams (DFDs) at multiple abstraction levels;
2. Identifying potential threats systematically using the STRIDE threat categorization model;
3. Assessing and quantifying identified threats through a structured economic risk assessment, assigning likelihood and economic impact scores to each threat;

4. Prioritizing threats based on their economic risk scores to efficiently allocate resources for implementing security measures;
5. Integrating appropriate mitigation and detection techniques based on the identified threats, explicitly guided by the MITRE ATT&CK framework.
6. Designing the system according to the prioritized results obtained from the threat analysis to ensure robust defense against identified and prioritized threats;

This structured approach ensures comprehensive threat identification and economically driven prioritization of mitigation strategies.

### *3.2.1. Analysis of data flow diagrams*

The STRIDE-based threat modeling process begins with an in-depth analysis of Data Flow Diagrams (DFDs) that represent the existing system architecture. Typically, DFDs are constructed at multiple levels, each depicting the system at different granularities of abstraction. Such multi-level diagrams are essential because they facilitate both broad and detailed threat identification, ensuring comprehensive coverage of security considerations from conceptual to operational perspectives. High-level diagrams facilitate the identification of threats related to external interactions and overall system boundaries, while detailed, low-level diagrams reveal threats associated with specific internal data operations.

The structured approach of this study strictly aligns with established methodologies outlined in *The Security Development Lifecycle* [21]. It consists of modeling data flows, systematically identifying potential threats, calculating risk scores, and ranking threats (Tables 2, 3, and Eqs. (1)–(3))— Specifically, our method involves distinctly numbered DFD elements, a precise mapping to STRIDE categories, explicit calculation of risk scores based on threat likelihood and impact, and quantifiable prioritization of threats.

**High-level DFD.** According to best practices described in [21], high-level DFDs, also known as context diagrams, clearly depict the system in question as a central entity interacting with external entities. Such diagrams precisely establish system boundaries and illustrate abstract interactions between the system and external entities. Data flows at this abstraction level represent generalized operations, highlighting conceptual relationships and high-level exchanges pertinent to primary usage scenarios.

Following these guidelines, we develop a high-level (context) diagram for FogBus2, illustrated in Fig. 3. This diagram effectively identifies external entities interacting with FogBus2, abstracting away internal complexities to emphasize the overall data exchange and system boundaries.

**Low-level DFD.** Further refinement of the high-level DFD yields Level-0 diagrams, which elaborate on the main internal processes and their interactions within the system. Each significant process represented in the context diagram is decomposed into subprocesses and has explicit data flows defined. Subsequently, Level-1 diagrams provide even greater detail by further decomposing Level-0 subprocesses, delineating intricate interactions among sub-components, data stores, and external entities. Data flows in these detailed diagrams explicitly indicate CRUD (Create, Read, Update, Delete) operations, clearly emphasizing precise data-handling activities.

In this study, Level-0 and Level-1 DFDs are integrated into a single comprehensive low-level diagram (Fig. 4). The rationale for this integration arises from the inherently tight coupling and interdependent interactions among FogBus2 components at multiple abstraction layers. Combining these diagrams provides greater clarity and coherence, facilitating a more practical and holistic threat analysis. Such integration is consistent with iterative threat modeling approaches recommended in established security practices [21].

**Derivation and Consolidation of DFD Elements.** A structured consolidation of DFD elements into tabular form is essential for systematically analyzing threats associated with resource scheduling in IoT application execution. Consolidating DFD elements into Tables 2 and 8 streamlines the analysis process by grouping related items that share common technological or functional characteristics. This consolidated list enhances clarity and efficiency in subsequent threat identification, mapping, and risk scoring processes.

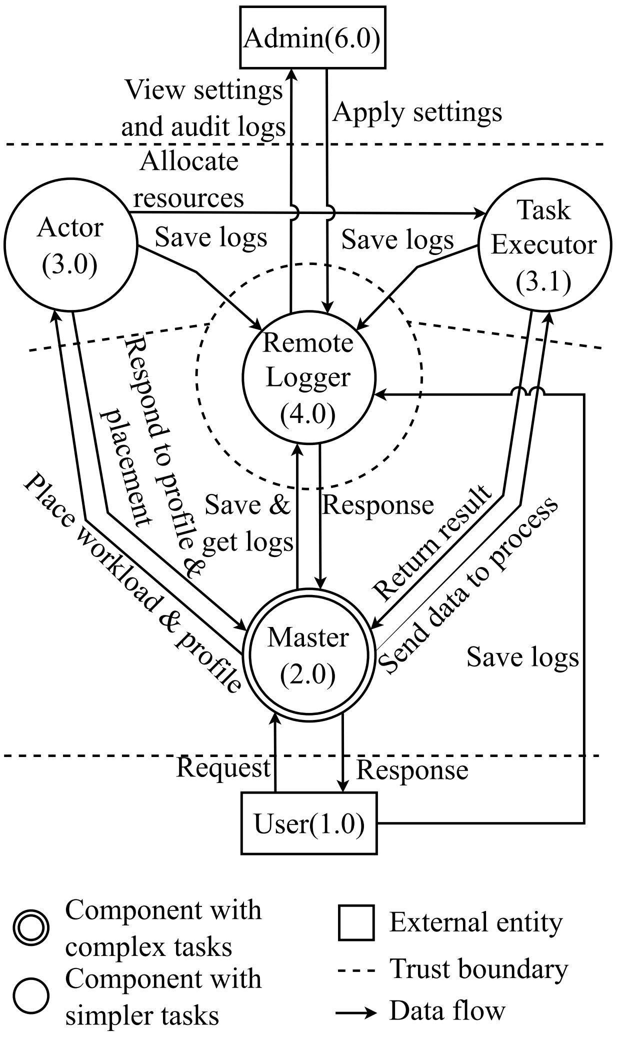

**Fig. 3.** High-level data flow diagram.

Table 3 presents a comprehensive mapping between identified threats and STRIDE categories with corresponding DFD elements. Threats effectively mitigated by current system designs are intentionally omitted to maintain focus on relevant vulnerabilities. For instance, threats related to information disclosure from data stores are excluded due to existing authentication mechanisms within FogBus2, provided that underlying host environments remain uncompromised.

**Quantification of Economic Risk Scores.** Having systematically analyzed the Data Flow Diagrams (DFDs) and consolidated all relevant threat mappings using STRIDE, we next quantify the economic risk scores for each DFD item. This step enables a more holistic view of overall economic risk, encompassing threat likelihood, potential losses (e.g., service downtime or data fines), and broader economic impact. Following the approach outlined in [22], we emphasize the importance of gathering assessments from a sufficiently large group of IT security experts with extensive domain experience. The median of their assessments may then be used to generate a representative likelihood and impact level for each identified threat. In this work, we serve as our own subject-matter experts for the foundation system under assessment, given our role as its principal designers.

**Table 2**
Consolidated list of data flow diagram items.

| DFD element category | DFD element name | DFD element ID |
|---|---|---|
| External entities | User<br>Admin | 1.0<br>6.0 |
| Components | Registry, Scheduler, Master Profiler<br>Task Executor Initiator and Actor Profiler<br>Executor<br>Logger Manager | 2.0.1–2.0.3<br>3.0.1, 3.0.2<br>3.1.1<br>4.0.1 |
| Data stores | System settings, performance, resources, and audit logs | 5.0.1–5.0.4 |
| Data flows | Detailed in Table 8 in Appendix A | |

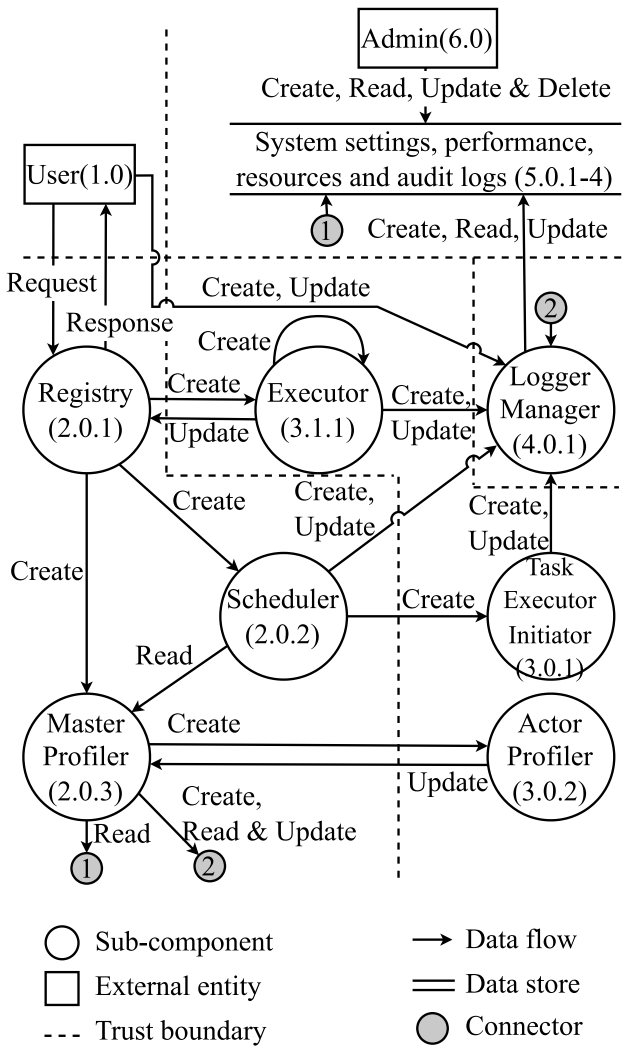

**Fig. 4.** Low-level data flow diagram.

Prior studies [22] propose categorizing threat impacts as Business Interruption, Financial Loss (including data reinstatement costs), Reputational Damage (including client attrition and sensitive data exposure), and Third-Party Claims and Regulatory Fines. We abstract all these forms of impact under a unifying notion of *economic impact*, thereby concentrating on assessing overall economic losses. This single, comprehensive metric translates threat likelihoods and impacts into numeric values, ultimately facilitating more straightforward but effective risk comparisons.

A summary of the relationship between threat types (such as spoofing and tampering) and components of the DFD (External Entities, Components, Data Stores, and Data Flows), along with corresponding likelihood and impact values, is noted in Table 3. In this table, $\mathbb{L}$ represents the overall likelihood of each threat category, $\mathbf{L}$ captures the item-specific likelihood, and $\mathbf{I}$ denotes the economic impact associated with that threat. The numeric ratings for likelihood and economic impact range from 1 to 5, where 5 indicates the highest likelihood or greatest impact. By emphasizing economic impact, our approach extends traditional risk assessments [21,30] to incorporate system-specific economic dimensions, building on the work of [22]. The precise assignment of $\mathbb{L}$, $\mathbf{L}$, and $\mathbf{I}$ necessitates deep domain knowledge, consideration of unique operational costs, and stakeholder-defined priorities; consequently, these values may differ substantially across organizations [21,30]. Detailed guidelines and quantitative frameworks for performing such valuations are available in the broader literature [22, 31,32]. Although somewhat subjective, an economic-based approach to risk scoring effectively foregrounds the economic implications of security threats, enabling system owners, security teams, and procurement decision-makers to fine-tune these values according to their particular organizational objectives and resource constraints.

