Autonomic Security Management for IoT Smart Spaces

CHANGYUAN LIN, University of Alberta, Canada
HAMZEH KHAZAEI, York University, Canada
ANDREW WALENSTEIN and ANDREW MALTON, BlackBerry, Canada

Embedded sensors and smart devices have turned the environments around us into smart spaces that could automatically evolve, depending on the needs of users, and adapt to the new conditions. While smart spaces are beneficial and desired in many aspects, they could be compromised and expose privacy, security, or render the whole environment a hostile space in which regular tasks cannot be accomplished anymore. In fact, ensuring the security of smart spaces is a very challenging task due to the heterogeneity of devices, vast attack surface, and device resource limitations. The key objective of this study is to minimize the manual work in enforcing the security of smart spaces by leveraging the autonomic computing paradigm in the management of IoT environments. More specifically, we strive to build an autonomic manager that can monitor the smart space continuously, analyze the context, plan and execute countermeasures to maintain the desired level of security, and reduce liability and risks of security breaches. We follow the microservice architecture pattern and propose a generic ontology named Secure Smart Space Ontology (SSSO) for describing dynamic contextual information in security-enhanced smart spaces. Based on SSSO, we build an autonomic security manager with four layers that continuously monitors the managed spaces, analyzes contextual information and events, and automatically plans and implements adaptive security policies.

As the evaluation, focusing on a current BlackBerry customer problem, we deployed the proposed autonomic security manager to maintain the security of a smart conference room with 32 devices and 66 services. The high performance of the proposed solution was also evaluated on a large-scale deployment with over 1.8 million triples.

CCS Concepts: • **Security and privacy** → **Distributed systems security**; • **Applied computing** → *Enterprise computing;*

Additional Key Words and Phrases: Autonomic security management, IoT, ontology, smart spaces

ACM Reference format:
Changyuan Lin, Hamzeh Khazaei, Andrew Walenstein, and Andrew Malton. 2021. Autonomic Security Management for IoT Smart Spaces. *ACM Trans. Internet Things* 2, 4, Article 27 (August 2021), 20 pages.
https://doi.org/10.1145/3466696

Authors' addresses: C. Lin, University of Alberta, Department of Electrical & Computer Engineering, 116 St & 85 Ave, Edmonton, Alberta, Canada, T6G 2R3; email: changyua@ualberta.ca; H. Khazaei, York University, Department of Electrical Engineering & Computer Science, 4700 Keele St, North York, Ontario, Canada, M3J 1P3; email: hkh@yorku.ca; A. Walenstein and A. Malton, BlackBerry, 2200 University Ave E, Waterloo, Ontario, Canada, N2K 0A7; emails: {awalenstein, amalton}@blackberry.com.
Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2021 Copyright held by the owner/author(s). Publication rights licensed to ACM.
2577-6207/2021/08-ART27 $15.00
https://doi.org/10.1145/3466696

1 INTRODUCTION AND BACKGROUND

Coupled with the accelerating development of computing and telecommunication technology is the fact that more and more interconnected computing devices, or IoT devices, are utilized by enterprises to facilitate business growth. These devices can be mobile, such as smartphones and smart bands, or stationary, such as smart doors and smart boards. In any case, these devices are equipped with sensors, software, and micro-controllers, that use the underlying network to transfer collected data and control points [Minerva et al. 2015]. However, the extraordinary heterogeneity of devices and protocols of IoT devices presents several daunting challenges to developers and managers in a commercial scenario. They need to handle an incredible diversity of hardware, software, and protocols to enable interactions between different systems. When a company wants to deploy new devices, developers have to redevelop some software embedded in previous systems to ensure the compatibility with new devices, making the delivery time very long. They also need to cope with complex relationships developed among different devices and correctly map them. Security is always one of the essential concerns for enterprises. How to ensure the safety of connected devices used in critical businesses and avoid disclosure of sensitive information is the vital issue [Misra et al. 2017]. There is a tradeoff between convenience and security. IoT devices can enable automation and intelligent process in enterprises, such as automated door access control and private messaging service. However, IoT devices are also vulnerable to different types of attacks due to high exposure, limited computational resources, and low reliability. In fact, because of the high cost of development, deployment, and maintenance of IoT devices, plus security concerns, the widespread use of IoT solutions in enterprises is just the aspiration, not the reality [Lee and Lee 2015].

To tackle the crux of popularizing and applying IoT solutions in enterprises, IoT systems must be able to meet the enterprise-level security bar without incurring prohibitive costs of development, deployment, and management. Systems built on IoT devices must be reliable and scalable and have fine-grained security management components. Fundamental changes should be made in system architecture and security management policies at different levels, including hardware, network, and software to address business pain points. The following example highlights the security challenges in a smart space:

Let us consider a smart factory embedded with smart devices. There is a smart camera to monitor the workshop, a smart door lock for entrance control, and several sensors to collect machine process data. It is 1:00 a.m., and one of the sensors finds that the temperature of the fluid in one pipeline is abnormally high, and it sends the warning data to the production management system. The production management system automatically shuts down one of the machines in the workshop and sends an alert to a maintenance specialist. Private messaging service on his smartphone informs the maintenance specialist, then he immediately opens his laptop, logs in to the web interface provided by the production management system, and sees real-time process data. After locating the problem, he grants entry permissions to several on-site technicians and asks them to fix the machine. The smart door lock authenticates those technicians by their smart bands. The specialist helps them remotely with smart cameras. In minutes, the team solves the problem, and the machine is running again. This application scenario is very straightforward and comprehensible. However, handling security risks in this scenario is a nightmare for developers and factory managers. For example, there is a risk of an intruder accessing the workshop using a stolen smart band. Developers determine to use two-factor authentication, detect intrusion by identifying the mismatched face using the smart camera, and then take appropriate action. They must redevelop the software of smart cameras to be able to recognize those technicians' faces. If the workshop manager purchases new IoT devices used for authentication, then developers must re-design the whole authentication control loop and develop features to support new authentication methods.

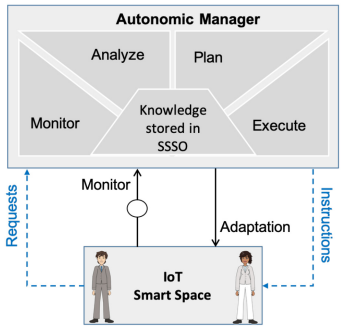


Fig. 1. Autonomic security management for IoT smart spaces.

What they do is only to tackle one possible security risk. If a sensor in the workshop malfunctions, if the smart door locker manufacturer adds fingerprint recognition, or if an interloper uses a hacked account to access machine process data, then developers have to repeatedly analyze the situation, develop new software, and deploy the new system for risk mitigation. Since such a procedure is mainly based on manual work, the possibility of human error is very high. Moreover, it is conceivable that there are many other security risks to be discovered by trusted developers and managers. That is to say, we can not always have on-site professionals monitoring the whole environment and finding all possible risks. Such a workflow is not suitable for IoT management in commercial scenarios. Hence, autonomic security management is indispensable for IoT systems, which could make IoT systems more secure, scalable, and reliable. The costs of developing, deploying, and managing such systems would also be considerably reduced.

The core of autonomic security management for IoT is that IoT systems can self-detect various types of vulnerabilities, automatically analyze situations, and autonomically learn and implement appropriate security policies. In this case, manual work will be minimized. Therefore, the chance of human failure is less, and security policies will be more concrete. Since IoT systems can be viewed as distributed computing systems, we can leverage the autonomic computing paradigm, introduced by IBM [Kephart and Chess 2003], to enable self-managed security by designing and implementing a **Monitor-Analyze-Plan-Execute-Knowledge (MAPE-k)** loop plus knowledge base, namely, the MAPE-k method. Hereafter, we use "MAPE-k engine" and "autonomic security manager" interchangeably. Figure 1 shows the high-level architecture of autonomic security management for IoT. In the following, we will describe this architecture in more detail.

Monitoring is the first step of the MAPE-k method, and the monitored objects include but are not limited to the working status of connected devices, changes in context, explicit user requests, and data streams. Obviously, the challenge here is to develop applications for heterogeneous IoT devices and collect data from them. Considering the IoT system as a microservice-based application composed of multiple microservices, we can adapt some technical patterns widely used in microservice-based web applications and use them in IoT systems, such as API, SDN, containers, access control, to enable high-level development and integrated security policies [Lu et al. 2017a]. Following the view of microservice-based IoT systems, we can assume the IoT system has the ability to sense changes and report anomalies through different microservices. We can also abstract the autonomic manager as a running microservice in the system. Once the anomaly or change is reported to the manager, it will automatically assess the security issue and plan how to mitigate against potential threats by either automatically adapting security policies and implementing them or asking people in the smart space to do explicit actions. All relevant data and actions are stored in the knowledge to help the system reason about threats and make appropriate responses to them. Microservice-based architecture can also offer more options for threat mitigation. For

example, when a sensor malfunctions, by unified service discovery, the IoT system can smoothly find the available alternative device that provides the same service.

