future internet

MDPI

Article

A Secure Auditable Remote Registry Pattern for IoT Systems

Antonio Maña *, Francisco J. Jaime and Lucía Gutiérrez

Institute for Software Engineering and Software Technology "Jose María Troya Linero", Universidad de Málaga, 29071 Málaga, Spain; franj@uma.es (F.J.J.); luciagutierrezmolina@uma.es (L.G.)
* Correspondence: amana@uma.es

Abstract: In software engineering, pattern papers serve the purpose of providing a description of a generalized, reusable solution to recurring design problems, based on practical experience and established best practices. This paper presents an architectural pattern for a Secure Auditable Registry service based on Message-Oriented Middleware to be used in large-scale IoT systems that must provide auditing capabilities to external entities. To prepare the pattern, the direct experience in applying the pattern solution in an industry-funded R&D project has been a key aspect because it has allowed us to gain a deep understanding of the problem and the solution, and it has contributed to the correctness and real-world applicability of the pattern as described. To further improve the quality of the paper, we have followed the commonly accepted practices in pattern development (including peer reviews) to ensure that the core aspects of the solution are correctly represented and that the description allows it to be applicable to similar problems in other domains, such as healthcare, autonomous devices, banking, food tracing or manufacturing to name a few. The work done in applying this pattern confirms that it solves a recurring problem for IoT systems, but also that it can be adopted in other domains, providing an effective solution in order to achieve enhancement of the auditability capabilities of the target systems. This pattern will be part of a pattern language (i.e., a family of related patterns) that we are developing for transitioning from legacy systems to IoT with an emphasis on security.

Keywords: IoT; security architecture; software security patterns; pattern language; design patterns; security patterns

Citation: Maña, A.; Jaime F.J.; Gutiérrez, L. A Secure Auditable Remote Registry Pattern for IoT Systems. *Future Internet* **2024**, *16*, 405. https://doi.org/10.3390/fi16110405

Academic Editors: Christos Tryfonopoulos and Nicholas Kolokotronis

Received: 11 September 2024
Revised: 24 October 2024
Accepted: 30 October 2024
Published: 4 November 2024

1. Introduction and Methodology

1.1. Introduction

The global trend towards connected IT systems and services capable of providing highly customizable information and analytics in near-real time requires fast and intelligent applications using new architectures and technologies. In this context, many legacy systems cannot meet the stringent requirements set by IoT applications or respond to the explosive growth of data generated by connected devices (IoT, IIoT (Industrial IoT) and OT (Operational Technology) are different abbreviations that are often used in combination. OT (sometimes used as a synonym for IIoT) is more industry-focused and refers to the hardware and software used to identify, monitor and control physical devices, processes and events in a manufacturing environment). The advent of the Internet of Things has been such a game changer that it has become one of the biggest drivers in the market to retire these legacy systems. In 2021, the value of the global IoT market was estimated at USD 213 billion, and forecasts suggest that it will reach between USD 2400 and USD 8100 billion by 2030 [1,2], with the number of IoT devices expected to potentially exceed 29 billion in a variety of industrial and consumer applications [3].

In the light of these numbers, and the corroboration of how fast legacy systems are becoming unfit for their purposes in the current IT context, it is clear that the decision to re-engineer legacy applications that are no longer able to meet their business objectives cannot be put off. Indeed, this situation is rather frequent in other domains involving

hardware devices, such as banking, healthcare, manufacturing, food-chain tracking and security [4].

We believe that the criticality and importance of this transition and the large number of systems that could be affected by such a process justifies the need to provide tools to facilitate and support the process of re-engineering and adapting legacy systems to new technologies such as IoT, microservices, etc. In fact, this need has already been recognized for other technological changes, and we can find, for instance, patterns for replacing legacy systems, for moving to the cloud [5,6] or even for developing systems [7].

Patterns are one of the most useful tools in software engineering in general and in re-engineering in particular. Patterns capture reusable solutions to common design and architectural problems. Patterns cover a wide range of problems and solutions on different levels, such as design (e.g., Singleton, Observer) and architectural levels (e.g., MVC, microservices), providing proven templates that developers can apply to solving problems while achieving specific properties like quality, maintainability and scalability. Patterns promote consistency, reduce complexity and improve collaboration across teams by providing a shared vocabulary for addressing recurring issues. Ultimately, patterns help streamline development processes, leading to more robust and flexible software systems.

The scientific contribution of this paper is the formulation of a new architectural pattern that is a software engineering tool to help other practitioners to better understand the problem, and to describe for them a specific proven and reusable solution to it. To produce this contribution, our work involved investigating in depth the solution, both in theoretical terms and through first-hand experience in a real industry-funded project, and extracting the essence of the problem and the solution in a form that would (1) help other practitioners and researchers learn the main aspects that are important in order to decide whether the solution is applicable to their system, and (2) facilitate the application of the solution to other systems, which, of course, requires research, generalization and polishing of the description (e.g., by removing implementation details that depend on specific technologies).

This paper presents an architectural pattern for a secure auditable registry service to be used by systems that require audit from external entities. This is a common requirement in IoT systems but is also relevant for many other distributed systems. The pattern has been crafted based on direct experience in solving the target problem in an R&D project focused on the re-engineering of a legacy large-scale high-security system in the domain of gambling, but we have followed the commonly accepted practices in pattern development (including peer reviews) to ensure that the core aspects of the solution are correctly represented and that the description allows it to be applicable to similar problems in other domains, such as healthcare, autonomous devices, banking, food tracing or manufacturing, to name a few.

To avoid confusion, in this paper we will refer to "systems" when we talk about general ones and will use the acronym SUD (System Under Design) to refer to a specific system that is being designed/developed.

The structure of the remainder of the paper is as follows: Section 2 introduces the context and background, and it presents the pattern ecosystem of the SARMOM pattern, which includes other patterns that may be used together with it, depending on the requirements of the SUD. Section 3 shows the pattern ecosystem around SARMOM. Section 4 contains the proposed pattern, described using a security-oriented variant of the POSA template, which is the one most frequently used in patterns dealing with security aspects [8]. Finally, Section 5 describes the conclusions and future work.

1.2. Methodology

Due to their practical and empirical nature, in the definition of patterns a typical scientific approach with hypotheses and experiments is not suitable, but this does not mean that this process lacks a methodology. There are well-established principles and practices to guarantee the quality of the pattern description, which have been perfected by the pattern community for a long time (the PLoP conference held its 30th edition this

year). In particular, it is common in the production of patterns to count on peer reviews and to produce several iterations of the pattern description. In the preparation of the pattern presented in Section 4, we have followed the methodology developed by the Hillside Group for the Pattern Languages of Programs (PLoP) conferences, which is centered on collaborative exploration and refinement of design solutions through a structured process. This process typically begins with the identification of a recurring problem in software design and the creation of initial pattern drafts. This step is taken during the development of an industry-funded project, and it includes a process of analysis of the solution in other existing real-world systems. Then, the drafts undergo peer review rounds, where authors present their patterns in small groups (we did this both internally, with members of our client staff, and externally, with the participation of experts in the definition of patterns). Feedback from these sessions allows for iterative improvements and helps authors articulate their patterns more clearly. This methodology emphasizes real-world applicability and collective insights, with a focus on refining patterns through discussions that highlight their context, forces and consequences. The Hillside Group's methodology is grounded in the belief that patterns are best understood and improved in a community setting, fostering a shared understanding of design challenges and solutions. This iterative, community-driven process has led to the establishment of a rich body of knowledge in software patterns, influencing both academic research and practical software development.

2. Motivation and Background

2.1. Motivating Scenario

Modern IT systems, and especially IoT systems, must comply with different standards and regulations, and they are expected to fulfill essential non-functional (a.k.a. quality) requirements, such as reliability and security. In particular, accountability, auditability and transparency are becoming essential security properties in these systems.