Using Table 3, we calculate the economic risk score ($R$) for each identified threat, component, and data flow item. A higher economic risk score indicates greater potential economic impact. The economic risk score is computed as the product of the likelihood $L$ of the threat occurring and its economic impact $I$, as expressed in Eq. (1).

$$\text{Economic Risk Score}(R) = L \times \text{Economic Impact}(I) \tag{1}$$

Let $T = \{t_1, t_2, \ldots, t_n\}$ be the set of identified threats, each with likelihood $L_j$ and economic impact $I_j$. The individual economic risk score $R_j$ for threat $t_j$ is calculated using Eq. (1). The total economic risk score $\tilde{R}$ for all threats of a given type is the sum of individual economic risk scores, as shown in Eq. (2).

$$\tilde{R} = \sum_{j=1}^{n} L_j \times I_j \tag{2}$$

Let $\mathbb{T} = \{T_S, T_T, T_R, T_I, T_D, T_E\}$ be the STRIDE threat categories: Spoofing ($T_S$), Tampering ($T_T$), Repudiation ($T_R$), Information Disclosure ($T_I$), Denial of Service ($T_D$), and Elevation of Privilege ($T_E$). For each category $T_i \in \mathbb{T}$, let $\mathbb{L}_i$ represent its overall likelihood, and let $R_{ij}$ be the economic risk score of the $j$th threat within $T_i$, calculated as $R_{ij} = L_{ij} \times I_{ij}$, where $L_{ij}$ and $I_{ij}$ denote the likelihood and economic impact of that specific threat.

The total economic risk score ($\mathbb{R}$) for the entire system is computed by summing the weighted economic risk scores of each threat category, as shown in Eq. (3).

$$\mathbb{R} = \sum_{i=1}^{m} \mathbb{L}_i \times \tilde{R}_i = \sum_{i=1}^{m} \mathbb{L}_i \times \sum_{j=1}^{n} L_{ij} \times I_{ij} \tag{3}$$

### 3.2.2. Threat ranking

Threat ranking is essential because administrators often face budgetary constraints limiting their ability to mitigate all identified threats comprehensively, even after systematic risk scoring [33]. Prioritizing threat categories based on their economic implications helps organizations efficiently allocate limited resources. Each organization

**Table 3**
Mapping and combination of threat types with data flow diagram items.

| Threat type | Affected DFD element categories (detailed within Table 9 in Appendix B) | Overall likelihood ($\mathbb{L}$) |
|---|---|---|
| Spoofing | External Entities | 2 |
| Tampering | Components, Data Flows | 3 |
| Repudiation | External Entities, Components | 1 |
| Information Disclosure | Data Flows | 4 |
| Denial of Service | Components, Data Stores, Data Flows | 5 |
| Elevation of Privilege | Components | 2 |

*This table provides a high-level overview. $\mathbb{L}$ represents the overall likelihood score of each threat type. For a detailed breakdown of the specific likelihood ($\mathbf{L}$) and potential economic impact ($\mathbf{I}$) on each DFD item, see Table 9 for details. These likelihood and impact values are subjective and contextual but serve to illustrate the economics-driven assessment approach.*

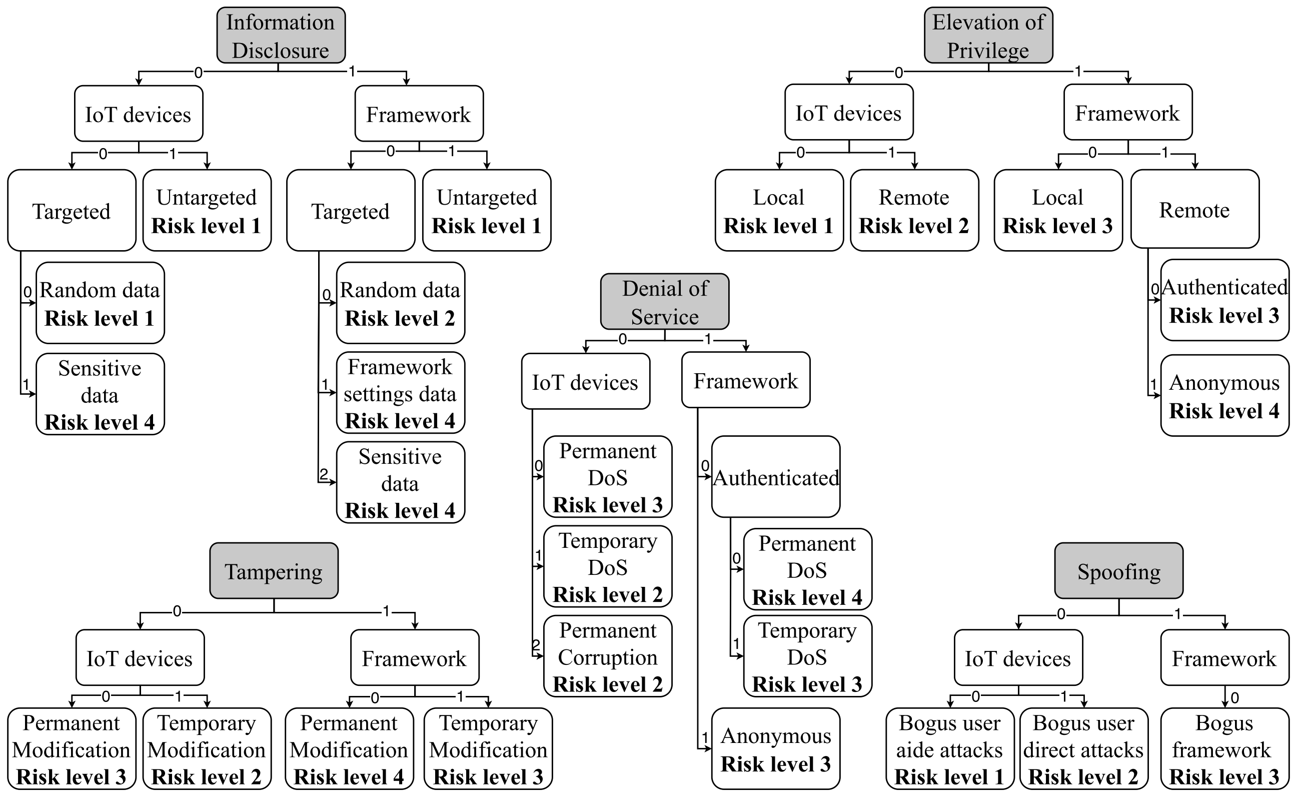
Information Disclosure
IoT devices
Framework
Targeted
Untargeted Risk level 1
Random data Risk level 1
Sensitive data Risk level 4
Targeted
Untargeted Risk level 1
Random data Risk level 2
Framework settings data Risk level 4
Sensitive data Risk level 4
Elevation of Privilege
IoT devices
Framework
Local Risk level 1
Remote Risk level 2
Local Risk level 3
Remote
Authenticated Risk level 3
Anonymous Risk level 4
Denial of Service
IoT devices
Framework
Permanent DoS Risk level 3
Temporary DoS Risk level 2
Permanent Corruption Risk level 2
Authenticated
Permanent DoS Risk level 4
Temporary DoS Risk level 3
Anonymous Risk level 3
Tampering
IoT devices
Framework
Permanent Modification Risk level 3
Temporary Modification Risk level 2
Permanent Modification Risk level 4
Temporary Modification Risk level 3
Spoofing
IoT devices
Framework
Bogus user aide attacks Risk level 1
Bogus user direct attacks Risk level 2
Bogus framework Risk level 3
0
1
2

**Fig. 5.** Determination of threat risk levels.

should evaluate and assign economic threat levels according to its economic constraints, economic objectives, system characteristics, physical installations, and staffing.

Fig. 5 illustrates our risk-level rankings derived from Microsoft's knowledge base [21], specifically tailored for integrated edge–cloud environments. In such contexts, minimizing economic losses associated with degraded service quality or compromised security is critical, especially given constrained edge resources. Risk levels are ranked from 1 (least severe economic impact) to 4 (most severe economic impact), prioritizing threats with substantial economic consequences. Denial of service (DoS) and resource hijacking receive the highest economic priority (level 4), as they can exhaust edge resources, significantly degrading service quality and increasing economic costs related to downtime and service recovery. Tampering also receives level-4 economic priority due to its ease of execution in open, multi-user edge environments, which can potentially cause substantial economic losses through data integrity breaches, regulatory fines, and remediation expenses. Repudiation is streamlined because it often results from tampering; thus, mitigating tampering threats inherently reduces economic risks from repudiation [21].

The weight ($\omega$) of each threat category is derived from the weights of its subnets ($\tau$), which are directly determined by their numerical threat levels. We calculate the economic weights for each threat category by averaging the economic weights for IoT devices and the framework, summing contributions from corresponding child branches or subnets. The calculated economic weights are: $\omega_S = 2.25$ for Spoofing, $\omega_T = \omega_R = 3.0$ for Repudiation and Tampering, $\omega_I = 1.958$ for Information Disclosure, $\omega_D = 2.792$ for Denial of Service, and $\omega_E = 2.125$ for Elevation of Privilege, with $\Omega = \{\omega_S, \omega_T, \omega_R, \omega_I, \omega_D, \omega_E\}$.

The total weighted economic risk score ($\hat{\mathbb{R}}$) for the entire system is computed by summing the weighted economic scores of each threat category:

$$\hat{\mathbb{R}} = \sum_{i\in\{S,T,R,I,D,E\}} \omega_i \times \tilde{R}_i \tag{4}$$

Using Eqs. (3) and (4), along with economic likelihood and impact values from Table 3, we calculate the overall economic risk score for the foundation system as $\hat{\mathbb{R}}_{Fdn.} = 1935.57 = 2.25{\times}10+3.0{\times}237+3.0{\times}10+1.958{\times}40+2.792{\times}375+2.125{\times}22$. Fig. 6 visualizes this calculation, clearly demonstrating that Tampering and Denial of Service threats demand prioritized mitigation due to their significantly higher economic risk scores compared to other categories.

This study employs static weights in the risk assessment during system design for clarity, an approach justified by recent research [27,34]. This simplification allows the analysis to focus on demonstrating the system's core design and evaluation.

**Table 4**
Categorized mitigation and detection for prioritized threats.

| Top Cat. | Resource Cat. | # | STRIDE | MITRE ATT&CK Tech. | Mitigation | Detection |
|---|---|---|---|---|---|---|
| Static assets | Code | 1 | T10 | Malicious Image T1204.003 | Code Signing M1045 | Container DS0032 |
| | | 2 | T11 | Implant Internal Image T1525 | Code Signing M1045 | Container DS0032 |
| | | 3 4 | I101 I102 | Credentials In Files T1552.001 | Audit M1047 | File DS0022 |
| Dynamic assets | Computation | 5 6 7 | D11 D100 D101 | Endpoint Denial of Service T1499 | Filter Network Traffic M1037 | Network Traffic DS0029 |
| | | 8 9 | E110 E111 | Escape to Host T1611 | Disable/Remove Feat/Prgm M1042 | Container DS0032 |
| | | 10 | D101 | Resource Hijacking T1496 | Our Procedure | Network Traffic DS0029 |
| | Storage | 11 | D100 | Resource Hijacking T1496 | Our Procedure | File DS0022 |
| | Network | 12 13 14 | D11 D100 D101 | Network Denial of Service T1498 | Filter Network Traffic M1037 | Network Traffic DS0029 |

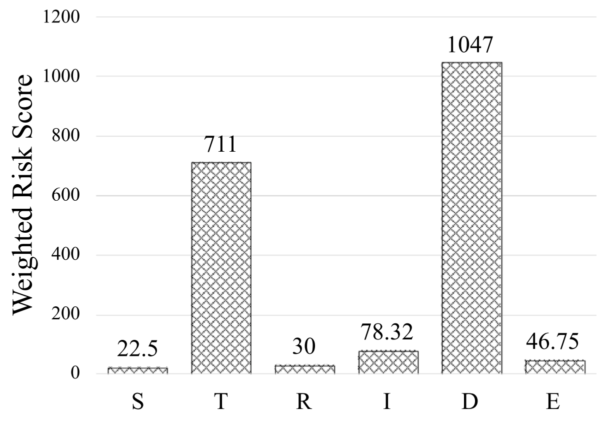

**Fig. 6.** Decoupled weighted economic risk scores for the foundation system.

### 3.2.3. *Mitigation and detection*

To clearly quantify the economic effectiveness of mitigation strategies, this section categorizes identified threats using MITRE ATT&CK techniques (Table 4). For each prioritized threat, specific economic mitigation and detection measures from the MITRE ATT&CK framework are outlined, and their economic effectiveness is implicitly validated through corresponding reductions in likelihood within our economic risk scoring model. Although economic impact values remain relatively stable due to inherent system characteristics and organizational economic context, applying mitigation measures effectively reduces threat likelihood, significantly decreasing the overall economic risk score as demonstrated in Section 4. This explicit economic linkage clarifies the connection between theoretical threat modeling and the practical evaluation of cost-effective mitigation mechanisms, directly addressing reviewers' concerns regarding clarity in economic assessment methods.