In our work, we conducted a case study on a current BlackBerry customer problem that focused on a smart conference room equipped with IoT devices. We followed the microservice architecture pattern to assume that each IoT device provides certain services. In this way, the interactions between users and smart spaces could be viewed as dynamically enabling and disabling services provided by different devices. Due to the heterogeneity of context, we proposed a generic ontology named **Secure Smart Space Ontology (SSSO)** for describing dynamic contextual information in security-enhanced smart spaces. Based on SSSO, we also developed a MAPE-k engine that can monitor and analyze the context and plan and execute countermeasures to achieve autonomic security control in smart spaces.

As a summary, the main contributions of this article are three-fold: (1) We propose, to the best of our knowledge, the first generic ontology specifically for describing security-enhanced systems backed by connected devices leveraging the microservice architecture. The proposed ontology is service-oriented, security-enhanced, event-driven, and context-rich. (2) Based on the proposed ontology and MAPE-k method, we design an autonomic security manager that can maintain the security of smart spaces adaptively. (3) Given the proposed ontology and autonomic security manager, we model a smart conference room with 32 devices, 66 services, and 160 events, and solve a current customer problem of BlackBerry. The remainder of the article is organized as follows: In Section 2, we overview the related work. Section 3 presents SSSO in detail. Section 4 elaborates on the proposed autonomic security manager. We evaluate the proposed method in Section 5. In Section 6, we discuss the proposed solution. Section 7 concludes the article.

2 RELATED WORK

In this section, we investigate related work on microservice-based IoT, ontology-based smart home management.

2.1 Microservice-based IoT

Making improvements in IoT architecture is one of the essential ideas for easing management and enhancing security. Recently, several works focusing on microservice-based IoT architecture have emerged. Following the view of microservice-based IoT systems, different IoT devices can be viewed as independent microservice providers, and we can leverage some microservice patterns to manage the IoT system. Butzin et al. [2016] proposed a microservices approach for the IoT to demonstrate how operating-system-level virtualization and open service gateway could ease service deployment and improve scalability and testability. Lu et al. [2017a] proposed a secure microservice framework for IoT. They considered the IoT system as a service-oriented system of many microservices and adapt some technical patterns widely used in web-centric systems for IoT systems, such as API, SDN, containers, and access control, to enable high-level development and integrated security policies. Based on such an architecture, Lu et al. [2017b] extended their work and developed a prototype of autonomous vehicles management system, which could help several vehicles form a physically local chain, and maintain close proximity while traveling down a road. The system employed six trucks, and each truck was an independent IoT subsystem offering event services to other vehicles and connecting to others' services. While most of the papers in this area just cover some basic designs and ideas of an IoT system with microservice architecture, such an implementation illustrated the feasibility of implementing microservice-based IoT systems in the production environment. Sun et al. [2017] proposed an open IoT framework based on the microservice architecture, which has nine components responsible for different functions. Different from directly leveraging microservice patterns, they first analyzed the possible functions required and

provided by IoT devices and extracted nine general components. Additionally, they also considered artificial intelligence, big data, and tenant services of IoT systems in the framework design.

2.2 Ontology-based Smart Home Management

An ontology is defined as a collection of high-level primitives that capture and model a knowledge domain [Liu and Özsu 2009]. The ontology usually adheres to the **Resource Description Framework (RDF)** data model [Lassila and Swick 1999], which utilizes a human-and machine-readable graph, expressed as a collection of triples, to represent the knowledge. In recent years, several works aiming at using ontology to ease the management of smart spaces have been published. Ontology has been proven to be an effective solution to tackle the heterogeneity and enable interoperability in IoT systems [Nagowah et al. 2018; Tao et al. 2018]. However, most of the ontologies proposed in this domain are focusing on either the context of smart space or human actions in the smart space. Only a very small proportion of work studied security management in smart spaces. These existing ontologies also failed to follow the changes in the IoT architecture, which is more service-oriented nowadays.

Latfi et al. [2007] proposed an ontology to describe the telehealth smart home. Chen and Nugent [2009] designed an ontology-based activity recognition technique in the context of assisted living within smart home environments. Evesti et al. [2011] proposed an information security ontology and implemented it into security measures of the password. It could be used for adaptive user authentication where the system is able to dynamically modify user authentication, depending on the monitoring authentication related measures [Evesti et al. 2013]. Borgo et al. [2015] developed an ontology for collaborative robots in the manufacturing domain, enabling the reconfigurable transportation system to adapt control loops based on context knowledge. Seydoux et al. [2016] proposed the IoT-O ontology aiming at tackling interoperability issues in the smart home scenario. Khan and Ndubuaku [2018] proposed a context-based security guideline ontology for describing vulnerabilities in smart homes. Since their proposed ontology could not describe services and relevant context, it is not suitable for the resource description of systems backed by the microservice architecture.

Korzun et al. [2013a] proposed the Smart-M3 platform with three key properties, namely, multi-device, multi-vendor, and multi-domain described by Balandin and Waris [2009]. The proposed platform have two types of components: The **knowledge processors (KPs)** representing information producers and consumers, such as devices and users, and **semantic information brokers (SIBs)** to handle interactions among knowledge processors. The Smart Space Access Protocol was implemented to handle communications between SIB and KPs for interoperability. Based on the Smart-M3 platform, many use cases, including smart home, smart city, and healthcare systems, were developed [Catania and Ventura 2014; Korzun et al. 2013b, 2015]. In our work, while we solve some similar problems, including security and ontology-based reasoning, the underlying nature and method are different. We follow the notion of the microservice-based and event-driven architecture where a central message broker is not required, and the interoperability can be well maintained through the replaceable request model. Besides, they did not give a solution to autonomic security management. Korzun et al. [2014] proposed the SmartRoom system and introduced the service and user profile ontologies. Although it is not directly comparable due to the different underlying IoT architecture and focuses on security, our proposed ontology also has *Service* and *User* classes. While several works focusing on IoT systems with the microservice architecture have emerged [Moeini et al. 2017, 2019; Tao et al. 2018], they focused more on the resource description at a different granularity level, such as smart city, and did not consider the security management either.

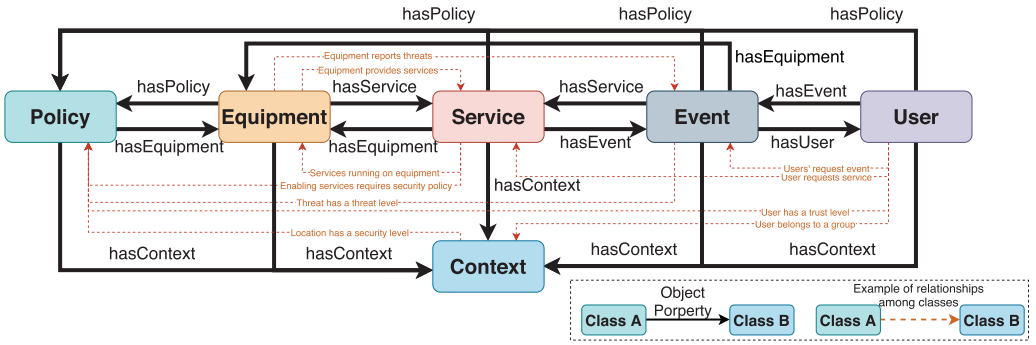


Fig. 2. Overview of Smart Secure Space Ontology (SSSO). Each box is a top-level class in SSSO. The solid lines represent object properties. Examples of relationships among classes are described by dotted lines.

The existent ontologies fail to describe services, devices, policies, events, and contexts in a security-enhanced smart space backed by the IoT system with the microservice architecture. Besides, an effective solution is required to integrate with the ontology and manage the security of the smart space while minimizing manual work. We fill both gaps by proposing and implementing the Secure Smart Space Ontology (SSSO) and autonomic security manager in this article.

3 SECURE SMART SPACE ONTOLOGY

3.1 Ontology Design

The ontology usually adheres to the RDF data model expressed as a collection of triples [Lassila and Swick 1999]. The underlying structure of the RDF data model can be abstracted as a graph that represents the entities and relationships developed among them. **Web Ontology Language (OWL)**, proposed by **World Wide Web Consortium (W3C)**, is an extension of RDF. Additionally, OWL provides a collection of standard relationships used for RDF [McGuinness et al. 2004]. The three components of the OWL ontology are *Classes*, *Properties*, and *Individuals*. Classes provide a basic abstraction of common concepts of things and group things with similar features together. Properties, as the name suggests, describe the relationship between two entities. There are two types of properties: object property, used for linking two non-data-value entities, and data property, used for linking a non-data-value entity to a data value entity. In our case, we used object properties to describe relationships among IoT devices, and used data properties to describe the points of devices/services (e.g., sensor points, communication endpoints) and detailed information of individuals described by the ontology (e.g., name, description, metadata in JSON). Individuals are class members (like instances in object-oriented programming), and an individual can belong to multiple classes. The statement to describe the relationship between two individuals is usually written as "*individualA property individualB*," and can be easily converted into the RDF triple like (individualA, property, individualB).