The commonly accepted definition of quality includes different aspects: efficiency, operability, compatibility, security or maintainability among others. We can refer to the ISO/IEC 25010:2011 software quality model [9] for an in-depth description of IT systems quality. In our case, the motivating scenario is a project for re-engineering a system to control slot machines in a casino. In this context, evolvability and maintainability are essential because the SUD must remain in operation for a long time and adapt to different changes (the current communication protocol for slot machines will soon be replaced, new functionalities are constantly added, etc.). To this end, the architecture proposed by the Secure Auditable Registry based on Message-Oriented Middleware (SARMOM) pattern is ideal, in order to achieve those properties and to reduce and isolate the impact of the future changes in the system.

Another critical requirement (which is also part of quality [9]) is security. Security properties cover an extremely wide spectrum of targets and goals ranging from the classic CIA trio (Confidentiality, Integrity and Availability) to more complex and specific properties, such as non-repudiation, access control mechanisms and policies, accountability, auditability, freshness, retention, etc. Together with the fact that there are agents interested in breaking those properties, the variety and complexity of the technical security solutions make them hard to select, design and implement.

Accountability is essential in our target domain. An accountable system provides mechanisms to understand the state of the SUD, and that, in turn, requires a guarantee that all significant actions and events are recorded and that it is possible to evaluate and trace both the state of the system and the reasons why it reached that state.

Auditability is a key requirement for the gambling domain, but also for many IoT applications and for other domains. It goes beyond accountability because it not only guarantees that we record traces of actions and events, but also that the SUD can provide trustworthy proofs of the authenticity of those traces for (normally external) entities called auditors. In this case, it is frequent to have dedicated interfaces and mechanisms for

auditors to access the proofs while maintaining the privacy and confidentiality of the system data.

Finally, transparency refers to a more general scenario, in which the facts, logs and/or reports of the SUD can be provided to any actor requesting them [10]. Obviously, transparency is usually limited or at least affected by privacy and confidentiality.

Moreover, in addition to technical requirements, our target scenarios are also subject to laws and regulations. In the case of the gambling domain, we find very different laws, such as the Gambling Act 2005 [11] for the UK, the Spanish Gambling Act or Decree-Law 422/89 or the Nevada Revised Statutes (mainly the NRS 463 series, such as the NRS 463.0129 "Public policy of state concerning gaming; license or approval revocable privilege").

Regulations governing IoT applications continue to evolve around the world, as governments gain a better understanding of the field and as they develop and evolve laws to protect citizens' rights in the expanding IoT landscape. It is not the goal of this section to provide a comprehensive overview but to highlight that regulations are converging and that the monitoring of relevant events, their protection and their use are part of those regulations. An extensive review of the laws and regulations affecting the IoT around the world is provided in [12]. As a relevant summary, we will discuss the cases of the EU and the USA.

In the European Union, the Cybersecurity Act [13] proposes a voluntary cybersecurity certification framework, which allows companies to certify their ICT products, services and processes according to EU standards. On the other hand, opposed to the voluntary character of the Cybersecurity Act, the recent EU Cyber Resilience Act [14] imposes legal obligations for stringent cybersecurity requirements on digital products, including IoT devices, to be fulfilled from design to marketing.

The European NIS2 Directive [15] extends the scope of the original NIS Directive, to include more sectors and types of entities that must comply with stricter cybersecurity standards. This directive had to be translated into national-level laws by member states by 17 October 2024. The NIS2 Directive proposes more concrete measures to reduce discrepancies between countries and ensure a common high level of cybersecurity across all member states. In particular, it establishes the obligation to facilitate "regular and targeted security audits carried out by an independent body or a competent authority" and further elaborates by establishing that "the results of any targeted security audit shall be made available to the competent authority". Finally, it establishes that "costs of such targeted security audit carried out by an independent body shall be paid by the audited entity", which means that it is in the interest of the entities managing those IT or IoT systems to implement measures to facilitate the audits and reduce the costs associated with them.

In the USA, on 12 May 2021, a Presidential Executive Order on Improving the Nation's Cybersecurity commisioned the definition of two labeling programs on the cybersecurity capabilities of Internet of Things (IoT) consumer devices and software development practices from the National Institute of Standards and Technology (NIST), which, in February 2022, released a White Paper on "Recommended Criteria for Cybersecurity Labeling in the IoT" [16] that contains guidance for IoT manufacturers and integrators. The content of this guide is voluntary, but it is important to mention that, apart from the basics, such as unique device identifiers (a serial number, for example), data protection or firmware updates, it refers to the need to maintain a cybersecurity event log. In particular, it mandates that "the IoT product captures and records information about the state of IoT components that can be used to detect cybersecurity incidents affecting or affected by IoT product components and the data they store and transmit".

2.2. Background

In this section, we provide background information that may be helpful to better understanding the SARMOM pattern. Readers who are familiar with these technologies can safely skip this section.

2.2.1. Message-Oriented Middleware (MOM)

The term Message-Oriented Middleware (MOM) is used to describe a software and/or hardware infrastructure used to send and receive messages between distributed systems and to provide storage (persistence) for those messages. The MOM decouples application components, so that they can use different operating systems and network protocols [17,18]. An important feature of modern MOM implementations, such as Apache Kafka, is that they provide persistent storage and mechanisms to organize messages, guarantee message queue properties (such as immutability, authenticity or ordering), balance the load between MOM servers, etc.

Figure 1 below shows a simplified architecture of an MOM. In particular, the figure does not show the distributed aspects of the MOM, but the logical structure. It is normal that an MOM is distributed among several servers (usually called "brokers"). The figure shows that messages are organized into "topics" and "partitions". A topic is a logical way to separate messages for processing, and each topic is maintained in at least one partition (sometimes more). According to the Kafka documentation "*Each partition is an ordered, immutable sequence of records that is continually appended to—a structured commit log. The records in the partitions are each assigned a sequential id number called the offset that uniquely identifies each record within the partition* ". To improve scalability and throughput, partitions may be managed in different replicas.

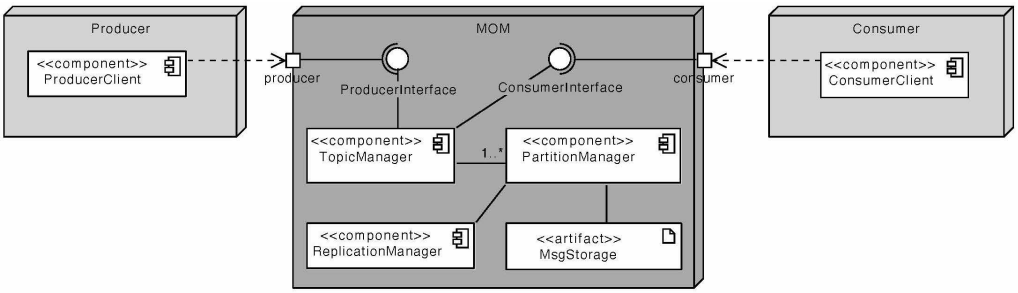

Figure 1. Architecture of a Message-Oriented Middleware (MOM).

Message-sourcing (i.e., Event-sourcing) is an effective way to maintain the state of an application, while at the same time greatly increasing trust in the current state of the system and opening the system to new and enhanced functionalities. Although they share some similarities, an MOM used in this way is not the same as blockchain. If used correctly, this solution can be made tamper-resistant, just like blockchain. Also, like blockchain, data can only be appended. But those are the main similarities. The main difference with blockchain is that this solution can scale to several millions of Transactions Per Second (TPS), while the fastest blockchains reach a maximum of 65,000 TPS.