To enhance the system's economic resilience, we aim to minimize the economic risk score defined in Eq. (2). The economic impact is usually stable, established through historical economic data and organizational expertise, whereas likelihood can be effectively decreased through proactive mitigation and detection techniques, thus significantly lowering the economic risk score.

Leveraging the MITRE ATT&CK framework [28], we devise mitigation and detection strategies targeting container-specific threats. MITRE ATT&CK provides a comprehensive, globally recognized reference for understanding, tracking, and economically managing cyber threats by detailing attacker methods and effective defense strategies, ultimately reducing potential economic losses.

Table 4[3] summarizes the mitigation and detection techniques associated with the prioritized threats analyzed via STRIDE and MITRE ATT&CK. From a resource procurement and management perspective, we classify threats targeting static assets (pre-runtime container images) and dynamic assets (runtime containers). This economic distinction is essential, as addressing static asset threats often incurs substantial manual effort, time, and higher costs, whereas dynamic asset threats can typically be addressed with automated, cost-effective detection and mitigation strategies.

This systematic analysis guides the design of our optimized system. We prioritize threats with significant economic impact, such as tampering and resource hijacking, and integrate their corresponding mitigation and detection techniques into the system architecture.

## 4. Architecture and design

This section presents the architecture and design of SecConEC, detailing how its optimized framework enhances security and operational efficiency. The discussion is structured into three subsections. Section 4.1 delineates the core components of the SecConEC Security Information and Event Management (SIEM) system. Section 4.2 elucidates the key optimizations to the foundation framework that enhance security and performance, focusing on secure scheduling and inter-component communication. Finally, Section 4.3 presents a security analysis of the optimized system, evaluating its effectiveness in mitigating identified threats.

### 4.1. *SecConEC components*

SecConEC integrates Security Information and Event Management (SIEM) [35]. This system collects and aggregates incidents and logs. It enables responsive actions to secure the system environment using embedded mitigation and detection techniques. SecConEC's SIEM consists of *SIEM — Static Assets (SSA)*, *SIEM — Dynamic Assets (SDA)*, and *SIEM Agent (SA)*. Fig. 8 illustrates these components and their interactions.

*SIEM Static Assets* and *SIEM Dynamic Assets* differ in their operational requirements, processing times, and granularity. *SIEM Static Assets* involves extensive manual operations and longer processing times, whereas *SIEM Dynamic Assets* handles more dynamic and variable data, operating more frequently. *SIEM Agents* interact with *SIEM Static Assets* and *SIEM Dynamic Assets* using a RESTful approach secured by HTTP Basic Authentication with transport layer security, ensuring lightweight and efficient operations. These components work together to monitor the environment, respond to incidents, and notify administrators based on predefined policies.

#### 4.1.1. *SIEM — Static assets*

*SIEM Static Assets* secures the static elements managed by SecConEC, utilizing four sub-components: (1) *Host Config Scanner (SSA1)* accesses relevant files with read permissions to scan host machine configurations; (2) *Network Config Scanner (SSA2)* detects suspicious host network configurations, requiring administrator training for mitigation; (3) *Storage Scanner (SSA3)* monitors for unauthorized storage device connections or disconnections; (4) *Code Scanner (SSA4)* ensures

[3] The STRIDE codes correspond to the risk levels depicted in Fig. 5 MITRE ATT&CK codes reference specific items in their database.

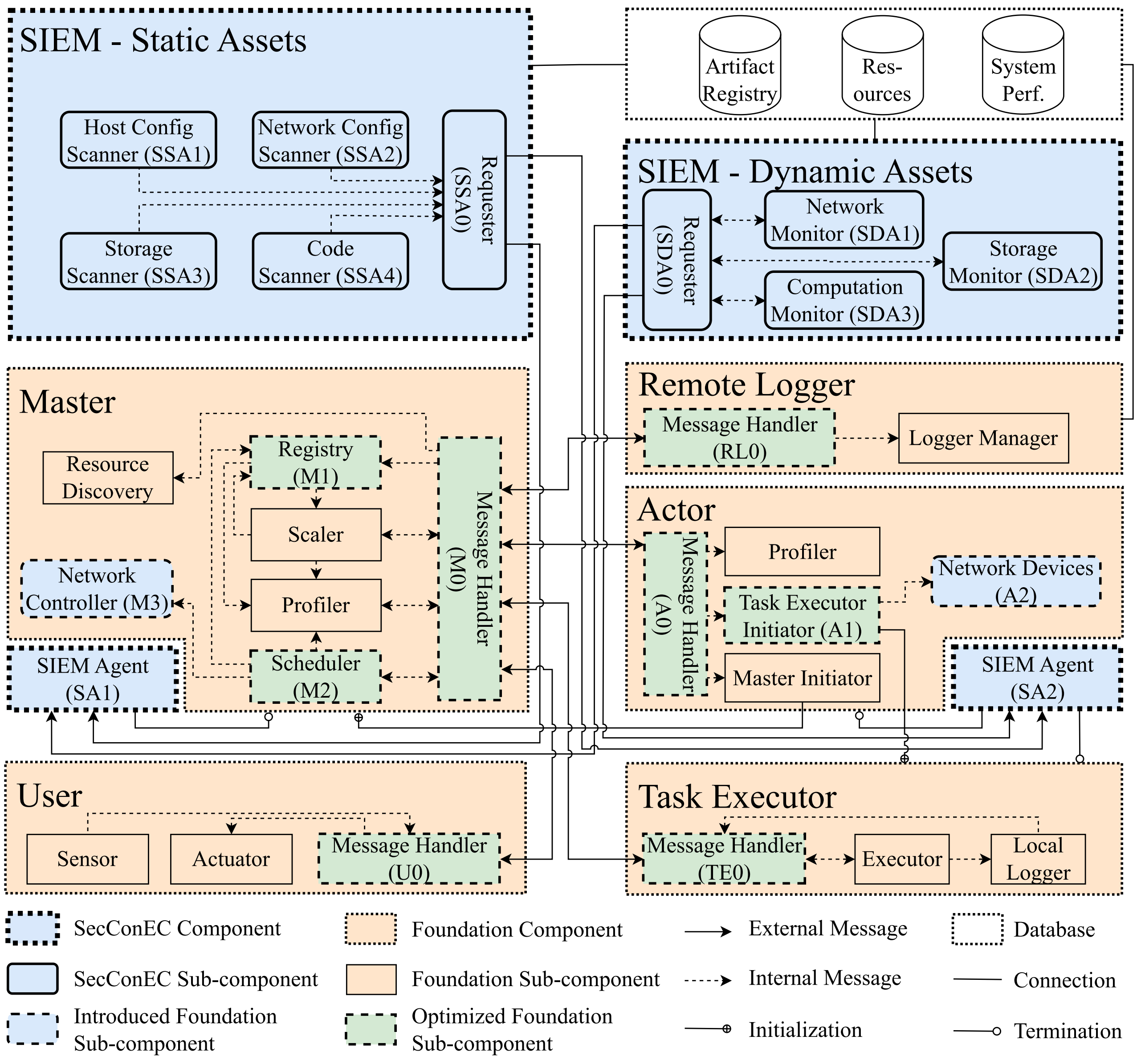

**Fig. 7.** SecConEC overview with software components and interactions.

container image integrity by verifying digital signatures before deployment. Security engineers manually sign a container image after resolving vulnerabilities and removing existing credentials. Before deployment, the digital signature is re-verified to ensure the integrity of the container image, thereby securing the data processed by the container (Algorithm 1 line 2, Table 4 #1–4).

The *SIEM Agents* are periodically queried by the *SIEM Static Assets* to retrieve relevant information and assess static resources (e.g., host configurations, host network configurations, host storage systems, and container images).

### *4.1.2. SIEM — Dynamic assets*

*SIEM Dynamic Assets* dynamically secures SecConEC's changing elements, detecting and mitigating threats in real-time, notifying administrators, and automatically taking action as per predefined policies. It includes three sub-components: (1) *Network Monitor (SDA1)* checks the network utilization of containers on *Master* and *Actor* host machines, ensuring no unrecognized container network overlay exists; (2) *Storage Monitor (SDA2)* aggregates volume information attached to containers and monitors their utilization; (3) *Computation Monitor (SDA3)* ensures CPU and memory utilization comply with defined policies, verifying the image digital signature of the running container as well as digitally signed scheduling arguments. The signature of the container arguments is stored within its metadata, maintaining lightweight and minimal complexity. The aforementioned steps are summarized in Algorithm 1, lines 21–23.

The *SIEM Agents* are periodically queried by the *SIEM Dynamic Assets* to monitor dynamic resources (e.g., container signatures, container networks, container volumes, and container resource usage). Policies are then applied to these results, triggering actions and notifications. For example, if a running container is not properly signed, it is removed and the administrator is notified.

### *4.1.3. SIEM agent*

*SIEM Agent* responds to *SIEM Static Assets* and *SIEM Dynamic Assets* invocations, operating on both *Master* and *Actor* hosts. It collects host information and performs commands from *SIEM Static Assets* or *SIEM Dynamic Assets*, managing configurations, storage, networks, container images, container networks, container volumes, and running containers (Algorithm 1 line 26).

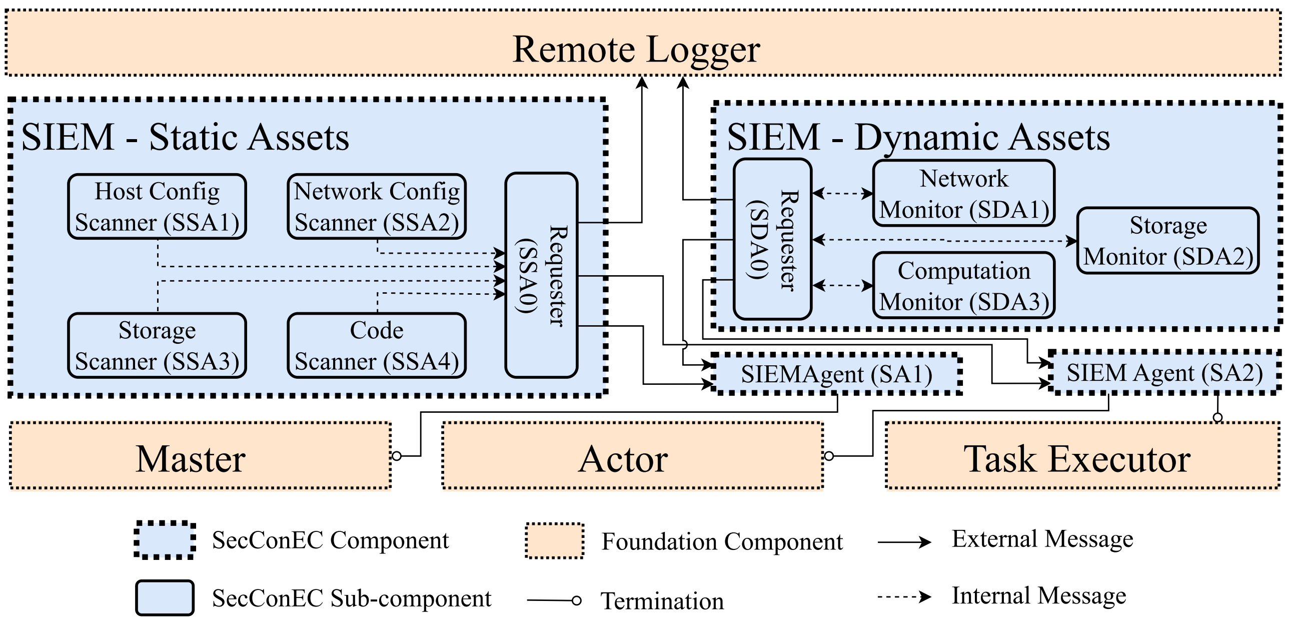

**Fig. 8.** Components and interactions of SIEM.

### 4.2. *Foundation framework optimization*

SecConEC optimizes its foundation framework for enhanced security and efficiency, as shown in Fig. 7.