Certainly, we adhere to the practice of OWL to design the **Secure Smart Space Ontology (SSSO)** shown in Figure 2. We use the Protege OWL tool [Knublauch et al. 2004] to design and validate the ontology that we propose in this article. The SSSO consists of five top-level classes, namely, *Service Class*, *Equipment Class*, *User Class*, *Policy Class*, and *Context Class*. Figure 3 depicts the class hierarchy of SSSO. Now, we discuss each class in more detail.

Service Class. We followed the service-oriented approach and microservice-based IoT architecture [Lu et al. 2017a] where each connected device provides certain services encapsulated as microservices. For example, the smart speaker provides *Play_Video* service (play audio from an external or internal audio source), *Record_Audio* service (record the voice and save the audio file

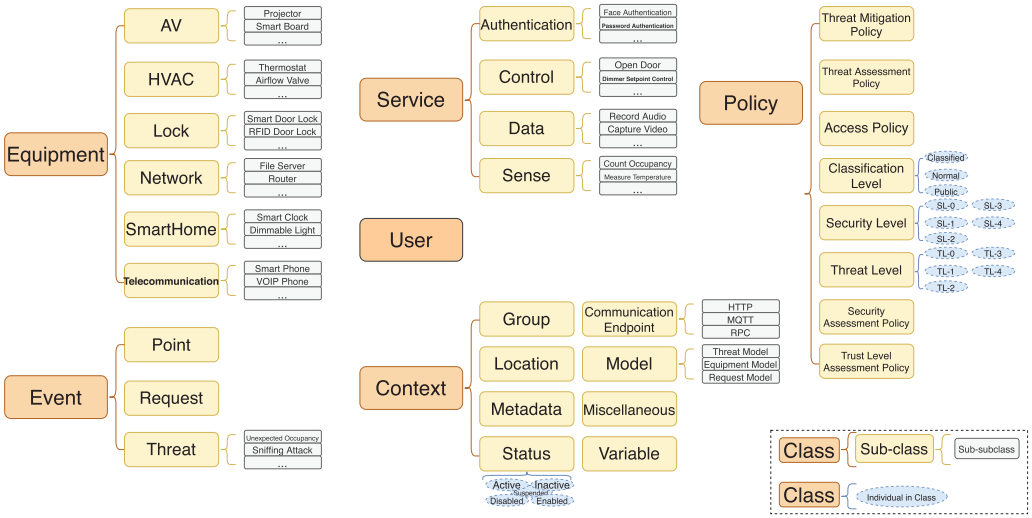


Fig. 3. The class hierarchy of SSSO.

to a local or remote location), and *Voice_Authentication* service (recognize identity by voice). IoT devices in the smart space can communicate with other components (e.g., other smart devices, control center, and MAPE-k engine) in the space through the publish-subscribe messaging service (e.g., MQTT), application programming interface (e.g., HTTP(s) API request), or **remote procedure call (RPC)**. The interactions between users and smart spaces can be viewed as dynamically enabling and disabling services provided by different devices. For instance, if a meeting attendee wants to use the smart speaker to record the meeting, then she can send a request from any available endpoints (e.g., recognized cell phone, control center, the smart speaker itself) to enable the *Record_Audio* service on the smart speaker.

Based on BlackBerry's customer needs, we comprehensively analyze microservices provided by various types of IoT devices deployed in security-critical businesses. By considering the potential risks of services and connections between services and critical data, we categorize services into four sub-classes, namely, *Authentication*, *Control*, *Data*, and *Sense*. *Authentication Class* contains security-critical services used for authentication (e.g., voice authentication, password authentication). *Control Class* describes services used for one-time control. The controlled resources can be security-critical (e.g., open a door) and non-critical (e.g., change the brightness of a dimmable light). Services regarding continuous critical data collection and access, such as voice recording and video capturing, are classified as *Data Class*. *Sense Class* contains services that can provide critical or non-critical contextual information, such as occupancy count and environmental temperature. Such services are typically offered by sensors in the smart space. The four sub-classes mentioned above also have their own sub-classes, which refer to specific types of service.

Equipment Class. It belongs to devices under control in the smart space. Following the same method used for designing *Service Class*, we categorize devices into six sub-classes. *AV Class* contains equipment relevant to multi-media content, including audio and video. *HVAC Class* describes devices of HVAC systems. *Lock Class* consists of critical devices for physical access control, such as smart door locks and RFID key readers. *Network Class* contains equipment responsible for accessing the Internet or the Intranet or storing and distributing digital content, such as file server and router. *Telecommunication Class* describes telecommunication devices, such as VOIP phone and smartphone. *SmartHome Class* contains smart devices that do not fall into the above categories and do not provide any services or access data that are security-critical. Examples are some

non-critical and auxiliary devices, such as dimmable light and smart sweeper. Similarly, each sub-class in *Equipment Class* has its own sub-classes, which refer to specific types of equipment.

User Class. It consists of users in the smart space. Each user is an individual (class instance) of *User Class*.

Policy Class. It refers to the adaptive security-relevant policy implemented in the security-enhanced smart space. *Policy Class* has eight sub-classes. We define three classification levels in *Classification_Level Class*, namely, *Classified*, *Normal*, and *Public*, to describe the classification level of instances in ontology, such as locations, groups, and services. We define five levels of security, from SL-0 to SL-4, in *Security_Level Class* to measure the security of the current environment and the trust level of users. Similarly, five levels of threat, from TL-0 to TL-4, are defined in *Threat_Level Class* to indicate how a threat compromises the security of the current environment. Generally, to ensure security, enabling service and keeping it running require both the security level of the smart space and the trust level of the service requester are equal to or greater than a certain threshold, depending on services and context. A threat may degrade the security level of the environment, depending on its severity and context, and may have a countermeasure described in *Threat_Mitigation_Policy Class*.

For better adaptability, the required security level for enabling a service and the threat level of a threat can be dynamic, depending on the context of the smart space. Besides, system managers may implement access policies to achieve dynamic access control, such as role-based, group-based, and context-based access. Therefore, we define *Security_Assessment_Policy Class* to store security policies used to assess the required security level of services. Similarly, *Threat_Assessment_Policy Class* consists of policies to assess the severity of threats. User-defined access policies belong to *Access_Policy Class*. Details about security/trust/threat levels and adaptive policies are discussed further in Section 4.

Context Class. It consists of contextual information that can help describe individuals of other classes more comprehensively. It is extendable so ontology users can define any additional information of entities using this class. *Context Class* also acts as a knowledge base of the MAPE-k engine proposed in Section 4. Currently, there are eight sub-classes in *Context Class*. *Location Class* describes the location of individuals, such as the room to which a device/user belongs. *Group Class* groups individuals, such as user groups and equipment groups. *Metadata Class* is for storing metadata of individuals, such as the metadata of a service. We define five status indicators, *Active*, *Inactive*, *Suspended*, *Disabled*, and *Enabled* in *Status Class* to describe the status of individuals, such as the active status of a service. *Variable Class* is for defining environmental variables of the system built on the ontology. *Model Class* has *Threat_Model Class*, *Equipment_Model Class*, and *Request_Model Class* as its sub-classes, used for storing the template of threats, devices, and requests, which are discussed further in Section 4. *Communication_Endpoint Class* describes the communication endpoint of services. The systems/services can communicate with each other through protocol and endpoint address described by *Communication_Endpoint Class*. Based on communication protocols, *Communication_Endpoint Class* has three sub-classes, namely, *HTTP*, *MQTT*, and *RPC*. Individuals that do not fall into the above classes belong to *Miscellaneous Class*. Ontology users can define any additional information of entities in it.

Event Class. The event-driven architecture is an effective solution to the software system, which is loosely coupled and highly distributed [Michelson 2006]. As the microservice-based system typically has the same features, we introduce *Event Class* in ontology to describe events in the smart space. *Event Class* has three sub-classes. *Point Class* refers to events regarding the value or status change of monitored endpoints. *Request Class* describes users' requests. *Threat Class* refers to the threat reported by equipment or the anomaly in the space. Each sub-class in *Threat Class* represents a specific type of threat/anomaly.