2.2.2. Secure Registries

From a high-level perspective, and in the field of IT, a registry is a (software/hardware) component, which is frequently used as a tool for system traceability and accountability. Registries provide interfaces to add information records, to consult the registry contents and provide storage capabilities, together with the mechanisms to guarantee the immutability and ordering (sometimes partial) of the records. The information format and size is greatly dependent on the environment where the registry is deployed, ranging from short text log records to any other type of textual or binary information. The access rate to the registry content also varies from one context to another. Built-in security measures within the registry should allow information consumers to verify and trust the information source and content, whereas it also helps to meet the overall system security and regulatory compliance requirements. Thanks to all these features, a secure registry becomes a valuable (if not the most valuable) asset for system (external) audits, which, usually, is the main purpose of a secure registry; hence, it is also called auditable.

Although regular security measures can be applied in order to provide a secure registry, there currently is an ongoing trend regarding the use of blockchain technology to achieve the desired security for registries. Blockchain itself may not be the only security measure, but it is commonly found as one layer in securing the information against unauthorized changes, the key feature of blockchain [19]. In this way, records added to the registry are kept within blocks, integrating hash values to ensure the chain (and, hence, the information it contains) order. This mechanism, together with the decentralization inherent to blockchain, makes it impossible to alter the stored records in any way, given the computational power of current computers. As a result, a secure registry intended for audits is provided.

2.3. Pattern Description Formats

Patterns in software engineering are documented using various formats, to ensure clarity and consistency. The most common formats include the following:

- Alexandrian format: Inspired by Christopher Alexander's A Pattern Language, it includes context, forces, solution and resulting context. This format highlights the balancing of competing forces in design [20].
- Gang of Four (GoF) format: Introduced by Gamma et al. in Design Patterns: Elements of Reusable Object-Oriented Software, this format includes sections like Name, Intent, Applicability, Structure, Participants, Consequences and Implementation [21].
- Pattern-Oriented Software Architecture (POSA) format: Introduced by Buschmann et al. [22], this format emphasizes the pattern's name, context, problem, solution and resulting context, with examples and consequences.
- Security patterns format [23]: This format, specific to security patterns, extends the typical pattern format with sections like Problem, Solution, Security Mechanism, Known Uses, Consequences and Potential Attacks. It focuses on how security concerns are addressed within a pattern.

Each format is tailored to fit the specific domain or design concern, ensuring clear communication of the pattern's intent, applicability and implications. In this paper, we use a variant of the security patterns format for the description of the (SARMOM) pattern and the Alexandrian pattern format for other related patterns.

2.4. Patterns for IoT

While many general patterns are applicable in IoT environments (for instance, encryption and key management patterns) new patterns specific to the IoT field are emerging to meet the specific challenges that arise due to its complex, distributed and heterogeneous nature. Design patterns have been proposed for the IoT, to address issues like scalability, interoperability, security and energy efficiency. The following survey papers summarize the advancements in design patterns for the IoT. Muccini et al. [24] presented a comprehensive collection of architectural patterns, focusing on scalability, energy efficiency and security. Washizaki et al. [25] included an extensive collection of 143 IoT architecture and design patterns for the IoT. On the specific aspect of security, Rajmohan et al. [26] reviewed common security patterns used to mitigate various IoT security threats.

3. The Pattern Ecosystem

SARMOM is a component of a larger ecosystem that is under development. However, there are some patterns that are already available and valuable for this context.

Figure 2 shows our Legacy-to-IoT secure ecosystem and includes some related patterns for generic secure IoT systems developed by other authors. For this diagram, we used UML and represent patterns as stereotyped collaborations. As the definition of collaboration in UML is "a society of classes, interfaces, and other elements that work together to provide some cooperative behavior that's bigger than the sum of all its parts", we believe that it is a perfect fit for representing a pattern using UML, and, therefore, we propose this representation to be adopted by the pattern community. Moreover, a collaboration is further specified using class and object diagrams, to show their structural aspects, and

using sequence and activity diagrams, to show their dynamic aspects. This is coherent with the way patterns have been described in the literature [8].

We note that the proposed UML-based representation does not alter the essence of the pattern diagrams used so far, but simply allows that representation to be integrated in UML, making it possible to use standard tools to create and manage such diagrams and to include other modeling aspects not directly visible in diagrams, thus becoming an integral part of the UML system documentation and enabling the possibility of automated processing of those diagrams.

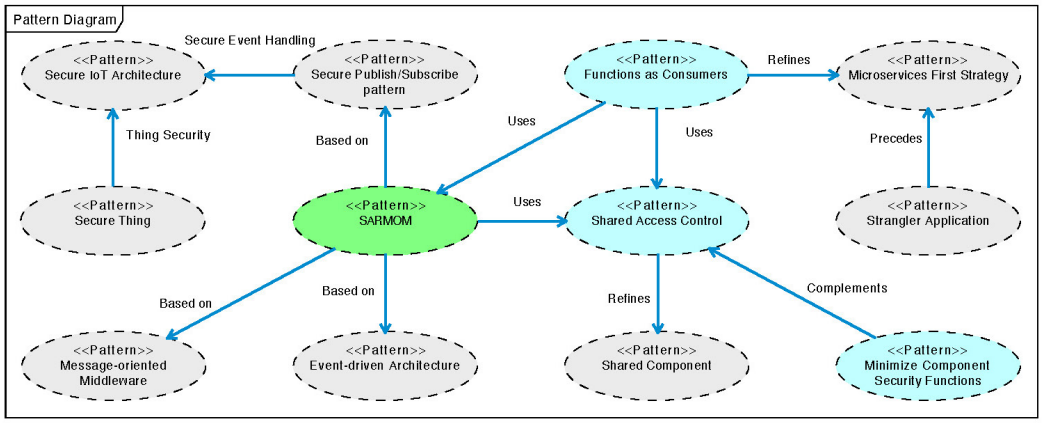

Figure 2. Pattern diagram relating the new pattern with other IoT-relevant patterns.

The patterns in blue in Figure 2 represent patterns that are part of our pattern language, with the SARMOM pattern highlighted in green. The rest of the patterns in gray are related patterns from other authors. The SARMOM pattern is based on Message-oriented Middleware [27], Event-Driven Architecture [6] and Secure Distributed Publish/Subscribe pattern for IoT [28], which, in turn, is related to Secure IoT Architecture and Secure IoT Thing [29]. SARMOM differs from Secure Distributed Publish/Subscribe pattern for IoT in several aspects. In particular, the former follows a strictly pull-based approach, while the latter can use a push-based strategy for messages, which was not convenient in our case. Additionally, SARMOM is tailored for auditability and includes mechanisms to ensure the immutability and ordering of the message chain. Moreover, Secure Distributed Publish/Subscribe pattern for IoT involves the use of Secure Channel, Authenticator, Signer, Encryptor, Decryptor and Signature Verifier patterns, which are not necessary in SARMOM. In particular, the authentication and access control functions are externalized in SARMOM. We rely on the Shared Access Control pattern, which, in turn, is a specialization of Shared Component and complements Minimize Component Security Functions, by outsourcing the access control functionalities, thus minimizing the security functions that are incorporated in SARMOM.

For the sake of completion, the pattern landscape (and the corresponding diagram) includes several patterns that we adopted in conjunction with the Secure Auditable Registry in the process of re-engineering the SUD to become an IoT SUD. These patterns are relevant in the process of implementing the SARMOM pattern. We first adopted Implement Monolith First [6] (not shown in the diagram because we do not consider it necessary in all our target systems), in order to provide maximum value for the clients as an MVP (Minimum Viable Product). We then proceeded to adopt the Strangler Application pattern [6] by means of Proxy Monolith Components to Microservices [6], and then defined a variant for MOM-based SUDs that we call Functions as Consumers, which is meant to be used when implementing SUDs that adopt SARMOM.