#### 4.2.1. *Secured communication*

Component communications are secured to prevent denial of service attacks and ensure integrity (Table 4 #5–7, 12–14). SecConEC employs virtual private networks at the network layer, secure sockets at the transport layer, and container network overlay (CNO) at the application layer. *Message Handlers* support secure sockets, and *Network Controller (NC)* and *Network Device (ND)* manage container network overlays.

#### 4.2.2. *Optimized interactions*

Interactions across trust boundaries are minimized (Table 4 #8–9) to reduce the attack surface. Only *Master* interacts with *Remote Logger* (shown in Fig. 8), forwarding logs from other components to mitigate tampering threats.

#### 4.2.3. *Optimized master*

This subsection outlines the enhancements and secure deployment processes implemented in the optimized *Master* component.

**Secure Deployment.** The optimized *Master* securely schedules container tasks. It includes a new sub-component (*Network Controller*) and the optimized *Message Handler*, *Registry*, and *Scheduler*. Upon receiving a *User* request for an IoT application (Algorithm 1 line 5), *Registry* identifies necessary signed images and searches for *Actors* supporting these images (line 9). *Scheduler* uses a genetic algorithm [36] to estimate response times, considering secure communication levels, predefined security constraints, and *User* constraints. If needed, *Network Controller* creates the container network overlay (line 11) before distributing commands, ensuring application container isolation. Once the schedule is determined, *Scheduler* signs and distributes allocation commands (line 12) to *Actors* to initialize *Task Executors*.

**Genetic Algorithm.** To demonstrate the feasibility of a security-aware scheduling algorithm within SecConEC, we incorporate a genetic algorithm as a proof of concept. The choice of a genetic algorithm is motivated by its widespread application and proven effectiveness in scheduling domains [37]. Genetic algorithms (GAs) are evolutionary optimization methods inspired by natural selection, characterized by iterative refinement of candidate solutions through processes such as selection, crossover, and mutation.

**Algorithm 1** Procedure for Secure Deployment And Execution

```
 1: SIEM Static Assets
 2:     $img\_signature \leftarrow ScanThenSign(image)$
 3:
 4: User
 5:     $token \leftarrow Request(iot\_app, sec\_constraints)$
 6:     $ReceiveResult(token)$
 7:
 8: Master
 9:     $actors \leftarrow LookUpImages(iot\_application)$
10:     $schd \leftarrow Schedule(sec\_constraints, actors)$
11:     $net \leftarrow AssignNetwork(schd)$
12:     $cmd\_signature \leftarrow Sign(schd, net)$
13:     $AllocateResource(schd, cmd\_signature)$
14:
15: Actor
16:     $s, app, net \leftarrow Verify(schd, cmd\_signature)$
17:     if $s$ is Valid:
18:         $Execute(img\_signature, app, net)$
19:
20: SIEM Dynamic Assets
21:     while True:
22:         $actions \leftarrow Monitor(resource\_utilization,$
23:                                 $signatures, policies)$
24:
25: SIEM Agent
26:     $perform(actions)$
```

1. **Selection.** Initially, a population of potential solutions is generated randomly. During the selection phase, candidate solutions are chosen based on their fitness – a measure evaluating their effectiveness or quality – ensuring that superior candidates are more likely to pass their traits to subsequent generations. This fitness-based selection is crucial as it drives the convergence of the algorithm towards optimal solutions.
2. **Crossover.** Subsequently, crossover combines pairs of selected solutions to produce offspring by exchanging subsets of their characteristics, fostering the exploration of the solution space.

3. **Mutation.** Mutation introduces random alterations to individual offspring, promoting diversity within the population and preventing premature convergence to local optima.

The iterative interplay among selection, crossover, and mutation enables genetic algorithms to efficiently navigate complex, constraint-rich search spaces. The widespread adoption and proven effectiveness of GAs in various scheduling contexts [37] underline their suitability for handling the complex constraints and dynamic requirements inherent in security-aware scheduling scenarios.

**Secure Scheduling.** To incorporate security constraints during the scheduling stage of a genetic algorithm, we focus on optimizing the fitness function. Let $l$ and $u$ represent the lower and upper bounds, respectively. Define the acceptance range $\mathcal{A} = [\alpha_l, \alpha_u]$ as the importance of response time and $\mathcal{B} = [\beta_l, \beta_u]$ as the importance of security score, where *User*'s acceptance $\mathcal{A}_{User}$ needs to be a subset of *Master*'s acceptance $\mathcal{A}_{Master}$. Let $FRT$ denote the fitness function for response time and $FRS$ denote the fitness function for risk score.

The goal is to find an optimal schedule $S$ among possible schedules $\{s_0, s_1, \dots, s_k, \dots\}$ that balances response time and security. This can be formulated as follows:

$$S_{\alpha,\beta} = \min\left(\alpha_i FRT(s_k) + \beta_j FRS(s_k)\right) \tag{5}$$

where $F_{RT}(s_k), F_{RT}(s_k) \in [0,1]$ after normalization, and $\alpha_i, \beta_j \in [0,1]$. Two primary algorithms are utilized to achieve this goal:

1. Algorithm 2 describes the approach to constraint security by dynamically assessing the risk scores of the resource placement. As it indicates, the fitness calculation requires the system's information to be analyzed and configured so that the algorithm understands the risk scores for each data flow item, which has been demonstrated in Section 3.
   The function FRT computes $N$, the fitness value related to response time, by determining the maximum total delay and computation time summed over each path in the execution map (Algorithm 2, lines 1–5).
   The function FRS computes $M$, the total economic risk score for all data flow items within a schedule $s$ (lines 7–11).
   Finally, the FITNESS function calculates the overall fitness value $r$ by applying intermediate acceptance weights $\alpha$ and $\beta$ to $N$ and $M$, respectively, and summing the results (lines 13–17).

2. Algorithm 3 outlines the integration of the acceptance ranges $\mathcal{A}$ and $\mathcal{B}$, representing the importance of response time and security score, respectively. While an intuitive approach might involve iterating through every possible combination of $\alpha$ and $\beta$ values with a specific granularity, this would likely be inefficient. Instead, by encoding $\alpha$ and $\beta$ as genes within the population, we enhance the algorithm's efficiency and effectiveness, enabling it to more rapidly discover optimal offspring with low response time and risk score, or suitable schedules in other words.
   *RandomlyGenerate* initializes the population with diverse solutions to ensure broad exploration of the search space using acceptance ranges $\mathcal{A}$, $\mathcal{B}$, and node IDs (Algorithm 3 line 1).
   *TournamentSelection* competitively selects superior individuals from the population based on $r$ (fitness) to guide convergence (line 3).
   *SimulatedBinaryCrossover* (SBX) effectively combines parent solutions, producing offspring by preserving and exploiting beneficial traits (line 4).
   *PolyMutation* introduces controlled random perturbations to maintain population diversity and prevent convergence to local optima (line 5).
   *DeDuplication* eliminates identical or highly similar solutions to ensure diversity (line 6).
   Finally, *MinFITNESS* enforces minimum fitness constraints (line 8), ensuring that solutions reach optimality after a predefined maximum number of iterations.

**Algorithm 2** Security-Aware Fitness

```
1:  FRT ($s_k$):
        /* $D_{xy}$: the delay from node x to y that read from
           data storage where the value is set to zero for the
           last node;
           $CT_x$: the computation time at node x for a task;
           $p_l$: $l_{th}$ data flow among all the data flows $P$ */
2:      $D_{xy} \leftarrow s_k$
3:      $CT_x \leftarrow s_k$
4:      $N \leftarrow \max_{p_l \in P}\left(\sum_{i=1}^{n} D_{i(i+1)} + CT_i\right)$
5:      return $N$
6:
7:  FRS ($s_k$):
        /* $T_i$: the effectiveness of a mitigation or detection
          technique of which the value is in [0, 1];
          $D_j$: the risk score of a data flow item putting on a
           node */
8:      $T_i \leftarrow s_k$ where $T_i \in [0,1]$
9:      $D_j \leftarrow s_k$
10:     $M \leftarrow \sum T_{i(i+1)} D_j$
11:     return $M$
12:
13: FITNESS ($\alpha, \beta, s_k$):
14:     $\mu \leftarrow \alpha FRT(s_k)$
15:     $\nu \leftarrow \beta FRS(s_k)$
16:     $r \leftarrow \mu + \nu$
17:     return $r$
```

**Algorithm 3** Security-Aware Genetic Algorithm

```
1: $pop \leftarrow$ RandomlyGenerate($nodes, \mathcal{A}, \mathcal{B}$)
2: for $i$ from 0 to max_iteration_num
3:     $parents \leftarrow$ TournamentSelection($pop, r$)
4:     $offsprings \leftarrow$ SBX($parents$)
5:     $offsprings \leftarrow$ PolyMutation($offsprings, \mathcal{A}, \mathcal{B}$)
6:     $pop \leftarrow$ DeDuplication($offsprings$)
7: end for
8: $best\_chromosome \leftarrow MinFITNESS(population)$
9: return $best\_chromosome$
```

**Scheduling Practical Feasibility.** In Algorithm 2, the term $D_j$ denotes the economic risk score associated with placing a particular data flow item on a given node. Determining each $D_j$ is non-trivial and often cannot be fully automated. System administrators, security engineers, and relevant stakeholders must collaborate to assign these values, drawing on domain knowledge, threat analyses, and risk tolerance. As a result, $D_j$ inherently exhibits a certain degree of subjectivity.

When applying this security-aware scheduling approach to a different integrated edge–cloud system, each $D_j$ must be comprehensively evaluated to reflect that environment's operational and economic priorities. Although our illustrative example assigns a lower risk to cloud resources (due to their regulated access controls) than to edge devices (which are more exposed to public or semi-public networks), other deployments may exhibit additional nuances. For instance, private edge nodes maintained by internet service providers in locked enclosures often entail markedly lower risk than publicly accessible nodes situated in hotels or train stations (with the same framework deployed). However, here we generalize a simple distinction between "cloud devices" and "edge devices" for demonstration purposes, without sacrificing general applicability. More granular categorizations can readily be incorporated by refining the economic risk scoring methodology, thereby accommodating numerous device classes with distinct security postures.

**Table 5**
Mitigation and detection integrated in SecConEC.

| # | STR IDE | MITRE ATT&CK Tech. | Mitigation | Detection |
|---|---|---|---|---|
| 1 | T10 | Malicious Image T1204.003 | SSA4 | SSA4, SDA3, SA2 |
| 2 | T11 | Implant Internal Image T1525 | SSA4 | SSA4, SDA3, SA2 |
| 3 4 | I101 I102 | Credentials In Files T1552.001 | *Training, Auditing* | SSA3, SSA4 |
| 5 6 7 | D11 D100 D101 | Endpoint Denial of Service T1499 | SSA2, U0, M0, RL0, A0, TE0, M1, M2, M3, A1, A2 | SDA1, SA1, SA2 |
| 8 9 | E110 E111 | Escape to Host T1611 | SSA4 | SSA1, SA1, SA2 |
| 10 | D101 | Resource Hijacking T1496 | M1, M2, M3, A1, A2 | SDA3, SA1, SA2 |
| 11 | D100 | Resource Hijacking T1496 | M1, M2, M3, A1, A2 | SDA2, SA1, SA2 |
| 12 13 14 | D11 D100 D101 | Network Denial of Service T1498 | SSA2, U0, M0, RL0, A0, TE0 | SDA1, SA1, SA2 |

**Balanced Scheduling.** The secure scheduling scheme integrates economic risk scores, dynamic system state information, and inter-node communication delays to make informed resource-allocation decisions under time-sensitive constraints. By weighing different economic risk scores of threat types, it aims to mitigate high-impact vulnerabilities without excessively compromising performance. Moreover, the security constraint is user-configurable, allowing each stakeholder to emphasize either responsiveness or risk aversion in alignment with evolving operational priorities. This design confers practical flexibility, enabling users to dynamically adjust scheduling objectives as threat levels, workloads, and business requirements change.