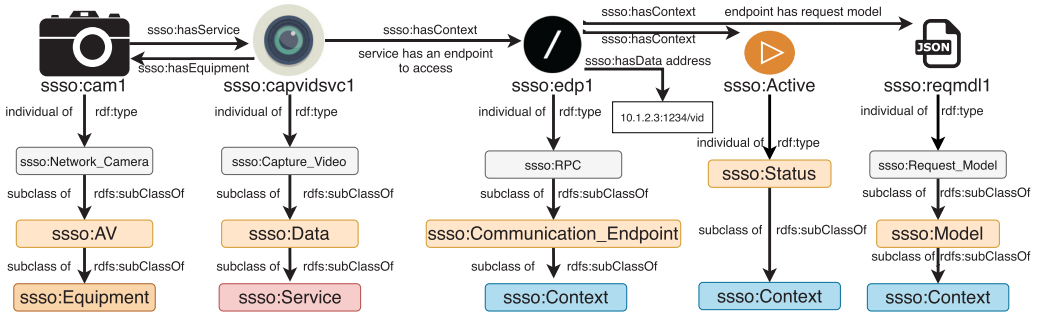


Fig. 4. Service-oriented: Each device provides specific services encapsulated as microservices. With the class hierarchy and object and data properties defined in SSSO, services in the smart space can be easily and comprehensively described.

Object and Data Properties. Properties that map the relationships developed among individuals and represent values are indispensable for describing resources, policies, and context in the smart space. For instance, the security team wants to enforce an access policy that only users belonging to the *Facility Manager* group can access services provided by HVAC devices deployed in the building *BLD-A*. Object properties are required to build connections among services, devices, users, policies, and context, including locations and groups. Data properties are also needed to provide the value of endpoints and the content of policies. Explicitly, we define the six object properties and six data properties in SSSO. Figure 2 illustrates relationships among classes and object properties defined.

For each top-level class in SSSO, we define an object property representing an individual having a relationship with an individual in that class. The six object properties are *hasService*, *hasEquipment*, *hasUser*, *hasPolicy*, *hasContext*, and *hasEvent*, respectively. For example, the statement that group *Facility Manager* has a user *Alice* can be written as "*Alice hasContext Facility_Manager*" or "*Facility_Manager hasUser Alice*." Besides, *hasName*, *hasDescription*, *hasMetadata*, *hasClass*, *hasValue*, and *hasData* are defined as six data properties. They are responsible for giving detailed information about an individual in terms of its name, description, metadata, relationship with a class, point value, and data content, respectively. For instance, an HTTP communication endpoint *edp1* with an address *http://10.1.2.3/svc/record* can be written as "*edp1 hasData "http://10.1.2.3/svc/record"^ rdfs:Literal*."

3.2 Features of Secure Smart Space Ontology

Overall, the proposed ontology has four features, namely, service-oriented, security-enhanced, event-driven, and context-rich. Now, we discuss each feature in more detail.

Service-oriented. We follow the ideas of microservice-based IoT architecture, where each device provides one or more services encapsulated as microservices. Each service has at least one communication endpoint with a specific protocol for service access. In the smart space-backed microservices, devices perform functions through services, users enable and disable specific services to make the best use of the space, services also detect and report status, events, and anomalies, and policies and preferences should be imposed on services to ensure security and achieve autonomic management. With the efforts considering the service-oriented system, in SSSO, *Service Class* is the core, and services in the smart space can be efficiently and comprehensively described through individuals in other classes and object and data properties. Figure 4 demonstrates the service-oriented feature and gives an example of a capture video service described by SSSO.

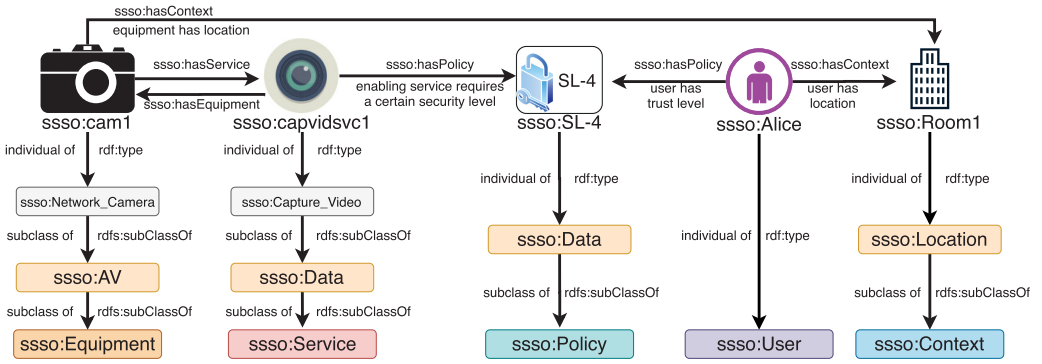


Fig. 5. Security-enhanced: Enabling and maintaining the capture video service requires both the security level of the space in which the service is located and the trust level of the requester reach SL-4.

As shown in Figure 4, the service named "capvidsvc1" is a capture video service provided by the network camera named "cam1." The service has an "Active' status, indicating that it is currently enabled and running. The service has a communication endpoint with an address specified by a data property for controlling the service through RPC. The communication endpoint also has a request model describing the metadata of the payload in the request. The system built on the ontology can follow the request model to interact with such a service. Therefore, the class hierarchy and object and data properties of SSSO can effectively describe the necessary information of services in smart spaces backed by devices with the microservice architecture. Figure 5 gives another example of describing details about the service regarding policies and more contextual information.

Security-enhanced. We introduce *Policy Class* and *Context Class* into SSSO for security enhancements. These two classes describe the access policy, security/trust/threat levels and assessment policies, threat mitigation policy, and contextual information regarding security. To prevent unauthorized activities, safeguard sensitive data, and take countermeasures against threats, these security-relevant attributes can be used to impose security control. Specifically, all activities in the smart space should conform to specific security policies and satisfy several conditions described by the ontology. For instance, enabling a service requires a certain security level and trust level. A threat has a certain threat level, which could degrade the current security level in the environment. Figure 5 demonstrates an example of using SSSO to describe the security policy regarding a capture video service hosted by a network camera. In this case Figure 5, enabling and maintaining the service *capvidsvc1* provided by *cam1* requires that both the security level of the environment *Room1* and the trust level of the service requester *Alice* satisfy the level of SL-4. In Section 4, we discuss the security-enhanced feature of SSSO and security policies in more detail.

Event-driven. The **event-driven architecture (EDA)** is an effective solution to the software system, which is loosely coupled and highly distributed [Michelson 2006]. As the microservice-based IoT system typically has the same features, EDA can also be applied to it. The event in EDA is a significant state change that the system should process and respond to. Similarly, in the microservice-based system, anything that happens in the system that changes the state or is going to change the state is the event. The event in EDA is a significant state change that the system should process, respond reasonably, and return the result. For instance, the user requesting to enable a service shown in Figure 5 is a request event. After receiving such a request event, the security manager built on SSSO adds such an event to *Request Class*, analyzes the situation, namely, checking if all required security policies are met in this case. If satisfied, then the system plans to enable the service and collect endpoint information described by SSSO. Finally, the system follows

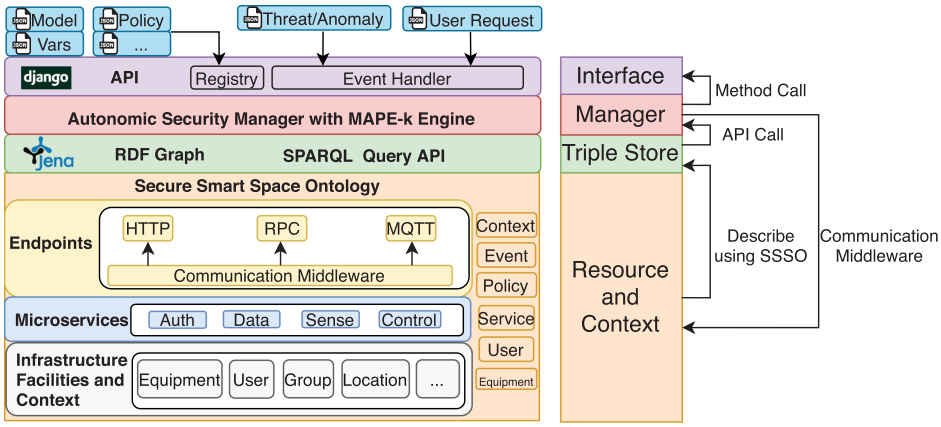


Fig. 6. Overview of the architecture of the proposed autonomic security manager. There are four layers in the system, namely, *Resource and Context*, *Triple Store*, *Manager*, and *Interface*, marked in different colors. Interactions between layers are shown on the right.

the address, protocol, request model to enable the service, and modify the status of the event and service described in SSSO. The request event triggers such a process, and enabling service is the result of the event. SSSO can well meet the needs of resources and information descriptions at all stages in the entire event processing logic.

Context-rich. SSSO explicitly has *Context Class* that describes contextual information that can help describe individuals of other classes more comprehensively. It is extendable so developers can define any additional information of entities using this class, thus making the ontology scalable and compatible with other smart space scenarios with different granularity levels, such as smart home and smart building. The rich context can also act as the knowledge base of the event processor, which would be discussed in Section 4.