In order to facilitate the understanding of some of the patterns that we are still developing, and which are, therefore, not yet available in the literature, we provide brief descriptions for the Shared Access Control, Minimize Component Security Functions

and Functions as Consumers patterns. Other solutions that could become patterns in the pattern language are still being revised, and it is not yet decided whether they will be developed as patterns.

3.1. Shared Access Control

An SUD involving different hardware devices is transitioning from monolith to IoT using a microservice architecture. Access to the new services must be controlled, but we need to make sure we provide consistent access control policies. At the same time, we want to avoid data duplication and to reduce the management overhead that would result from using independent access control mechanisms in each service.

Therefore,

we factor the access control functionalities to a common set of services and components that serve the remaining services.

Related patterns:

Shared Component [30], Sidecar [8], XACML Architecture [31], Role-Based Access Control (RBAC) [28], Access Token [32].

3.2. Minimize Component Security Functions

Each component or service in a distributed system has different security requirements, which means different security functions. We want to minimize system redundancy, avoid inconsistent treatment of security, reduce security management overhead and reduce attack vectors. The size of a component is strongly correlated with its security risks. Reducing the component size is good for understandability (which, in turn, improves security). Increasing security by means of partitioning code is a proven and mature strategy [33,34].

Therefore,

we identify security functions that can be externalized to a shared component and we move them there. Additionally, we use "by-design" approaches to reduce the number of security functions required by a service. Finally, we reduce the attack surface of each service by dividing functionalities into separate components, to avoid a service having to implement several security-critical functions.

Related patterns:

Minimize The Functionality of A Component [35], Minimization Principle [36], Shared Component [30].

3.3. Functions as Consumers

We are in the process of transitioning our SUD from a monolith to a microservice event-driven architecture. We want to avoid situations in which new functions continue to be added to the monolith because, on the one hand, we want to avoid duplication of work (those functions will eventually have to be extracted to microservices in the future), and, on the other hand, when implemented in the monolith these functions will become affected by the level of coupling of the monolith and will, therefore, introduce changes in other functions of the monolith that may have already been implemented as microservices, and their understanding and subsequent extraction will become more difficult and risky.

Therefore,

we require that all new functions are specified as consumers of the MOM (i.e., specifying the function as a consumer that retrieves its input data from the MOM) to facilitate and encourage developers to implement them as loosely coupled services fed by information from the MOM.

Related patterns:

Microservices First [6], Replace as Microservice [6].

4. The Secure Auditable Registry Based On Message-Oriented Middleware (SARMOM) Pattern

This section presents the SARMOM pattern using a variant of the security pattern format used in [8]. The reader is referred to [8], in order to obtain a description of each of the pattern sections.

4.1. Intent

Build your system around a Secure Auditable Registry if your system must maintain a record of its operations and must also prove compliance to an external auditor. Complement it by using a Message-Oriented Middleware as the system hub connecting all components, so that all events/messages are consistently treated and can be accessed by the registry.

4.2. Context

The different components of a distributed mission-critical IoT application need to exchange large amounts of information, but we want to maintain these components loosely coupled. The application information can be represented as messages or events. Most information is produced by a specific type of component (i.e., *Things*) and then consumed by multiple components that provide different functionalities. The application operates in a regulated environment and needs to provide access to external auditors, to prove compliance with the applicable regulations.

4.3. Problem

An IoT application is used to obtain event information from different things or machines (e.g., devices and sensors). The amount of information generated is enormous and the rate of production is also extremely fast. The generated information is diverse and is used for many different purposes. Part of the information must be protected, to preserve privacy, to protect business intelligence or because it is considered confidential information, but at the same time it must also serve as proof of the integrity of the system and its information for external auditors.

To represent the problem and the target SUDs, we investigated other systems and domains that could potentially benefit from adopting this pattern. One of those is part of a collaboration we have with an autonomous drone company. Autonomous drones are used for many different tasks, but because they are not controlled by humans their actions must be auditable in every moment. The gambling system and the drone system have very little in common at the functional level, and while there are similarities at the level of components and architecture we found that the main reason for the use of the pattern is not related to the components or the architecture but to the goals and restrictions of the system. For this reason, in order to represent the target SUDs that could benefit from using SARMOM, we concluded that we had to complement the normal UML (class, components or architecture) diagrams showing the architecture of the target systems by further specifying the SUD goals. This was achieved by means of the Secure Tropos Goal Analysis model [37]. Secure Tropos is an independent formalism that is not part of the standard UML. Consequently, to be able to integrate the Secure Tropos model into our modeling environment we developed a specific UML profile to represent Secure Tropos in UML. This profile has been made available as an Open Source artifact.

Figure 3 shows a simplified and generalized architecture of target SUDs before the application of the SARMOM pattern. The depicted architecture is a generalization of several systems that are candidates for the application of the SARMOM pattern. In particular, we have considered scenarios for gambling, autonomous drones and a trusted

mesh communication network for military vehicles. Please note that the dashed lines represent dependencies and do not imply any direction in the communication.

In all these systems, there are "Machines" (i.e., gaming machines, drones or military vehicles) that generate the events that are central to the system operation. Each machine has a controller component that is used to integrate it with the SUD. Machines may send events to a central component (Event Collector) that has a central event store or to specific modules controlling different functions of the SUD, which may also have local event stores. An SUD Management module also exists, to coordinate the rest of the SUD.

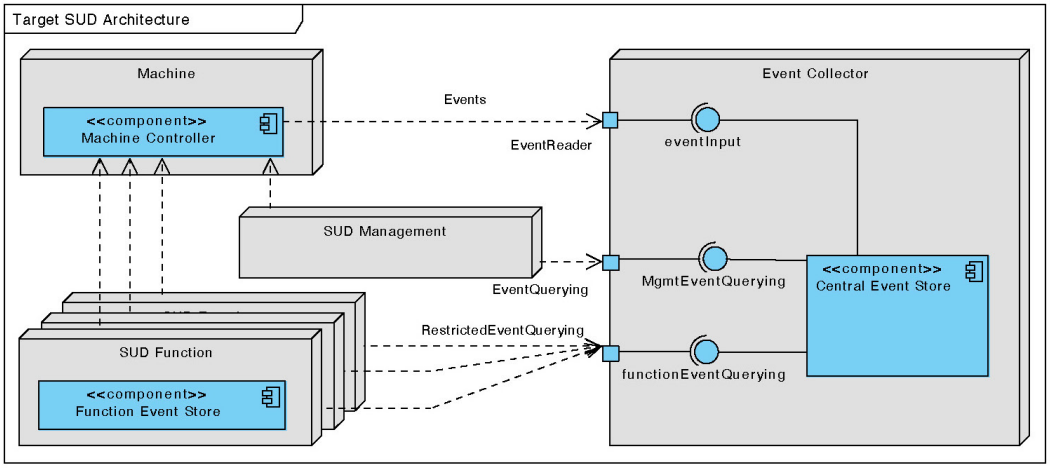

Figure 3. Architecture of SUDs that may use the SARMOM pattern.

While the previous figure may allow the readers to determine whether their SUD can benefit from the application of the SARMOM pattern, we feel it misses important information about the goals of the system if it is taken as the only source for identifying target SUDs for the SARMOM pattern.

To complement the previous architecture diagram, in Figure 4 we present the Secure Tropos Goal Analysis diagram highlighting goals of legitimate and malicious agents and showing how the proposed pattern can help mitigate most of the threats (represented with the «Threatens» relation stereotype) and contribute positively to legitimate goals for the SUD. The diagram uses a representation based on our own UML Profile (the ITIS-UMA-SwEngProfile), which extends UML with different security aspects: for instance, allowing us to represent Secure Tropos concepts and diagrams in UML. In this profile, the stereotype «Agent» is used to represent legitimate stakeholders of the SUD, while «Malicious Agent» denotes entities with illegitimate interests in the system. «Goal» and «Malicious Goal» are used as high-level objectives of agents and malicious agents, respectively. «Mitigation Measure» is a secondary objective, included in the SUD to counteract a «Malicious goal». Relations «+» and «−» indicate, respectively, a positive or negative contribution of one goal to another goal. The «Threatens» relation is used to indicate that one malicious goal is focused on preventing another legitimate goal, and the «mitigates» relation is used to indicate that one legitimate goal is focused on preventing another illegitimate goal. Finally, the stereotypes «include», and «extend» have the standard UML meaning.