**Comparison with Foundation Scheduling.** Our approach differs from FogBus2 [5], which primarily employs OHNSGA for scheduling optimization based on past execution profiles and response time metrics. While OHNSGA effectively enhances convergence and resource utilization through historical data, it does not incorporate security considerations into its evaluation criteria.

In contrast, our proposed genetic algorithm extends OHNSGA by explicitly integrating security constraints into the scheduling process. Specifically, our fitness function (Eq. (5)) simultaneously optimizes for two objectives: response time ($FRT$) and risk score ($FRS$). This integration is reinforced by dynamically computed risk metrics and user-defined security acceptance thresholds ($\mathcal{A}$, $\mathcal{B}$). Consequently, our approach ensures not only efficient scheduling performance but also strict adherence to security requirements, addressing a critical dimension that FogBus2 currently overlooks.

### 4.2.4. Optimized actor

*Actors* verify allocation commands before initializing *Task Executors*. Upon receiving an allocation from the *Master*, *Task Executor Initializer* verifies digital signatures (Algorithm 1 line 16). If valid, the container is initialized with these arguments and attached to the specified container network overlay using *Network Device* (line 17 & 18), ensuring command integrity and mitigating tampering and resource hijacking risks (Table 4 #10–11).

## 4.3. Analysis of optimized system and discussion

We analyze the data flow diagram of the optimized system, shown in Fig. 9. Interactions between components have been minimized to reduce the attack surface. Notably, only the *Master* interacts with the *Logger Manager*, eliminating other component interactions to reduce exposure. By limiting these interactions, the *Logger Manager*'s interfaces are less exposed, mitigating potential attacks. The mapping of threat types to data flow diagram items (as in Table 3) is omitted here to avoid duplication and conserve space. We calculate[4] the risk score of the optimized system as $\mathbb{R}_{Sec.} = 285.354$, resulting in an 85.26% reduction using the same $\Omega$.

Table 5 summarizes components designed or optimized to counter threats analyzed in Section 3.[5] Notably, rows 3 and 4 list mitigations

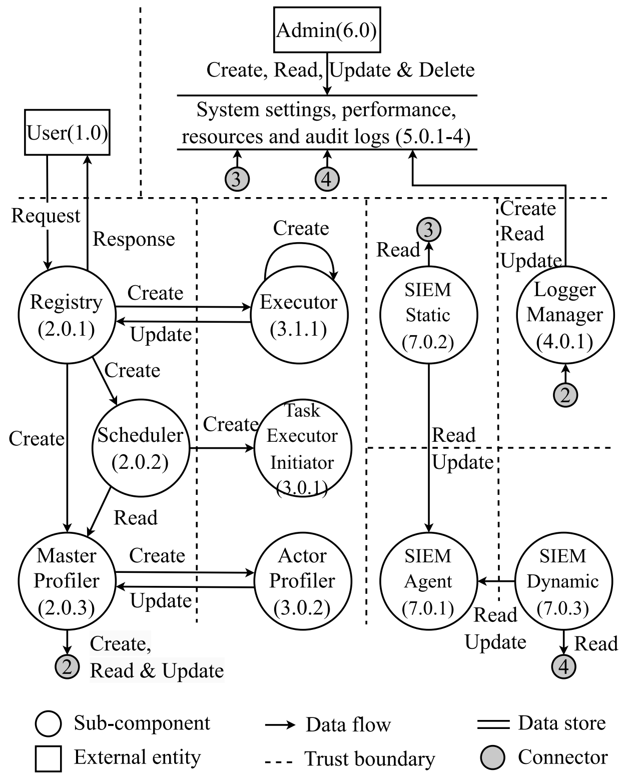

**Fig. 9.** Low-level data flow diagram of optimized system.

as *Training* and *Auditing* due to the need for developer education and auditing for credential-related threats, while detection is addressed in software design.

## 5. Performance evaluation

In this section, we validate mitigation and detection and evaluate SecConEC's performance in real-world environments using two sample applications. One application relies on high computing power and good network quality, while the other requires good network quality but minimum computing power.

## 5.1. Experiment setup and sample applications

This section outlines the experimental setup and the sample applications used. The setup involves various devices with specific roles and configurations, while the applications demonstrate different IoT scenarios, emphasizing diverse requirements for network bandwidth and computational resources.

[4] The reduction in each threat type varies according to the specific mitigation and detection measures implemented, each requiring different resources.

[5] Mitigation and detection codes are from Fig. 7.

**Table 6**
Roles, configurations, and specifications of devices in experiments setup.

| Device | Role | Amt. | Network | CPU Arch | Cores@Freq (GHz) | Mem (GiB) |
|---|---|---|---|---|---|---|
| RPi 4B | User | 1 | WiFi@5G | aarch64 | 4@1.5 | 2 |
| RPi 4B | Actor, SA2 | 1 | WiFi@5G | aarch64 | 4@1.5 | 2 |
| RPi 4B | Actor, SA2 | 2 | WiFi@5G | aarch64 | 4@1.5 | 4 |
| EVM | Master, SA1 | 1 | Ethernet | x86_64 | 16@3.6 | 64 |
| EVM | Actor, SA2 | 2 | Ethernet | x86_64 | 16@3.6 | 64 |
| EC2 C6I | Actor, SA2 | 1 | Ethernet | x86_64 | 64@3.5 | 128 |
| EC2 G5 | Remote Logger | 1 | Ethernet | x86_64 | 8@3.6 | 32 |
| EC2 M5 | SSA, SDA | 1 | Ethernet | x86_64 | 32@3.1 | 128 |
| EVM | Traffic Light | 4 | Ethernet | x86_64 | 1@3.6 | 4 |

### 5.1.1. *Experiments setup*

The devices for the experiments include 4 Raspberry Pis (RPi), 7 Edge Virtual Machines (EVM), and 3 Amazon Elastic Compute Cloud (EC2) instances. Table 6 details their roles, configurations, and specifications.

### 5.1.2. *Object detection (OD)*

This application identifies objects in images and is helpful for traffic management, home security, robotics, and healthcare. It transfers significant image data, requiring high bandwidth and computational power. In our experiment, the application integrates Yolov7 [38] with their tiny model. The image frames are from freeway traffic footage.

### 5.1.3. *Smart traffic light (STL)*

This application aggregates traffic light statuses for possible use by pedestrians, drivers, and smart vehicles. It transfers minimal data, needing low-latency networks and minimal CPU and memory. In our experiment, 4 servers generate traffic light statuses, and the application aggregates upon request.

## 5.2. *Analysis of mitigation and detection*

In this section, we evaluate the effectiveness of our mitigation and detection mechanisms, which are built on the design strategies and scheduling algorithms introduced in Section 4. We begin by describing controlled experiments in which adversaries attempted unauthorized container executions, resource starvation attacks, and network-based data exfiltration. Our findings confirm that the integrated mitigation techniques successfully block attackers when they violate predefined policies or jeopardize system stability.

Moreover, we observe that detection performance critically depends on how frequently the Security Information and Event Management (SIEM) dynamic asset module queries the SIEM agent for real-time status updates on system and container activity. While a higher polling frequency lowers detection latency, it also increases system overhead, illustrating the trade-off between rapid response and resource consumption. Once an anomaly is identified, the reaction mechanism – introduced in Section 4 – promptly suspends or removes the compromised container and notifies administrators. Our experimental measurements demonstrate that this swift response is vital for sustaining overall system continuity. Notably, the reaction times depicted in the performance graphs primarily indicate how quickly the approach mitigates the impacts of discovered attacks.

Sections 5.2.1 and 5.2.2 detail the detection and mitigation of Resource Hijacking attacks. SecConEC's real-time monitoring, powered by *SIEM Dynamic Assets* and the *SIEM Agent*, enables rapid anomaly detection, which triggers an automated mechanism to terminate compromised containers and issue alerts. We observe a corresponding recovery in response and computation times, highlighting the practical benefits of this integrated detection and mitigation framework.

To stay with space limitations, we present only the validation of resource hijacking detection for network (RHN) and computation (RHC) attacks within SecConEC. By monitoring the response time (RT) and computation time (CT) for object detection and smart traffic lights, we observe the impact of these attacks and validate our detection methods. Response time is defined as the duration from when the *User* sends data to when the *User* receives the result. Computation time is defined as the duration from when a *Task Executor* receives the data to when the *Task Executor* completes the computation. All experiments were conducted with identical setups, using the foundation system as a baseline for comparison; any legend starting with "Fdn." represents results using the foundation system.

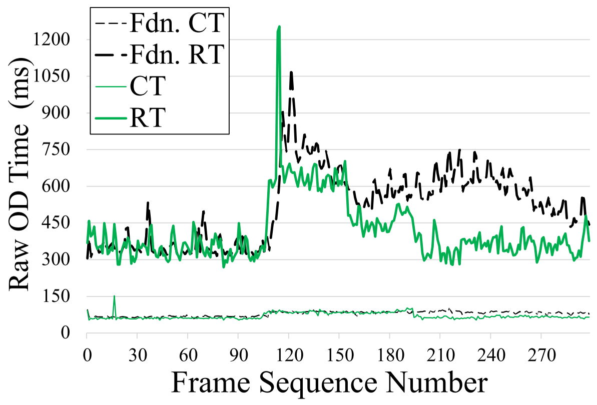

**Fig. 10.** Object detection response time with detection of resource hijacking on network.

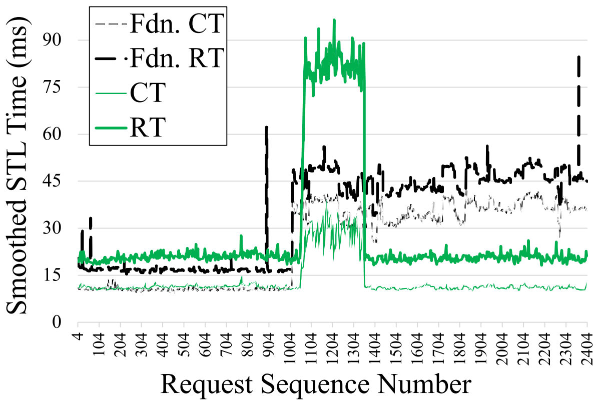

**Fig. 11.** Smart traffic light response time with detection of resource hijacking on network.

### 5.2.1. *Detection of resource hijacking on network*

We initiate a resource hijacking on a network attack by signing an image with code that fully utilizes network bandwidth and monitoring its impact. A container of this image is executed with signed commands (thus will pass Algorithm 1 line 16 & 22–23) on the host where object

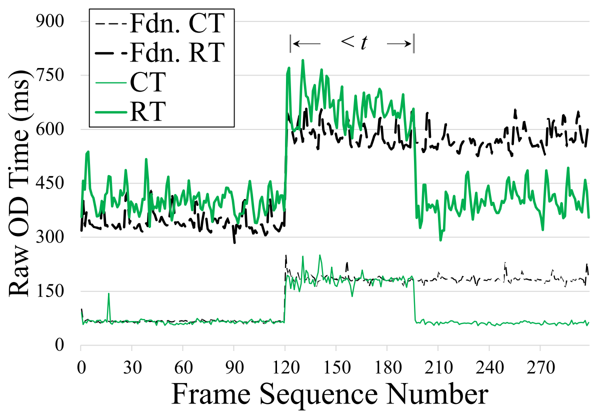

**Fig. 12.** Object detection response time with detection of resource hijacking on computation.

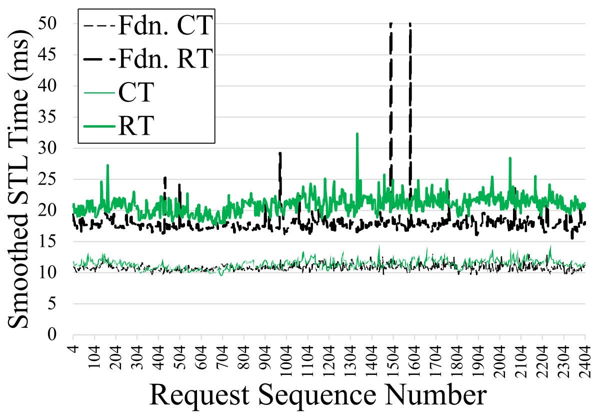

**Fig. 13.** Smart traffic light response time with detection of resource hijacking on computation.

detection or smart traffic light is running. Fig. 10 shows the raw times of each frame for object detection, while Fig. 11 shows smoothed times for smart traffic light by calculating the median (for better presenting) of a data point with its previous four data points. Smoothing was applied to address the substantial variability present in the raw data, which obscured clear visual identification of trends. Importantly, this process preserves the trend patterns without altering the underlying data structure.