4 AUTONOMIC SECURITY MANAGER

In this section, we follow the MAPE-k method and propose an autonomic security manager for IoT smart spaces built on top of resources described by SSSO. The manager maintains the security of smart spaces adaptively and can be encapsulated as a service running in the environment backed by microservice-based devices.

4.1 Overall Architecture

The overview of the architecture of the proposed autonomic security manager is shown in Figure 6. Now, we elaborate on four layers in the system. The first layer is *Resource and Context*, which represents the physical infrastructure, facilities, and context in smart spaces. Devices provide functionalities through services encapsulated as microservices hosted on them. Each service exposes a communication endpoint through which the service can be accessed, controlled, and configured. For time series data provided by services in *Sense Class*, the endpoints keep the records of their address and protocol for ease of retrieval by other components in the system. Using the SSSO, all resources and contextual information in this layer are converted into the RDF graph expressed as a collection of RDF triples.

Triple Store layer is responsible for storing those RDF triples generated in *Resource and Context* layer. We leverage Apache Jena to develop this layer. Apache Jena is an open-source semantic web framework with various components that can provide high-performance RDF storage and query

services [Jena 2007]. To fetch the knowledge expressed by the RDF data model, SPARQL was used to query the RDF graph [Prud et al. 2006]. Human-readable constraints and patterns of triples can be defined in SPARQL queries, and the RDF graph will be traversed to find the match. Hence, we deploy this layer as a container and exposes an API for querying the RDF graph using SPARQL.

Manager layer is where the MAPE-k method is implemented to achieve autonomic security management of resources in smart spaces. The autonomic security manager running in this layer monitors/receives events, analyzes situations, and autonomically learns and implements appropriate actions. During such a process, the manager queries/updates the resource and context RDF graph through the query API provided by the *Triple Store* layer. If needed, the manager can also utilize the communication middleware embedded in it and follow the address, protocol, and request model stored in RDF triples to communicate with endpoints. Details about the implemented MAPE-k method and examples of manager's interactions are discussed in Section 4.4. *Manager* layer is encapsulated as a Python package and is convenient for developing components upon it.

Interface layer is a Django-based API encapsulated as a containerized service built on the *Manager* layer. It is acting as an interface to communicate with the autonomic manager, responsible for exposing the methods provided by *Manager* layer in the form of API, and receiving requests and returning results in JSON. In this way, security management becomes a service in the smart space, and other resources in the space can interact with it as if they interact with other services. *Interface* layer has two main components, namely, registry and event handler, which are detailed in Section 4.2

4.2 Equipment Model and Device Registration

The registry in *Interface* layer is for registering new devices/services and modifying policies/context in the SSSO RDF graph. While adding a new device, either an equipment model that describes the metadata about the device or the UUID of the equipment model already stored in the RDF Graph is required. The equipment model contains the information including name, version, UUID, description, class the device belongs to, provided services, communication endpoints together with protocols, addresses, and request models, threats the device may report, and applicable security policies. Figure 7 illustrates the schema of the equipment model JSON format and gives an example of describing a type of smart board using the equipment model. As in practice, it is common to deploy a large number of devices of the same model, using the equipment model to describe devices can ease the workload of register and manage devices. Besides registering devices, the equipment model can provide important information to facilitate system development in the smart space.

When the event to register a new device received by the interface passes to the manager, the first step is to parse the equipment model from the input and store it in *Equipment_Model Class*. If the UUID of an existent equipment model is provided, then the system will retrieve the model from the RDF graph through the SPARQL query. The manager creates an individual in the equipment class specified by the model with a newly generated UUID as its IRI in the ontology. For future reference, the UUID of an individual is the same as its IRI, and we use them interchangeably in this article. Then, for each service defined in the equipment model, the manager inserts an individual in the corresponding service class and adds detailed information such as endpoints and provided contextual information in corresponding classes defined by SSSO. Object and data properties are used to link newly added individuals. The procedure is similar for adding information regarding threats specified in the model. Figure 4 and Figure 5 together give a simple example about individuals and properties added in the graph when a new device is registered. Following a similar idea, when the registry receives the event to register/modify policies, users, and any other

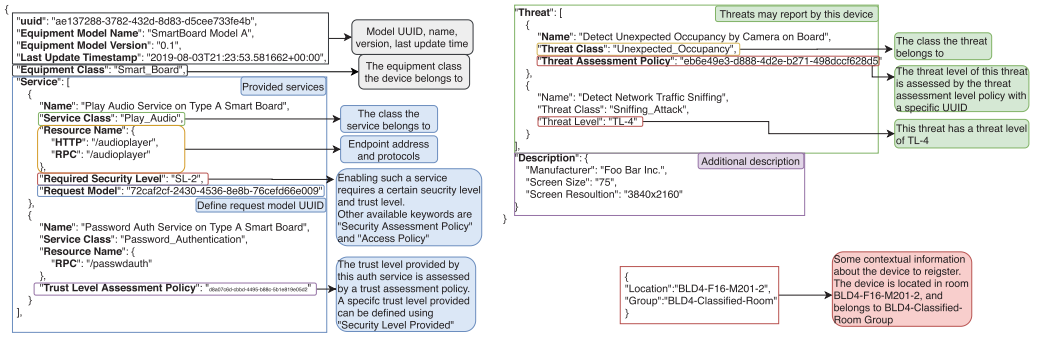
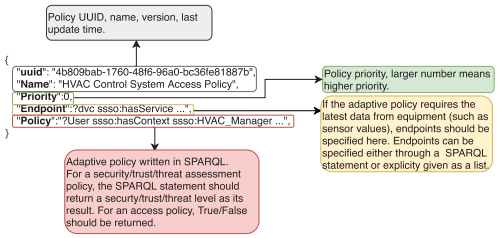


Fig. 7. An example of an equipment model of a smart board. Attributes in the red box are contextual information of a new device to register.



(a) The schema of adaptive policies

Type	Reserved Keywords	Evaluation Result
Security Assessment Policy	?Security_Level, ?User, ?Service	?Security_Level binds to a security level.
Trust Assessment Policy	?Security_Level, ?User, ?Service	?Trust_Level binds to a security level.
Threat Assessment Policy	?Threat_Level, ?Equipment, ?Threat_Class	?Threat_Level binds to a threat level.
Access Policy	?User, ?Service	True or False
Threat Mitigation Policy	?Suspend, ?Disable, ?Enable, ?Threat_Level, ?Equipment, ?Threat_Class, ?Env	?Suspend binds to services to suspend. ?Disable binds to services to disable. ?Enable binds to services to enable ?Threat_Level binds to a threat level
Endpoint SPARQL Statement	?Endpoint, ?User, ?Service, ?Equipment, ?Threat_Class	?Endpoint binds to endpoints

(b) Rules for writing adaptive policy statements

Fig. 8. (a) The schema of adaptive policies and its description. (b) Guidelines for writing adaptive policy statements. For each policy type, the table presents the reserved keywords in the statement, the variable to which the result binds to, and the expected result.

contextual information, the manager leverages the SPARQL query to add/delete/modify individuals and properties in the SSSO RDF graph.

4.3 Adaptive Security Policy

The security manager and SSSO support both deterministic and adaptive policies. As shown in Figure 3 and Figure 7, there are seven types of security policies applicable to services and threats in SSSO. In this section, we discuss each type of security policy and elaborate on the schema defining a security policy.

For safely accessing services, the most straightforward approach is to define a specific required security level using individuals in *Security_Level Class*. One example is the service demonstrated in Figure 5. However, a fixed security level may not be a good option, especially when the space manager desires to impose dynamic access control based on context. The same problem holds for the threat level of threats. We introduce adaptive policies in SSSO and the autonomic security manager to enable adaptability to various contexts. The adaptive policy is written in SPARQL, which can be evaluated by the manager and return a dynamic required security level/posed threat level, depending on the information stored in the SSSO RDF graph. Figure 8(a) depicts the schema of the adaptive policy. For the sake of performance, real-time time series data is not synchronously updated in the SSSO graph. As the adaptive policy may rely on the real-time data provided by services running on equipment (usually sensors), we introduce an optional attribute named "Endpoint" in the schema to let the manager first retrieve the latest data of the endpoint following the protocol, address, and request model stored in the SSSO graph, and use "hasValue" data property

to insert/refresh the data of the endpoint. The UUID (IRI) of endpoints to refresh can be specified by a SPARQL statement or given explicitly as a list. After refreshing the value of endpoints, the security manager evaluates the statement defined in the "Policy" attribute, obtains the dynamic result as security/trust/threat level, and continues to process the event.