The OT/IoT SUD agent represents the legitimate stakeholders of the SUD in general, who need to manage machines in their SUD, and who are also interested in goals such as obtaining SUD intelligence to maximize SUD efficiency and to optimize operations. Guaranteeing operations and the resilience of the SUD are also important goals that may be negatively affected by the marketing goal of lowering the cost of the system. Privacy is a goal of the SUD stakeholders but also of the end users. Another goal of the end users is to gain trust in the SUD, which is positively influenced by the goal of external auditors (representing authorities and regulatory boards) to audit the SUD. It is important to note that this legitimate goal of auditing the SUD can represent a threat to privacy. Likewise, it is important to note that some agents may act selfishly or dishonestly.

For instance, malicious end users may want to circumvent accountability mechanisms to avoid being held accountable for their actions.

The diagram shows how using SARMOM can contribute to mitigating most of the threats and can also contribute positively to achieving goals such as guaranteeing operations and proving correctness.

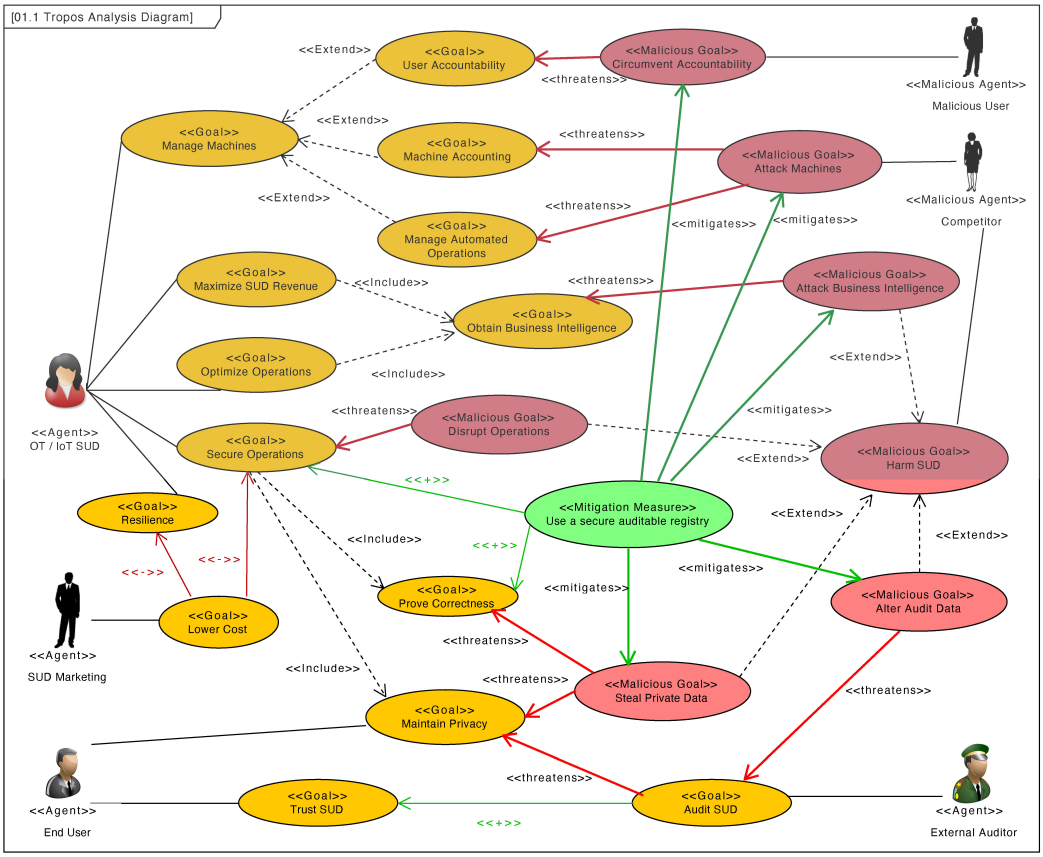

Figure 4. Secure Tropos diagram describing the goals of a typical SUD that may use the SARMOM pattern.

4.4. Solution

The solution consists in using a Secure Auditable Registry (SAR) service that is supported by Message-Oriented Middleware (MOM). The architecture of the system after applying the solution is depicted in Figure 5 below. Please note that the Machine Controller node will be replicated.

A Machine Controller node can manage one machine (i.e., an IoT device), but can also manage a group of machines. In our specific case study, we use a Machine Controller node for a group of slot machines called a "block" that can consist of up to 128 machines, but the pattern can be used with any arbitrary number of machines per controller. In fact, for other domains, such as autonomous drones, there is one Machine Controller per machine (drone).

SUD Client nodes represent the SUD microservices that need to access the data in the registry to provide their functions. Examples of these nodes are Reporting, Accounting, Business Analytics and Anomaly Detection. It is possible to have more than one Auditing Authority node.

The solution entails several steps that can be taken incrementally:

1. In the first place, a Secure Auditable Registry (SAR) service should be integrated with the SUD, using an independent component (e.g., microservice). This component implementing SAR must also implement security mechanisms to protect the integrity and the ordering of the records it contains. See implementation section for details.

2. Then, to free the SAR component from dealing with other requirements, such as concurrent access, scalability and resilience, the access to the SAR must be conveyed through an MOM service. In our case, the ideal solution is to use a pull-based MOM (such as Kafka) so that the registry can retrieve the messages at its own pace, knowing that there is no risk of messages being lost because of the lower performance of the SAR component. In this way, we externalize the management of topics and we have guarantees for the ordering of messages (at least, at the topic level). Additionally, dealing with issues like concurrency, high input throughput and differences in production and consumption speeds is also solved by the MOM. The MOM also helps with error recovery, since it can provide persistence of the records received.

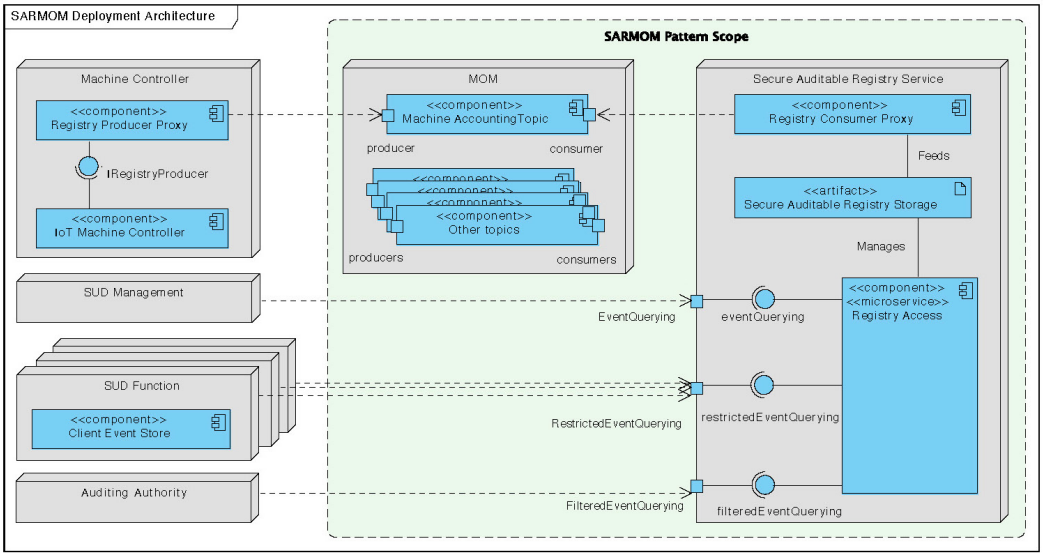

Figure 5. System architecture after applying the SARMOM pattern.