Fig. 10 indicates that resource hijacking on network doubles the response time of object detection, primarily due to the transfer of frames and results. A slight drop in response time during the second half of the attack is observed, likely because the operating system prioritized network IO for object detection more. Computation time is slightly affected as the detection of objects consumes computing power, with the attacking container consistently handling transferring data, causing a slight computation time increase. In comparison, the foundation (Fdn.) system cannot mitigate the impact of the attack, resulting in prolonged response time and computation time.

Fig. 11 shows that resource hijacking on a network quadruples the response time of smart traffic lights, with the main increase from transferring data. Computation time nearly doubles, as the aggregation of traffic lights is significantly affected by the poor network conditions during the attack. In comparison, due to the sensitivity of smart traffic lights to network conditions, both the response time and computation time for the application running on the foundation system also increased and remained high. Notably, the increased response time of the smart traffic lights is lower on the foundation system than on SecConEC. This discrepancy is attributed to the overhead introduced by the new secure communication mechanisms, which we discuss in detail in Section 5.4.

The resource hijacking on the network is detected and mitigated by terminating the attacking container, returning response times for both object detection and smart traffic light to normal. The time ($t$) to detect such attacks is linked to the frequency ($f = 1/t$) that *SIEM Dynamic Assets* invokes *SIEM Agent*. Thus, attacks cannot last more than $t$. We set $t$ at 1 min, balancing high incident response speed and resource consumption.

### 5.2.2. Detection of resource hijacking on computation

We ran a resource hijacking on computation attack using a signed image (thus pass Algorithm 1 line 16 & 22–23) with code that fully utilizes the CPU.

Fig. 12 shows that a resource hijacking on computation attack doubles the response time of object detection and triples the computation time, as the attacking container consumes computing power, causing longer scheduling delays for object detection computations. In contrast, the foundation (Fdn.) system is unable to detect or mitigate this impact.

However, as shown in Fig. 13, both response time and computation time for smart traffic lights are minimally affected by resource hijacking on computation since they require little computing power.

### 5.2.3. Additional validation

To comprehensively assess the resilience and efficacy of SecConEC, we conduct a series of manually executed experiments aimed at replicating realistic intrusion scenarios. These experiments were specifically designed to validate the integrated detection and mitigation mechanisms under diverse attack vectors in containerized IoT environments. Table 7 summarizes the results, and the following points elaborate on each scenario:

1. **Deployment of malicious images without valid signatures.** Attackers may attempt to introduce malicious images during the application deployment phase. In our tests, we intentionally deployed unsigned images to confirm that the container execution tool immediately rejected these images, thus preventing unauthorized code execution. While advanced adversaries may exploit unknown vulnerabilities to bypass signature checks, our system still successfully identified and flagged the malicious container by verifying its metadata against the *Master* node's signature. An administrative alert was generated upon detecting a running container with invalid provenance.
2. **Exploitation of security flaws in legitimate (signed) images.** Even legitimate containers can become attack vectors if they are paired with malicious or unverified input. We emulate such a scenario by injecting hostile parameters into a signed container image. The system promptly detected the suspicious runtime arguments and notified the administrator. This confirms that SecConEC not only enforces container authenticity via signing but also continuously monitors for anomalies indicative of exploit attempts.
3. **Resource starvation using legitimate signed containers.** An attacker may deploy a validly signed container with verified parameters, yet still aim to overwhelm system resources. To test SecConEC's response, we launch a resource-intensive task that attempted to starve available system capacity. The monitoring tools in SecConEC accurately flagged the anomalous resource consumption, halting the attack and alerting the administrator. This result demonstrates that even authorized containers are scrutinized for runtime behavior.
4. **Tampering with system communications while operating a compromised container.** Attackers who successfully compromise a container may remain stealthy and attempt to intercept sensitive information in transit. In our experiments, we employ traffic-dumping techniques within a verified container to access

**Table 7**
Tested deployment and execution of iot application.

| # | IoT app security | Application container | Outcome |
|---|---|---|---|
| 1 | Deployment | Unsigned random image | Mitigated & Detected |
| 2 | Deployment | Signed image with unsigned arguments | Mitigated & Detected |
| 3 | Execution | Overloading signed image & arguments | Detected |
| 4 | Execution | Tampering with communication | Secured |

system communications. However, because all data exchange within SecConEC relies on TLS, VPN, and isolated container networking, no actionable information could be retrieved by the attacker. This outcome demonstrates the robustness of SecConEC's network isolation and encryption measures.

Although some of these experiments are inherently difficult to visualize, they serve as critical validation points for assessing SecConEC's layered security strategy under adversarial conditions. The detailed test results in Table 7 further confirm the system's capability to detect, mitigate, and alert on a range of potential attacks in containerized IoT ecosystems.

### *5.3. Analysis of security-aware scheduling*

Recall from Section 3.2.1 that the quantitative assessment of economic risk scores inherently involves subjective assessments [22,31, 32], but the underlying methodology remains consistently applicable across different systems. In our system, this economic risk scoring approach is directly integrated into the scheduling process, wherein each computational node is assigned a distinct economic risk score based on its susceptibility to prioritized threats, primarily denial-of-service (DoS) attacks as outlined in Section 3.2.2. For example, considering the nodes listed in Table 6, the economic risk scores assigned reflect the relative security and susceptibility of each node: `RPi 4Bs` connected via public WiFi networks have a economic risk scores (e.g., 7 out of 10) due to their elevated exposure to DoS attacks. Conversely, edge nodes (`EVMs`) in dedicated, secured edge data centers have lower scores (5), while cloud nodes enjoy the lowest risk scores (3) due to robust infrastructural security measures commonly deployed by cloud providers.

This assignment of risk scores forms the critical parameter $D_j$ used in the fitness evaluation described in Algorithm 2. The effectiveness factor $T_i$ of security mitigations was uniformly assumed to be 0.8 for simplification and demonstration purposes. It is important to emphasize again that while the specific values of the economic risk scores might vary across deployments based on expert assessment, the structured methodology remains universally applicable.

Building upon these risk assessments, we test our security-aware scheduling policy under varied acceptance ranges $\mathcal{A}$ and $\mathcal{B}$, representing the importance placed on response time and economic security score, respectively.

Fig. 14 shows the average (5 tests) response time of the object detection application under a security-constrained scheduling policy, utilizing Algorithms 2 and 3, with three predefined ranges of $\mathcal{A}$ (acceptance range of the importance of response time) and $\mathcal{B}$ (acceptance range of the importance of economic security score) introduced in Section 4.2.3. Ranges I, II, and III correspond to $\alpha \in [0.1, 0.3]$, $\beta \in [0.7, 0.9]$; $\alpha \in [0.4, 0.6]$, $\beta \in [0.4, 0.6]$; and $\alpha \in [0.7, 0.9]$, $\beta \in [0.1, 0.3]$, respectively. Range I prioritizes security over response time, assuming cloud layer resources are less exposed than edge layer resources and thus have lower risk scores for the same codebase. Range II balances security and response time equally, while Range III prioritizes lower response time over security.

The results indicate a notable trade-off between security and performance. When security ($\beta$) is significantly prioritized over response time ($\alpha$), the response time nearly doubles from 419.05 to 786.81 ms. This occurs because the scheduling policy intentionally allocates tasks

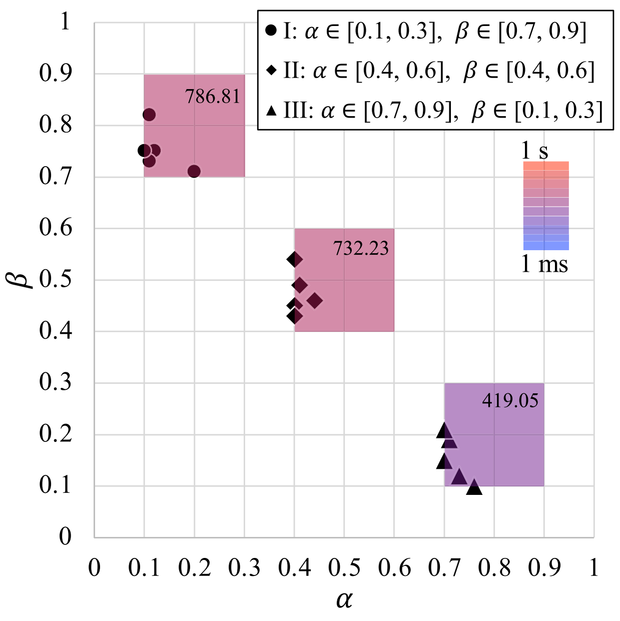

**Fig. 14.** Object detection average response time with $\alpha$–$\beta$ pairs given by security-aware scheduling.

to cloud resources rather than proximate edge resources, reflecting the policy's dependence on risk scores that favor the secure infrastructure provided by cloud environments. This strategy is particularly justifiable when the perceived economic impact of security breaches significantly outweighs the benefit of reduced response time, thus aligning with an economically rational mitigation approach to DoS threats (prioritized by threat ranking in Section 3.2.2.)

Importantly, this observation does not contradict the purpose of developing robust attack detection and mitigation mechanisms at the edge layer. Instead, it highlights the complementary nature of edge and cloud security measures within the integrated framework. Security mechanisms at both edge and cloud layers are essential for early threat detection, local mitigation, and maintaining service continuity under normal operational scenarios, while the security-aware scheduler leverages controlled resources (dedicated edge resources and cloud resources) as a strategically secure fallback when the economic impacts of security breaches become critically significant. Thus, the reliance on cloud resources when prioritizing security does not diminish the necessity or effectiveness of edge security measures. Instead, it provides an additional, strategic security layer by leveraging the inherently more secure cloud infrastructure when high-security assurance is mandated.

Fig. 15 presents the scheduling time (ST) for both the foundational system and the security-aware scheduling of SecConEC. Scheduling time is defined as the interval from when the *Master* begins scheduling to the moment a decision is made, just before notifying the involved components. The results indicate that the $\alpha$–$\beta$ security-aware scheduling takes approximately 8 ms longer (an increase of 45.67%) compared to the foundational system. This increase primarily originates from the normalization (each of 100 iterations) process involving unavoidable sequential computations. However, the total scheduling time remains short at 25.248 ms.