The table shown in Figure 8(b) presents the rules for writing adaptive policy statements. The reserved keywords are the name of SPARQL variables that have pre-defined value or to which the result binds. Listing 1 and Listing 2 give examples of a security assessment policy and an access policy, respectively.

```
OPTIONAL {
    ?loc rdf:type ssso:Location.
    ?cl rdf:type ssso:Classification_Level.
    ?eq ssso:hasService ?Service.
    ?eq ssso:hasContext ?loc.
    ?loc ssso:hasPolicy ?cl.
    FILTER (?cl NOT IN (ssso:Classified)). }.
BIND (IF(BOUND(?cl), ssso:SL-1, ssso:SL-2) as ?Security_Level)
```

Listing 1. Example of a security assessment policy. The required security level for enabling services depends on the classification level of the room where the equipment is located. If the classification level is "Classified," then the security level of SL-2 is required. Otherwise, SL−1 is needed. *?Service* is a reserved variable that has the UUID of the requesting service as its pre-defined value. *?User* is pre-defined as the service requester UUID. The evaluation result binds to the reserved keyword *?Security_Level*.

```
?sl rdf:type ssso:Security_Level.
?eq ssso:hasService ?Service.
?eq rdf:type ?class.
?class rdfs:subClassOf* ssso:HVAC.
?User ssso:hasPolicy ?sl.
FILTER (?sl IN (ssso:SL-1, ssso:SL-2, ssso:SL-3, ssso:SL-4)).
?User ssso:hasContext ssso:HVAC_Manager.
```

Listing 2. Example of an access policy. Only authenticated users in the HVAC manager group can access services provided by HVAC devices. The evaluation result of the access policy is True/False, representing having access or not. No result bindings are needed.

4.4 MAPE-k Method for Autonomic Security Management

In this subsection, we discuss each phase of the MAPE-k method for autonomic security management in smart spaces.

***Monitor Phase*:** We follow the microservice architecture pattern and assume that the IoT system has the ability to sense changes through various microservices and report anomalies to the autonomic security manager through communication protocols. In this case, there is no need to consider the low-level monitoring part. We could view the manager as an event-driven engine, as what has been discussed in Section 3. Once the anomaly or status change is reported to the autonomic manager, the even processing loop will be triggered.

***Analyze Phase*:** Once the event is reported to the manager, the manager will analyze the event. We consider two types of events: the user's request and the threat. Now, we discuss the two cases separately.

When a user requests for enabling a service on a certain device, the endpoint will report such a request that contains the requested service UUID and requester UUID to the manager through the interface. After receiving the request, the manager first analyzes the situation to determine if the service can be enabled and running in a secure environment by querying the SSSO RDF

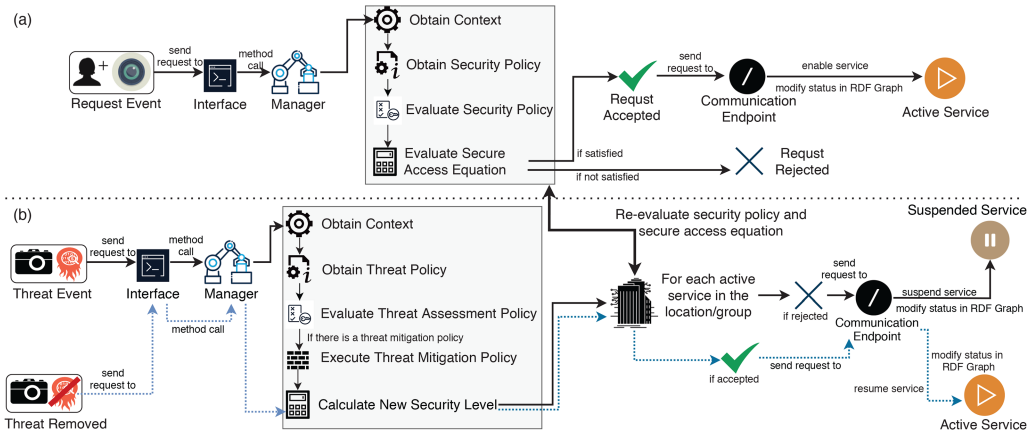


Fig. 9. (a) The process for handling a user request event. (b) The process with black arrows demonstrates the procedures in the event of a device reporting an active threat. The steps connected by blue dotted arrows depict the process when a device reports a threat is no longer active or has been resolved.

graph using SPARQL. As each step shown in the grey box in Figure 9(a), the analysis contains the four processes, which are detailed in the following: (1) Get the environmental context regarding the requested service. Precisely, the location or the group of the service, the classification level of the location or the group, and the trust level of the requester are acquired. (2) Obtain the security policy that applies to the service. If multiple security policies are applicable, then the manager chooses the one with the highest priority. If the service does not have any security policy applicable to it, then a default security assessment policy will be applied. For the sake of security, the default policy stipulates that services in *Data Class* and *Control Class* provided by equipment in *AV Class*, *Lock Class*, *Network Class*, and *Telecommunication Class* require a security level of SL-4; otherwise, a security level of SL-2 is required. (3) Evaluate the applicable adaptive security policy. For a security assessment policy, the evaluation result is the required security level. For an access policy, the request will be directly accepted if the evaluation returns True. If the service has a fixed required security level, then the manager skips this step. (4) Check whether the condition satisfies the secure access equation defined in Equation (1) and plan appropriate actions in the following phase, depending on the equation evaluation result. If the user authenticates through service in *Authentication Class*, then the process is similar, but the manager obtains and evaluates the corresponding trust assessment policy and gives the user a trust level by modifying the SSSO RDF graph through SPARQL.

In the event of a device reporting a threat, the manager generally goes through five steps, as shown in the grey box in Figure 9(b). (1) Get the environmental context regarding the threat. Specifically, the location or the group of the device, the current security level, the class to which the threat belongs, and the active threats and services in the location. (2) Obtain the threat policy that applies to the threat. Similarly, if multiple policies are available, then the manager chooses the one with the highest priority. If the threat does not have any policy applicable to it, then a default threat assessment policy will be applied. To ensure security, any threat reported by a device belonging to *AV Class*, *Lock Class*, *Network Class*, and *Telecommunication Class*, will pose a threat level of TL-4. Otherwise, the threat level of TL-2 will be applied to the threat. (3) Obtain the threat mitigation policy if applicable. If there is a threat mitigation policy, then the manager will suspend/disable/enable services as stipulated by the policy as countermeasures. (4) Evaluate the applicable adaptive threat policy. For a threat assessment policy, the evaluation result is the posed threat level. If

the threat has a fixed threat level or the mitigation policy already returns the threat level after applying countermeasures, then the manager skips this step. (5) Calculate the new security level of the environment(location or group) using Equation (2) and Equation (3). Then, the manager adds the threat to the RDF graph and re-evaluates all active services running in the environment by applying the same steps used in the analyzing phase for requests to enable them. All active services that fail to satisfy its security policy or the secure access equation will be suspended to avoid information disclosure and ensure security. If a device reports that a threat with a specific UUID is no longer active or has been resolved, then the manager will remove the threat, update the security level, and re-evaluate all suspended services in the environment following the same steps. If the security level has improved due to the removed threat, then the suspended threats may be resumed.

$$\begin{cases} \text{Trust Level} \geq \text{Security Level} \\ \text{Security Level} \geq \text{Required Security Level} - \text{CL Tuning Param} \end{cases} \tag{1}$$

where

$$\text{CL Tuning Param} = \begin{cases} 0, \text{if Classification Level is Classified} \\ 2, \text{if Classification Level is Normal} \\ 4, \text{if Classification Level is Public} \end{cases}$$

$$New\ Security\ Level = Current\ Security\ Level - max(Threat\ level\ of\ active\ threats) \tag{2}$$

besides, while calculating the new security level, the manager also follows

$$\begin{cases} \text{Four TL} - 1 \text{ threats in different classes are equivalent to one TL} - 2 \text{ threat} \\ \text{Three TL} - 2 \text{ threats in different classes are equivalent to one TL} - 3 \text{ threat} \\ \text{Two TL} - 3 \text{ threats in different classes are equivalent to one TL} - 4 \text{ threat} \\ New\ Security\ Level = 0, \text{if Current Security Level} - \max(\text{Threat level of active threats}) < 0 \end{cases} \tag{3}$$

***Plan Phase*:** During the plan phase, the manager plans appropriate actions to be executed for maintaining security in the execute phase, depending on the results of the analyze phase, and minimizes manual work. As shown in Figure 9, after analyzing the request and the situation, the manager needs to change the status of the service and update the SSSO RDF graph if the user's request is accepted. Therefore, the manager needs to retrieve relevant triples, including the status of the requested service, communication endpoint, protocol, and request model from the SSSO RDF graph, and prepare commands to enable the service through the request and update the graph for the execute phase. In the event of handling a new threat, if the security level has degraded, then the manager has to re-evaluate the security policy for each active service in the current environment. In case there are any services with an unmet condition, the manager plans to suspend these services that may indeed pose security risks. The manager has to evaluate relevant SPARQL queries to retrieve relevant triples from the graph and prepare commands to suspend services. Similarly, such plan phase procedures are also applicable to a threat removal event. In that case, the threat will be removed, and the manager plans to resume previously suspended service automatically if the security level has improved.