4.5. Implementation

As stated above, a proper context for the SARMOM pattern requires a Message-Oriented Middleware to decouple the secure registry from the target IT system. Even though there are several middleware solutions capable of event streaming, we will refer to Kafka for describing the most relevant implementation considerations that the pattern adopter must take into account. The conclusions in this section are also valid for other MOM implementations.

One key aspect is complexity of configuration and management. Since Kafka supports a distributed operation but only handles the actual data streams and connectivity to clients, a configuration manager component is required. Prior to version 2.8, the Kafka cluster required Zookeeper to manage its configuration. In April 2021, Kafka became self-sufficient, simplifying the architecture by consolidating all configuration management into Kafka itself.

Kafka also provides a set of key–value pairs in a properties file for configuration, which can be supplied either from a file or programmatically. There is a high number of supported properties, which are out of the scope of this work, allowing to configure any aspect or behavior of the middleware: brokers, topics, producers, consumers, connectivity, streams, etc.

The flexibility and versatility of Kafka makes it an excellent choice for many domains, and, consequently, there is a huge ecosystem around it. These properties make Kafka complex to manage (see Refs. [38,39] for details).

The security of the SAR component is also very dependent on the way this component is implemented. The main aspects to consider are the security of the hash chaining system and the security of the registry storage. The ideal implementation for this is to apply a

chained-hash approach [40], in which each record includes the secure hash of the previous record, similarly to what is done with blockchain technologies but without the need to have an infrastructure of peers, conflict resolution, distributed verification, etc. As already mentioned, additional security mechanisms (encryption, anonymization, pseudonymization, redundancy, etc.) can be adopted, depending on the specific requirements of the record contents. The service must also provide secure storage for the records.

The interfaces of the SAR component must be designed not to provide any modification or deletion functions. Only append and read interfaces must be provided. Error handling in this setting must be done by appending error-detection and eventually error-correction events. For instance, a slot machine may report wrong values for its meters (in the gambling domain, meters record things like the amount of money cashed in and out, the number of games played, etc.). When this is detected, the system is not allowed to change the records stored in the registry to reflect the correct values. Instead, new error-detection and error-correction records are inserted, to allow client services to reconstruct the correct state of the system. In this way, the authenticity of the events and the accuracy of the state of the system are guaranteed and can be proved.

Two main read interfaces must, at least, be included. One is a filtered querying interface (FilteredEventQuerying) specifically tailored for the auditing authorities to access (read) the registry contents that are relevant for their role while respecting the privacy requirements. The other one is an unfiltered querying interface, allowing unrestricted read access to the registry contents for the management of the system (named SUD Management in the figure). Please note that this is a conceptual definition of the interface, but, normally, the unfiltered querying interface will be implemented by means of smaller interfaces, which may be restricted for some management functions that do not require full access, following the Interface Segregation Principle. In Figure 5, we have illustrated this idea by including two interfaces, an unrestricted interface (EventQuerying) for the SUD Management, and a restricted interface (RestrictedEventQuerying) that supports only the necessary access for the different SUD Functions. This interface can be further divided when necessary.

Having different SUD Functions with different capabilities may require setting up access control mechanisms, at least for these interfaces, ideally using an external access control service (hence the suggestion to use a shared access control pattern). This access control component is not shown in the diagram because it is not strictly necessary for SARMOM, because its implementation is very dependent on different criteria and because it is covered by external patterns. The materialization of the Secure Auditable Registry (SAR) component as a service allows the target SUD to reduce coupling and facilitates the development of new functions, as they all will use the registry as the main source of data.

4.6. Known Uses

Currently, there are many systems in a variety of industries using Message-Oriented Middleware, such as Kafka, RabbitMQ and Amazon SQS. Many of these systems must also support auditing processes, which means they also need to use different forms of secure registries, a combination that fits the SARMOM pattern proposed in this work.

Below, we list some known uses of systems that combine the use of a Message-Oriented Middleware and a Secure Registry [40]:

- Netflix deals with different IoT devices (ranging from set-top boxes to smart TVs) and uses an MOM system (to convey its real-time monitoring and to support its event-processing pipeline) and a secure registry that is audited by independent processes implemented using HIVE [41].
- Uber deals with car-tracking devices and end-users' devices, and it uses MOM as part of the overall infrastructure stack powering various online and near-real-time use cases. Moreover, Uber uses the distributed cluster of Kafka nodes as a registry when performing internal audits and investigations [42]. While the security needs that arise from the requirement to prove the event sequence to external entities are not

all present there, the internal auditors still need the most important ones that we have identified for the SAR component, referenced in the implementation section.
- Several banks (e.g., Barclays and Rabobank) connect to devices such as ATMs and use both an MOM and a registry that provides the event recording and the audit functions of our SAR to support several financial processes and services, including those for proving compliance and for fraud tracking. Their combined use also fits the present pattern.

4.7. Consequences

After applying the pattern, we can expect different consequences, both positive and negative. Among the positive consequences, we can highlight the fact that we can solve our original problem with additional qualities, such as scalability, performance, ease of evolution and management of a large number of concurrent producers. More specifically, the SAR is the core to achieving the requirements of secure system registry and auditability, while the MOM is the key to achieving the applicability of the concept of the SAR to our target scenarios. Other positive consequences of the application of the SARMOM pattern include the following:

- Enabling asynchronous messaging from producers to the registry through the MOM allows the system to accept high rates of input messages regardless of the consumption capability of the registry. It also improves integrity and fault tolerance.
- MOM is normally an externally developed system, which frequently has control over the policies for evolution of features and security patching.
- The use of the SAR as the source of information for other SUD functions makes the system less vulnerable to data-alteration attacks.
- Likewise, centralizing the sourcing of information in the registry facilitates the adoption of centralized and consistent access control for the different SUD functions (clients).
- Centralizing the sourcing of system events in the registry facilitates the adoption of AI-powered monitors for early detection and prevention of attacks.
- The architecture of the pattern fosters the decoupling of components, thus facilitating maintenance and evolution.
- The intermediation of the MOM makes the registry independent from the source data formats. In particular, most MOM implementations provide the means to add specific interfaces (connectors).

On the downside, we can highlight the following considerations:

- It becomes necessary to install, configure and manage a relatively complex infrastructure, such as the MOM.
- The fact that the MOM is normally an externally developed system reduces the control over the policies for evolution of features and security patching and creates a dependency that may affect our system in case the MOM is discontinued.
- Despite the security mechanisms adopted, the SAR becomes a single point of failure and a single point of attack.
- If not carefully designed, the centralization of the sourcing of information through the SAR may have a negative impact on the performance.

It is important to note that most of the negative considerations can be reduced or eliminated through careful design and integration strategies.

4.8. Threats

Apart from the specific threats described in the previous sections, we present a brief threat analysis in the domain of IoT systems, and, in particular, on their messaging systems. Security is always a major concern regarding distributed applications, and this is especially true for IoT systems. Due to their openness and having their components loosely coupled, IoT systems are continuously exposed to vulnerabilities and threats in different layers and scopes. For messaging-based IoT systems, threats cover aspects such as interceptions, mod-

ifications or even interruption of the services, but also time altering, evidence elimination, etc. The most common threat categories for these systems are the following:

- Unauthorized access: the first fear regarding the use of an IoT system (regardless of whether it is centered on a registry or not) is the unauthorized access to the stored data, allowing attackers to read confidential information, possibly also involving data tampering, corruption or destruction attacks. These attacks have evolved from simple data corruption or injection to become more sophisticated and difficult to detect, and for a system relying on a messaging or registering system the consequences can be fatal.
- Denial of Service (DoS): a frequent goal for attackers is to prevent clients from accessing the resources or services of the system. The most common way to achieve this is by flooding the network or the application servers with a huge amount of information and traffic. For instance, it may result in the registry being unable to write new information or access the information already stored.
- Integrity in multi-user access scenarios: most IoT systems operate in a multi-user environment—that is, a scenario involving several users accessing the same information and/or resources (typically a database or registry) simultaneously. Users can be humans, devices or other IT systems. Keeping data integrity in this context (not only resulting from malicious actions but also accidents and errors) is a key concern. In registry-based systems, integrity also affects the timing and ordering of events and messages.