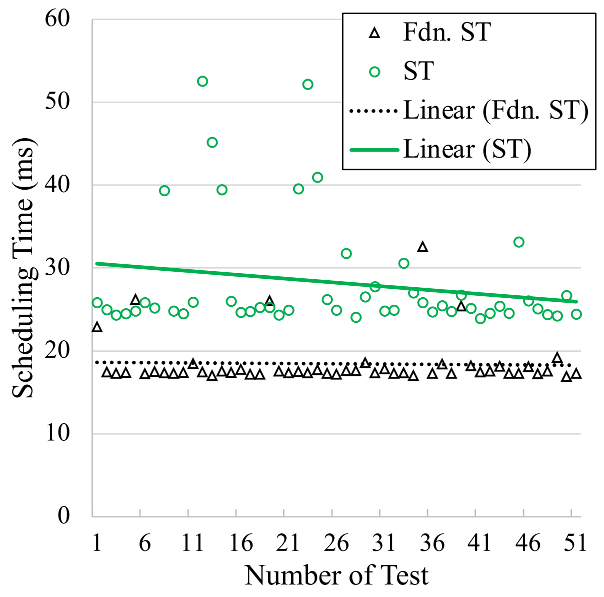

**Fig. 15.** Time comparison of object detection between SecConEC security-aware and foundation scheduling.

In summary, the security-aware scheduling algorithm effectively demonstrates the dynamic consideration of both security and performance requirements. Rather than conflicting with edge security mechanisms, the scheduler strategically supplements them by intelligently leveraging the more secure resources when economically justified, thereby creating a coherent, multilayered security posture tailored for varied operational needs.

## 5.4. Analysis of secure communication overhead

This section evaluates the overhead introduced by various secure communication techniques and their combinations. P, O, T, and V denote **p**laintext, container network **o**verlay, **t**ransport layer security, and **v**irtual private network communication, respectively. Combinations of them reflect the use of multiple techniques. The increased ratio of response time (IRRT) is used to compare the response times of other techniques with plaintext. Fig. 16 shows that the increased ratio of response time of object detection for the techniques ranges from 5.92% to 29.48%, averaging 16.2%. Similarly, Fig. 17 illustrates that the increased ratio of the response time of smart traffic lights ranges from 1.7% to 21.3%, with an average of 10.64%. Both figures demonstrate that virtual private network communication introduces the highest overhead, more than doubling that of container network overlay and transport layer security. Interestingly, the increased ratio of response time of VOT is slightly lower than VO for both object detection and smart traffic light, likely because the container network overlay[6] skips encryption upon detecting transport layer security use.

In conclusion, container network overlay is recommended for IoT applications like smart traffic light that require low latency and minimal security. For applications like object detection that demand real-time processing of sensitive data, container network overlay with transport layer security is advised. Virtual private network should be employed when higher security is essential.

[6] We use overlay network driver of Docker for container network overlay.

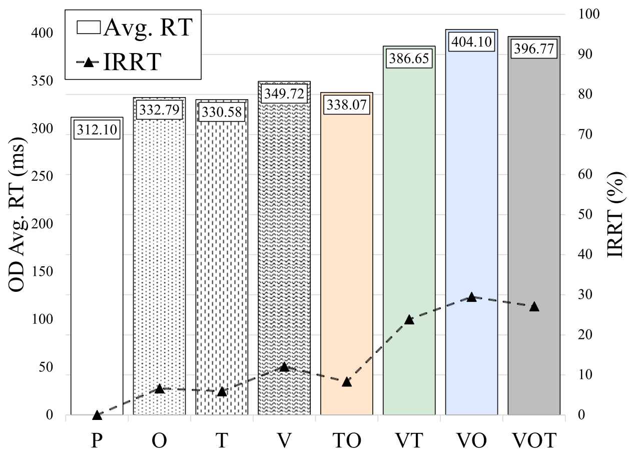

**Fig. 16.** Object detection average response time.

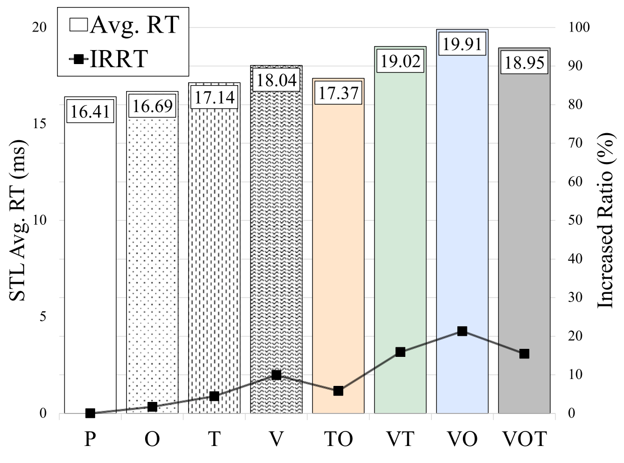

**Fig. 17.** Smart traffic light average response time.

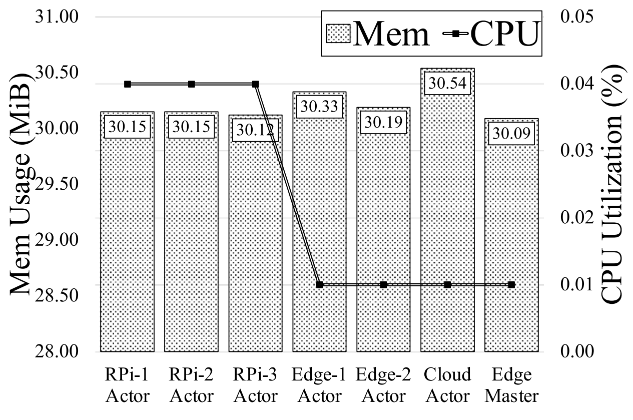

**Fig. 18.** Resource usage of SIEM agent.

## 5.5. Analysis of SIEM overhead

This section examines the CPU and memory usage of SecConEC's main components. The components were monitored for SIEM Agent, SIEM Static Assets, and SIEM Dynamic Assets running on experimental devices.

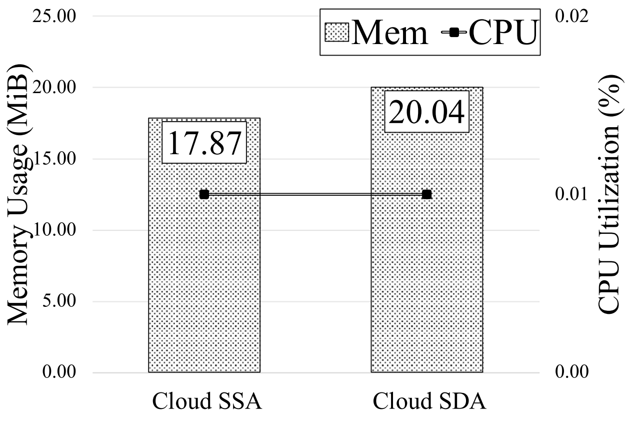

**Fig. 19.** Resource usage of SIEM.

Fig. 18 shows that SIEM Agents running on the devices consume around 30 MiB of memory and utilize 0.04% of CPU on RPis, compared to 0.01% on Edge or Cloud devices, reflecting the different CPU performances.

Fig. 19 compares the overhead of SIEM Static Assets and SIEM Dynamic Assets. Although both have the same minimum CPU utilization, SIEM Dynamic Assets uses more memory than SIEM Static Assets because the library is used to verify digital signatures.

In summary, the overhead of SIEM is low. This is due to the use of threshold-based policies with fine granularity, but higher overheads can be expected with more complex policies, such as those involving machine learning algorithms.

## 6. Conclusions and future work

This paper introduced SecConEC, an economically driven, lightweight framework designed for the secure deployment and execution of containerized IoT applications in integrated edge–cloud environments. SecConEC integrates comprehensive threat modeling using STRIDE and MITRE ATT&CK frameworks with explicit economic risk assessments, prioritizing tampering and resource-hijacking threats due to their significant economic impact arising from the open, multi-user nature, and limited resource characteristics of edge environments. The proposed framework enhances security by systematically embedding economic considerations into system design and implementing robust yet lightweight security policies, ensuring container integrity and facilitating secure communications with economically justified overhead. Furthermore, SecConEC incorporates a novel, economically-aware dynamic scheduling algorithm that effectively balances service latency requirements and security constraints, proactively mitigating the economic consequences of potential security incidents. Performance evaluations demonstrated SecConEC's effectiveness in economically mitigating critical threats, preserving system performance with minimal latency overhead (approximately 1.7%).

Future work includes (1) automatic mechanism which dynamically udpate threat weights based on the latest risk assessments, (2) enhancing the scalability of SecConEC to handle larger and more complex IoT deployments, (3) integrating machine learning techniques for dynamic threat detection, (4) leveraging consensus algorithms for decentralized and tamper-proof logging of security events, (5) exploring machine learning algorithms for intelligent, security-aware scheduling, and (6) incorporating recommendation mechanisms into scheduling to smartly minimize overall costs and improve the general performance of the system.

## Software availability

The code of SecConEC can be accessed from https://github.com/Cloudslab/SecConEC.

## CRediT authorship contribution statement

**Qifan Deng:** Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Resources, Methodology, Formal analysis. **Mohammad Goudarzi:** Methodology. **Arash Shaghaghi:** Methodology. **Majid Sarvi:** Supervision. **Rajkumar Buyya:** Conceptualization.

## Declaration of Generative AI and AI-assisted technologies in the writing process

During the preparation of this work, the first author used Gemini (2.5 Pro) and ChatGPT (o3) to improve language and readability, format LaTeX source code, and debug LaTeX compile errors. After using the tools, the first author reviewed and edited the content as needed and takes full responsibility for the content of the publication.

## Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## Appendix A

Table 8 enumerates the specific data flows between system components, detailing the source, destination, and corresponding path for each interaction. This comprehensive mapping provides the granular detail necessary for the systematic STRIDE threat analysis.

**Table 8**
Detailed data flows within the DFD.

| Data flow description | Source | Destination | DFD flow path |
|---|---|---|---|
| User requests resources and submits data | User (1.0) | Master (2.0.1) | 1.0 → 2.0.1 |
| Master schedules workload and initializes Executor | Master (2.0.1) | Actor (3.0.1) | 2.0.1 → 2.0.2 → 3.0.1 |
| Master forwards data to Executor for processing | Master (2.0.1) | Executor (3.1.1) | 2.0.1 → 3.1.1 |
| Master reads host profiles from data stores | Master (2.0.3) | Data Stores | 2.0.3 → 4.0.1/5.0.1-4 |
| Master profiles actor host and receives response | Master (2.0.3) | Actor (3.0.2) | 2.0.1 → 2.0.3 → 3.0.2 → 2.0.3 |
| Master responds to user registration | Master (2.0.1) | User (1.0) | 2.0.1 → 2.0.2 → 1.0 |
| Executors collaborate on tasks | Executor (3.1.1) | Executor (3.1.1) | 3.1.1 → 3.1.1 |
| Executor returns results to Master | Executor (3.1.1) | Master (2.0.1) | 3.1.1 → 2.0.1 |
| Master forwards results to User | Master (2.0.1) | User (1.0) | 2.0.1 → 1.0 |
| Components save logs to the Logger Manager | All Components | Logger Manager (4.0.1) | All Components → 4.0.1 |
| Logger Manager persists logs to Data Stores | Logger Manager (4.0.1) | Data Stores (Table 2) | 4.0.1 → 5.0.1-4 |
| Admin manages system and audits logs | Admin (6.0) | Data Stores (Table 2) | 6.0 → 5.0.1–5.0.4 |