To further reduce manual work, the manager will reason about the root cause of the failed attempt to enable service and try to resolve it, in case it rejects a user's request due to any violated security policies or the unsatisfied secure access equation in the analyze phase. Suppose the trust level of the requester is under the threshold. In that case, the manager will calculate the additional trust level the user needs, obtain the context, find available authentication services in the environment that can provide such an amount of trust level through SPARQL queries, and prompt the user for performing authentication using a particular authentication service on a specific device. If there are multiple choices, then the manager will randomly choose one of them. For the sake of

safety, relying on a single authentication method is not recommended. Hence, the introduction of randomness could also reduce the likelihood of system breaching. During this process, the engine will also ignore the authentication services that have been used by this user to avoid duplicated authentication methods and ensure security. However, suppose the failed attempt is due to the insufficient security level of the space. The engine will try to execute any applicable countermeasures defined by threat mitigation policies to mitigate threats. If not applicable, then the manager will suggest the user disable the device that incurs threats. The security level of the space may improve, and consequently, the request may be accepted.

***Execute Phase*:** During the execute phase, the manager executes commands prepared in the plan phase. For the system with the microservice architecture, the execute phase can be simplified. We implement a communication middleware in the manager so it can send requests to controlled services and execute the planned actions following the communication protocol, endpoint address, and request payload obtained in the plan phase. Besides, the manager executes SPARQL queries and updates the SSSO RDF graph.

***Knowledge Base*:** The whole RDF graph based on SSSO acts as the knowledge base of the autonomic security manager. The RDF graph can be queried and updated using SPARQL through the triple store API, representing retrieving information and updating the knowledge base.

5 IMPLEMENTATION AND EVALUATION

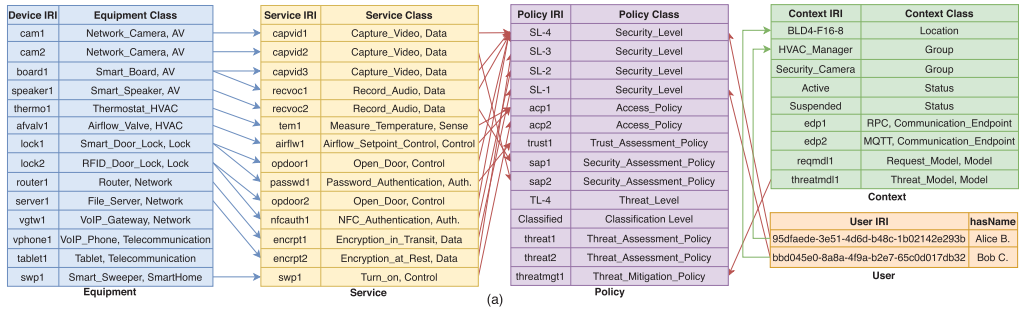
We use Python 3.8.2 to implement the autonomic security manager and the interface layer. For the triple store layer, we leverage Apache Jena with the Fuseki component. All three implemented layers are containerized using Docker and can be deployed as services in the microservice-based smart space. The containerized autonomic security manager can be scaled horizontally and vertically to meet the demands of large-scale deployments and serve a large volume of requests. As the evaluation, based on a current BlackBerry customer problem, we model a smart conference room with 32 devices, 66 services, 30 potential threats, and 28 adaptive policies, using SSSO, and deploy the implemented autonomic security manager. The partial overview of the modeled space is shown in Figure 10(a). All source code, ontologies, descriptions, and results are publicly available in the artifact repository.[1]

Based on BlackBerry and the customer's input, we design a series of 160 events, as shown in Figure 10(b). The series of events covers various event types. We validate the autonomic security management through such an event series. The autonomic security manager can adaptively respond to the events and maintain the security of the smart space through the MAPE-k method proposed in Section 4.4. The responses of the manager are partially presented in Figure 10(b). We also evaluate the performance of the proposed solution for managing security on a large-scale deployment with 20,000 devices, 180,000 services, 260,002 contextual entities, and in total, 1,860,701 triples. We deploy the system on a laptop with a 2.50 GHz Intel Core i5-2520M processor and 8 GB of memory without scaling any layer. Without considering the communication delay for sending requests to the endpoint, the autonomic security manager can respond to new device registration, request handling, and threat handling within two seconds, on average. This shows the applicability of the proposed system for a large-scale smart space. The manager is applicable to other smart space scenarios with different granularity levels, such as smart home and smart building.

6 DISCUSSION

We proposed the Secure Smart Space Ontology for describing resources and context in security-enhanced smart spaces. The ontology is service-oriented, security-enhanced, event-driven, and

[1] https://github.com/pacslab/SSSO-AutoSecMng.



Device IRI	Equipment Class
cam1	Network_Camera, AV
cam2	Network_Camera, AV
board1	Smart_Board, AV
speaker1	Smart_Speaker, AV
thermo1	Thermostat_HVAC
afvalv1	Airflow_Valve, HVAC
lock1	Smart_Door_Lock, Lock
lock2	RFID_Door_Lock, Lock
router1	Router, Network
server1	File_Server, Network
vgtw1	VoIP_Gateway, Network
vphone1	VoIP_Phone, Telecommunication
tablet1	Tablet, Telecommunication
swp1	Smart_Sweeper, SmartHome

Equipment

Service IRI	Service Class
capvid1	Capture_Video, Data
capvid2	Capture_Video, Data
capvid3	Capture_Video, Data
recvoc1	Record_Audio, Data
recvoc2	Record_Audio, Data
tem1	Measure_Temperature, Sense
airflw1	Airflow_Setpoint_Control, Control
opdoor1	Open_Door, Control
passwd1	Password_Authentication, Auth.
opdoor2	Open_Door, Control
nfcauth1	NFC_Authentication, Auth.
encrpt1	Encryption_in_Transit, Data
encrpt2	Encryption_at_Rest, Data
swp1	Turn_on, Control

Service

Policy IRI	Policy Class
SL-4	Security_Level
SL-3	Security_Level
SL-2	Security_Level
SL-1	Security_Level
acp1	Access_Policy
acp2	Access_Policy
trust1	Trust_Assessment_Policy
sap1	Security_Assessment_Policy
sap2	Security_Assessment_Policy
TL-4	Threat_Level
Classified	Classification Level
threat1	Threat_Assessment_Policy
threat2	Threat_Assessment_Policy
threatmgt1	Threat_Mitigation_Policy

Policy

Context IRI	Context Class
BLD4-F16-8	Location
HVAC_Manager	Group
Security_Camera	Group
Active	Status
Suspended	Status
edp1	RPC, Communication_Endpoint
edp2	MQTT, Communication_Endpoint
reqmdl1	Request_Model, Model
threatmdl1	Threat_Model, Model

Context

User IRI	hasName
95dfaede-3e51-4d6d-b48c-1b02142e293b	Alice B.
bbd045e0-8a8a-4f9a-b2e7-65c0d017db32	Bob C.

User

(a)

Event ID	Event Description	Event Type	Response
1	Register a new device "Smart Board".	Register Device	1 device, 9 services, 11 endpoints, 1 equipment model... added
2	Register a new device "Smart Speaker".	Register Device	1 device, 3 services, 3 endpoints, 1 equipment model... added
3	System administrator initializes a conference room and modifes the classification level of the room BLD4-F16-8 as "Classified".	Modify Policy	BLD4-F16-8 hasPolicy Classified, BLD4-F16-8 hasPolicy SL-4
4	Alice authenticates through the password authentication service porvided by the smart door lock.	User Request	Alice hasPolicy SL-2. Alice has the trust level of SL-2.
5	Alice requests to enable the capture video service provided by the smart board.	User Request	Request rejected as the trust level is insufficient. Prompt for Face Auth on Smart Board.
6	Alice authenticates through the face authentication service provided by the smart board.	User Request	Alice hasPolicy SL-4. Alice has the trust level of SL-4.
7	Alice requests to enable the capture video service provided by the smart board.	User Request	Request accepted. Enable the capture video service capvid1.
7	The network camera in BLD4-F16-8 reports the unexpected occupancy threat.	New Threat	BLD4-F16-8 hasPolicy SL-0. The security level degrades to SL-0. capvid1 service is suspended.
...	...	...	...