4.9. Known attacks

The threats above are not just theoretical risks but have actually materialized as attacks on several IoT systems. A cyber attack targeting a critical infrastructure can be catastrophic or even fatal. This section mentions a few recent attacks that were related to the SARMOM pattern, to help the reader understand the pattern better [43,44].

- The Triton attack (2017) was an attack on industrial cyber security systems, in particular, on a Schneider's Triconex safety instrument that was injected with malicious code that could eventually led to the release of toxic gas, which would have been fatal to human life. SARMOM would have contributed to preventing this attack by means of the minimization of data exchange between components of the SUD.
- Data leakage from a North American casino (2017). In this case, attackers gained access to a casino system via a high-tech "smart" fish tank featuring internet connectivity for remote management (temperature, salinity and automated feedings). Although quickly detected, this attack succeeded at retrieving up to 10 gigabytes of data, including the casino's database of high rollers. SARMOM would have contributed to preventing this attack by means of the isolation and centralization of data exchange between components of the SUD.
- The Oldsmar cyberattack on the US water system (2021) targeted the water treatment system of Oldsmar, Florida, gaining access to the control process of the treatment system. The attackers were successful in increasing the sodium hydroxide content in the water systems to toxic levels. However, a human operator was alert and immediately reduced the toxin levels in the water system. SARMOM would have prevented this attack by means of the trusted event sourcing and by minimization of data exchange between the components of the SUD. Moreover, monitoring capabilities enabled by SARMOM would have made it possible to detect the problem and stop the attackers before they could do harm.
- In the Colonial Pipeline cyberattack (2021), the control system of a pipeline carrying jet fuel and gasoline in the US was attacked by ransomware, and the infrastructure was down for nearly six days before service restarted. Although this attack focused on the billing infrastructure of the company, it brought down the entire oil pipeline as they could not issue customers with appropriate bills, affecting the entire east coast of the US. SARMOM would have made this attack much less severe, by means of the

independence of the SUD components, which would have allowed it to contain the effects of the attack in the affected subsystem and would have allowed the system to continue in operation, even if producing bills was not possible, because consumption data would have remained safely stored in the registry for later processing.

5. Conclusions and Future Work

In this paper, we have presented a pattern for a Secure Auditable Registry based on Message-Oriented Middleware to be used in large-scale IoT systems that require audit from external entities. This pattern is part of a pattern language that we are developing for transitioning from legacy systems dealing with hardware devices to IoT. These patters have been applied in an industry-funded R&D project focused on the re-engineering of a legacy high-security system in the domain of gambling. We have worked on generalizing and describing the pattern, so that it is usable in other emerging IoT-enabled domains, such as autonomous vehicles, military operations, banking, healthcare, smart cities, transportation, etc.

Regarding our future work, we plan to work on extending the pattern ecosystem by completing the development of the patterns referred to in Section 1.2 and adding new patterns as they are identified in our current IoT projects. We also plan to prepare computer-oriented versions of these patterns based on COSSPs [45] and to use those to develop pattern-oriented extensions for system engineering tools, to facilitate the adoption of those patterns.

Author Contributions: Conceptualization, A.M.; Software, L.G.; Validation, F.J.J.; Investigation, A.M., F.J.J. and L.G.; Writing—original draft, A.M.; Writing—review & editing, F.J.J. and L.G.; Project administration, A.M.; Funding acquisition, A.M. All authors have read and agreed to the published version of the manuscript.

Funding: This research was partially funded by the Junta de Andalucía, Spain, under contract QUAL21 010UMA.

Data Availability Statement: The original contributions presented in the study are included in the article, further inquiries can be directed to the corresponding author.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Straits Research. Internet of Things (IoT) Market: Information by Component (Hardware, Connectivity, Services), End-User Vertical (Manufacturing, Transportation, Healthcare), and Region—Forecast Till 2030. 2022. Available online: https://straitsresearch.com/report/internet-of-things-market (accessed on 19 June 2024).
2. Al-Sarawi, S.; Anbar, M.; Abdullah, R.; Hawari, A.B.A. Internet of Things Market Analysis Forecasts, 2020–2030. In Proceedings of the 2020 Fourth World Conference on Smart Trends in Systems, Security and Sustainability (WorldS4), London, UK, 27–28 July 2020; pp. 449–453. [CrossRef]
3. Vailshery, L.S. *Number of Internet of Things (IoT) Connected Devices Worldwide from 2019 to 2021, with Forecasts from 2022 to 2030*; Statista: Hamburg, Germany, 2022. Available online: https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/ (accessed on 10 September 2024).
4. Priyadarshini, U. Real World IoT Applications in Different Domains. 2022. Available online: https://www.edureka.co/blog/iot-applications/ (accessed on 19 June 2024).
5. Cartwright, I.; Horn, R.; Lewis, J. Patterns of Legacy Displacement. 2022. Available online: https://martinfowler.com/articles/patterns-legacy-displacement/ (accessed on 19 June 2024).
6. Brown, K.; Woolf, B.; Groot, C.D.; Yoder, J. Cloud Adoption Patterns. 2021. Available online: https://kgb1001001.github.io/cloudadoptionpatterns/ (accessed on 19 June 2024).
7. Fernandez, E.B.; Washizaki, H.; Yoshioka, N.; Okubo, T. The design of secure IoT applications using patterns: State of the art and directions for research. *Internet Things* **2021**, *15*, 100408. [CrossRef]
8. Fernandez, E.B. *Security Patterns in Practice: Designing Secure Architectures Using Software Patterns*; Wiley Publishing: Hoboken, NJ, USA, 2013.
9. *ISO/IEC 25010:2011*; Systems and Software Engineering—Systems and Software Quality Requirements and Evaluation (SQuaRE)—System and Software Quality Model. ISO: Geneva, Switzerland, 2017. Available online: https://www.iso.org/standard/35733.html (accessed on 19 June 2024).