**Table 9**
Detailed threat analysis on DFD items.

| Threat type | DFD element category | DFD item/path | Likelihood (**L**) | Impact (**I**) |
|---|---|---|---|---|
| Spoofing | External entity | 1.0 | 4 | 1 |
| | | 2.0.1 | 1 | 1 |
| Tampering | Component | 1.0 | 3 | 1 |
| | | 2.0.1 | 1 | 1 |
| | | 2.0.3 | 2 | 5 |
| | | 3.0.1 | 2 | 5 |
| | | 3.0.2 | 2 | 1 |
| | | 3.1.1 | 2 | 5 |
| | | 4.0.1 | 1 | 5 |
| | Data flow | 1.0 → 2.0.1 | 3 | 4 |
| | | 2.0.1 → 2.0.2 → 3.0.1, 2.0.1 → 3.1.1 | 3 | 2 |
| | | 2.0.3 → 4.0.1/5.0.1-4, 2.0.1 → 2.0.3 → 3.0.2 → 2.0.3 | 3 | 2 |
| | | 2.0.1 → 2.0.2 | 3 | 2 |
| | | 3.1.1 → 3.1.1 → 2.0.1 → 1.0 | 2 | 2 |
| | | 1.0/2.0.2/3.0.1/3.1.1 → 4.0.1 → 5.0.1–4 | 4 | 1 |
| Repudiation | External entity | 1.0 | 1 | 1 |
| | Component | 2.0.1 | 1 | 2 |
| | | 2.0.3 | 1 | 2 |
| | | 3.0.1 | 1 | 3 |
| | | 3.0.2 | 1 | 1 |
| | | 3.1.1 | 1 | 2 |
| | | 4.0.1 | 1 | 1 |
| Info. disclosure | Data flow | 1.0 → 2.0.1 | 2 | 1 |
| | | 2.0.1 → 2.0.2 → 3.0.1, 2.0.1 → 3.1.1 | 1 | 1 |
| | | 2.0.3 → 4.0.1/5.0.1-4, 2.0.1 → 2.0.3 → 3.0.2 → 2.0.3 | 1 | 1 |
| | | 2.0.1 → 2.0.2 | 1 | 1 |
| | | 3.1.1 → 3.1.1 → 2.0.1 → 1.0 | 1 | 2 |
| | | 1.0/2.0.2/3.0.1/3.1.1 → 4.0.1 → 5.0.1–4 | 3 | 1 |
| Denial of service | Component | 2.0.1 | 3 | 3 |
| | | 3.0.1 | 2 | 3 |
| | | 3.0.2 | 2 | 1 |
| | | 3.1.1 | 2 | 3 |
| | | 4.0.1 | 4 | 1 |
| | Data store | 5.0.1 | 4 | 1 |
| | | 5.0.2 | 4 | 1 |
| | | 5.0.3 | 4 | 1 |
| | | 5.0.4 | 4 | 1 |
| | Data flow | 1.0 → 2.0.1 | 3 | 3 |
| | | 2.0.1 → 2.0.2 → 3.0.1, 2.0.1 → 3.1.1 | 2 | 2 |
| | | 2.0.3 → 4.0.1/5.0.1-4, 2.0.1 → 2.0.3 → 3.0.2 → 2.0.3 | 2 | 3 |
| | | 2.0.1 → 2.0.2 | 2 | 2 |
| | | 3.1.1 → 3.1.1 → 2.0.1 → 1.0 | 1 | 3 |
| | | 1.0/2.0.2/3.0.1/3.1.1 → 4.0.1 → 5.0.1–4 | 3 | 2 |
| Elev. of privilege | Component | 2.0 | 1 | 5 |
| | | 3.0 | 1 | 3 |
| | | 3.1 | 1 | 3 |

## Appendix B

Table 9 provides a granular breakdown of the STRIDE threat analysis, mapping each threat type to specific system components and data flows from the DFD. It lists the quantitative likelihood (**L**) and economic impact (**I**) scores to each potential vulnerability, forming the basis for the overall economic risk score calculation.

## Data availability

Data will be made available on request.

## References

[1] T. Liu, S. Ni, X. Li, Y. Zhu, L. Kong, Y. Yang, Deep reinforcement learning based approach for online service placement and computation resource allocation in edge computing, IEEE Trans. Mob. Comput. 22 (2023) 3870–3881.

[2] X. Zhu, W. Yao, Y. Hou, S. Li, J. Luo, Z. Huang, S. Fu, RDRM: Real-time dynamic replica management with joint optimization for edge computing, IEEE Trans. Serv. Comput. (2024) 1–14.

[3] Z. Wang, M. Goudarzi, M. Gong, R. Buyya, Deep reinforcement learning-based scheduling for optimizing system load and response time in edge and fog computing environments, Future Gener. Comput. Syst. 152 (2024) 55–69.

[4] Y. Zhuang, Z. Zheng, Y. Shao, B. Li, F. Wu, G. Chen, Nebula: An edge-cloud collaborative learning framework for dynamic edge environments, in: Proceedings of the 53rd International Conference on Parallel Processing, ICPP '24, Association for Computing Machinery, New York, NY, USA, 2024, pp. 782–791.

[5] Q. Deng, M. Goudarzi, R. Buyya, FogBus2: A lightweight and distributed container-based framework for integration of IoT-enabled systems with edge and cloud computing, in: Proceedings of the International Workshop on Big Data in Emergent Distributed Environments, BiDEDE '21, ACM, New York, NY, USA, 2021.

[6] C. Pahl, A. Brogi, J. Soldani, P. Jamshidi, Cloud container technologies: A state-of-the-art review, IEEE Trans. Cloud Comput. 7 (2019) 677–692.

[7] S. Noor, B. Koehler, A. Steenson, J. Caballero, D. Ellenberger, L. Heilman, IoTDoc: A docker-container based architecture of IoT-enabled cloud system, in: Big Data, Cloud Computing, and Data Science Engineering, Springer, Cham, 2020, pp. 51–68.

[8] W. Zhang, Y. Zhang, H. Fan, Y. Gao, W. Dong, A low-code development framework for cloud-native edge systems, ACM Trans. Internet Technol. 23 (2023).

[9] G. Merlino, G. Tricomi, L. D'Agati, Z. Benomar, F. Longo, A. Puliafito, Faas for IoT: Evolving serverless towards deviceless in I/Oclouds, Future Gener. Comput. Syst. 154 (2024) 189–205.

[10] A. Yousefpour, A. Patil, G. Ishigaki, I. Kim, X. Wang, H.C. Cankaya, Q. Zhang, W. Xie, J.P. Jue, FOGPLAN: A lightweight QoS-aware dynamic fog service provisioning framework, IEEE Internet of Things J. 6 (2019) 5080–5096.
[11] A.J. Ferrer, J.M. Marques, J. Jorba, Ad-hoc edge cloud: A framework for dynamic creation of edge computing infrastructures, in: 2019 28th International Conference on Computer Communication and Networks, ICCCN, 2019, pp. 1–7.
[12] Y. Zhang, H.-Y. Wei, Risk-aware cloud-edge computing framework for delay-sensitive industrial IoTs, IEEE Trans. Netw. Serv. Manag. 18 (2021) 2659–2671.
[13] R. Roman, J. Lopez, M. Mambo, Mobile edge computing, Fog et al.: A survey and analysis of security threats and challenges, Future Gener. Comput. Syst. 78 (2018) 680–698.
[14] P. Ranaweera, A.D. Jurcut, M. Liyanage, Survey on multi-access edge computing security and privacy, IEEE Commun. Surv. Tutorials 23 (2021) 1078–1124.
[15] G. Nencioni, R.G. Garroppo, R.F. Olimid, 5G multi-access edge computing: A survey on security, dependability, and performance, IEEE Access 11 (2023) 63496–63533.
[16] Q. Fan, Y. Xie, C. Zhang, X. Liu, L. Zhu, An authentic and privacy-preserving scheme towards E-health data transmission service, IEEE Trans. Serv. Comput. 17 (2024) 1969–1982.
[17] J. Zhou, J. Sun, P. Cong, Z. Liu, X. Zhou, T. Wei, S. Hu, Security-critical energy-aware task scheduling for heterogeneous real-time mpsocs in IoT, IEEE Trans. Serv. Comput. 13 (2020) 745–758.
[18] W. Chorfa, N.B. Youssef, A. Jemai, Threat modeling with mitre ATT&CK framework mapping for SD-IOT security assessment and mitigations, in: 2023 IEEE Symposium on Computers and Communications, ISCC, 2023, pp. 1323–1326.
[19] M. Goudarzi, A. Shaghaghi, S. Finn, B. Stillerd, S. Jha, Towards threat modelling of IoT context-sharing platforms, in: 2024 22nd International Symposium on Network Computing and Applications, NCA, 2024, pp. 87–96.
[20] F. Swiderski, W. Snyder, Threat Modeling, Microsoft Press, USA, 2004.
[21] M. Howard, The security development lifecycle, 2006.
[22] A.B. Aissa, R.K. Abercrombie, F.T. Sheldon, A. Mili, Quantifying security threats and their impact, CSIIRW 9 (2009).
[23] A. Dawoud, S. Finster, N. Coppik, V. Ashiwal, Better left shift security! framework for secure software development, in: 2024 IEEE European Symposium on Security and Privacy Workshops, EuroS&PW, 2024, pp. 642–649.
[24] R. Hasan, M. Hossain, R. Khan, Aura: An incentive-driven ad-hoc IoT cloud framework for proximal mobile computation offloading, Future Gener. Comput. Syst. 86 (2018) 821–835.
[25] S.N. Srirama, F.M.S. Dick, M. Adhikari, Akka framework based on the actor model for executing distributed fog computing applications, Future Gener. Comput. Syst. 117 (2021) 439–452.
[26] G. Merlino, R. Dautov, S. Distefano, D. Bruneo, Enabling workload engineering in edge, fog, and cloud computing through OpenStack-based middleware, ACM Trans. Internet Technol. 19 (2019).
[27] B. Sheehan, F. Murphy, A.N. Kia, R.K. and, A quantitative bow-tie cyber risk classification and assessment framework, J. Risk Res. 24 (2021) 1619–1638.
[28] B.E. Strom, A. Applebaum, D.P. Miller, K.C. Nickels, A.G. Pennington, C.B. Thomas, Mitre ATT&CK: Design and Philosophy, Technical Report, The MITRE Corporation, 2018.
[29] A. Alwarafy, K.A. Al-Thelaya, M. Abdallah, J. Schneider, M. Hamdi, A survey on security and privacy issues in edge-computing-assisted Internet of Things, IEEE Internet of Things J. 8 (2021) 4004–4022.
[30] M. Souppaya, J. Morello, K. Scarfone, Application Container Security Guide, Technical Report, National Institute of Standards and Technology, 2017.
[31] M. Jouini, L. Ben Arfa Rabai, R. Khedri, A quantitative assessment of security risks based on a multifaceted classification approach, Int. J. Inf. Secur. 20 (2021) 493–510.
[32] M.B. Lindsey, A method for estimating the financial impact of cyber information security breaches utilizing the common vulnerability scoring system and annual loss expectancy, 2010.
[33] X. Ou, A. Singhal, Quantitative Security Risk Assessment of Enterprise Networks, Springer, 2011.
[34] A. Zhylin, Methodology of quantitative assessment of network cyber threats using a risk-based approach, Appl. Cybersecur. Internet Gov. 3 (2024) 227–260.
[35] S. Bhatt, P.K. Manadhata, L. Zomlot, The operational role of security information and event management systems, IEEE Secur. Priv. 12 (2014) 35–41.
[36] K. Deb, A. Pratap, S. Agarwal, T. Meyarivan, A fast and elitist multiobjective genetic algorithm: NSGA-II, IEEE Trans. Evol. Comput. 6 (2002) 182–197.
[37] J. Jiang, Z. Sun, R. Lu, L. Pan, Z. Peng, Real relative encoding genetic algorithm for workflow scheduling in heterogeneous distributed computing systems, IEEE Trans. Parallel Distrib. Syst. (2024) 1–14.
[38] C.-Y. Wang, A. Bochkovskiy, H.-Y.M. Liao, YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2023, pp. 7464–7475.

**Dr. Rajkumar Buyya** is a Redmond Barry Distinguished Professor and Director of the Cloud Computing and Distributed Systems (CLOUDS) Laboratory at the University of Melbourne, Australia. He has authored over 850 publications and seven text books including “Mastering Cloud Computing” published by McGraw Hill, China Machine Press, and Morgan Kaufmann for Indian, Chinese and international markets respectively. He is one of the highly cited authors in computer science and software engineering worldwide (h-index=174, g-index=385, 160300+ citations).