(b)

Fig. 10. (a) The partial overview of the modeled security-enhanced smart conference room. (b) The partial description of the series of 160 events. The full information, including individuals, properties, policy definitions, descriptions, and event responses, is available on our artifact page (footnote 1).

context-rich. It is extendable and adheres to the principles of the IoT system with microservice architecture. Based on SSSO, we proposed an autonomic security manager with the MAPE-k method, which can autonomically manage the security while minimizing manual work. Through the MAPE-k method, the manager is able to reason about the event happening in the smart space, analyze the context, and take appropriate measures. The manager can dynamically manipulate services in response to the event, including enabling, disabling, suspending, and resuming services, thus ensuring the security of the system. We proposed a case study on a smart conference room with 32 devices and 66 services and evaluated the proposed manager using a series of 160 events. The manager can adaptively respond to all events and autonomically manage the security of the space. We also assessed the performance of the system under a large-scale deployment with over 1.8 million triples.

Several limitations are identified: (1) We use Python to simulate behaviors of IoT devices, and we do not use the actual equipment to implement such an autonomic security management system. Since the microservice-based IoT device can be treated as a piece of software, we do not expect any major impact on our results. (2) We only use a series of events to evaluate our security solution and do not evaluate the solution in regular IoT systems. Using the IoT security testbed is an effective method for solution evaluations [Waraga et al. 2020]. However, there is no available IoT testbed suitable for microservice architecture systems. Implementing such a testbed will be our future work. (3) We design a default security assessment policy and a default threat assessment policy based on the relationships among services, devices, and critical data. Such default policies may not be the best practices in some scenarios in which the underlying system architecture, space granularity, or on-premise security guidelines differ considerably. In such a case, an on-site security professional may be advised to optimize default security and threat assessment policies.

7 CONCLUSION

In this article, we first proposed a Secure Smart Space Ontology (SSSO) to describe and abstract smart spaces. Such a formal description of the IoT environments facilitates analysis and reasoning

about the current state of the space in a machine-understandable fashion. We used SSSO as the knowledge base in the MAPE-k loop engine to achieve autonomic security management in IoT smart spaces. We implemented an autonomic security manager that has four layers with scalability. The autonomic manager could monitor and analyze events and context and plan and execute adaptive countermeasures with minimum human intervention at a large scale. Based on the current BlackBerry customer problem, we modeled a smart conference room and evaluated our work through a series of events. The performance of the proposed solution was also assessed through a large-scale deployment.

REFERENCES

Sergey Balandin and Heikki Waris. 2009. Key properties in the development of smart spaces. In *International Conference on Universal Access in Human-computer Interaction*. Springer, 3–12.

Stefano Borgo, Amedeo Cesta, Andrea Orlandini, and Alessandro Umbrico. 2015. An ontology-based domain representation for plan-based controllers in a reconfigurable manufacturing system. In *28th International Flairs Conference*.

Björn Butzin, Frank Golatowski, and Dirk Timmermann. 2016. Microservices approach for the internet of things. In *IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*. IEEE, 1–6.

Vincenzo Catania and Daniela Ventura. 2014. An approach for monitoring and smart planning of urban solid waste management using smart-M3 platform. In *15th Conference of Open Innovations Association FRUCT*. IEEE, 24–31.

Liming Chen and Chris Nugent. 2009. Ontology-based activity recognition in intelligent pervasive environments. *Int. J. Web Inf. Syst.* 5, 4 (2009), 410–430.

Antti Evesti, Reijo Savola, Eila Ovaska, and Jarkko Kuusijärvi. 2011. The design, instantiation, and usage of information security measuring ontology. In *2nd International Conference on Models and Ontology-based Design of Protocols, Architectures and Services*. 1–9.

Antti Evesti, Jani Suomalainen, and Eila Ovaska. 2013. Architecture and knowledge-driven self-adaptive security in smart space. *Computers* 2, 1 (2013).

Apache Jena. 2007. Semantic web framework for Java. https://jena.apache.org/.

Jeffrey O. Kephart and David M. Chess. 2003. The vision of autonomic computing. *Computer* 1 (2003), 41–50.

Yasir Imtiaz Khan and Maryleen U. Ndubuaku. 2018. Ontology-based automation of security guidelines for smart homes. In *IEEE 4th World Forum on Internet of Things (WF-IoT)*. IEEE, 35–40.

Holger Knublauch, Ray W. Fergerson, Natalya F. Noy, and Mark A. Musen. 2004. The Protégé OWL plugin: An open development environment for semantic web applications. In *International Semantic Web Conference*. Springer, 229–243.

Dmitry Korzun, Ivan Galov, and Sergey Balandin. 2013b. Development of smart room services on top of Smart-M3. In *14th Conference of Open Innovation Association*. IEEE.

Dmitry Korzun, Ivan Galov, Alexey Kashevnik, and Sergey Balandin. 2014. Virtual shared workspace for smart spaces and M3-based case study. In *15th Conference of Open Innovations Association FRUCT*. IEEE, 60–68.

Dmitry G. Korzun, Sergey I. Balandin, and Andrei V. Gurtov. 2013a. Deployment of smart spaces in internet of things: Overview of the design challenges. In *Internet of Things, Smart Spaces, and Next Generation Networking*. Springer, 48–59.

Dmitry G. Korzun, Alexander V. Borodin, Ilya V. Paramonov, Andrey M. Vasilyev, and Sergey I. Balandin. 2015. Smart spaces enabled mobile healthcare services in internet of things environments. *Int. J. Embed. Real-Time Commun. Syst.* 6, 1 (2015), 1–27.

Ora Lassila and Ralph R. Swick. 1999. Resource description framework (RDF) model and syntax specification. https://www.w3.org/TR/PR-rdf-syntax/.

Fatiha Latfi, Bernard Lefebvre, and Céline Descheneaux. 2007. Ontology-based management of the telehealth smart home, dedicated to elderly in loss of cognitive autonomy. In *OWLED*, Vol. 258.

In Lee and Kyoochun Lee. 2015. The internet of things (IoT): Applications, investments, and challenges for enterprises. *Bus. Horiz.ons* 58, 4 (2015).

Ling Liu and M. Tamer Özsu. 2009. *Encyclopedia of Database Systems*. Vol. 6. Springer New York, NY.

Duo Lu, Dijiang Huang, Andrew Walenstein, and Deep Medhi. 2017a. A secure microservice framework for IoT. In *IEEE Symposium on Service-Oriented System Engineering (SOSE)*. IEEE, 9–18.

Duo Lu, Zhichao Li, and Dijiang Huang. 2017b. Platooning as a service of autonomous vehicles. In *IEEE 18th International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM)*. IEEE, 1–6.

Deborah L McGuinness, Frank Van Harmelen, et al. 2004. OWL web ontology language overview. *W3C Recomm.* 10, 10 (2004).

Brenda M. Michelson. 2006. Event-driven architecture overview. *Patric. Seyb. Group* 2, 12 (2006), 10–1571.

Roberto Minerva, Abyi Biru, and Domenico Rotondi. 2015. Towards a definition of the internet of things (IoT). *IEEE Internet Init.* 1 (2015), 1–86.

Sridipta Misra, Muthucumaru Maheswaran, and Salman Hashmi. 2017. *Security Challenges and Approaches in Internet of Things*. Springer.

Hessam Moeini, I-Ling Yen, and Farokh Bastani. 2017. Routing in IoT network for dynamic service discovery. In *IEEE 23rd International Conference on Parallel and Distributed Systems (ICPADS)*. IEEE, 360–367.

Hessam Moeini, Wenxi Zeng, I-Ling Yen, and Farokh Bastani. 2019. Toward data discovery in dynamic smart city applications. In *IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*. IEEE, 2572–2579.

Soulakshmee D. Nagowah, Hatem Ben Sta, and Baby Gobin-Rahimbux. 2018. An overview of semantic interoperability ontologies and frameworks for IoT. In *6th International Conference on Enterprise Systems (ES)*. IEEE, 82–89.

Eric Prud'hommeaux, Andy Seaborne. 2007. SPARQL query language for RDF. https://www.w3.org/TR/rdf-sparql-query/.

Nicolas Seydoux, Khalil Drira, Nathalie Hernandez, and Thierry Monteil. 2016. Autonomy through knowledge: How IoT-O supports the management of a connected apartment. In *Semantic Web Technologies for the Internet of Things (SWIT)*. CEUR-WS.

Long Sun, Yan Li, and Raheel Ahmed Memon. 2017. An open IoT framework based on microservices architecture. *China Commun.* 14, 2 (2017).

Ming Tao, Jinglong Zuo, Zhusong Liu, Aniello Castiglione, and Francesco Palmieri. 2018. Multi-layer cloud architectural model and ontology-based security service framework for IoT-based smart homes. *Fut. Gen. Comput. Syst.* 78 (2018), 1040–1051.

Omnia Abu Waraga, Meriem Bettayeb, Qassim Nasir, and Manar Abu Talib. 2020. Design and implementation of automated IoT security testbed. *Comput. Secur.* 88 (2020), 101648.

Received October 2019; revised August 2020; accepted May 2021