10. Anisetti, M.; Ardagna, C.; Damiani, E.; Maña-Gomez, A.; Spanoudakis, G. Towards transparent and trustworthy cloud. *IEEE Cloud Comput.* **2017**, *4*, 40–48. [CrossRef]
11. Gambling Act 2005. Latest Available Revised Version. 2005. Available online: https://www.legislation.gov.uk/ukpga/2005/19/contents (accessed on 19 June 2024).
12. Thales. IoT Cybersecurity: Regulating the Internet of Things. Thales Digital Identity and Security. 2024. Available online: https://www.thalesgroup.com/en/markets/digital-identity-and-security/iot/inspired/iot-regulations (accessed on 10 September 2024).
13. European Economic and Social Committee. Opinion of the European Economic and Social Committee on the 'Proposal for a Regulation of the European Parliament and of the Council on ENISA, the "EU Cybersecurity Agency", and Repealing Regulation (EU) No 526/2013, and on Information and Communication Technology Cybersecurity Certification ("Cybersecurity Act")'. Regulation (E.U.) 2019/881 of 17 April 2019 (COM(2017) 477 final/2 2017/0225 (COD)) OJ C, C/227, 28.06.2018. 2018; p. 86. Available online: https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=uriserv:OJ.C_.2018.227.01.0086.01.ENG (accessed on 10 September 2024).
14. European Commission; Directorate-General for Communications Networks, Content and Technology. Cyber Resilience Act—New EU Cybersecurity Rules Ensure More Secure Hardware and Software Products 2024. Available online: https://data.europa.eu/doi/10.2759/543836 (accessed on 10 September 2024).
15. European Parliament. Directive (EU) 2022/2555 of the European Parliament and of the Council of 14 December 2022 on Measures for a High Common Level of Cybersecurity Across the Union, Amending Regulation (EU) No 910/2014 and Directive (EU) 2018/1972, and Repealing Directive (EU) 2016/1148 (NIS 2 Directive). L 333/80 EN Official Journal of the European Union 27.12.2022. 2022. Available online: https://eur-lex.europa.eu/eli/dir/2022/2555/oj (accessed on 10 September 2024).
16. National Institute of Standards and Technology. *Recommended Criteria for Cybersecurity Labeling for Consumer Internet of Things (IoT) Products*; Nist Cybersecurity White Paper; National Institute of Standards and Technology: Gaithersburg, MD, USA, 2022. [CrossRef]
17. Liu, Y.; Zhang, L.J.; Xing, C. Review for Message-Oriented Middleware. In *Internet of Things—ICIOT 2020*; ICIOT 2020. Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2020; Volume 12405, pp. 12–23. [CrossRef]
18. Fu, G.; Zhang, Y.; Yu, G. A Fair Comparison of Message Queuing Systems. *IEEE Access* **2021**, *9*, 421–432. [CrossRef]
19. Rajasekaran, A.S.; Azees, M.; Al-Turjman, F. A comprehensive survey on blockchain technology. *Sustain. Energy Technol. Assess.* **2022**, *52*, 102039. [CrossRef]
20. Alexander, C. *A Pattern Language*; Oxford University Press: Oxford, UK, 1977.
21. Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J. Design patterns: Abstraction and reuse of object-oriented design. In Proceedings of the ECOOP'93—Object-Oriented Programming: 7th European Conference, Kaiserslautern, Germany, 26–30 July 1993; Proceedings 7; Springer: Berlin/Heidelberg, Germany, 1993; pp. 406–431.
22. Buschmann, F.; Henney, K.; Schimdt, D. *Pattern-Oriented Software Architecture: On Patterns And Pattern Language*; John Wiley & Sons: Hoboken, NJ, USA, 2007; Volume 5.
23. Schumacher, M.; Fernandez-Buglioni, E.; Hybertson, D.; Buschmann, F.; Sommerlad, P. *Security Patterns. Integrating Security and Systems Engineering*; Wiley: Hoboken, NJ, USA, 2006.
24. Muccini, H.; Moghaddam, M.T. IoT architectural styles: A systematic mapping study. In Proceedings of the Software Architecture: 12th European Conference on Software Architecture, ECSA 2018, Madrid, Spain, 24–28 September 2018; Proceedings 12; Springer: Berlin/Heidelberg, Germany, 2018; pp. 68–85.
25. Washizaki, H.; Ogata, S.; Hazeyama, A.; Okubo, T.; Fernandez, E.B.; Yoshioka, N. Landscape of Architecture and Design Patterns for IoT Systems. *IEEE Internet Things J.* **2020**, *7*, 10091–10101. [CrossRef]
26. Rajmohan, T.; Nguyen, P.H.; Ferry, N. A decade of research on patterns and architectures for IoT security. *Cybersecurity* **2022**, *5*, 2. [CrossRef]
27. Fehling, C.; Leymann, F.; Retter, R.; Schupeck, W.; Arbitter, P. *Cloud Computing Patterns*; Springer: Berlin/Heidelberg, Germany, 2014.
28. Fernandez, E.B.; Astudillo, H.; Orellana, C. A pattern for a Secure IoT Thing. In Proceedings of the 26th European Conference on Pattern Languages of Programs (EuroPLoP'21), Graz, Austria, 7–11 July 2021; Association for Computing Machinery: New York, NY, USA, 2022; pp. 1–6. [CrossRef]
29. Fernandez, E.B.; Uzunov, A.V. Secure middleware patterns. In Proceedings of the 4th International Symposium on Cyberspace Safety and Security (CSS 2012), Melbourne, Australia, 12–13 December 2012; pp. 470–482.
30. Steel, C.; Nagappan, R.; Lai, R. *Core Security Patterns: Best Practices and Strategies for J2EE, Web Services, and Identity Management*; Pearson: Hong Kong, China, 2005.
31. Fernandez, E.B.; Yoshioka, N.; Washizaki, H. Secure Distributed Publish/Subscribe (P/S) Pattern for IoT. In Proceedings of the AsianPLoP, Online, 12–16 October 2020.
32. Richardson, C. *Microservices Patterns With Examples in Java*; Manning Publications: New York, NY, USA, 2018.
33. Huang, Z.; Jaeger, T.; Tan, G. Fine-grained program partitioning for security. In Proceedings of the 14th European Workshop on Systems Security, Scotland, UK, 26 April 2021; pp. 21–26.
34. Maña, A.; Pimentel, E. An Efficient Software Protection Scheme. In Proceedings of the IFIP SEC'01, Paris, France, 11–13 June 2001; Kluwer Academic Publishers: Dordrecht, The Netherlands, 2001.
35. Wheeler, D.A. Secure Programming HOWTO. v3.72 Edition. 2015. Available online: https://dwheeler.com/secure-programs/Secure-Programs-HOWTO.pdf (accessed on 19 June 2024).

36. Jackson, C.; Russell, S.; Sons, S. The Information Security Practice Principles. Foundational Whitepaper. Version 0.9. Indiana University Center for Applied Cybersecurity Research. 2017. Available online: https://cacr.iu.edu/principles/ISPP-Foundational-Whitepaper-2017.pdf (accessed on 20 October 2024).
37. Paja, E.; Dalpiaz, F.; Giorgini, P. STS-tool: Security requirements engineering for socio-technical systems. In *Engineering Secure Future Internet Services and Systems: Current Research. Lecture Notes in Computer Science (LNCS)*; Springer: Berlin/Heidelberg, Germany, 2014; Volume 8431, pp. 65–96.
38. Kreps, J. Kafka Ecosystem. 2022. Available online: https://cwiki.apache.org/confluence/display/KAFKA/Ecosystem (accessed on 19 June 2024).
39. Powered by Kafka. Latest Revised Version. Available online: https://kafka.apache.org/powered-by (accessed on 19 June 2024).
40. Han, M.; Jiang, W. A Secure Communication Method Based on Message Hash Chain. *Appl. Sci.* **2022**, *12*, 4505. [CrossRef]
41. Confluent. Featuring Apache Kafka in the Netflix Studio and Finance World. 2020. Available online: https://www.confluent.io/blog/how-kafka-is-used-by-netflix/ (accessed on 19 June 2024).
42. Blog, U.E. Presto® on Apache Kafka® At Uber Scale. 2022. Available online: https://uber.com/en-ES/blog/presto-on-apache-kafka-at-uber-scale/ (accessed on 19 June 2024).
43. Jain, S.; Lakshmi, V.; Srivathsa, D.R. *IoT and OT Security Handbook*; Packt Publishing: Mumbai, India, 2023.
44. Lee, M. Criminals Hacked a Fish Tank to Steal Data from a Casino. Forbes Newsletters. 2017. Available online: https://www.forbes.com/sites/leemathews/2017/07/27/criminals-hacked-a-fish-tank-to-steal-data-from-a-casino/ (accessed on 19 June 2024).
45. Mana, A.; Fernandez, E.B.; Ruiz, J.F.; Rudolph, C. Towards Computer-oriented Security Patterns. In Proceedings of the 20th International Conference on Pattern Languages of Programs PLoP'13, Monticello, IL, USA, 23–26 October 2013.